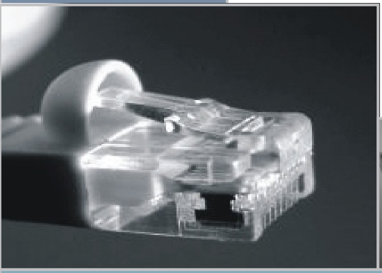
Network Architectures
and Services
NET 2010-08-1

IPTComm 2010

Proceedings of IPTComm 2010
Principles, Systems and Applications of IP Telecommunications

Location	Leibniz Supercomputing Center, Munich, Germany
Chairs	Georg Carle, Helmut Reiser
TPC Chairs	Gonzalo Camarillo, Vijay K. Gurbani
Sponsorship	Technically co-sponsored by ACM SIGCOMM and IFIP TC6 WG6.2
Organisation	Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Munich Chair for Network Architectures and Services, Department of Computer Science, Technische Universität München

Technische Universität München



Network Architectures
and Services
NET 2010-08-1

IPTComm 2010

Proceedings of IPTComm 2010
Principles, Systems and Applications of IP Telecommunications

August 2nd and 3rd, 2010
Leibniz Supercomputing Center, Munich, Germany

Editors:
Georg Carle, Helmut Reiser,
Gonzalo Camarillo, Vijay K. Gurbani

Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Munich

Chair for Network Architectures and services, Department of Computer Science, Technische Universität München

Technische Universität München

IPTComm 2010
Proceedings of IPTComm 2010
Principles, Systems and Applications of IP Telecommunications

Editors:

Georg Carle
Chair for Network Architectures and Services
Department of Computer Science
Technische Universität München
D-85748 Garching b. München, Germany
Email: carle@in.tum.de
Web: http://www.net.in.tum.de/~carle/

Helmut Reiser
Leibniz Supercomputing Center
Boltzmannstr. 1
D-85748 Garching b. München, Germany
Email: reiser@lrz.de
Web: http://www.lrz.de/~reiser/

Gonzalo Camarillo
Ericsson
Advanced Signalling Research Lab.
FIN-02420 Jorvas, Finland
Email: Gonzalo.Camarillo@ericsson.com
Web: http://users.piuha.net/gonzalo/

Vijay K. Gurbani
Bell Laboratories, Alcatel-Lucent
1960 Lucent Lane, Rm. 9C-533
Naperville, Illinois 60566, USA
Email: vkg@bell-labs.com
Web: http://ect.bell-labs.com/who/vkg/

Cataloging-in-Publication Data

IPTComm 2010
Proceedings of IPTComm 2010
Principles, Systems and Applications of IP Telecommunications
Munich, Germany August 2 and 3, 2010
Georg Carle, Helmut Reiser, Gonzalo Camarillo, Vijay K. Gurbani
ISBN: 3-937201-15-7

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
Network Architectures and Services NET 2010-08-1
Series Editor: Georg Carle, Technische Universität München, Germany

Preface

These are the proceedings of IPTComm 2010, the fourth of a successful series of conferences on Principles, Systems and Applications of IP Telecommunications. This year's edition of the conference is held in Garching b. München, Germany, on August 2 and 3, 2010.

The scope of the conference covers new services and service models, management and resilience, mobility, and a special focus on security.

The call for papers asked for submission of full papers, short papers. It attracted 50 paper submissions.

The technical program committee chairs ensured a rigid review process. For each paper, at least three reviews were received.

The technical programm committee decided to accept 12 papers as regular papers for the conference, resulting in an acceptance ratio of 24 %.

Additionally, 4 papers were selected to be presented as "Work-in-Progress" papers for presentation at the conference.

We are very grateful to the all members of the technical program committee and to all external reviewers for their hard work, in particular to the TPC co-chairs Gonzalo Camarillo and Vijay K. Gurbani, who made a tremendous effort, and who ensured that the conference selected highly attractive papers.

A separate call for industrial talks and demonstrations attracted a number of highly interesting submissions, of which the industrial talks and demonstrations committee selected 5 industry talks and 8 demonstrations.

The conference is hosted by the Leibniz Supercomputing Centre (LRZ) of the Bavarian Academy of Sciences and Humanities, which is the scientific computer centre for all Munich Universities and other research organizations within the greater area of Munich. It supplies more than 100.000 users with IT services and acts as an IT competence centre for all its customers.

We would like to thank all authors, conference organizers sponsors for their support of IPTComm 2010!

Munich, August 2010

Georg Carle

Helmut Reiser

Executive Commitee

Steering Committee Members

Gregory Bond (AT&T Research)
Dorgham Sisalem (Tekelec)
Saverio Niccolini (NEC Laboratories Europe)
Radu State (University of Luxembourg)
Henning Schulzrinne (Columbia University)

General Chairs

Georg Carle, Technische Universität München
Helmut Reiser, Leibniz Supercomputing Center, Munich

Technical Programm Committee Chairs

Gonzalo Camarillo, Ericsson Research
Vijay K. Gurbani, Bell Laboratories/Alcatel-Lucent

Technical Programm Committee

John Buford, *Avaya Labs Research*
Eric Chen, *NTT Corporation*
Eric Cheung, *AT&T Labs Research*
Tasos Dagiuklas, *Technological Educational Institute of Mesolonghi*
Carol Davids, *Illinois Institute of Technology*
Ali Fessi, *Technical University of Munich*
Rosario Garroppo, *University of Pisa*
Aniruddha Gokhale, *Vanderbilt University*
Swapna Gokhale, *University of Connecticut*
Carmen Guerrero, *University Carlos III of Madrid*
Christian Hoene, *University of Tübingen*
Alan Jeffrey, *Bell Laboratories, Alcatel-Lucent*
Cullen Jennings, *Cisco Systems*
Salvatore Loreto, *Ericsson*
Jouni Mäenpää, *Ericsson*
Enrico Marocco, *Telecom Italia*
Joerg Ott, *Helsinki University of Technology*
Victor P. Avila, *Acme Packets*
Joachim Posegga, *University of Passau*
Anand Prasad, *NEC Corporation*
Ivica Rimac, *Bell Laboratories, Alcatel-Lucent*
Ronaldo Salles, *Military Institute of Engineering (Brazil)*
Stefano Salsano, *University of Rome "Tor Vergata"*
Jan Seedorf, *NEC Laboratories Europe*
Jose Solar, *Technical University of Denmark*
Ivan Vidal, *University Carlos III of Madrid*
Xiaotao Wu, *Avaya Labs Research*
Pamela Zave, *AT&T Labs Research*

Conference Web Site

http://www.iptcomm.org

Table of Contents

Session 1: Security

Session 2: Deployment considerations and services architecture track (I)

Session 3: Performance of VoIP systems and networks

Session 4: Deployment considerations and services architecture track (II)

Session 5: Peer-to-Peer in IP Telephony

Session 6: Deployment considerations and services architecture track (III)

Introducing a Cross Federation Identity Solution for Converged Network Environments

Konstantinos Lampropoulos
University of Patras
Patras, Greece
+302610969863
klamprop@ece.upatras.gr

Daniel Diaz- Sanchez
University Carlos III of Madrid
Madrid, Spain
+34916246233
dds@it.uc3m.es

Florina Almenares
University Carlos III of Madrid
Madrid, Spain
+34916248799
florina@it.uc3m.es.

Peter Weik
Fraunhofer FOKUS
Berlin, Germany
+493034637196
peter.weik@fokus.fraunhofer.de

Spyros Denazis
University of Patras
Patras, Greece
+302610969863
sdena@upatras.gr

ABSTRACT

The Future Internet architecture, based on the integration of existing networks and services, and the addition of many new devices like sensors, face a series of important technical challenges, one of them being the management of diverse user identities. The diversity and plethora of the services and procedures affected by the unassociated existing user identities stress the necessity for a holistic solution to deal with the different aspects of the identity management problem. Existing efforts propose limited identity solutions that can only be applied within well defined boundaries and cannot extend their functionality to support converged network environments and service operations across different administrative domains. This paper presents a Dynamic Identity MApping N' Discovery System (DIMANDS) as a holistic identity solution for large scale heterogeneous network environments. This solution offers cross federation identity services and is based on a universal discovery mechanism which spans across different networks, layers and federations. It is also empowered with a unified trust framework which can collect and process diverse trust information to provide trust decisions on a widely accepted format.

Categories and Subject Descriptors

C.2.m [**Computer Systems Organization**]: Computer-Communication Networks – *Miscellaneous*

General Terms

Management, Design, Security.

Keywords

Identity Management, Trust Management, Privacy, Discovery.

1. INTRODUCTION

Future Internet promises to offer a unified network environment able to provide innovative services agnostic to the underlying infrastructure. To achieve this, current network concepts, must be redesigned, translated or mapped in such a way that diverse data may travel and be processed among different networks, contexts and administrative domains. Presently, users are owners of diverse identities and of unrelated identity information, valid and used in different contexts and for different purposes. We see this fragmentation as one of the major obstacles for developing cross-domain user-centric services in the Future Internet.

Despite the fact that Future Internet advocates network convergence, existing research efforts examine the identity problem partially, placing it in very specific and narrow contexts. Proposed solutions often implement identity frameworks which are usually applicable only within well defined administrative boundaries resulting in the creation of "Identity Management islands with interoperability issues" [1]. Cross-domain IdM systems have also been proposed to support environments with multiple co-operative providers and technologies (e.g. converged networks, clouds, federated testbeds etc), but the vast majority of them also suffer from the same symptom: they introduce customized identity formats; preconfigured trust/business relations; custom procedures. These peculiarities make them applicable only within the federation of domains (e.g. federation identities) thus any attempt for interoperating across federations becomes impossible. These practices have just shifted the problem from the isolation of domains to the isolation of federations and certainly away from network convergence.

The IdM problem in the Future Internet must be approached from a different perspective. Numerous domains, federations, cloud-hosted applications etc, which apply different identity schemes, adjusted to their internal procedures, will always exist e.g. for the Internet of Things which will connect not only users but additionally also huge amounts of sensors and slave-labour devices.

Enforcing new identity management systems or new identity formats which must be adopted by everyone is therefore not a feasible solution. Towards the Future Internet, the only way to address the IdM problem is to permit today's administrative domains to create customized identities, based on their needs and technologies, and support network convergence by creating the appropriate dynamic identity associations between these domains. Trust/business relations should dynamically flourish in an ad-hoc and autonomously way free of closed and centralized mechanisms. The proposed solution must act as the glue between the existing diverse identity concepts and support interoperability allowing diverse identity data to travel across different domains and federations. Such a solution can be realized only through an autonomous and independent system that exclusively provides identity services across different domains, federations, technologies and layers without affecting internal network procedures.

Motivated by the aforementioned arguments, this work proposes a Dynamic Identity MApping N' Discovery System (DIMANDS). DIMANDS is an autonomous and independent system designed to organize all kinds of identities that a single user may own as a member of various providers (e.g. government organizations, applications, service or network providers). With DIMANDS, the end user can form a dynamic online profile and allow third parties to automatically discover identity information about him irrespectively of the identity he is currently using or his network availability. The system is also able to setup and negotiate new trust relations breaking the trust staticity of current identity systems. Accordingly, the system does not collect or store any private identity data but points to authorized places that hold and manage this information.

The rest of the paper is organized as follows. In section 2 we present the current state of the art and in section 3 we describe our identity management and discovery framework. In section 4 we introduce our trust dynamic framework which deals with trust establishment inside and outside DIMANDS borders while in section 5 we present how the overall system supports critical cross domain identity issues. In section 6 we discuss security privacy and trust issues about DIMANDS. Section 7 presents the evaluation of our system and finally in section 8 we conclude this paper discussing open issues and future work.

2. PREVIOUS WORK

Identity management is an intensively researched topic in many academic, enterprise and standardization bodies.

Liberty Alliance [2] group has proposed Liberty Federation, a framework for federated identity management. Based on its specifications, OASIS formed the Security Assertion Markup Language (SAML) 2.0, an XML based standard for data exchange between Identity Provider and Service Providers. With federated identities, providers that reside in a Federation Group bind together various login identities assumed by one user. In this framework providers form static bindings between some of these identities, while there is no identity blinding since the closed Federation Group ensures the desired trust.

According to the OpenID [3] proposed solution, when a user contacts an OpenID-enabled web site instead of his username, he inserts a URL. The OpenID site redirects the user to a site that corresponds to the submitted URL, which in turn, performs the user login operation. OpenID is fast becoming the de-facto solution for secure login in the internet as it is user-friendly, user centric and supports features like Single Sign On (SSO). Nevertheless, this approach (due to the required URL login), is restricted only to application layer identity solutions excluding integration with lower layer identities. Its functionalities are limited to user authentication while its framework does not support management of multiple identities.

Information Cards [4], a Microsoft-initiated solution, is a user-centric mechanism to store and manage online identities. Using a client software, users can create, delete, and modify the identity profiles (Information Cards) they use in the network thus controlling the kind and amount of information revealed in the network. With this approach however, features like cross-layer SSO are not supported while its architecture does not link the different names assumed by one user. Privacy is granted through identity isolation and there is no formation of a single network entity to provide cross-layer identity solutions in the NGN.

Project Higgins [5] is another identity framework which unifies all identity interactions across multiple heterogeneous systems through a common user interface metaphor also called Information Cards (i-cards). The represented identities (Digital Subjects) and their Identity Attributes are exposed in a Context through a data model, described by OWL. It is a user centric system with the end users having a single point of control over multiple heterogeneous identities preferences and relationships. Even though the project presents a unified representation for every type of identity, the identity framework is again limited in the application layer identities and attributes without making clear how lower level protocols may be able to comprehend and adopt it.

The DAIDALOS Project [6] proposes a cross layer identity management system based on the management of all different profiles a user may have in the network. These profiles are linked together into multiple groups, creating the so called Virtual Identities (VIDs). Two or more VIDs cannot be associated with each other, providing strong privacy and security. Project SWIFT [7], a European Union funded project started in 2008, is also based on VIDs. Formation and submission of VIDs though, is a static procedure. Every new service has its unique restrictions and requirements and users must always create, organize and remember numerous VIDs. A complete unified network entity cannot be constructed and existing identities are replaced with VIDs forcing all systems to adopt handle and transport this new type of identity.

Project PRIME [8] defines a system which consists of two parts: a service-side module which mainly provides access control functionalities, and handles trust and policy negotiations and a user-side module (PRIME middleware) which runs on the user's computer. The PRIME management console enables the user to control the disclosure of his personal data. PRIMElife [9] is the follow up project launched in 2008. These projects offer many solutions in identity management, especially in privacy. But again their proposal rejects identity convergence for security reasons, not making clear how it can be integrated in the NGN environment.

Other identity systems designed to face the identity challenges are Shibboleth [10] and Athens [11]. Both these systems propose solutions through a secure login procedure but like the majority of the research are limited to application layer identities.

In [12] authors acknowledge the need for a widely deployed IdM system that must encompass identities of different layers, formats and business areas. This work describes a valid discovery mechanism but all network interactions assume end-users' intervention and require from them to take all the final decisions about who they should trust and the amount of identity information they are willing to reveal.

Within standardization bodies, ETSI is developing the Universal Communication Identifier (UCI) [13]. This identifier is bound to a Personal User Agent (PUA) that negotiates with other PUAs to deliver communication services between two end parties. UCIs are globally unique identifiers that can be solely resolved to a unique resource where a complete user profile resides. UCI is a cross layer proposal for the identity management system that is mapped to the NGN requirements. It requires though the use of a unique identifier, imposing the need for modification to existing systems and procedures, something that is not easily acceptable especially in protocols levels lower than the application. Furthermore the solution is restricted within a single domain, and it is not clear if and how external identities can be authenticated and then adopted by the system. Also the recently founded industry specification group Identity Management for Networks and Services (INS) of ETSI does not address the aspect of cross-layer identity resolution.

The Focus Group on Identity Management (FG IdM) of ITU-T NGN GSI (SG13) is designing the NGN User Identity (NUI) [1], a new type of identity that will meet a series of NGN requirements and also provide means for identification, authentication, ubiquitous access to network and services, profile organization etc. NUI may include a public user identity for communication with other NGN users, and a private user identity for networking purposes like authentication with providers. Until now current specifications from FG IdM outline the current state of the identity management system presenting existing identity management solutions, also indicating scenarios to target areas and gaps that still remain unsolved.

Finally the Kantara Initiative [14], an evolution of Liberty Alliance, is an effort to address the identity problem in a much wider landscape. But as stated in [15] "*the reality is that it is one more layer of bureaucracy on top of already top-heavy structures*". Kantara proposes that smaller Identity solutions and also the persons creating these projects should become members of its large identity framework. Thus the identity management solution is again based on the formation of large scale trust group something that has already proved to be insufficient.

The main problem behind current initiatives is their inability to transport and manage identity data across predefined trust areas (e.g. federations) thus they are incapable to expand their functionality to unconditionally support converged network environments.

3. DIMANDS ARCHITECTURE

The DIMANDS architecture is based on an innovative Distribute Hash Table (DHT) overlay infrastructure which combines the routing capabilities of DHT networks and the security benefits of individual Identity Providers (IdPs). The basic characteristics of DIMANDS overlay are:

- Only nodes (individual IdPs) exist in DIMANDS overlay and not objects (identity data).
- The overlay is used only for routing purposes (and not for storage of any kind of data)
- Participating nodes cannot change their position in the overlay.

DIMANDS infrastructure (Figure 1) is formed by a number of CHORD [16] circles, placed on top of each other to create a cylindrical overlay. This cylinder forms a torus, meaning that the upper circle and the lower circle can directly communicate with each other irrespective of their location in the torus. Newly created circles can be placed anywhere in the cylinder allowing DIMANDS to expand infinitely. One non-profit global organization (e.g. ICANN) will have the responsibility to define and distribute these circles. This organization will have no further involvement in DIMANDS functionality or its stored information.

Each CHORD circle represents a predefined geographical area (e.g. a country) and is assigned to a Regional Authority (RA) that resides in this area (e.g. government). RAs' responsibility is to check, evaluate, provide the required security credentials and finally monitor the behavior of the IdPs that participate in the CHORD circle of their geographic area. The RAs will have no further involvement in DIMANDS functionality or DIMANDS stored information.

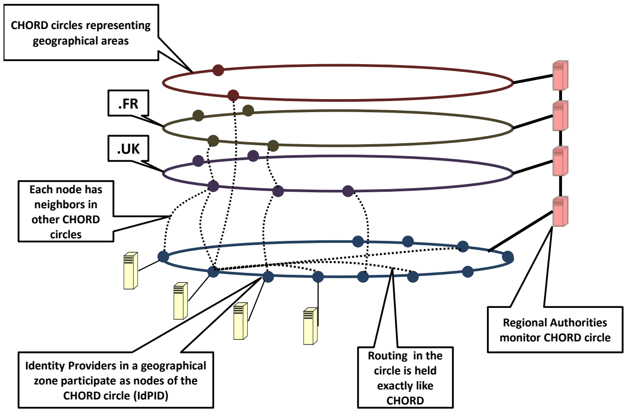

CHORD circles representing geographical areas
.FR
.UK
Each node has neighbors in other CHORD circles
Identity Providers in a geographical zone participate as nodes of the CHORD circle (IdPID)
Routing in the circle is held exactly like CHORD
Regional Authorities monitor CHORD circle

Figure 1. DIMANDS overlay.

DIMANDS' functionality is deployed through independent Identity Providers. Any kind of provider, organization, authority etc that wishes to join DIMANDS as an IdP, must first be evaluated by RA of its geographical zone. Upon successful validation, this IdP joins, as a new node, the corresponding CHORD circle and reserves a specific and permanent point. This point is called Identity Provider ID (IdPID) and it is a numeric value which indicates its position in the overlay. CHORD overlay structure ensures uniform distribution on the circle for practically an infinite number of nodes.

Each overlay node (IdP) sustains a number of neighbors and constantly maintains secure connections with each one of them. These neighbors exist in its own circle and in other circles and their number may vary based on system's size and traffic. Selecting neighbors in the same circle is carried out exactly like CHORD. To be able to select neighbors in different circles a node requests from the corresponding RA a list, containing the IdPIDs of the nodes that exist in the same point like itself in the rest of the CHORD circles (or close to that point). Since DIMANDS overlay is a torus, the nodes in this list form a virtual vertical circle thus the selection of neighbors in other circles is held again exactly like CHORD. To deliver a message destined to a specific point on the overlay, a node must simply forward it vertically to a node in another circle or horizontally to a node in its own circle using CHORD routing.

The proposed architecture is designed to satisfy a number of key requirements in terms of performance, privacy, trust and security. DIMANDS is based on a hierarchical 3-level architecture where multiple organizations and providers contribute and assume well defined and separate roles, without being able to access data or functions that are not supposed to. In the two top levels of DIMANDS architecture reside distinguished organizations (ICANN, Regional Authorities) which only purpose is to provide the necessary orchestration to the IdPs that host DIMANDS actual functionality and compose the lower third level of the DIMANDS architecture. End-users' private data are safely stored in the IdP of their choice without being accessible to any of the organizations of the two upper levels.

The DHT architecture offers minimal management overhead and robustness to the system and was selected over other centralized or hierarchical architectures to support the hard trust and security requirements of an IdM system. Nodes (IdPs) in the overlay will rarely join or leave DIMANDS. Thus each node will sustain stationary neighbors and build long term trust relations with them. Any message, destined to a node in DIMANDS, **will safely travel through a path of trusted neighbors**. In any other centralized of hierarchical architecture the destination node would have to process incoming messages from unknown sources without always being able to validate them. Furthermore, in DIMANDS overlay neighboring nodes exchange data through secure connections, thus providing high levels of security to the system. In any other centralized of hierarchical architecture it is impossible to store credentials and establish secure connections between all of the participating IdPs to securely exchange identity information. New circles and nodes can constantly join the overlay without affecting the functionality or the architecture of the system. Failure of a node has only temporary and local effects and does not affect the overall system. Its neighbors may temporarily route messages from alternative paths, and there is no data loss since each node is responsible for maintaining the data of its own users.

Assigning CHORD circles to specific geographical areas provides locality to DIMANDS overlay and advances system's performance. End-users' identities will mainly exist in providers located in their geographical area (e-government, e-health, telco services) thus message exchanging between entities that reside in large distances in the actual network is minimized. Better locality can be achieved, if the nodes in each geographical circle are organized in a locality aware CHORD. It must be clarified that CHORD was selected among other existing DHTs due to its ability to improve locality by only modifying the neighbors on each node on the circle and not the position of the node, thus satisfying DIMANDS requirement for stationary nodes.

3.1 User Account

In DIMANDS a user may select the IdP he prefers and trust the most, and create an account. Each DIMANDS account is defined by an identifier called User account ID (UsID) which is a numeric value assigned and known only by the IdP. Linked to this UsID exists a unique database where the user can store the identity information he wishes (Figure 2). This database has four fields. The “RID” and “TRID” fields which hold representations of all user’s identities, the “Domain Name” field where all the domains that host the profiles for the corresponding user’s identities are stored and a variable set of fields called “Attributes”.

DIMANDS obfuscates the real identities and replaces them with Random Identity Numbers (RIDs). An RID is a unique representation of a single identity and is stored in both DIMANDS’ IdP and the organization’s (the one that holds the profile for this specific identity) databases. Any communication between the organization and DIMANDS regarding an identity is carried out by using the corresponding RID, providing robust identity obfuscation.

UsID: 1311652232			
TRID	**RID**	**Domain Name**	***Attributes***
35124	1234.6334124463	umts.com	***Service:*** Tel, video ***Network Authorization:*** Yes
61234	1234.1254576222	telco.com	***Service:*** Tel, video, IM ***Network Authorization:*** Yes
78123	1234.6982145781	gov.uk	***User Validation:*** Yes ***Age Validation:*** Yes ***Location Validation:*** Yes

Figure 2. The database stored in DIMANDS’ User Account.

RIDs are formed as the concatenation of the Identity Provider’s IdPID (L bits) and the output of a private key cryptographic function (like e.g. AES or RC4) that receives as input a number composed of the user’s UsID in the higher N bits, followed by a consecutive number, in the lower M bits (N+M bits).

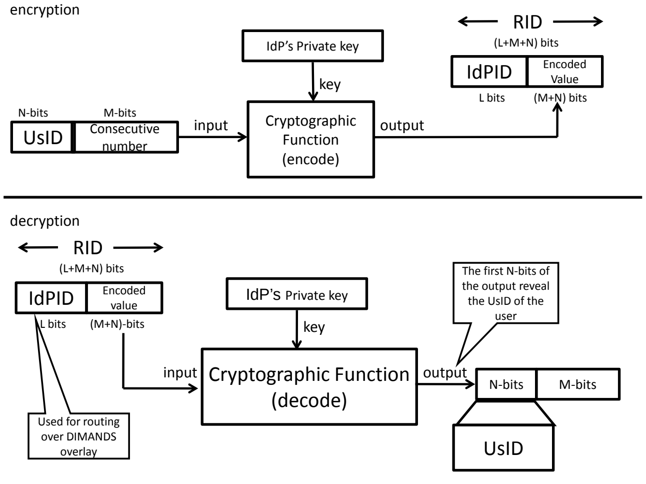


Figure 3. Random Identity Number (RID).

The use of the consecutive number is to produce 2^M random and uncorrelated RID’s making it impossible for someone to link network identities with real end-users. Using the first L bits of an RID contained in a request, DIMANDS may route on the overlay and locate the corresponding IdP that this request is targeted to. Since DIMANDS nodes are stationary, the request will always be transmitted to the correct IdP. This IdP, by decoding the remaining (M+N) bits of the RID (using its own cryptographic key), may retrieve user’s UsID, and thus identify the user that the request is referred to (Figure 3). It must be noted that decoding an RID reveals an end-user and not the identity that represents. A TRID is an identifier generated by the organization (the one that holds the profile for this specific identity) and corresponds to a specific RID. It has been introduced to avoid revealing the RID value outside DIMANDS borders. Any communication between the organization and an entity (other organization, provider etc) outside DIMANDS regarding an identity is carried out using the corresponding TRID. The RID and TIRD values are **NOT** introduced as new global identifiers and remain agnostic internal network procedures. The providers and DIMANDS exchange the RIDs or TRIDs only in the IP network, which in turn can be mapped on to existing internal local identifiers.

The “Attributes” field is an expandable set of optional fields that contains descriptions about the stored RIDs. The “Attributes” field does **NOT** contain any privacy or security data that may expose user’s identity information. For instance the Attribute “Service” describes what service can this RID support and not any information e.g. about user authentication to this service. The Attribute “Age Validation” indicates that the corresponding provider - in the “Domain Name” field - can validate user’s age but it does not contain user’s actual age. No real identity data reside in this database.

3.2 DIMANDS-Client

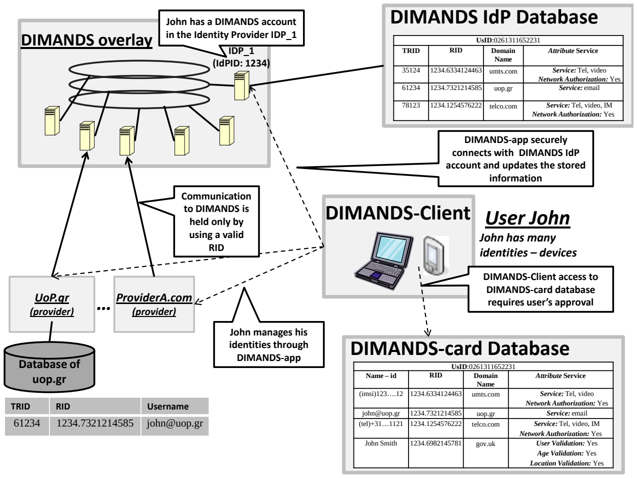
The DIMANDS-Client is a web-based application designed to help the end-users to manage their digital identity data with DIMANDS IdPs from many end devices like e.g. PCs, netbooks or mobile phones. Access to this application is governed by a hardware token, the DIMANDS-card. DIMANDS-card can be a smart card activated by a PIN i.e. a mobile phone Universal Integrated Circuit Card. End-users may download and install the DIMANDS-Client application locally in one or more of their end-devices and before launching the app the end-user is asked to connect to the end-device his DIMANDS-card and insert the correct credentials or PIN.

In the DIMANDS-card an encrypted database is stored which holds the same identity data as the database in the DIMANDS IdP but instead of the value TRID, it contains a field with users’ real identities. Based on this database the end-users can see their actual identities and perform basic functions and procedures like registration of new identities, deletion or modification of existing data, creation of new rules etc. After completing the management of their identity data the users may upload to their IdP account the updated database and the specific rules and policies that organize the data stored in it. This upload is held over a secure connection between DIMANDS-Client and the IdP.

The responsibility for developing and distributing the DIMANDS-clients and associated DIMANDS-cards is assigned to the IdPs that participate in the DIMANDS infrastructure. Both of these components must meet a predefined and very specific set of obligatory security and privacy requirements. Each IdP may modify them only by adding improvements in security. The exact interface of the application and its full capabilities are not described since they are outside of the scope of this paper.

3.3 DIMANDS overall architecture

Figure 4 depicts how the different DIMANDS components and elements thereof form the overall DIMANDS architecture, and how this architecture interacts with other external entities like service providers.


DIMANDS overlay
John has a DIMANDS account in the Identity Provider IDP_1
IDP_1 (IdPID: 1234)
DIMANDS IdP Database
DIMANDS-app securely connects with DIMANDS IdP account and updates the stored information
Communication to DIMANDS is held only by using a valid RID
DIMANDS-Client
User John
John has many identities – devices
DIMANDS-Client access to DIMANDS-card database requires user's approval
UoP.gr (provider)
...
ProviderA.com (provider)
John manages his identities through DIMANDS-app
Database of uop.gr
DIMANDS-card Database

UsID:0261311652231			
TRID	RID	Domain Name	*Attribute* Service
35124	1234.6334124463	umts.com	***Service:*** Tel, video ***Network Authorization:*** Yes
61234	1234.7321214585	uop.gr	***Service:*** email
78123	1234.1254576222	telco.com	***Service:*** Tel, video, IM ***Network Authorization:*** Yes

TRID	RID	Username
61234	1234.7321214585	john@uop.gr

UsID:0261311652231			
Name – id	RID	Domain Name	*Attribute* Service
(imsi)123….12	1234.6334124463	umts.com	***Service:*** Tel, video ***Network Authorization:*** Yes
john@uop.gr	1234.7321214585	uop.gr	***Service:*** email
(tel)+31…1121	1234.1254576222	telco.com	***Service:*** Tel, video, IM ***Network Authorization:*** Yes
John Smith	1234.6982145781	gov.uk	***User Validation:*** Yes ***Age Validation:*** Yes ***Location Validation:*** Yes

Figure 4. DIMANDS overall architecture.

Only validated providers may issue requests to retrieve information from DIMANDS. Requests can be sent to any one of the DIMANDS' servers, and then forwarded to their final destination. DIMANDS seeks to be a totally independent system thus discovery of a DIMANDS server is not held through DNS (Domain Name System). Each service provider, organization or any other entity that wants to submit requests to DIMANDS must acquire a list of available DIMANDS servers only by the RA of its geographical area and establish a long term secure connection with one of them. The long term connection is required for two reasons. The first one is to minimize traffic to RAs for the retrieval of available DIMANDS nodes and the second one is that long term connections build gradual and sufficient trust relations between DIMANDS nodes and outside entities that ask for information, thus enhancing system's security. It must be noted that the response of a submitted request is returned back by the DIMANDS node that sustains a long term secure connection with the requester, and not by the IdP in the overlay that actually processed the request. This is required for security reasons to avoid man in the middle attacks.

3.4 New Identity Registration

Each time a user creates a new account in a service provider or an organization, a profile with a username is created in the corresponding provider's database. If the user wants to register this new username to DIMANDS he must complete the following procedure (The provider must have a Web Page compatible with DIMANDS architecture).

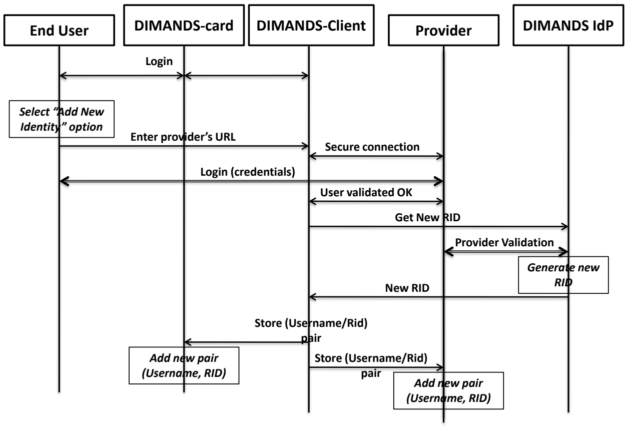


Figure 5. New Identity Registration message flow.

The user logs in to the DIMANDS-Client and selects the "Register new identity" option (Figure 5). He is then asked to insert the URL of the provider that holds the profile for the corresponding username. The DIMANDS-client establishes a secure connection with provider's Web Page and the user is asked to undergo a second login in provider's Web Page to prove that he is the legitimate owner of the username he wants to register. (Credentials for the second login must have been provided to the user with the creation of his new account). As soon as the user is validated, DIMANDS-client communicates with the IdP and requests the generation of a new RID. Before the IdP produces the new RID it is mandatory to validate the provider that holds the profile for the corresponding username. This validation is required to ensure that no malicious party registers in DIMANDS false links or data to perform phishing attacks. After the provider is validated a new RID is generated and transmitted back to the DIMANDS-client which binds this new RID with the username and stores it in the local encrypted database (without informing the IdP of the username or its binding with the generated RID). Finally, DIMANDS-client transmits the RID/username pair to the provider to be stored in its database too. This procedure must be held only once. The provider must then generate and frequently update the TRID for this identity.

4. TRUST FRAMEWORK

Identity discovery is the first step in every single interaction but afterwards there is another issue to solve, the lack of dynamic trust support.

IdM systems manage trust in different ways as stated in [17] or [18], but it is always handled in a very static fashion. For instance, SAML employs pre-existing trust relationship, by means of PKI, between the Relying Party and the Attribute Provider [19]. Shibboleth inherits from SAML this model. Thus, Shibboleth federations imply the aggregation of large lists of providers that agree to use common rules and contracts. The process might require human intervention being even more rigid. The drawbacks of this kind of trust model are well known: hard to deploy and maintain, and high dependence on central authorities [20].

OpenID did not consider trust in the beginning (trust-all-comers model). However, a new OpenID extension called PAPE (Provider Authentication Policy Extension) [21] has been approved in order to enforce trust mechanisms. PAPE provides means for RPs and OpenID Providers to request and advertise previously agreed policies. Others, as WS-Federation and Liberty Alliance, resemble PKI trust models between Certification Authorities (CAs). These models are typically implemented by means of trust lists containing trustworthy authorities that are manually configured by an administrator.

To overcome the aforementioned trust staticity, two different paradigms are applied in the trust framework: trust management and trust negotiation. So, flexibility in trust is provided while security and privacy are guaranteed. In a high dynamic ecosystem, as Future Internet would be, trust might be handled mimicking humans' behavior, considering though the history of interactions, the environment, and the scope to derive trust levels for every request.

The trust framework does not store data in DIMANDS user's accounts and is not combined with the identity organization and discovery mechanism. In fact, our aim is to minimize the dependence on central authorities or previous configuration to allow entities to be more autonomous and capable of making P2P trust decisions.

DIMANDS exploits this framework's functionality to address the trust issues that arise inside and outside its borders. In this way, interactions between providers, IdPs, and users are seamlessly achieved. Such interactions could imply bridging trust models across disparate domains, as well as negotiating several options such as protocols, multiple identifiers, flexible attributes, a common set of policies, obligations, and procedures regarding access control, information disclosure or treatment, etc. No trust data must exist in user's DIMANDS accounts, in order for the system to meet its hard privacy and security requirements.

The system provides trust services to allow new comers as, IdP, service providers or external entities, to negotiate trust by exchanging requirements and credentials. The trust framework can be instantiated in every single entity willing to participate, can be shared by several entities, or used as a trust broker for an administrative domain.

The following sections describe our trust framework. The Pervasive Trust Manager is in charge of handling trust in stationary state, in other words, maintains a trust relation after it has been set up. Existing trust relations are monitored using evidences, context and preferences dealing with cooperative attacks, botnets and virus. The Pervasive Trust Negotiation module is the bootstrapping module. It handles interactions with estrangers deriving a trust relation for the first time, enables dynamic trust establishment to increase privileges or handles important context (or preferences) changes that require a new relation to be established. Both modules manage the risk associated to establish and manage relationships with a certain uncertainty degree.

4.1 Pervasive Trust Negotiation

This module assists entities to select policies (requirements), credentials, and resources to disclose, according to strategies, preferences, and context. The objective is to achieve a fair P2P trust negotiation. The module uses a human-mimicking decision engine able to simplify problems. Moreover, if a human is involved in the process, the module can graphically present those problems in comprehensive way allowing he/she to understand what is happening despite his/her technical training.

In order to authenticate and authorize estrangers, trust negotiation rely on the fact that any resource is protected by a policy that express which credential(s) should be disclosed to obtain access to it. [22] describes the requirements that trust negotiation systems should cope with. Requirements should be disclosed gradually, according to the level of trust reached until the moment, since they might contain sensible information [23]. However, to protect entities against rogue or greedy peers, that harvest unnecessary credentials from others, the process should be driven by a decision engine.

The trust framework is agnostic in terms of policy or credential language and encoding. The negotiation is governed by policies from different editors that protect resources. A resource can be protected by several policies and a policy can protect several resources. For that reason, policies are split into parts that are called "policy items". A policy item is a formal definition, therefore expressed with adequate semantics, for a requirement. The formal definition of a resource, guarantees that other peers would be able to find out which credential(s) should be disclosed in order to satisfy it. Finally, a resource is any information, service, mechanism or credential, in general any object, which its disclosure implies a risk. Policy items are considered also resources since it disclosure might be dangerous.

To drive the credential exchange towards a fair negotiation, for either bringing up a new trust relation or to increase privileges, we place every object (resources or policy items), subject to be part of the decision, into a decision set, as shown in Figure 6. The decision set is composed by object descriptions that are collections of attributes. In a highly dynamic scenario, is unfeasible to find a common model to describe every object, especially if objects can be different in nature. Thus, the amount of attributes used to describe objects depends on the nature and the available information. The Pervasive Trust Negotiation module fetches then context and preferences the environment. It derives an agnostic decision set from the input by transforming attributes using generic attribute taxonomy. The taxonomy is able to cope with quantitative, ordinal and membership data.

Once the engine derives the generic attribute taxonomy, it includes a virtual object called “status” into the decision space. The status object contains attributes, according the derived taxonomy, which reflect the information known up to the moment, as disclosed credentials, environment information, and preferences. Then the engine computes dissimilarities among entities and produces a dissimilarity matrix, similar to a covariance matrix.

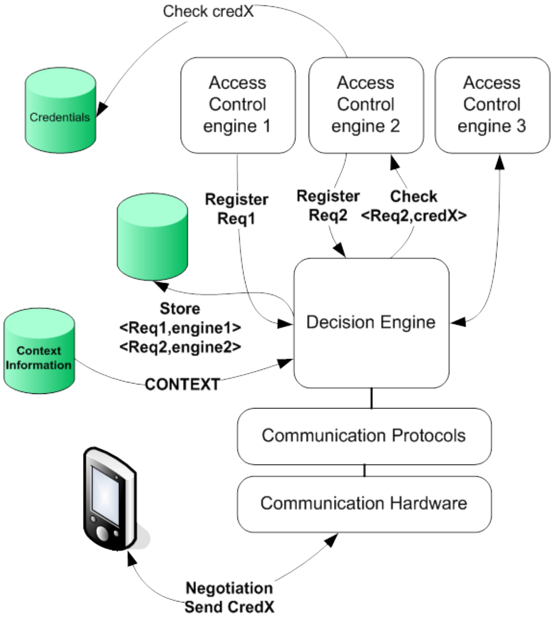


Figure 6. Pervasive Trust Negotiation Engine

The Pervasive Trust Negotiation module uses multivariate statistics to simplify the problem by reducing the dimensionality of the input space to a common set of attributes. To achieve that, the engine process the matrix with ALSCAL Multidimensional Scaling (MDS) [24], which uses alternate least squares, together with weighted dissimilarities, to combine both metric and non-metric analysis. The algorithm deals with spare matrixes (with different number of attributes) so it is suitable for the problem we face with highly dynamic environments (absence of some data).

In order to find out the next credential to exchange or the next requirement to commit, the Pervasive Trust Negotiation engine takes the simplified decision space and measure the distance between any object and the status object. This distance is directly interpreted as a measure of risk. After a successful requirement fulfillment, a change on the preferences or environment data, the engine computes again the dissimilarity matrix, simplifies the problem, and computes the risk again obtaining the next step.

MDS has been successfully used to solve similar complex problems as classifying music [25], or selecting the most appropriate network from a heterogeneous set [26]. When applied to trust negotiation, it provides robustness (works in the absence of data) and agnosticism (works with any object) if compared to Petry Network driven trust negotiation [27]. Compared to solutions that make policies publicly available but obfuscate them including fake policies [28], our system make policies private and release them according to a risk model, thus it does not suffer from cooperative attacks that end up finding the fake policies. Other solutions define their own language as [29] and make peers to exchange complex graphs but they lack of agnosticism or the context is not well addressed.

4.2 Pervasive Trust Manager

This module is responsible for managing internal and external trust information, and maintaining dynamically the trust and distrust lists updated. The internal trust information is obtained from a local repository, which contains data related to the entities’ behavior. On the other hand, external trust information is obtained from trusted third parties (TTPs), making use of the common knowledge by means of requesting and collecting reputation information, so maintaining a history of the interactions and collecting recommendations from other entities.

From such trust information, the Trust Manager models trust evolution over time, as it has a clear impact in risk management and trust decisions. Trust evolution represents that trust learning is gradual, subjective and dynamic. This takes into account the environment as well as historical evidences or reputation information. So, this module is enriched with more complex functionality such as risk and policies management, and uses of techniques applied to cooperation and collaboration models. These advanced functionalities allow considering timing, analysis of cached trust material, update to policies, agreements fulfillment, etc., in order to achieve a better trust management.

The trust metrics used are specified in order to make easier the mapping from different trust models applied to every domain similar to the problem of identity formats.

5. DISCOVERY – IDENTITY CONVERGENCE

Being an independent and autonomous entity, DIMANDS has the ability to provide cross domain/federation identity services. Figure 7 presents a cross domain service delivery scenario that requires the co-operation of providers which reside in multiple and unassociated federations. User "user@webstore.com" logs on his account in provider "webstore.com" and requests a specific service (e.g. an online purchase). In order to complete the transaction, provider "webstore.com" must contact other providers (e.g. paypal.com, supplier.com etc) which all participate in the federation F1.

The "supplier.com" provider though needs to validate user's age information against an entity that might not be part of the federation F1 but that should be trusted.

Existing literature fails to support the above operation because the "supplier.com" cannot autonomously discover where the desired information resides (is restricted to use only information that exists in the federation or the cloud) and even if it somehow knew that the information existed in the gov.com" organization, the username "user@webstore.com" means nothing to "gov.com".

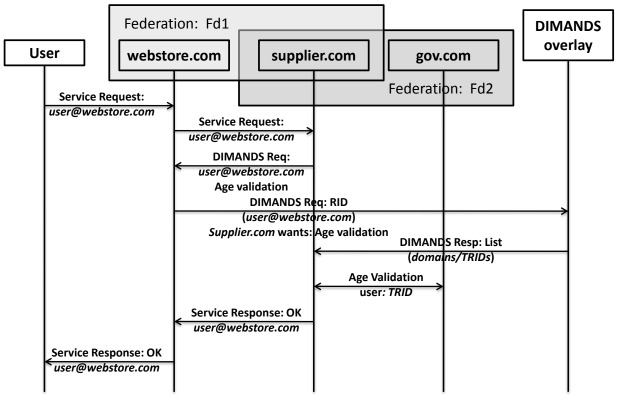

Federation: Fd1
User
webstore.com
supplier.com
gov.com
DIMANDS overlay
Federation: Fd2
Service Request: *user@webstore.com*
Service Request: *user@webstore.com*
DIMANDS Req: *user@webstore.com* Age validation
DIMANDS Req: RID (*user@webstore.com*) *Supplier.com* wants: Age validation
DIMANDS Resp: List (*domains/TRIDs*)
Age Validation user*: TRID*
Service Response: OK *user@webstore.com*
Service Response: OK *user@webstore.com*

Figure 7. Cross federation service delivery using DIMANDS.

Using DIMANDS functionality, the provider "supplier.com" constructs a DIMANDS request asking for age validation. Since it has no knowledge of a valid RID (the user does not have an active account in supplier.com) to directly contact DIMANDS, the request is transmitted to the provider "webstore.com", which in turn retrieves from its database the corresponding RID and forwards the request to DIMANDS on behalf of the "supplier.com". DIMANDS validates that the "supplier.com" is a legitimate provider and responds a message containing a list with domains, which can validate user's age and a number of TRIDs. The "supplier.com" provider receives the list and among the containing providers chooses the "gov.com" organization that trusts the most (participate in a different federation). Using the corresponding TRID, can now directly contact the organization and acquire the desired information

However, it cannot be assumed that the above list will always contain trusted providers. Accordingly, if the "supplier.com" and the organization and "gov.com" had no previous trust relation and interacted for the very first time, they should set up a trust relation dynamically.

To accomplish that task based on our proposed trust framework, both entities would engage in a trust negotiation. For instance, "supplier.com" derives its dissimilarity matrix and looks for the next step in the negotiation. It sends a basic authentication requirement to "gov.com". "gov.com" authenticates with a PKI certificate and requests authentication to "supplier.com". After the basic authentication phase, both entities know they are talking to the appropriate machine. According to "supplier.com" policy, "gov.com" must demonstrate that is a valid source for verifying the age. Nevertheless, before going further, "gov.com" calculates its decision space and requests "supplier.com" to send a credential asserting it is a valid supplier and requests it to agree on the privacy policy.

In this way, "gov.com" protects itself from rogue peers. Once "supplier.com" fulfills those requirements, "gov.com" sends a set of attributes within a SAML sheet that demonstrates it is a valid source and validates the age. Both entities will convey their brand new trust relations to the trust management module so next interactions, under the same scope, would be faster.

6. SECURITY, PRIVACY & TRUST ANALYSIS

Security: Even though DIMANDS is formed by multiple individual IdPs its architecture provides maximum security to the system since all communication, is held over secure connections. Encryption for all messages provides data integrity and monitor of the participants' behavior by the RAs protects the system from internal malicious parties. Monitoring the IdPs of a geographical area may be held through our proposed Pervasive Trust Manager, part of the trust framework. Thus, each IdP might be evaluated by its neighbors and inappropriate actions would be reported to the RAs to generate evidences for future trust decisions. Any kind of unaccepted behavior from an IdP may result in its permanent removal from the system. Only validated providers can register or acquire information from DIMANDS system. This validation can be achieved through certificates and ensures that no malicious entity will have the ability to insert false links or data to perform e.g. phishing attacks, or illegally collect identity information. Misbehaving outside entities may be denied of DIMANDS services. DIMANDS-Client is one of the most essential parts in DIMANDS architecture thus any security flaw may cause serious data exposure. Firstly it must be ensured that the application cannot be modified and distributed from any unauthorized party. Its distribution should be performed only by the IdPs through a secure site which supports SSL client side authentication. As mentioned above, DIMANDS-Client is a web-interfaced application. The web interface advances the usability of the system making it more user-friendly. However this imposes many critical security threats e.g. cross-script attacks. DIMANDS-Client cannot be developed on top of existing browsers because the system will constantly be subjected to the plethora of different security flaws that these browsers may have. Thus we propose that DIMANDS-Client should be an independent application designed in such a way that meets specific security requirements, capable of supporting generic web browsing, used only for DIMANDS purposes. Furthermore for DIMANDS-card, it must be ensured that its connection with DIMANDS-Client is absolutely secure. Many secure frameworks exist in literature that provide security and data integrity for the use of smart cards. Mutual PKI (Private Key Infrastructure) authentication is mandatory and frequent re-authentication is advised, to ensure that DIMANDS-card is connected with a valid and secure DIMANDS-Client. Finally, for the protection of DIMANDS from threads that apply in all large scale networks e.g. DDoS attacks, many security solutions exist in literature that can be exploited based on the nature of the attack. This kind of security analysis is outside of scope of this paper.

Privacy: Considering privacy, the 3-level architecture of DIMANDS ensures that each user's data will only remain under the control of the Identity Provider of his choice, and will not be accessible by any other entity in the system (e.g. ICANN or RAs). The obfuscation of the real identities (with the use of RIDs and TRIDs) ensures maximum security from monitoring and data collection even within DIMANDS. However the strongest part of DIMANDS is the nature of the stored information. DIMANDS holds irresolvable identity mappings and descriptions of their capabilities. No actual identities or identity data exist in DIMANDS. The actual information are distributed and well protected in the providers and the organizations that issued user's real identities.

Trust: DIMANDS assigns CHORD circles in geographical areas to advance trust with its users. End-users most likely will trust IdPs that reside in the same geographic region, share the same culture, and obey the same security policies and rules enforced by a locally distinguished Authority. Of course the strongest evidence on why should the end-users trust DIMANDS, is the fact that stored information in DIMANDS do not carry any actual identities or identity data and cannot lead to personal information exposure. DIMANDS proves its trustworthiness to the IdPs that participate in its infrastructure and the outside providers - organizations that exchange information with, through the careful design of its architecture that ensures maximum privacy and security.

The long-term connections between DIMANDS nodes and external providers combined with the trust relations between DIMANDS overlay neighbors provide high levels of trust and security for the overall system. Furthermore with the adoption of our proposed trust framework that provides trust services to allow new comers (e.g. IdPs, service providers or external entities) to negotiate trust by exchanging requirements and credentials DIMANDS succeeds in supporting the essential issue of trust not only inside, but also outside its borders

7. EVALUATION

DIMANDS' overlay is formed by servers connected by means of a DHT overlay. Organizing a system as a DHT though, imposes additional packet delay and traffic load because of the generated requests due to the overlay routing. For the reliability of our measurements we used a widely accepted simulator tool, the OPNET Modeler v.14 and real round trip time measurements taken from the Meridian King data set which provides RTT measurements among 2000 nodes and reflect RTT latencies among globally distributed DNS servers.

In our scenario each node in DIMANDS randomly accepts requests destined to a random node in the overlay. Once a request reaches its destination a response is generated and transmitted back to the node that initially accepted the request. Even though this response can be directly transmitted back to the initial node, our scenario examines the worst possible (and absolutely secure) scenario where DIMANDS responses **follow back the same overlay path of trusted nodes as the requests**. The measured values in all the examined scenarios were the number of hops for message delivery in DIMANDS overlay and message delay for the same process.

Two kinds of evaluations were performed. The purpose of the first evaluation was to investigate if the performance of DIMANDS is affected by the number of CHORD circles or the distribution of the IdPs in the overlay. Two different overlays were created. In the first structure 2000 nodes were equally distributed in 40 geographical regions (CHORD circles), thus each circle contained 50 nodes. Each node sustained 6 neighbors in its own circle and 6 neighbors in the others circles. In the second structure the 2000 nodes were equally distributed in 4 CHORD circles, thus each circle contained 500 nodes. Each node sustained 9 neighbors in its own circle and 2 neighbors in the others circles. It must be clarified that in this evaluation, the nodes were distributed totally random not following a specific locality optimization algorithm.

Figures 8 and 9 present the cumulative density function for the number of hops and message delay of DIMANDS requests. As depicted, the average delay for routing a message over the overlay for both structures is about 0,8 - 0,9 seconds and the average number of hops between 11 to 12. Based on these results we can accept that the performance of DIMANDS is not affected by its structure.

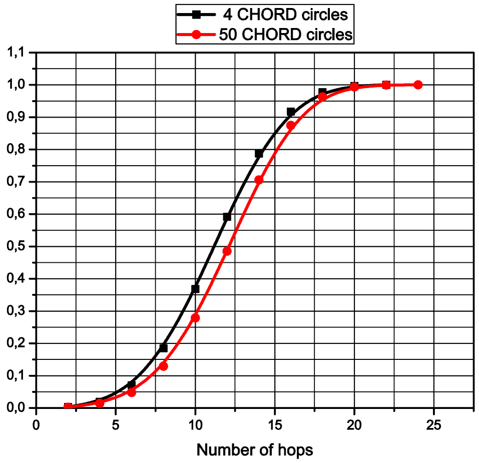


Figure 8. Cumulative density function for the number of hops.

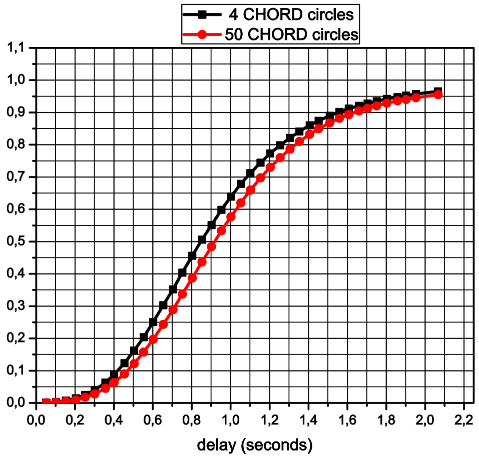


Figure 9. Cumulative density function for message delay.

The purpose of the second evaluation was to examine DIMANDS performance if the distribution of its nodes is based on a specific locality algorithm. The selected locality algorithm was the Proximity Neighbor Selection (PNS) algorithm. In this simulation the overlay was composed by 4 CHORD circles each one containing 500 nodes. Figures 10 and 11 present the cumulative density function for the number of hops and message delay for two different overlay structures: a locality aware DIMANDS overlay and a DIMANDS overlay that its nodes were distributed totally random. As depicted, the average number of hops is not affected by the locality algorithm, but the average delay for routing a message over the overlay decreases from 0.85 to 0.7 seconds.

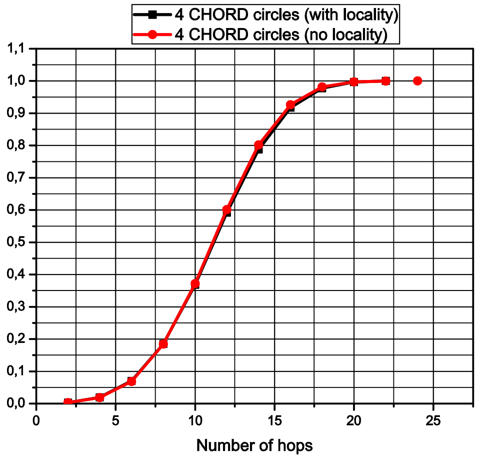


Figure 10. Cumulative density function for the number of hops.

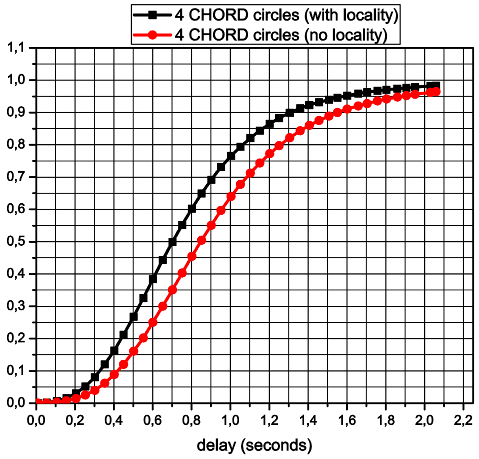


Figure 11. Cumulative density function for message delay.

We must clarify that the above measurements present the worst case scenario and prove that the system's performance can improve with the adaptation of better routing algorithms. Furthermore it must be noted that in all the above scenarios, the destination node of a generated request was randomly selected and could be any node in the overlay. In real conditions, the destination nodes of DIMANDS requests are expected to reside close to the requester due to the fact that DIMANDS architecture is organized in geographical areas to ensure that message exchanging between providers that reside in large distances in the actual network is minimized.

8. CONCLUSION

In this paper we presented DIMANDS, a global identity and trust framework free of any vertical (network layers, protocols etc) or horizontal (services, domains etc) limitations that spans across different domains and federations, and binds together different types of identities of the same user without compromising his privacy. We illustrated the role that DIMANDS can play to support innovative cross domain and cross federation user services. Through a detailed security and trust analysis we described how DIMANDS ensures absolute security and privacy to all its components and provides the means for dynamic trust establishment across different administrative borders. Finally we evaluated our system by means of simulation to present that its performance is acceptable even in the worst possible scenarios.

Moving towards the Future Internet the identity management problem will become more complex and will have to deal with not only with the management of users' identities but also all with the identifiers of interconnected devices, machines and software components (Internet of Things). Future work will try to extend DIMANDS to address this important issue and provide a large scale framework which deals with the identity management problem as a whole.

9. ACKNOWLEDGMENTS

This work is partly funded by the Greek General Secretariat for Research and Technology in the context of PENED 2003 03ED723 project, (75% EC, 25% Greek Republic, according to 8.3, 3rd Framework program).

REFERENCES

[1] Focus Group on Identity Management, “Report on Identity Management Use Cases and Gap Analysis”, ITU-T, 2008

[2] Liberty Alliance, Liberty ID-FF architecture overview, version 1.2, 2004-09.

[3] http://openid.net/specs/openid-authentication-2_0/html.

[4] http://informationcard.net/technical-information-center

[5] http://www.eclipse.org/higgins

[6] http://www.ist-daidalos.org

[7] http://www.ist-swift.org

[8] https://www.prime-project.eu

[9] http://www.primelife.eu

[10] https://spaces.internet2.edu/display/SHIB2/Home

[11] http://www.athensams.net/

[12] M. Dabrowski, P. Pacyna, “Cross-Identifier Domain Discovery Service for Unrelated User Identities”, DIM Workshop, 2008

[13] ETSI EG 284 004 v1.1.2, Universal Communications Identifier (UCI) http://ftp3.itu.ch/fgidm/Deliverables/0295-att-1.doc

[14] http://kantarainitiative.org/

[15] http://www.networkworld.com/newsletters/dir/2009/062209id2.html

[16] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, Sigcomm 2001

[17] J. Hodges. (2009) Technical Comparison: OpenID and SAML - Draft 06. [Online]. http://identitymeme.org /doc/ draft-hodges-saml-openid-compare.html

[18] E. and Reed, D. Maler, "Options and Issues in Federated Identity Management," in IEEE Security & Privacy, 2008, pp. 16-23.

[19] S. Boeyen, G. Ellison, N. Karhuluoma, W. MacGregor, P. Madsen, S. Sengodan, J. Linn(Ed). (2004) Trust Models Guidelines. [Online]. http://www.oasis-open.org/committees/download.php/6158/sstc-saml-trustmodels-2.0-draft-01.pdf

[20] Florina Almenarez Mendoza, Andres Marin Lopez and Daniel Diaz Sanchez Patricia Arias Cabarcos, "Enabling SAML for Dynamic Identity Federation Management," in Wireless and Mobile Networking Conference , Gdansk, 2009.

[21] D., Jones, M., Bufu, J., Daugherty, J. and Sakimura, N Recordon. (2009) OpenID Provider Authentication Policy Extension 1.0. [Online]. http://www.openid.net

[22] Bertino, E., Khan, L.R., Sandhu, R., Thuraisingham, B.: Secure knowledge management: confidentiality, trust, and privacy. Systems, Man and Cybernetics, Part A, IEEE Transactions on 36 (2006)

[23] Bhatti, R., Bertino, E., Ghafoor, A.: An integrated approach to federated identity and privilege management in open systems. Commun. ACM 50 (2007) 81{87

[24] Takane, Y., Young, F.W., de Leeuw, J.: Nonmetric individual di®erences multidimensional scaling: an alternating least squares method with optimal scaling features. In: Psychometrika 42. (1977)

[25] Platt, J.C.: Fast embedding of sparse music similarity. In: Advances in Neural Information Processing Systems vol. 16. (2004)

[26] Díaz Sánchez, D., A. Marín López, F. Almenárez Mendoza, C. Campo Vázquez, and C. García-Rubio. "Context awareness in network selection for dynamic

environments." Journal/Magazine: Telecommunication Systems. Vol:36. Issue: 1 (2007): Pages:49–60

[27] Yan He and Miaoling Zhu. “A complete and efficient strategy based on petri net in automated trust negotiation”. Infoscale, June 2007.

[28] Keith Irwin and Ting Yu. “Preventing attribute information leakage in automated trust negotiation”. CCS'05, 12th ACM conference on Computer and communications security , November 2005.

[29] Jiangtao Li, Ninghui Li, and William H. Winsborough. “Automated trust negotiation using cryptographic credentials”. CCS'05, 12th ACM conference on Computer and communications security, November 2005.

Hidden VoIP Calling Records from Networking Intermediaries

Ge Zhang
Karlstad University, Karlstad, Sweden
ge.zhang@kau.se

Stefan Berthold
Karlstad University, Karlstad, Sweden
stefan.berthold@kau.se

ABSTRACT

While confidentiality of telephone conversation contents has recently received considerable attention in Internet telephony (VoIP), the protection of the caller–callee relation is largely unexplored. From the privacy research community we learn that this relation can be protected by Chaum's mixes. In early proposals of mix networks, however, it was reasonable to assume that high latency is acceptable. While the general idea has been deployed for low latency networks as well, important security measures had to be dropped for achieving performance. The result is protection against a considerably weaker adversary model in exchange for usability. In this paper, we show that it is unjustified to conclude that low latency network applications imply weak protection. On the contrary, we argue that current Internet telephony protocols provide a range of promising preconditions for adopting anonymity services with security properties similar to those of high latency anonymity networks. We expect that implementing anonymity services becomes a major challenge as customer privacy becomes one of the most important secondary goals in any (commercial) Internet application.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

VoIP, Mixes, Anonymity, Traffic analysis attacks

1. INTRODUCTION

Addressing on the network layer in the Internet is by no means secure: neither is it a simple task to validate a given address, since forking is easy, nor is it simple to hide addresses reliably from network intermediaries, and thus establish anonymity, since explicit addresses are an inherent part of most Internet protocols. Chaum's mixes [1] are designed to build an anonymity layer upon such protocols. A mix is a forwarding proxy which obfuscates addresses, sending and receiving time, and contents of network messages. A typical way of obfuscating the sending or receiving time in high latency applications, such as e-mail (Mixminion [2]), is to wait for several messages arriving at the mix before forwarding them all together in random order. However, in low latency protection services (e.g., AN.ON [3], Tor [4], and ISDN-Mixes [5]), delaying messages is no option. Even for web mixes, practical experience shows that the packet delay needs to be close to zero. The smaller the delay of a packet may be, however, the smaller is the set of packets that can be sent out in random order or lexicographical order for practical reason, and thus the higher becomes the probability of successful traffic analysis attacks. A common proposal to avoid this kind of attacks is to generate artificial cover traffic in the network, if otherwise the mix does not receive enough packets.

Voice over IP (VoIP) applications are natural competitors of classic public switched telephone network (PSTN), but fall short when it comes to preserving the anonymity of users on the network level, since explicit addresses are used. At first sight, it seems hard to establish an anonymity layer between network layer and existing VoIP protocols due to their bandwidth demands and the low latency which is allowed. A few assumptions about the mode of operation, however, allow to create a view on VoIP protocols which is quite appealing for establishing the anonymity layer. These assumptions are (1) VoIP media flow is sent in constant rate, (2) it is sent continuously, i. e., silence suppression is not applied, and (3) each media packet is with the equal size, i. e., a fixed bit rate codec is employed.

In this paper, we perform an analysis of the passive traffic analysis attacks on VoIP systems, considering both, signaling flow and media flow, and moreover demonstrate how to eliminate or equalize the flow patterns to make the attacks more difficult.

We deem privacy concerns as one of the major hurdles in the large-scale adoption of VoIP technology. Not only the telecommunication legislation in many countries declares the telecommunication contents and the caller-callee relation as sensitive data in general, but also companies may specifically worry about business secrets such as confidential negotiations, they may allow anonymous whistle-blowing within

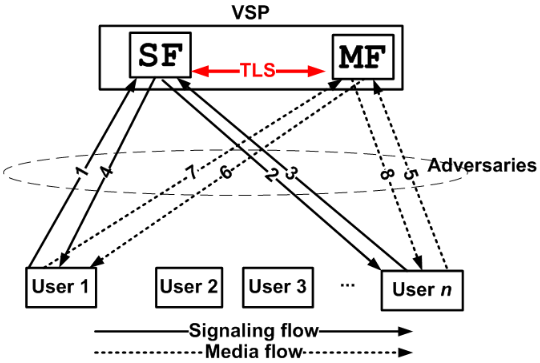


Figure 1: A simple VoIP architecture

the company to improve working conditions as well, and scientists may provide anonymity to those who participate in their surveys.

The remainder of this paper is organized as follows: Section 2 provides the system model, notions, and a calling scenario; Section 3 describes our adversary model and attacking methods. Prevention methods are discussed in Section 4; Section 5 addresses open problems and future work; Section 6 presents a overview of related work; in Section 7, we summarize our conclusions.

2. MODELS

2.1 Preliminaries: a VoIP model

Typically two types of flows are involved to realize a VoIP call: signaling flows for call setup and termination (e.g., the Session Initiation Protocol (SIP) [6]) and media flows for coded voice packets transmission (e.g., the Realtime Transport Protocol (RTP) [7]). In this paper, we focus on the Client/Server based VoIP model, which consists of a VoIP Service Provider (VSP) and a set of clients. The VSP mainly provides two functional components: *(1)* a Signaling Function (SF) to authenticate user, locate user, forward signaling messages and manage billing records; *(2)* a Media Function (MF) to relay media flows between users. The motivation of employing a MF is to help media flows to traverse Network Address Translation (NAT) [8] devices since users may not own a valid public IP address. To solve the problem, the SF first needs to contact with the MF using a middlebox management protocol (e.g., MIDCOM [9] or TURN [10]) to reserve corresponding resources (e.g., ports) on the MF. Secondly, the SF replaces the original peer transport address (TA)[1] information appeared in signaling messages with the reserved TA of the MF. In this way, the users do not need to setup a direct bidirectional media flow with each other. Instead, the media flows are relayed by the MF as a rendezvous point.

Given the VoIP architecture illustrated in Figure 1, there are n users registered on the VSP: Each user may setup calls with another one. The channels between users and the VSP are insecure and can be intercepted by adversaries. However, we assume that each user shares a secrte key with the VSP. The channels can be encrypted by using the shared key between the user and the VSP. It is flexible for users to select user-agents by themselves. Two properties of a user-agent are worth being mentioned:

- Silence suppression: Some VoIP user-agents allow discontinuous voice packets transmission [11], which is a capability of user-agents to stop sending media packets during silent periods of its owner. In this circumstance, bandwidth can be significantly saved. If silence suppression is not applied, the media packets are generated constantly with a fixed time interval (e.g., 20 ms).

- Encoding bit rates: Two types of encoding bit rates can be distinguished: *Fixed Bit Rate (FBR)* and *Variable Bit Rate (VBR)*. With FBR codec (e.g., G.711 [12]), the generated media packets are always the same size. On the other hand, VBR codec (e.g., Speex [13]) means that the encoding bit rate varies according to the voice. In this way, user agents produce media packets with different sizes.

2.2 A calling scenario

Let us take the scenario illustrated in Figure 2[2]. The user, Alice (denoted as $a \in \mathcal{U}$), launches an INVITE request targeting to Bob (denoted as $b \in \mathcal{U}$). User a first initializes the request to the SF. The SF processes this request and then forwards it to b. It takes a while for b to decide whether this request should be accepted and the time period for deciding is highly indeterministic. Then we assume that b sends a positive response to SF to accept the call. The response is relayed by SF). These signaling messages compose one *signaling transaction.*

After a acknowledges the request by an ACK message, a will start the conversation by continuously sending media packets. The packets are relayed by MF to b. Meanwhile, b continuously originates media packets. All these media packets compose a *media session.*

Finally, a sends termination signaling request to tear down the media flows. This request and the response to this request is relayed by SF. User a and b stop sending media packets respectively. The signaling messages for terminating the call compose another signaling transaction and all the signaling messages in this scenario forms a *signaling dialog.*

We assume that each user shares a secret key with the VSP. All the signaling and media packets in this scenario are encrypted by using the keys. Therefore, when the SF or the MF receives a signaling or a media packet, it will decrypt the packet at first and then encrypt it again using the shared keys with the sender and recipient respectively. The following sections will analyze attacks and attack prevention based on this scenario.

3. TRAFFIC ANALYSIS ATTACKS

3.1 Adversary model

We first assume that Alice trusts the VSP and her contact, Bob. In this way, the calling records of their conversations are legitimate to be known by them. We consider a *global adversary model*: the intermediaries (e.g., routers) in the

[1] A transport address is a pair of IP address and port

[2] To simplify, we do not consider optional signaling and RTCP traffic in this paper.

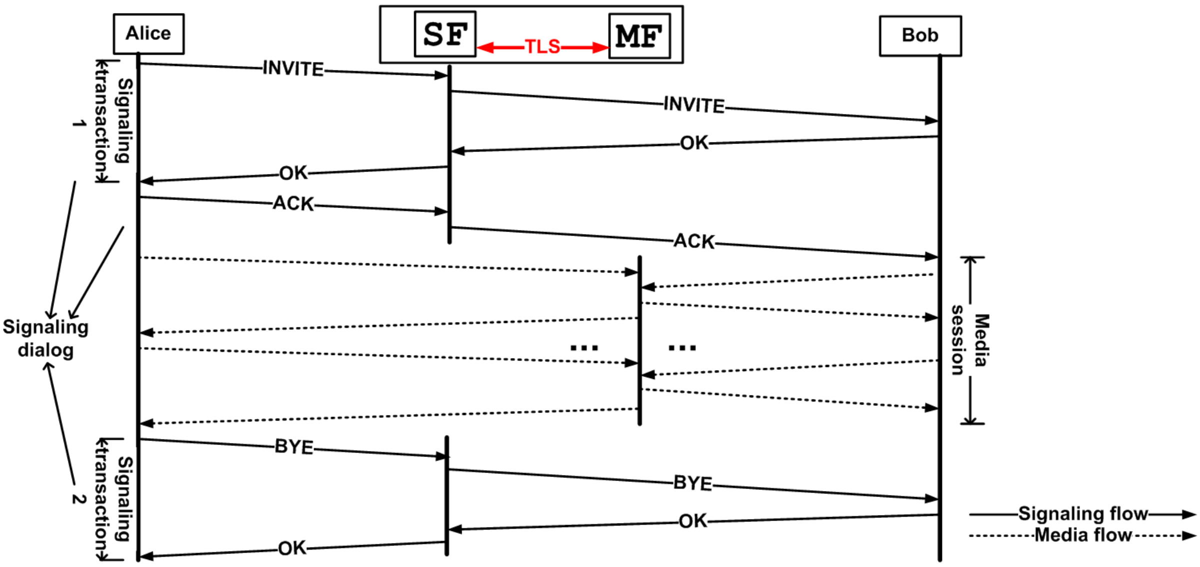


Figure 2: A calling scenario

network are potential adversaries, who form a grand coalition to wiretap the flow information over any links in the network during a period of time. However, we assume that it is computational impossible for the attackers to decipher any flow. Thus, the attackers are unable to read the plain text of their intercepted packets. Nevertheless, the attackers do not care about the content of conversations, but only aim to profile the calling records (caller-callee relation). What the attackers can observe are header, payload, size and arrival time of packets in any flow. Furthermore, the attackers have an experimental knowledge of the packet loss and delay on each channel. Readers please note that this paper only focus on passive traffic analysis attacks, which means that the attackers do not modify, drop and delay any packet in any flow.

3.2 Basic notions

Let $\mathcal{U}$ to be the set of the n users: $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ and we use m and s to indicate the MF and the SF respectively. We use $\mapsto$ to denote "who called whom". For example, If u_1 called u_n, then we write $u_1 \mapsto u_n$. Moreover, a vector $\overrightarrow{xy}$ $(x, y \in \mathcal{U} \cup \{m, s\}; x \neq y)$ represents the flow from x to y for a given time period T. The flow consists of a set of IP packets or no packets at all, formalized by $\overrightarrow{xy} = (\langle\overrightarrow{xy}\rangle_1, ..., \langle\overrightarrow{xy}\rangle_{|\overrightarrow{xy}|})$, $|\overrightarrow{xy}|$ denotes the number of packets in $\overrightarrow{xy}$. Then $|\overrightarrow{xy}| = 0$ means that no packet sent from x to y during T. Otherwise, a set $(\langle\overrightarrow{xy}\rangle_1, ..., \langle\overrightarrow{xy}\rangle_{|\overrightarrow{xy}|})$ can represent $\overrightarrow{xy}$. All the $\langle\overrightarrow{xy}\rangle_i, (1 \leq i \leq |\overrightarrow{xy}| \wedge i \in \mathbb{Z})$ are the data packets contained in $\overrightarrow{xy}$ which are numbered in the order in which they were sent.

Each packet has its own properties (e.g., size, arrival time, etc). For a given packet $\langle\overrightarrow{xy}\rangle_i$, let $S(\langle\overrightarrow{xy}\rangle_i)$ to be the size of $\langle\overrightarrow{xy}\rangle_i$; And $T(\langle\overrightarrow{xy}\rangle_i)$ to be the arrival time of $\langle\overrightarrow{xy}\rangle_i$.

Moreover, each channel also has its own features (e.g., packet loss, transmission delay). We use δ_{xyz} to denote the maximal packet loss rate and $d_{xyz} \pm \varepsilon_{xyz}$ to denote the transmitting delay over the channel from x to z relayed by y.

We further define:

- Given two flows $\overrightarrow{xy}$ and $\overrightarrow{yz}$, we define that it is a **packet amount match** $(\overset{A}{\equiv})$ between the two flows if and only if the two flows have the similar amount of packets.

$$
\begin{aligned}
&(|\overrightarrow{xy}| - |\overrightarrow{yz}| \leq \delta_{xyz} \cdot |\overrightarrow{xy}|) \\
&\iff (\overrightarrow{xy} \overset{A}{\equiv} \overrightarrow{yz});
\end{aligned} \tag{1}
$$

- Given two flows $\overrightarrow{xy}$ and $\overrightarrow{yz}$, we define that it is a **size match** $(\overset{S}{\equiv})$ between the two flows if and only if: (1) $\overrightarrow{xy} \overset{A}{\equiv} \overrightarrow{yz}$, and (2) a subvector $\overrightarrow{xy}'$ of $\overrightarrow{xy}$ can be found: The packets in $\overrightarrow{xy}'$ and $\overrightarrow{yz}$ with the same sequence number have the same payload.

$$
\begin{aligned}
&(\overrightarrow{xy} \overset{A}{\equiv} \overrightarrow{yz}) \wedge (\exists \overrightarrow{xy}' : \forall i : S(\langle\overrightarrow{xy}'\rangle_i) = S(\langle\overrightarrow{yz}\rangle_i)) \\
&\iff (\overrightarrow{xy} \overset{S}{\equiv} \overrightarrow{yz});
\end{aligned} \tag{2}
$$

- Given two flows $\overrightarrow{xy}$ and $\overrightarrow{yz}$, we define that it is a **relative time match** $(\overset{T}{\equiv})$ between the two flows if and only if: (1) $\overrightarrow{xy} \overset{A}{\equiv} \overrightarrow{yz}$; and (2) a subvector $\overrightarrow{xy}'$ of $\overrightarrow{xy}$ can be found: The packets in $\overrightarrow{xy}'$ and $\overrightarrow{yz}$ with the same sequence number have the arrival time difference with in the predicted transmitting delay $d_{xyz} \pm \varepsilon_{xyz}$.

$$
\begin{aligned}
&(\overrightarrow{xy} \overset{A}{\equiv} \overrightarrow{yz}) \wedge \\
&(\exists \overrightarrow{xy}' : \forall i : |T(\langle\overrightarrow{yz}\rangle_i) - T(\langle\overrightarrow{xy}'\rangle_i) - d_{xyz}| \leq \varepsilon_{xyz}) \\
&\iff (\overrightarrow{xy} \overset{T}{\equiv} \overrightarrow{yz});
\end{aligned} \tag{3}
$$

3.3 Attack methods

Some people believe that their calling records are withheld from intermediaries as long as both the signaling and the media flows are encrypted as well as relayed by the VSP

[14]. However, it is not the case if we apply the global adversary model. If we take the calling scenario in Figure 2 as an example, the calling record can be successfully detected by the attackers as long as they can link any two relayed flows ($\overrightarrow{as}$ with $\overrightarrow{sb}$, $\overrightarrow{bs}$ with $\overrightarrow{sa}$, $\overrightarrow{am}$ with $\overrightarrow{mb}$, or $\overrightarrow{bm}$ with $\overrightarrow{ma}$) in time window of a call. The attackers just need to observe some linkable patterns of the flows on both sides of the VSP without deciphering any packet. This kind of attacks is named as *passive traffic analysis attack*.

The traffic analysis attacks take advantage of the linkability of the two relayed flows (e.g., $\overrightarrow{as}$ and $\overrightarrow{sb}$). As defined by Pfitzmann et al., [15] linkability of two flows from an attacker's perspective means that the attacker can sufficiently distinguish whether the two flows are related or not. Actually, within the VoIP model, some patterns of the relayed flows are highly deterministic. Here we address these linkable features:

1. Non-empty pattern: If a called b, there must be a sequence of signaling request packets sent from a to s and from s to b. Also some response packets from b to s and s to a should be originated. However, it is uncertain whether there are media packets generated or not, depending on whether such a request is accepted or rejected by b. This pattern is formalized as

$$\begin{aligned}(a \mapsto b) \implies (|\overrightarrow{as}| > 0) \wedge (|\overrightarrow{sb}| > 0) \wedge (|\overrightarrow{bs}| > 0) \wedge (|\overrightarrow{sa}| > 0)\\ \wedge \begin{cases}(|\overrightarrow{am}| > 0) \wedge (|\overrightarrow{mb}| > 0) \wedge (|\overrightarrow{bm}| > 0) \wedge (|\overrightarrow{ma}| > 0), & accept\\ (|\overrightarrow{am}| = 0) \wedge (|\overrightarrow{mb}| = 0) \wedge (|\overrightarrow{bm}| = 0) \wedge (|\overrightarrow{ma}| = 0), & reject;\end{cases}\end{aligned} \tag{4}$$

We assume that attackers have already known that a involved in a conversation during a time T. Considering Equation 4, the attackers know that the contact of a must be bound to a set $\mathcal{X}_a \subseteq \mathcal{U}/\{a\}$ which is formalized as:

$$\begin{aligned}\forall_{x\in\mathcal{X}_a} : (|\overrightarrow{sx}| > 0) \wedge (|\overrightarrow{xs}| > 0) \wedge\\ \begin{cases}(|\overrightarrow{mx}| > 0) \wedge (|\overrightarrow{xm}| > 0), & accept\\ (|\overrightarrow{mx}| = 0) \wedge (|\overrightarrow{xm}| = 0), & reject;\end{cases}\end{aligned} \tag{5}$$

The calling record can be confirmed by the attackers if $\mathcal{X}_a = \{b\}$ and $\mathcal{X}_b = \{a\}$. For example, Figure 1 gives a scenario in which only two users (u_1 and u_n) generates signaling and media packets. Thus, it is easy for attackers to find out that the conversation is between u_1 and u_n because $\mathcal{X}_{u_1} = \{u_n\}$ and $\mathcal{X}_{u_n} = \{u_1\}$.

2. Packets amount pattern: If a called b, the $\overrightarrow{as}$ and the $\overrightarrow{sb}$ should be packet amount match. This rule is also applied to $\overrightarrow{bs}$ with $\overrightarrow{sa}$, $\overrightarrow{am}$ with $\overrightarrow{mb}$, and $\overrightarrow{bm}$ with $\overrightarrow{ma}$. This pattern is formalized as

$$\begin{aligned}(a \mapsto b) \implies (\overrightarrow{as} \overset{A}{\equiv} \overrightarrow{sb}) \wedge (\overrightarrow{bs} \overset{A}{\equiv} \overrightarrow{sa}) \wedge\\ (\overrightarrow{am} \overset{A}{\equiv} \overrightarrow{mb}) \wedge (\overrightarrow{bm} \overset{A}{\equiv} \overrightarrow{ma})\end{aligned} \tag{6}$$

The attackers have already known the $|\overrightarrow{as}|$, $|\overrightarrow{sa}|$, $|\overrightarrow{am}|$ and $|\overrightarrow{ma}|$ since they can intercept the flows over the link between a and the VSP. Moreover the attackers know the packet loss rate over the channels. Using Equation 6, the attackers know that the contact of a must be bound to a set $\mathcal{Y}_a \subseteq \mathcal{U}/\{a\}$ which is formalized as:

$$\begin{aligned}\forall_{y\in\mathcal{Y}_a} : (\overrightarrow{as} \overset{A}{\equiv} \overrightarrow{sy}) \wedge (\overrightarrow{ys} \overset{A}{\equiv} \overrightarrow{sa}) \wedge\\ (\overrightarrow{am} \overset{A}{\equiv} \overrightarrow{my}) \wedge (\overrightarrow{ym} \overset{A}{\equiv} \overrightarrow{ma})\end{aligned} \tag{7}$$

3. Packets size pattern: The signaling packets are usually modified by the SF (e.g., to insert or remove packet header fields), they are immunized to the packet size pattern. However, the packets size of relayed media flows should be matched if MF only decrypts and encrypts received packets without changing them. This pattern is formalized as

$$(a \mapsto b) \implies (\overrightarrow{am} \overset{S}{\equiv} \overrightarrow{mb}) \wedge (\overrightarrow{bm} \overset{S}{\equiv} \overrightarrow{ma}) \tag{8}$$

The attackers have already intercepted the packet sizes of $\overrightarrow{am}$ and $\overrightarrow{ma}$. Taking Equation 8 into account, the attackers know that the contact of a must be bound to a set $\mathcal{V}_a \subseteq \mathcal{U}/\{a\}$ which is formalized as:

$$\forall_{v\in\mathcal{V}_a} : (\overrightarrow{am} \overset{S}{\equiv} \overrightarrow{mb}) \wedge (\overrightarrow{vm} \overset{S}{\equiv} \overrightarrow{ma}) \tag{9}$$

4. Packets arrival time pattern: The packets arrival time of relayed flows are also highly deterministic. As mentioned above, $\overrightarrow{as}$ with $\overrightarrow{sb}$, $\overrightarrow{bs}$ with $\overrightarrow{sa}$, $\overrightarrow{am}$ with $\overrightarrow{mb}$, and $\overrightarrow{bm}$ with $\overrightarrow{ma}$ should be relative time match.

$$(a \mapsto b) \implies (\overrightarrow{as} \overset{T}{\equiv} \overrightarrow{sb}) \wedge (\overrightarrow{bs} \overset{T}{\equiv} \overrightarrow{sa}) \wedge (\overrightarrow{am} \overset{T}{\equiv} \overrightarrow{mb}) \wedge (\overrightarrow{bm} \overset{T}{\equiv} \overrightarrow{ma}) \tag{10}$$

The attackers have already intercepted the packets arrival time of $\overrightarrow{as}$, $\overrightarrow{sa}$, $\overrightarrow{am}$ and $\overrightarrow{ma}$. Considering Equation 10, the contact of a must be bound to a set $\mathcal{W}_a \subseteq \mathcal{U}/\{a\}$ which is formalized as:

$$\forall_{w\in\mathcal{W}_a} : (\overrightarrow{as} \overset{T}{\equiv} \overrightarrow{sw}) \wedge (\overrightarrow{ws} \overset{T}{\equiv} \overrightarrow{sa}) \wedge (\overrightarrow{am} \overset{T}{\equiv} \overrightarrow{mw}) \wedge (\overrightarrow{wm} \overset{T}{\equiv} \overrightarrow{ma}) \tag{11}$$

Let us say that a and b had a conversation using VoIP. Taking above patterns (Equation 5, 7, 9 and 11) into account, the contact of the user a must be bound in such the set $\mathcal{C}_a = \mathcal{X}_a \cap \mathcal{Y}_a \cap \mathcal{V}_a \cap \mathcal{W}_a$ from the attackers' view. Similarly, the contact of b must be bound in such the set $\mathcal{C}_b = \mathcal{X}_b \cap \mathcal{Y}_b \cap \mathcal{V}_b \cap \mathcal{W}_b$. Therefore, the success of passive traffic analysis attacks totally depends upon the sizes of $\mathcal{C}_a$ and $\mathcal{C}_b$. We define two cases as follows:

- The worst anonymity case: The attackers can confirm that a called b when $(\mathcal{C}_a = \{b\}) \wedge (\mathcal{C}_b = \{a\})$, which is defined as the worst case.

- The best anonymity case: The contact of a can be any one in the $\mathcal{U}$ except a and the contact of b can be any one in the $\mathcal{U}$ except b when $(|\mathcal{C}_a| = n - 1) \wedge (|\mathcal{C}_b| = n - 1)$, which means that it is insufficient for attackers to distinguish. This case is defined as the best case.

Unfortunately, however, there are a variety of user-agents are available in reality. Some user-agents support silence suppression or VBR codec, which makes the flow patterns highly depending on specified scenarios. Moreover, each VoIP user has unique characteristics and preferences for making calls. And all the calls can be established and terminated at arbitrary time independent from others. As a result, the sizes of $\mathcal{C}_a$ and $\mathcal{C}_b$ usually is rather small and leading to the worst case.

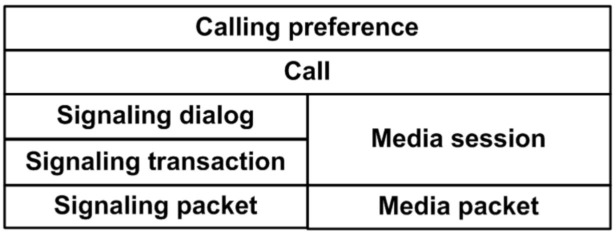
4. PROTECTION METHODS

While the attackers aim to minimize the size of $\mathcal{C}_a$ and $\mathcal{C}_b$, the goal of the protection is to increase them so that it is difficult for attackers to locate who is the real contact. Countermeasures against traffic analysis are based on the mixes concept [1], in which one or several mix nodes serve to relay packets and meanwhile hide the relationship between incoming and outgoing packets. A mix can apply a variety of techniques including broadcasting, multi-layer cryptography, reordering packets, delaying and forwarding packets as a batch, generating additional cover traffic and replay determination schemes to reduce the linkable patterns of the flows. Many anonymity services based mixes concept are existing in the Internet. For example, AN.ON [3] mixes HTTP flows from and to its web browsing users. However, most of these anonymity services are designed for web surfing, FTP, or Email applications. Nevertheless VoIP has different characteristics to these applications:

- Flow types: To achieve a VoIP call, a signaling communication and a media communication must be established. There are different flow patterns for these two types of communications.
- Performance: VoIP users usually have different performance requirements on signaling flows and media flows. For example, the end-to-end latency for transmitting media packets over 350 ms can interrupt the conversation [16]. The latency requirement for signaling flows are lower and depends on how long a user would like to wait for establishing a call.
- Conversation mode: VoIP users mostly build conversation following a 1:1 mode. That is, one user can only call or can be called by another user in a given time.[3]
- Packets rate: VoIP media flows can be sent in constant rate continuously when silence suppression is not applied.
- Packets size: Each media packet can be the equal size when a fixed bit rate codec is employed.

Taking these characteristics into account, the anonymity solutions for other applications are difficult to be reused for VoIP. A specific scheme should be designed combined with the characteristics of VoIP. Moreover, we consider to apply some mixes techniques on the SF and the MF by rewriting the processing logics on the SF, the MF and user-agents. The logics can be rewritten for two considerations:

- Generalizing patterns: We apply policies on the SF and the MF to enforce that all their served flows must follow predefined patterns. Thus, the patterns of the flows are equalized.



<table>
<tr><td colspan="2">Calling preference</td></tr>
<tr><td colspan="2">Call</td></tr>
<tr><td>Signaling dialog</td><td rowspan="2">Media session</td></tr>
<tr><td>Signaling transaction</td></tr>
<tr><td>Signaling packet</td><td>Media packet</td></tr>
</table>

Figure 3: A layered model of VoIP

- Eliminating patterns: Another alternative is to rewrite the calling processing rules to break the original deterministic patterns. In this way, the linkable patterns are not valid anymore.

4.1 Anonymity preference

Technically, VoIP can be considered as a layered model illustrated in Figure 3. In this model, the overlay represents the higher logic based on the underlay. For example, a media session contains bidirectional flows of media packets. A signaling transaction consists of a SIP request packet and a response packet (e.g., INVITE and 200 OK). A signaling dialog includes a “starting” transaction and a “terminating” transaction. A call is composed of a signaling dialog and a media session. Finally, calling preference is the highest logic indicating the frequency of a callee for a given caller.

Anonymity preference can be enforced on any layer in this model. However, a low-layer protection does not mean the high-layer is well protected. For example, the call anonymity is not necessarily achieved even if the media session anonymity is protected since attackers might find clues from the signaling flows. Thus, it is more difficult to protect the anonymity on the higher layer. In this paper, we focus on the anonymity protection on the call layer, which aims to withhold the caller/callee relationship of a single call. Our solution cannot guarantee the protect calling preference. For example, the anonymity protection might be broken if Alice calls Bob for many times. The detail on this issue is discussed in Section 5 and we leave the calling preference protection counteracting long term intersection attacks for future work.

4.2 Methods

Taking the two approaches (generalizing patterns and eliminating patterns) into account, we discuss countermeasure methods.

Enforcing to use the same FBR codec: We force all user-agents to be applied the same FBR codec when users search for traffic analysis resistance. In this way, the media packets generated from user-agents always have the same size, which means that the packet size pattern is equalized for any media flow.

Dropping media packets generated in silence periods: This method is based on the “defensive dropping” concept [17]. We name the media packets generated in silence periods as *silent packets*. In this scheme, the user-agents do not apply silence suppression but Voice Activity Detection (VAD): The user-agents generate media packets to the MF in a constant rate whatever they detect silence or speech. However, the user-agent can instruct the MF to drop some randomly selected silent packets according to a dropping rate. This can be easily achieved by putting one

[3] We do not consider to address VoIP conference mode in this paper.

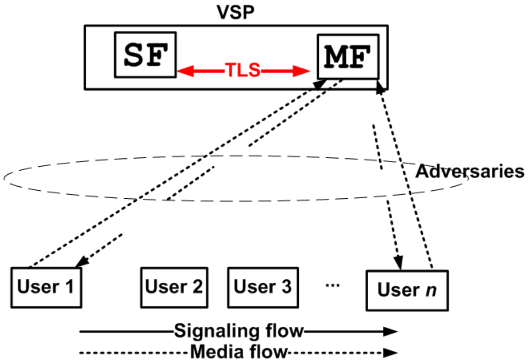


Figure 4: Dropping media packets generated in silence periods

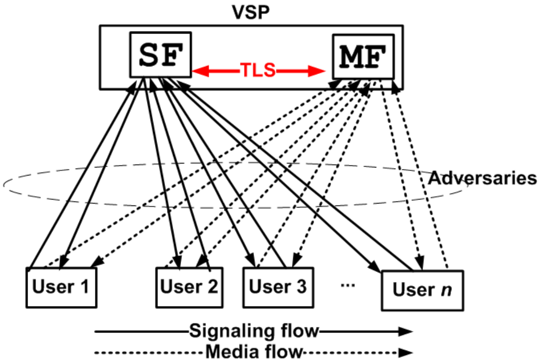


Figure 5: Enforcing global dummy traffic to cover media flows

bit ('0' for keeping and '1' for dropping) inside the encryption layer of each media packet. This method is helpful to eliminates packets amount pattern on media flows. Moreover, dropping these silent packets introduces less impact on the performance of a VoIP conversation. Figure 4 depicts this method. The dots in the figure denote media packets. From the figure, we can see that $\overrightarrow{u_1m}$ and $\overrightarrow{mu_1}$ are with different amount of packets as silent packets are dropped by the MF. Nevertheless, the dropping rate must be carefully selected according to our previous work in [18]. Especially, the attackers can find out the silence periods and speech periods of both users by observing the "gaps" in $\overrightarrow{mu_1}$ and $\overrightarrow{mu_n}$ if all the silence packets are dropped by the MF. In this way, attackers can therefore match $\overrightarrow{mu_1}$ and $\overrightarrow{mu_n}$ using human conversation pattern "When one speaks, the other listens [19]".

Enforcing global dummy traffic to cover media flows: The method of enforcing global dummy traffic in VoIP has been discussed in [20]. Dummy traffic is the traffic consisting of encrypted garbage packets. Since all packets are encrypted, the attackers cannot distinguish a captured packet is a media packet or a garbage one. There are two options for users in the scheme proposed in [20]:

- The users constantly send dummy media packets to the MF, meanwhile the MF constantly sends all users with dummy media packets. Both the MF and the users decrypt received packets, and drop them if they are recognized as garbage.

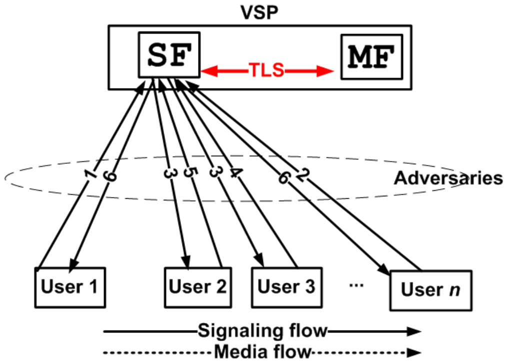


Figure 6: An example of batch method with $k = 2$

- When a user a wants to have a conversation with user b, a replaces the encrypted garbage packets with media packets. The MF then decrypts the packets from a and recognizes that they are not garbage packets, but media packets targeting to b. Therefore, the MF forwards these media packets to b instead of garbage packets. After decrypting them, b accepts these packets since they are not garbage packets. And b sends media packets relayed by the MF following the same rule.

The idea behind this method is that all idle users pretend communicating to cover the users who are really making calls. In this way, the non-empty pattern for media flows is equalized. However, we need to mention that global dummy traffic can introduce high bandwidth overhead. Figure 5 illustrates this method.

Enforcing batch scheme: This scheme does not rely on dummy traffic to equalize the non-empty pattern, but aims to process k calls as a batch with the same starting and terminating time. The constant k is a quantitative requirement of anonymity and can be arbitrarily set according to specific contexts. The detailed operations are as follows:

1. The SF waits for the calling requests (INVITE) until k requests are received.

2. The SF takes these k requests as a batch (e.g., recording Call-IDs of all k requests) and then flushes the batch at once (forwarding all requests).

3. The SF waits for the responses to the k requests until all of them are received.

4. The SF flushes the responses, and waits for the terminating requests (BYE) for the batch until a timeout t_{out} occurs.

5. If all terminating requests for the batch have been received and the timeout does not occur, the SF flushes all the terminating requests.

6. Otherwise, if the timeout occurs and not all terminating requests for the batch have yet been received, the SF generates terminating requests to force terminating all calls within the batch.

7. Similarly, the SF batches all the terminating responses.

An example of this method is illustrated in Figure 6, with $k = 2$. The signaling flows are numbered in chronological order. Let us say that u_1 called u_2 and u_n called u_3. User u_1 first initialize a calling request to the SF. However, the SF does not forward it immediately, instead, the SF waited until the second calling request is received. Then, the SF takes these two requests as a batch and flushes them to u_2 and u_3 respectively. The batch scheme is applied to the responses, too. With this scheme applied, the attackers at most know that u_1, u_2, u_3, and u_n are involved in 2 conversations. However, they cannot distinguish who called whom in detail, because u_1 might call u_2 or u_3, and it is the same for u_n. The complexity increases with the k. The major side effect of this method is that the time of establishing and terminating a call is beyond the user's control and depending other users (whether there are other users join the batch or not). Nevertheless, this method is more flexible than the global dummy traffic method since the requirement k is tunable.

4.3 An example solution

We will give an example solution based on the above mixes methods. Our solution is not designed to achieve the best anonymity case since the cost to achieve it is rather high [20]. Instead it aims to maintain $(|\mathcal{C}_a| = k) \vee (|\mathcal{C}_b| = k)$ with $1 \leq k \leq m$. We assume that each user shares a secrete key k_{u_i} with the VSP, which can be used both by the SF and the MF.

User-agents behavior: The user-agents need to apply a pre-defined FBR codec with VAD. The user-agents should send signaling packets to the SF over encrypted channels using k_{su_i}. The user-agents should embed a flag in each media packet to inform the MF to drop (encoded '1') or forward (encoded '0') it. Only silence packets are legitimate to be dropped and they are selected for dropping by a random function. The media packets should be protected with a layer encryption using k_{mu_i}. Furthermore, if a calling request is rejected, the user-agent of the caller is required to generate dummy media packets with the same standard targeting to itself relayed by the MF until it receives a terminating request from the SF. The operations on making a call by the user-agents is shown in Algorithm 1.

The MF behavior: The MF need to decrypt the received media packets. From embedded flag, the MF knows whether this packet should be dropped or forwarded. The operations on the MF is shown in Algorithm 2

The SF behavior: The SF process the signaling messages mainly based on the batch method. The calling processing logic on SF is listed in Algorithm 3. Let us say each user and the SF have built a encrypted channel using k_{su_i}, thus we do not repeat stating the cryptographic operations in Algorithm 3.

5. OPEN ISSUES

Here, we briefly summarize some open problems which are important for the future research on VoIP calling anonymity.

Performance: Countermeasures to traffic analysis attacks are usually high cost. On the other hand, there is no doubt that performance is one of the most critical aspects for VoIP applications if we consider the natural of voice communication. As said, the SF needs to delay $k-1$ requests until the k^{th} request is received for counteract against traffic analysis. Delaying signaling for a period of time will not

Algorithm 1 The operations on making a call by the user-agents

```
Encrypts the calling request using $k_{su_i}$;
Send it to the SF;
Wait for the response;
if Get a calling response then
  Decrypts it using $k_{su_i}$;
  if response == OK then
    Send a encrypted ACK;
    repeat
      Send a media packet;
      if It is a silence packet then
        Decide whether this packet should be dropped;
        if Decide to drop then
          Mark it's flag as 1;
        end if
      end if
      Encrypt media packets using $k_{mu_i}$;
      Send and receive media packet;
      Decrypt media packets using $k_{mu_i}$;
      if Want to tear down the call then
        Send terminating request;
        Break;
      end if
    until Calling terminating received
  else if response == rejected then
    repeat
      Generate randomized a dummy packet;
      Decide whether this packet should be dropped;
      if Decide to drop then
        Mark it's flag as 1;
      end if
      Encrypt dummy packets using $k_{u_i}$;
      Send and receive dummy packets;
    until Calling terminating received
  end if
end if
```

Algorithm 2 The operations on the MF

```
Receive a media packet;
Deciphering it using $k_{mu_i}$;
if flag is 1 then
  Drop this packet;
else
  Encrypt this packet using $k_{mu_j}$; $\{u_j$ is the destination
  of the packet.$\}$
  Forward it;
end if
```

Algorithm 3 The operations on the SF

```
repeat
    Get 1 calling request;
until Received $k$ calling requests;
Take these $k$ requests as a batch;
flush them;
Wait for calling responses for the batch;
if All the $k$ responses are received then
    Reserve resources on the MF;
    flush the responses;
end if
Wait for terminating requests for the batch;
if All terminating requests for the batch received then
    Free the resources on the MF;
    flush the terminating requests;
else if the maximal calling duration is reached then
    Free the resources on the MF;
    Generate terminating requests;
    flush the terminating requests;
end if
Wait for final responses for the batch;
if All responses received then
    flush them;
end if
```

interrupt the whole conversation: Users just need to wait for a while to setup or terminate the call. It is reasonable to tradeoff if user want to achieve privacy. However, different to signaling flows, a heavy end-to-end latency on media flows can interrupt the whole conversation. According to [16], the one-way latency on media flows under 150 ms is considered highly desirable. Our solution requires the MF to synchronize and re-encrypt its relayed media flows. These additional operations certainly introduce overhead on end-to-end latency of media flows. In future work, we will evaluate the performance of a prototype implementation.

Active traffic analysis: A comprehensive anonymity solution should also take the *active* traffic analysis attacks into account. In this context, the intermediaries not only can wiretap their transmitted flows, but also can modify, delay, drop any packets in the network. The idea behind the active traffic analysis attacks is to introduce specific linkable patterns for correlation. Sophisticated detection and prevention schemes need to be designed to defend against active traffic analysis.

Replay attacks: The MF does not read and check the content of its relayed media packets, but only decrypts, synchronizes and forwards them. Attackers can take advantage of this by replaying intercepted media packets several times later. Thus, attackers can identify who is the recipient of the packets by the intersection of different batches if the replayed packets cannot be detected. In future work, we will use timestamp and timeout scheme to discard replayed media packets since late-arrival media packets are useless.

Malicious users: In this paper we assume that only the networking intermediaries are adversaries while all the users from $\mathcal{U}$ are honest and cooperative. However, it is usually not the case in reality. Malicious users might frequently setup meaningless calls batched with other users. With the assistant of malicious users, the intermediaries can easily exclude the malicious users from a given batch. On the other hand, some users might be nonmalicious but just uncooperative. For example, they do not follow the designed rule to initialize dummy media packets if their calling requests are rejected. We will evaluate this threat model in the future work.

More realistic traffic models: To simplify, we employ an idealistic model in this paper without considering optional signaling packets, RTCP packets and re-INVITE request, etc. In future work, we will take these packets into account to find out how they affect the attacks and prevention solutions.

Long-term intersection attacks: As introduced in Section 4.1, our solution in this paper only provides anonymity for a single call. However, the provided anonymity may be broken in a long run. For example, Alice participates in several calling batches. Let us assume that Alice always calls Bob in these batches while the other users participating in these batches have different callees. In this way, it is easy for attackers to intersect the $\mathcal{C}_a$ in different rounds to find Alice's real contact, Bob. We will investigate the long-term intersection attacks in the future.

Lawful interception: Telecommunication providers are required to support Communications Assistance for Law Enforcement Act (CALEA) in many countries for national security and the investigation of serious crime. Lawful interception means that a law enforcement agency is authorized to intercept both the conversation content and the calling records for a particular user. For example, the EU Council Resolution of 17 January 1995 on the lawful interception of telecommunications (96/C 329/01) [21] declared the surveillance on telecommunication as a mandatory requirement. The solution discussed in this paper does not contradict the requirement of lawful interception since the VSP is still able to learn everything about the conversations it served. What the solution prevents is only the unauthorized intermediaries on the communication channels. Nevertheless, the interface for lawful interception based on this solution is still needed to be taken into account.

6. RELATED WORK

Recent work on VoIP privacy protection is mainly focused on the following fields:

Information hiding on signaling flows: Peterson [22] and Shen et al. [14] demonstrated a comprehensive summary of privacy-sensitive message fields in SIP. Some optional headers (e.g., Subject) is not essential for achieving the intended purpose of the messages and can be removed by users without any side-effects for privacy purpose. Moreover, non-optional headers (e.g., To, From, Via and Contact) can be replaced by the VSP or a trusted third party with randomized values. However, the relationship between the original values and the random values must be cached on the VSP or the trusted third party for routing responses. Karopoulos et al. [23] proposed a framework to separate caller and callee's identities based on encryption in multi-domain environments. The caller encrypts the identities of the caller and the callee in a SIP message by the keys shared with the caller's domain proxy and the callee's domain proxy respectively. In this way, no such a single party exists which can see both the identities of the caller and the callee. Unfortunately, even the identity information in signaling and media flows are well protected, it does not prevent intermediaries from profiling "who called whom" using traffic analysis attacks.

Traffic analysis attacks on media flows: Some VoIP users make calls over commercial relays for anonymity. Wang et al. [24] demonstrated that such a solution is vulnerable to active traffic analysis attack: An attacker can embed unique watermark into the encrypted VoIP flow by slightly delaying of random selected packets. In this way, an attacker can find out who called whom by encoding and decoding the watermarks on both side of the relay. Verscheure et al. [25] proposed a method to reveal calling records by exploiting the human conversation pattern: When one speaks, the other usually listens. This "alternate in speaking and silence" represents a probabilistic rule of VoIP communication. Taking this into account, the caller and the callee's flows are probabilistic linkable if the attackers can detect the silence period and voice period for a flow. This attack is mainly against those VoIP systems which support silence suppression. These two papers are only focused on attacks and no countermeasure solutions are given.

Information leaking of VBR codec: As introduced in Section 2, VBR technique allows the codec to change its bit rate dynamically according to the speech audio. Thus, the user-agents generate media packets with different sizes if they apply VBR codec. Wright et al. [26, 27] demonstrated attacks to identify the spoken language or partial conversation content of encrypted media packets by using the packet-length information. Moreover, the packet-length information may also enable attackers to recognize the speaker [28].

Solution based on unlinkable identity: Munakata et al. [29] proposed a user-driven privacy mechanism by introducing Globally Routable User Agent URIs (GRUU)[30] and Traversal Using Relays around NAT (TURN)[10]. The users can obtain a SIP URI (temp GRUU) and a IP address (IP address of a TURN server) which are unlinkable to their real identities. The proposed mechanism in [29] enables VoIP users themselves to achieve anonymity by using unlinkable identities that are functional yet anonymous. However, this method does not mitigate traffic analysis as well: Intermediaries on both side of a TURN server can still profile the mapping relationship of its relayed flows.

Solution based on MIX concept: Melchor et al. [20] discussed three MIX techniques for VoIP media flows providing strong resistance against traffic analysis. The three MIX techniques are based on dummy traffic, broadcasting and private information retrieval respectively. They further evaluated the performance of these techniques using a theoretical model. Nevertheless, their techniques are too expensive to be deployed in reality.

Our paper addresses VoIP anonymity using different system model and adversary model. We do not focus on the confidentiality of message content, but consider a global adversary model of passive traffic analysis attacks. We performed a formalized analysis taking both the signaling and media flows into account.

7. CONCLUSION

VoIP data are usually transmitted over large-scale networks with untrusted intermediaries. The intermediaries can thus wiretap the packets passing by and then easily find out the calling records ("who called whom") from the destination and source fields of the packet-headers. However, many Voice Service Providers (VSP) relay both the signaling packets and media packets between users. Thus, the intermediaries cannot observe the calling records from packet-headers directly. Instead, passive traffic analysis attacks can be mounted: the intermediaries need to correlate the flows entering and leaving the VSP using patterns of the flows. A flow usually shares some unique patterns with the flow being relayed. In this paper, we proposed a formalized model to address the patterns (e.g., packet size, payload, arrival time) for both signaling flows and media flows. The result shows that the attacks can be easily succeed without additional security measurements.

There are currently no such a practical solution to mitigate passive traffic analysis attacks for VoIP users. Some existing ideas (e.g., broadcasting flows or generating long-term dummy flows) are too heavy for VoIP and waste resources. Our paper discussed countermeasures to eliminate or equalize the flow patterns while taking VoIP context into account. We also find that it is helpful to defend against traffic analysis attacks by enforcing VoIP configuration parameters (e.g., use FBR codec and do not use silence suppression). Moreover, we proposed a example defending solution by integrating some Mix-based methods into the VSP components.

Our future work will be mainly in two directions: *(1)* We are going to investigate the performance issues of the proposed solution, especially on the end-to-end latency of media flows, and *(2)* We will address more sophisticated adversary models including active traffic analysis attacks, malicious users, etc. We hope that this paper motivates the community of researchers in the area of both IP telecommunication and networking anonymity to work towards practical solutions for VoIP anonymity protection.

8. REFERENCES

[1] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[2] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy (SP '03)*, Washington, DC, USA, 2003. IEEE Computer Society.

[3] O. Berthold, H. Federrath, and S. Köpsell. Web MIXes: a system for anonymous and unobservable internet access. In *Proceedings of International workshop on Designing privacy enhancing technologies*, pages 115–129, New York, NY, USA, 2001. Springer-Verlag.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proceedings of the* 13^{th} *conference on USENIX Security Symposium (SS '04)*, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[5] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes: Untraceable communication with small bandwidth overhead. In *Kommunikation in Verteilten Systemen, Grundlagen, Anwendungen, Betrieb, GI/ITG-Fachtagung*, pages 451–463, London, UK, 1991. Springer-Verlag.

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol, 2002. RFC 3261.

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications, 2003. RFC 3550.

[8] K. Egevang and P. Francis. The IP Network Address Translator (NAT), 1994. RFC 1631.

[9] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox communication architecture and framework, 2002. RFC 3303.

[10] J. Rosenberg, R. Mahy, and P. Matthews. Traversal using relays around nat (TURN): Relay extensions to session traversal utilities for NAT (STUN), 2010. RFC 5766.

[11] R. Zopf. Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN), 2002. RFC 3389.

[12] G.711. http://www.itu.int/rec/T-REC-G.711/e, visited at 21th-Oct-2009.

[13] Speex. http://www.speex.org/, visited at 21th-Oct-2009.

[14] S. Chen, X. Wang, and S. Jajodia. On the anonymity and traceability of peer-to-peer VoIP calls. *Network, IEEE*, 20(5):32–37, 2006.

[15] A. Pfitzmann and M. Hansen. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management - a consolidated proposal for terminology. Technical report, February 2008.

[16] ITU-T. Recommendation G.114 - One-way Transmission Time, 2003.

[17] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright. Timing attacks in low-latency mix systems (extended abstract). In *FC '04: Proceedings of the* 8^{th} *international conference on Financial Cryptography*, pages 251–265, Berlin, Heidelberg, 2004. Springer-Verlag.

[18] G. Zhang and S. Fischer-Hübner. Peer-to-peer VoIP communications using anonymisation overlay networks. In *Proceedings of the* 11^{th} *Conference on Communications and Multimedia Security (CMS '10)*, Linz, Austria, 2010. Springer-Verlag.

[19] M. Vlachos, A. Anagnostopoulos, O. Verscheure, and P. S. Yu. Online pairing of VoIP conversations. *The VLDB Journal*, 18(1):77–98, 2009.

[20] C. A. Melchor, Y. Deswarte, and J. Iguchi-Cartigny. Closed-circuit unobservable Voice over IP. In *Proceedings of the* 23^{rd} *Annual Computer Security Applications Conference (ACSAC '07)*, pages 119–128, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[21] Council Resolution of 17 January 1995 on the lawful interception of telecommunications. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31996G1104:EN:HTML, visited at 21th-Jun-2010.

[22] J. Peterson. A Privacy Mechanism for the Session Initiation Protocol (SIP), 2002. RFC 3323.

[23] G. Karopoulos, G. Kambourakis, S. Gritzalis, and E. Konstantinou. A framework for identity privacy in SIP. *J. Netw. Comput. Appl.*, 33(1):16–28, 2010.

[24] X. Wang, S. Chen, and S. Jajodia. Tracking anonymous peer-to-peer VoIP calls on the Internet. In *Proceedings of the* 12^{th} *ACM conference on Computer and communications security (CCS '05)*, pages 81–91, New York, NY, USA, 2005. ACM.

[25] O. Verscheure, M. Vlachos, A. Anagnostopoulos, P. Frossard, E. Bouillet, and P. S. Yu. Finding "who is talking to whom" in VoIP networks via progressive stream clustering. In *Proceedings of the Sixth International Conference on Data Mining (ICDM '06)*, pages 667–677, Washington, DC, USA, 2006. IEEE Computer Society.

[26] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of* 16^{th} *USENIX Security Symposium on USENIX Security Symposium (SS '07)*, pages 1–12, Berkeley, CA, USA, 2007. USENIX Association.

[27] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy (SP '08)*, pages 35–49, Washington, DC, USA, 2008. IEEE Computer Society.

[28] L.A. Khan, M.S. Baig, and A. M. Youssef. Speaker recognition from encrypted VoIP communications. *Digital Investigation*, In Press, Elsvier, 2009.

[29] M. Munakata, S. Schubert, and T. Ohba. User-agent-driven privacy mechanism for SIP, 2010. RFC 5767.

[30] J. Rosenberg. Obtaining and using globally routable user agent uris (GRUUs) in the session initiation protocol (SIP), 2009. RFC 5627.

Work in Progress: Inter-Domain and DoS-Resistant Call Establishment Protocol (IDDR-CEP)

Patrick Battistello[*]
Orange Labs
2 av. Pierre Marzin, 22307 Lannion, France
patrick.battistello@orange-ftgroup.com

ABSTRACT

VoIP security is a tricky issue in inter-domain *open* context where interconnection proxies are reachable from anywhere on the public Internet and may be the subject of DoS and SPIT attacks. This paper proposes a secure call establishment protocol designed for this context with a particular focus on DoS protection. The mechanism performs session key agreement in the signalling plane and can be integrated to SIP call establishment. It is based on symmetric cryptography algorithms and implicit transaction identifiers to protect against DoS attacks. We provide heuristic analysis of various security properties among which privacy and resistance to off-line passive attacks. The IDDR-CEP protocol is presented in a three party architecture but can be adapted to a two party architecture; it may also be adapted to non-VoIP applications.

Categories and Subject Descriptors

C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks—*General, Security and Protection*

General Terms

Security

Keywords

VoIP, security, inter-domain, DoS-resistant, call establishment, key agreement, authentication, privacy, SPIT, token, ticket, Kerberos

1. INTRODUCTION

VoIP has become a major technology, both for residential and professional customers. It enables voice and data integration along with new services (notification, presence, WebPhone, ClickToPhone,...). On the other hand, VoIP raises new issues amongst which the security of communications [6], which is the subject of this paper.

1.1 Background

Three main architectures can be found in VoIP networks. First the *initial architecture*[1] which comes from the incumbent telco operators and is a flat centralized architecture based on registrar and proxy servers. Secondly, the *P2P architecture* which distributes the registrar and proxy functions over a set of nodes and combines the DHT (Distributed Hash Table) concepts with usual VoIP protocols as described in [12]. Finally, the *WEB architecture* which brings the VoIP endpoint directly in the client's WEB browser thus leading to the WebPhone and ClickToPhone concepts. Among the various protocols for VoIP, SIP [24] has become the fundamental signalling brick in association with SDP for session description and RTP for conveying the media flows. Further in this article, we will use the `SIP-INVITE` request to indicate the beginning of the call establishment process.

1.2 Current stakes in VoIP security

Several threats have been pointed out in many studies like [6], [2], [10], [13] but their impact strongly depends on the context, especially intra-domain or inter-domain contexts.

In the *intra-domain context*, VoIP communications remain confined in the same administrative domain, whatever the underlying network architecture[2]. Each user endpoint is authenticated by a unique identifier (ID) and the operator proxies usually stand in the call signalling path. Because of these characteristics, specific VoIP risks are limited to DoS and SPIT (SPam over Ip Telephony). DoS risks come from the protocols complexity and the need to maintain call context. SPIT originates from (possibly compromised) softphones thus primarily in VoIP P2P and WEB architectures but now also in incumbent operator architectures with new mobile endpoints.

The threats become higher in the *inter-domain context* where several of the previous architectures are interconnected; an exhaustive taxonomy is provided in [17]. However, the risk level depends on the chosen interconnection approach.

A first approach, noted *open model* hereafter, assumes that IP connectivity between endpoints, proxies or domains and DNS lookup are sufficient to establish multimedia communications, just like the e-mail model. The first major issue here is that VoIP identifiers are designed to convey a domain

[*] Also temporally affiliated to Telcom Bretagne, France.

[1] In reference to the first ITU-T and IETF VoIP standards.

[2] However interconnection with PSTN is not precluded.

part but, as explained in [23], because of the huge hardphone installed base and of the PSTN predominance, most VoIP calls originate from, or terminate in, a PSTN cloud. Consequently, caller (or callee) identifiers lack the domain part, so inter-domain calls are tricky to route and vice versa to verify. Where a public-ENUM-like system[3] would solve this problem, it is foreseen in [23] that such a solution is unlikely to appear in the public domain. The second major issue is the accumulation of DoS and SPIT risks over the interconnection points which is called the "pinhole problem" in [23] and is very similar to the e-mail vulnerabilities.

An alternative approach, noted *closed model* or *private federations* hereafter, is based on a contractual agreement between a set of operators to share a secure interconnection architecture. The IMS (IP Multimedia Subsystem) standards [1] define such an architecture with secure links (IPSec) between domains and network topology hiding to protect against DoS attacks towards the interconnection proxies. The phone number problem is solved by sharing securely `E.164` information between the operators.

Finally, a very recent approach *VIPR* [23] proposes an *hybrid model* combining VoIP, P2P and PSTN components. In brief, it relies on PSTN call information to build secure routing and authentication information which are then used to place direct VoIP calls.

1.3 Problem statement

It is unlikely that VoIP calls remain confined in intra-domain context, although it offers the highest security. For inter-domain calls, the *closed model* reproduces the PSTN principles and is expected to reach comparable security. However, its architecture is not designed for "any-to-any" sporadic communications between domains (like the e-mail architecture). Therefore, we anticipate that this model will coexist with *open model* or *hybrid model* where the security risks are much higher because VoIP proxys are reachable from anywhere on the public Internet and also because of the difficulty to route and verify the `E.164` call identifiers.

As explained in section 2, the current solutions available for *open* or *hybrid model* show some limitations. Consequently, we identify the need for a secure call establishment protocol for inter-domain context which addresses these issues and takes into account the VoIP specificities, especially: its real-time nature and the regulatory constraints[4] which may have a strong impact on deployment [11]. It should also meet the "usual" security requirements of key establishment protocols: mutual authentication, session key freshness, access control, privacy, DoS protection, Perfect Forward Secrecy (PFS) and anti-replay[5]. The key freshness requirement is taken in its broader scope meaning a new session key shall be established at each new call. The PFS requirement shall apply to both passive off-line attacks (like password guessing) and compromission attacks where the adversary has obtained the long term secrets of one or several parties; under these hypothesis the adversary shall still not be able to recover any of the past session keys. It is assumed that the adversary has all the required capabilities for protocol interaction (message interception, tampering, replay, forging or deletion) and may be also an *insider* as described in [7].

The remaining of this article is organized as follows: section 2 analyses the related work in VoIP secure call establishment. In section 3, we provide the specifications of the IDDR-CEP protocol. Section 4 shows some possible implementation of this protocol in a VoIP context. Finally, section 5 provides the discussion and the conclusions.

2. RELATED WORK

2.1 Authentication and key establishment

As often done in the litterature [7], we associate in this section the authentication and key establishment aspects. The `E.164` call identifiers problematic is also implicitely associated to the authentication phase. We consider mainly authentication between VoIP domains, assuming each domain is responsible for authenticating its own endpoints.

The first possible solution is the use of TLS (or DTLS or IPSec) on a hop-by-hop basis with inter-domain authentication. Once the secure link is established, the session key can be transported in the `SIP-INVITE` request by using SDP specific security parameters [4]. In addition to the analysis provided in [9], this approach does not solve the `E.164` identifiers problematic and it does not meet the PFS (Perfect Forward Secrecy) requirement since compromission of a private key will expose all the previous session keys. Also, it is unclear how an inbound proxy facing the public Internet would behave in case of a DoS attack at the TLS level.

The *SIP Identity* protocol [22] supports end-to-end authentication of the calling domain by adding a digital signature in the `SIP-INVITE` request. Called domain authentication can be supported in return with the extension defined in [8]. The overall solution supports neither privacy nor key establishment (because the `SIP-INVITE` request is not encrypted). Also, as explained in [9], this mechanism seems vulnerable to DoS attacks if the called domain is flooded with spoofed requests containing invalid signatures. The *SIP Identity* protocol was also proposed in conjunction with the *E.164-RRC* (Return Routability Check) mechanism [26] to perform `E.164` caller ID verification. The main idea was to return a verification request towards the `E.164` calling number with a random token (nonce) to be signed. Unfortunately, this verification phase assumes that the `E.164` routing problematic has been solved previously.

More recently, the *VIPR* proposal [23] claims to solve the phone number routing problem by combining VoIP, P2P and PSTN technologies. It requires that each domain joins a global open P2P network and publishes in the DHT the list of its phone identifiers and at least one of its VoIP proxy. Once an inter-domain PSTN call is completed, if the called number can be found in the DHT, the calling domain contacts the called domain and obtains a "cryptographic call token" bounded to the specific called number and specific calling domain, along with SIP routing information to place direct VoIP calls in the future. While this mechanism offers an incremental approach, we envision some limitations: the called endpoint has no guarantee that the calling number

[3] ENUM stands for E.164 to Uniform Resource Identifiers.

[4] Especially the capability to disclose session keys when required by legal procedure.

[5] The reader is referred to [7] for a more detailed definition and illustration of theses requirements.

has not been spoofed, each domain has to store potentially a large number of tokens and besides all it requires PSTN *endlessly*. Actually, each time a token validity has expired or a new destination is being called, a PSTN verification is required. Even worse, if a signature key gets compromised, the process involving PSTN shall be repeated for all the previously issued tokens. It should be noted that some *VIPR* ideas were previously proposed in [20] where authentication tokens are also inserted by the caller in the `SIP-INVITE` requests. The constraints (token storage and security) are almost the same since the tokens have a long-term validity.

Another approach, still in the signalling plane, is MIKEY [5] which can be integrated into the VoIP SIP call establishment[6] and supports three authentication modes: pre-shared key (PSK), public key (PKI) or Diffie-Hellman (DH) exchange. Recently, a new MIKEY mode denoted *KMS* was proposed in [14]. It is a ticket-based approach with a trusted third party, inspired from Kerberos [16], which can also be integrated into the VoIP call establishment and addresses the call forking specificity. According to [9], the MIKEY protocol is subject to DoS attack and we anticipate the same issue with KMS extension[7]. Also, except for MIKEY-DH, the PFS security property is not met since the other modes use session key transportation.

Specific inter-domain protocols were also proposed for authentication and key exchange. Among them, [27] is a four-party protocol organized in three tiers, using identity-based cryptography and secret public keys. While this protocol provides PFS and resistance against off-line and active attacks, it may not be resistant to DoS attacks since a single fake message from sender A will generate a total of 6 messages from both the responder B, the domain server S_B and the domain server S_A. Another inter-domain protocol is proposed in [21]; it is based on an improved proxy ElGamal encryption scheme which enables two users in two distinct domains to exchange a cipher text through a proxy server in each domain. This scheme assumes that a shared inter-domain key $K_{DM1,DM2}$ is set up; consequently PFS is not achieved because if this key is compromised all the previous exchanges are revealed.

Session key establishment may also be performed in the media plane with the ZRTP [29] or DTLS-SRTP [15] protocols. Operating the key exchange in the media plane may conflict with regulatory requirement of key disclosure and, as explained in [9], this does not save the need for a security infrastructure at the signalling plane in most of the cases.

2.2 DoS protection

Several DoS-resistant protocols have been proposed with the common attempt to limit resources consumption by the responder at the very first step. For this purpose, a cookie is often returned by the responder and has to be acknowledged by the originator. This is done for example in [3] where the cookie includes the initiator information and the responder's current exponential. Although the responder does not commit memory resources at the very first step, it still has to return a message to the initiator and compute one exponentiation each time a spoofed initiation message is received. Even worse, if the adversary acknowledges the cookie, the responder has to verify a fake public-key signature on the third message. The same analysis and conclusions hold for [25] where the responder can still not authenticate the first initiator message thus exposing him to DoS attack in subsequent messages.

An alternative approach is described in IPACF [28] where each message conveys an access filter value which is "trivial" to check. This filter value is updated at each new frame and it depends on a shared secret key established between the server and each client. When the server receives a frame with a valid filter value, it responds to the client with a new responder filter value and updates the client filter value for the next frame; the same process holds for the client. This mechanism also provides user privacy by sending a pseudo-ID which is user specific and changes at each frame. However, it is unclear how the protocol behaves if one (or several) frames are lost or disordered and whether strictly bidirectional exchange has to be maintained between the server and each client.

3. PROTOCOL SPECIFICATIONS

3.1 General overview and technical novelty

Starting from the architectural view, and the simplified case, the protocol runs between entities A and B. Entity A is the initiator, that is the caller or calling entity in a VoIP context. It may be the user endpoint itself or a proxy acting on its behalf which is able to authenticate the caller identity ID_A. Entity B is the responder, that is the callee or the called entity. It may be the user endpoint itself or a proxy acting on its behalf which is able to authenticate the callee identity ID_B. Entities A and B share a secret from which they can perform secure transactions. Since installing shared secrets between each couple of entities is not scalable, an intermediary server S is involved in the general case. Entity S is responsible for authenticating A and B and takes part in each transaction from A to B, provided it receives a valid transaction request from A. This means that S is online and has shared-secrets K_{AS} and K_{SB} with respectively entities A and B. A protocol transaction is defined as the set of IDDR-CEP messages leading to the receipt by B of the Authenticated Message (AM) related to the original message (MES) held by A. Finally, S is responsible for performing phone number routing and verification between domains when necessary. This design choice comes from the previous statements that phone number authentication always requires a trusted third party; in *VIPR* [23] this is explicitly the PSTN network whereas with *e164-RRC* [26] this is SIP routing and thus implicitly some kind of underlying PSTN routing.

This architectural setting is similar to Kerberos, but the protocol is different and it brings several properties that Kerberos is lacking: PFS is achieved even if K_{AS} and K_{SB} are compromised, the protocol is DoS resistant and the size of the final message from A to B is not expanded significantly. This last property is achieved because no key transportation is used and consequently the UDP transport can be preserved. As detailed in section 4, this three party architecture is quite flexible because entities A and B can be

[6] It does not require prior establishment of a secure link.

[7] This is because the receiver has no means to check the ticket validity prior to contacting the KMS.

from different domains or from the same domain; the roaming situation where entity A or B is in a visited domain is also supported. Finally, entity S may be located in a third domain or in the calling domain or in the called domain. The rest of the description keeps the general three entities case but it can be easily adapted to the simplified two entities case by assuming A = S.

There are two ways to integrate this protocol with VoIP. The first one is when the (final) IDDR-CEP Authenticated Message (AM) is a `SIP-INVITE` request conveying the security information. This "on-top" approach is illustrated in all the implementation examples given in section 4. The second way is to consider IDDR-CEP as a key establishment protocol in itself. In that case, the Authenticated Message (AM) is the first (and only) message required from A to B for authentication and key agreement. This alternative approach may be used to set-up a secure link for exchanging VoIP trafic; it is not further described in this paper.

The DoS resistant property is achieved mainly by using symmetric cryptographic algorithms (for both entities authentication and key agreement) and by inserting an identification value $TRID_N$ in each AM message. This identification value (also called token) serves for filtering purposes on the responder side (B entity). It can be checked straightaway by B entity by comparing the received value to the pre-computed (expected) value. Consequently, any AM message with invalid $TRID_N$ value is immediately discarded and also, since the filter value changes at each transaction, the responder is protected from replay attacks. The B entity is also protected from blind attacks because it is not engaged in any processing (cryptographic computation, context handling or message generation) before a valid $TRID_N$ value is detected. This contrasts with protocols like SIP Identity [22], Kerberos [16] or MIKEY [5] where the responder has to perform at least one cryptographic operation before further processing of the message.

From this perspective, IDDR-CEP follows the same filtering principle as the IPACF protocol [28] and extends its usability from the access to the inter-domain area. However, the $TRID_N$ value also serves as a transaction identifier among a possible set of transactions (called a *transaction window*) thus achieving two properties which seem to be lacking in IPACF: the handling of transactions loss or disordering. More precisely, the $TRID_N$ value is the result of a one-way cryptographic function which depends on the current transaction index N maintained by the responder. The value of N is kept secret and only its public image $TRID_N$ is sent as a reference to the current transaction index.

Finally, the DoS protection is increased by the structure of the protocol exchange. Unlike KMS [14] where B has to contact the trusted third-party (S) for checking each message it received from A and for obtaining the session key, in IDDR-CEP the responder B can retreive all the cryptographic material from the single message sent by A. More precisely, the session key $K_{AB,N}$ between A and B is established with a key agreement scheme: it is pre-computed by B as a function of the current transaction index N and therefore it is uniquely identified from the received $TRID_N$ value. This approach contrasts with several protocols like TLS, Kerberos, MIKEY or KMS where the (encrypted) session key is provided to the responder by the initiator or by a trusted third party. The first benefit is that the final AM message sent to B is much shorter. Moreover, this removes the risk of off-line passive attacks since no encrypted key is transported to B.

As a summary, the IDDR-CEP approach improves the properties of both Kerberos and IPACF protocols by adding respectively the DoS protection, the PFS property and the support for transaction loss or disordering. Other security properties are achieved as explained at the end of this section.

3.2 Definitions and notations

$TRID_N$: public image (or identifier) of transaction index N. This identifier shall depend (at least) on the value N and be the result of a cryptographic function such that knowing any number of $TRID_{N-i}(i \geq 0)$ identifiers it is impossible to determine either N or any of the following $TRID_{N+j}(j > 0)$ identifier [8].

$K_{AB,N}$: the session key shared between A and B after completing transaction of index N. This key shall depend (at least) on N and K_{SB} and be the result of a cryptographic function such that knowing $K_{AB,N}$ and N it is impossible to infer K_{SB}.

$H(V)$: hash of value V where H is a cryptographically secure one-way hash function.

$MAC_K(V)$: Message Authentication Code applied to message V and based on key K.

$\{V\}_K$: encryption of value V using symmetric key K.

$V_1 \oplus V_2$: the result of V_1 XOR V_2.

$||$ or , : concatenation operator.

$len(V)$: binary length of value V.

$[V]$: optional value V.

$trunc(V, n)$: the n leftmost bits of value V.

3.3 Operations on the responder side

This section describes the operations between entities S and B assuming the shared secret K_{SB} has been previously established. Along with the shared secret, entities S and B agree on the initial transaction index N_0 which shall also be kept secret. From the shared secret K_{SB} and the current transaction index N, entities S and B simultaneously derive the $TRID_N$ and $K_{AB,N}$ values associated to this transaction (cf. section 3.2). An additionnal key $K_{SB,N,MAC}$ is also derived from N and K_{SB} through a cryptographic function such that knowing $K_{SB,N,MAC}$ and N it is impossible to infer K_{SB}. The $K_{SB,N,MAC}$ symmetric key is used for authentication and integrity protection of information sent from S to B.

[8] In this context, and further on, impossible means "computationaly non-feasible with non-negligible probability".

At each transaction, B receives one of the following Authenticated Message (AM):

$$\begin{aligned} x \rightarrow B \quad &: \quad TRID_N, TRCheck_N, MES, \\ & \quad\; MAC_{K_{AB,N,A}}(TRID_N, TRCheck_N, MES) \\ x \rightarrow B \quad &: \quad TRID_N, TRCheck_N, \{MES\}_{K_{AB,N,E}}, \\ & \quad\; MAC_{K_{AB,N,A}}(TRID_N, TRCheck_N, \\ & \quad\; \{MES\}_{K_{AB,N,E}}) \end{aligned}$$

The only difference between these two AM variants is that in the first case the original message MES is sent in clear whereas in the second case it is encrypted. When MES is encrypted, the symmetric key $K_{AB,N,E}$ is used; this key shall be derived from $K_{AB,N}$ through a public function. In both messages, the sender x stands either for entity A or for entity S. Usually the AM message will be sent directly by entity A but, in some cases, the authorization server S would prefer staying in the signalling path. When the AM message is sent by entity A this implies that A has previously obtained all the transaction material ($TRID_N$, $TRCheck_N$ and $K_{AB,N}$) from S (cf. section 3.4).

When receiving the AM message, entity B first checks that the $TRID_N$ value matches the current identifier for transaction of index N. This check is "computation-free" for B since it just has to compare the received value with the pre-computed value of $TRID_N$ for transaction of index N. If the transaction identifier is valid, then B checks the MAC code which applies to all the information contained in the message. The MAC code is based on the symmetric key $K_{AB,N,A}$; this key shall be derived from $K_{AB,N}$ through a public function. When $K_{AB,N}$ is only used to compute this MAC no derivation is required ($K_{AB,N,A} = K_{AB,N}$).

Finally, entity B has to check the $TRCheck_N$ information (cf. section 3.4) which contains at least a MAC based on the $K_{SB,N,MAC}$ symmetric key. The $TRCheck_N$ information is computed by S and proves to B that the transaction was authorized by S along with the main authentication parameters used by S. When the AM message is sent by S to B, the $TRCheck_N$ information is no longer necessary since S can guarantee itself the integrity of the AM message.

Most of the time, entity S will authorize "concurrent" transactions towards entity B from a set of entities A_i. In this context, transaction loss or disordering may occur meaning that AM messages sent by each A_i entity may not be received by B in incremental transaction order or some of them may be lost. This ordering problem may be the consequence of heterogeneous processing powers among A_i, of network message loss, or of compromised A_i entity requesting transaction to S but not sending the corresponding AM message to B. For this reason, it is suggested that B maintains a sliding *transaction window.*

More precisely, if the current transaction index between entities S and B is N, B creates a transaction window of size Δ, and pre-computes the transaction identifiers $TRID_{TI}$ for the Δ consecutive transactions starting from index N. For performance optimization, B may also pre-compute the keys $K_{AB,TI}$ and $K_{SB,TI,MAC}$ associated to each $TRID_{TI}$ value[9]. When receiving an Authenticated Message with transaction identifier $TRID_X$, B verifies if $TRID_X$ matches one of the pre-computed $TRID_{TI}$ values inside the transaction window. If no match is found, the message is silently ignored, otherwise it is processed and the transaction window is shifted forward according to the TI value. This checking process is still "computation-free" for B since he has to perform *at most* Δ comparisons (instead of one without the transaction window). It is important to note that entity S *does not have* to maintain a transaction window on its own since it simply authorizes transactions consecutively.

Several algorithms may be applied to iterate the transaction index N starting from its initial value N_0 set up between S and B. The simplest way is to increment the N value by 1 at each new transaction. Unfortunately, such a linear scheme does not achieve the PFS property because if entity S or B is compromised the adversary obtains the K_{SB} and N secrets and consequently all the previous session keys associated to the past transaction indexes ($N - i$, $i > 0$) are revealed . For this reason, the transaction index shall be iterated through a one-way function like $N_{+1} = H(N)$.

3.4 Operations on the originator side

This section describes the operations between entities A and S leading to the transmission to B of the Authenticated Message (AM). Since the DoS protection is also important on the originator side, especially for S which is the trusted third-party (and has to be online), we retain the same principle of transaction identifiers which serve as filtering values for both A and S. This implies that during the initial configuration stage, in addition to establishing the shared secret K_{AS}, entities A and S also agree on a start-up value M_0 for the transaction index M. The current transaction index M between entities A and S has the same semantic and usage as the transaction index N between entities S and B. To be more precise, we should note M_i the (current) transaction index between entity A_i and S[10], but for simplicity of notation we omit the i sub-index considering there is one single A entity in the protocol description. The transaction index M between A and S should be iterated also with a one-way function to achieve the PFS property. It should be noted that the M value between entities A and S is independent from the N value between entities S and B.

From the shared secret K_{AS} and the current transaction index M, the following information is computed independently by entities A and S:

$TRIDA_M$: transaction identifier (or public image) of transaction index M inserted in the authorization request from A to S. This identifier shall depend at least on the M value and be the result of a function such that knowing any number of $TRIDA_{M-i} (i \geq 0)$ identifiers, it is impossible to determine either M or any of the following $TRIDA_{M+j} (j > 0)$ identifier.

$TRIDS_M$: transaction identifier (or public image) of transaction index M inserted in the authorization response from

[9] An efficient algorithm to pre-compute all the transaction material is provided in section 4.1.

[10] And similarly N_j the (current) transaction index between entities S and B_j.

S to A. This identifier has the same semantic and properties as $TRIDA_M$.

$K_{AS,M,MAC1}$: this symmetric key depends on K_{AS} and M. It is used to compute the MAC code inserted in the authorization request.

$K_{AS,M,MAC2}$: same as $K_{AS,M,MAC1}$ for the authorization response.

$K_{AS,M}$: this key is used as a one-time pad to convey securely the $K_{AB,N}$ and $TRID_N$ information computed by S to A. The $K_{AS,M}$ key shall depend at least on M and K_{AS} and be the result of a function such that knowing $K_{AS,M}$ and M it is impossible to infer K_{AS}. Its length shall verify: $len(K_{AS,M}) = len(TRID_N) + len(K_{AB,N})$.

The protocol exchange between A and S is compounded of the two following messages:

$$\begin{aligned} A \rightarrow S \quad &: \quad TRIDA_M, ID, \\ &\quad\; MAC_{K_{AS,M,MAC1}}(TRIDA_M, ID) \\ S \rightarrow A \quad &: \quad TRIDS_M, [ID'], OP, TRCheck_N, \\ &\quad\; MAC_{K_{AS,M,MAC2}}(TRIDS_M, [ID'], OP, \\ &\quad\; TRCheck_N) \end{aligned}$$

In the first message, which is called the authorization request, entity A indicates to S its wish to send to B the original message MES[11]. The ID set of information contains at least ID_A and ID_B identities; in some circumstances, entity A may not know the identity of B which has to be set by S. Additional information such as the characteristics or purpose of message MES as well as previous transaction information may be inserted in ID, especially if S has to set the identity of B. The authorization request also contains the transaction identifier $TRIDA_M$ which changes at each transaction. The whole message is authenticated and integrity protected with a MAC based on the $K_{AS,M,MAC1}$ key. When privacy is required, the ID set of information may be encrypted by using a symmetric key derived from (at least) K_{AS}.

When receiving the authorization request, entity S first checks that it contains a valid transaction identifier $TRIDA_M$. If the transaction identifier $TRIDA_M$ is valid, this enables S to identify A and to retrieve (or compute) the keying material associated to this transaction with A. Based on the $K_{AS,M,MAC1}$ key, S is able to check the MAC code and thus to authenticate A and verify the authorization request integrity. Assuming the transaction is accepted by S, it retrieves (or computes) the transaction material it shares with B for the current transaction index N ($TRID_N$, $K_{AB,N}$ and $K_{SB,N,MAC}$). Then S forms the authorization response for A which contains the information shown in the second message:

$TRIDS_M$: transaction identifier provided by S in response to $TRIDS_A$ for the current transaction of index M.

OP: public operand computed by S to convey securely the $TRID_N$ and $K_{AB,N}$ values needed by A to contact B. The OP operand combines these two values with the $K_{AS,M}$ key which is used as a one-time pad. A possible scheme for producing OP is:

$$OP = K_{AS,M} \oplus (TRID_N || K_{AB,N})$$

$TRCheck_N$: authentication information provided by S to B to prove that S has authenticated and allowed the transaction of index N for entity A. The set of information $TRCheck_N$ shall be constructed in such a way that it can not be manipulated by A and also that it can not be used by an adversary to impersonate A. For this purpose, $TRCheck_N$ includes at least a MAC based on the $K_{SB,N,MAC}$ key and computed over the transaction main identifiers: ID_A, ID_B and optionally other fields like the N value or the contact addresses of entity A or B ($@IP_A$, $@IP_B$). However, the transaction index N shall never appear in clear text. The various ways to produce $TRCheck_N$ can be expressed as follows:

$$\begin{aligned} TRCheck_N \quad = \quad & [ID_A], [ID_B], [@IPA], [@IPB], ..., \\ & MAC_{K_{SB,N,MAC}}(ID_A, ID_B, [@IP_A], \\ & [@IP_B], ..., [N]) \end{aligned}$$

ID': optional subset derived from ID in the authorization request. This may be used to convey the ID_B identity when this information is set by S.

When receiving the authorization response, entity A checks the $TRIDS_M$ identifier against the expected one and verifies the MAC code based on the $K_{AS,M,MAC2}$ key which is specific to this transaction. If the message is valid, entity A then extracts the $TRCheck_N$ value and computes the $TRID_N$ and $K_{AB,N}$ values with the reverse operation:

$$(TRID_N || K_{AB,N}) = OP \oplus K_{AS,M}$$

Then A is able to form and send the AM message to B as explained in the previous section.

Compared to the operations on the responder side, entity S has to manage transaction identifiers $TRIDA_M$ and $TRIDS_M$ for all the possible A_i entities. Since there might be a large number of A_i entities, it is not possible for S to maintain a sliding transaction window with each A_i. This means that each Ai entity must respect the incremental order of the transaction index M it shares with S and initiate transactions sequentially rather than in parallel.

It should be noted that S may need to remain in the signalling path, in which case the protocol exchange between A and S is slightly modified as shown below. The ID information is replaced by MES (which may be sent in clear or encrypted), the $TRCheck_N$ and $TRID_N$ values are no longer required in the authorization response[12]:

$$\begin{aligned} A \rightarrow S \quad &: \quad TRIDA_M, MES, MAC_{K_{AS,M,MAC1}}(\\ &\quad\; TRIDA_M, MES) \\ S \rightarrow A \quad &: \quad TRIDS_M, OP, MAC_{K_{AS,M,MAC2}}(\\ &\quad\; TRIDS_M, OP) \end{aligned}$$

[11] When the protocol is used solely for key establishment, the MES message may be void.

[12] Consequently $OP = K_{AS,M} \oplus K_{AB,N}$.

3.5 Complete protocol exchange

The complete protocol exchange is shown in Figure 1, assuming that the AM message is sent by entity A, that the original message MES does not need to be encrypted and that the $K_{AB,N}$ session key is solely used to ensure AM integrity (no key is derived from $K_{AB,N}$). From this three party architecture, the simplified case with two entities is obtained by assuming A = S and consequently the first two messages are no longer necessary.

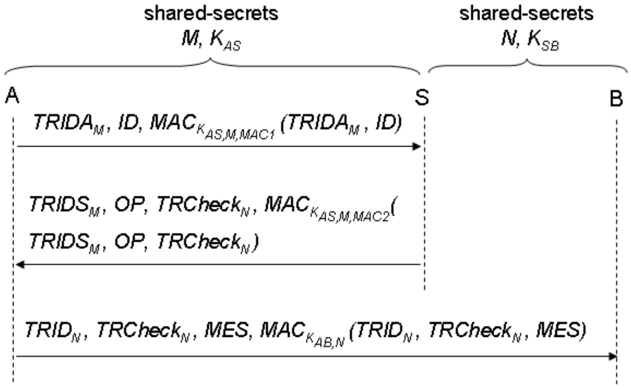

shared-secrets
M, K_{AS}
shared-secrets
N, K_{SB}
A
S
B
$TRIDA_M$, ID, $MAC_{K_{AS,M,MAC1}}(TRIDA_M, ID)$
$TRIDS_M$, OP, $TRCheck_N$, $MAC_{K_{AS,M,MAC2}}($
$TRIDS_M$, OP, $TRCheck_N)$
$TRID_N$, $TRCheck_N$, MES, $MAC_{K_{AB,N}}(TRID_N, TRCheck_N, MES)$

Figure 1: Complete protocol exchange.

3.6 Analysis of requirements

The IDDR-CEP protocol is now analysed regarding security requirements of section 1.3:

Mutual authentication: mutual authentication is under the responsibility of the trusted third party S. Entity S asserts to B that A has been authenticated under identity ID_A with the $TRCheck_N$ information. Let us suppose that A inserts a wrong identity $ID_{A'}$ in the MES section of the AM message then B computes $MAC_{K_{SB,N,MAC}}(ID'_A, ...)$ and detects that this does not match the MAC value contained in $TRCheck_N$. On the other way round, in the authorization response, entity S authenticates the identity of B or its contact address when necessary.

Access control: entity A has to authenticate towards S to communicate with B. Therefore, S can throttle the number of requests towards B in case A is an adversary or compromised entity. It is clear, although not detailed for simplification purposes, that the protocol works just the other way round when B wants to communicate with A.

Session key: the session key $K_{AB,N}$ meets the freshness property since a new key is generated at each transaction. With regards to key authentication, only entity A is able to compute the $K_{AS,M}$ key and thus retrieve $K_{AB,N}$ from the OP operand. Also B has the assurance to be speaking with A since $TRCheck_N$ authenticates ID_A but also possibly the contact address for A. Session key confirmation is not explicitly included in the protocol, but we assume it is realized in further messages sent from B to A[13].

Privacy: starting from the AM message, privacy is enforced as soon as $TRCheck_N$ does not carry any explicit identifier and MES is encrypted using a symmetric key derived from $K_{AB,N}$. Looking at messages between A and S, privacy is enforced as soon as the ID and ID' sets of information are encrypted (with a symmetric key derived from K_{AS}) while the $TRIDx$ values are sufficient to identify the sender.

DoS protection: the first protection comes from the use of symmetric cryptography and the chosen protocol exchange which does not require the responder B to contact a third party to check the AM message. The second protection comes from the use of transaction identifiers (i.e. $TRID_N$, $TRIDA_M$, $TRIDS_M$) which serve as a "computation-free" first level of verification for the receiving entity. Finally, all the transaction keys can be pre-computed thus *limiting the need for "on-line" sequential computation to just the OP operand and the MAC codes.* It should be noted that the OP value is trivial to obtain as soon as the $K_{AB,N}$ and the $K_{AS,M}$ keys are pre-computed.

Perfect Forward Secrecy: starting from the AM message and looking at off-line passive attacks, an eavesdropper may try to recover $K_{SB,N,MAC}$ and $K_{AB,N}$ by trying some kind of brute-force attack. For $K_{AB,N}$ this may be achievable, but this will not reveal any other $K_{AB,N\pm i}$ key. For $K_{SB,N,MAC}$ this seems impossible if the associated MAC incorporates the N index. Looking at messages between A and S, if an off-line adversary has previously guessed the $K_{AB,N}$ key, it can retrieve the $K_{AS,M}$ key from the OP value (which is passed in clear) but none of the others $K_{AS,M\pm j}$ keys. Concerning the transaction identifier $TRID_N$ (same analysis applies to $IDTRA_M$ and $IDTRS_M$), if it is the truncated result of a one-way hash function over a large N value, the probability to recover N from $TRID_N$ is almost null. However, for increased security, it is recommended that $TRID_N$ depends on both N and K_{SB} (cf. section 4.1). Now assuming an adversary has compromised entity S or B (same analysis applies to A), it has access to K_{SB} and the *current* N value but it can not retrieve the previous session keys because the N value can not be inverted[14]. In summary, the PFS property is obtained because IDDR-CEP is a key-agreement scheme and because the various keys used by the protocol are changed at each new transaction based on a one-way function.

Anti-replay: assuming the current transaction of index N is completed on the responder side (B entity), then the transaction window is shifted forward meaning the transaction index N and the associated values ($TRID_N$, $K_{AB,N}$ and $K_{SB,N,MAC}$) are no longer valid. Consequently, if a valid AM message is replayed, it will be silently ignored by B without requiring further processing. The same protection applies to the protocol exchange between entities A and S. Finally, because the keys are automatically renewed at each transaction, it seems impossible to inject in the protocol a previous session key.

[13] Assuming MES is a `SIP-INVITE` request, A will later receive a call establishment response from B which implicitly confirms the correct key establishment.

[14] It is implicitely assumed that all the keying material from previous transactions is automatically erased.

4. PROTOCOL IMPLEMENTATION

In this section we describe three modes of implementation of the protocol in a VoIP context. The first one is based on a trusted third party whereas the last two operate between two domains sharing a long term secret. The last two modes show a three party architecture from which the simplified case (with only two entities) can be easily deduced by assuming S = B (in mode 2) or S = A (in mode 3). Beforehand we propose an efficient way for deriving the keying material required for each transaction.

4.1 Cryptographic material

The $MAC_K(V)$ values are computed following the HMAC standard [19] with the SHA-256 hash function. The $\{M\}_K$ encrypted values are computed following the AES-128 standard [18]. The key lengths chosen in this section, as well as the transaction index lengths, are for illustration only and should be adapted depending on the application specific security requirements:

K_{SB}, K_{AS}: 128-bit symmetric keys.

N, M: 120-bit values.

Let DC_X be some 8-bit public constants used for key derivation purpose. From the previous secrets and DC_X values, the following keying material is obtained:

$K_{AB,N} = \{DC_1||N\}_{K_{SB}}$: 128-bit key.

$K_{SB,N,MAC} = \{DC_2||N\}_{K_{SB}}$: 128-bit key.

$TRID_N = trunc(\{DC_3||N\}_{K_{SB}}, 64)$: 64-bit (public) transaction identifier.

$K_{AS,M} = trunc((\{DC_4||M\}_{K_{AS}}||\{DC_5||M\}_{K_{AS}}), 192)$: 192-bit key[15].

$K_{AS,M,MAC1} = \{DC_6||M\}_{K_{AS}}$: 128-bit key.

$T = \{DC_7||M\}_{K_{AS}}$: 128-bit value which is logically split in two parts of 64 bits each: $T = TRIDA_M||\ TRIDS_M$.

$K_{AS,M,MAC2} = K_{AS,M,MAC1} \oplus LC$: 128-bit key (LC is a 128 bit non null constant).

The next transaction indexes are computed as: $N_{(+1)} = trunc(\text{SHA-256}(N), 120)$, $M_{(+1)} = trunc(\text{SHA-256}(M), 120)$.

4.2 Implementation mode 1

As shown in Figure 2, entity A is one of the outbound VoIP proxies in domain A, entity B is one of the inbound VoIP proxies in domain B and S is the trusted third party which is responsible for domain authentication and for phone number routing and verification. Most of the time, the endpoints are not A and B themselves but rather A' or B' which might be a VoIP terminal or another proxy. In this mode, entity A sends the authorization request to S in the form of a `SIP-OPTIONS` message with a text body part containing the protocol information $TRIDA_M$, ID, $MAC_{K_{AS,M,MAC1}}(TRIDA_M, ID)$. Then S responds to A with a SIP 200OK message containing the required information. Finally, A sends the `SIP-INVITE` request to B including the $TRID_N$, $TRCheck_N$ values and the MAC code. Since the length of these fields is short, they can be easily conveyed in the `SIP-INVITE` header, for example as an extension of the `User-Agent` field. Further SIP dialogue takes place between entities A and B which now share the $K_{AB,N}$ key and can confirm it in the forthcoming messages. It should be noted that *this mode also applies to intra-domain context* where A, B and S are from the same domain.

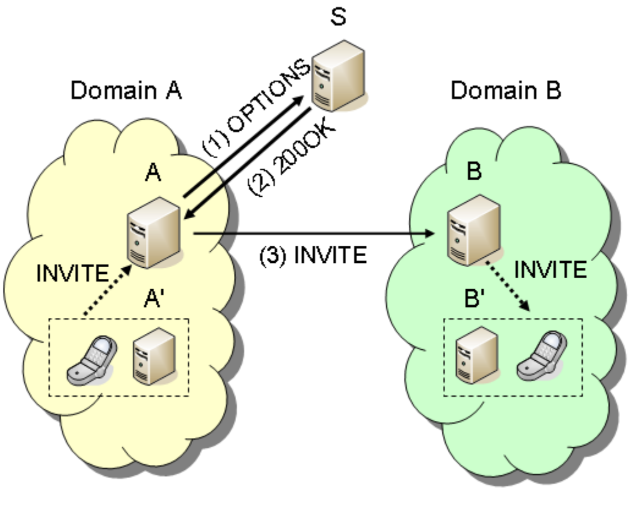


Figure 2: First mode of implementation.

4.3 Implementation mode 2

As shown in Figure 3, entity A is one of the outbound VoIP proxies in domain A, entity S is one of the inbound proxies in domain B. Entity B may be a user endpoint in domain B, another proxy in domain B or even a proxy in a visited domain C where the responder endpoint is currently attached. In this mode, entity A queries authorization to S with a `SIP-OPTIONS` request and in the `SIP-200OK` response S indicates the contact address where the final `SIP-INVITE` request shall be sent. It may be either the B entity (for optimized routing) or S itself if it needs to remain in the signalling path.

In this mode, domains A and B need to share the K_{AS} and M secret values which raises scalability issues when domain B may be accessed from any other Internet domain. Therefore, it is proposed to mix this mode with the previous one. In extended, domain B has a shared secret with trusted domains with whom significant traffic is exchanged. Untrusted domains, or domains with whom sporadic traffic is exchanged must go through a trusted third party as described in the previous mode.

4.4 Implementation mode 3

As shown in Figure 4, entity S is one of the outbound VoIP proxies in domain A, entity B is one of the inbound proxies in domain B. Entity A may be a user endpoint in domain A, another proxy in domain A or even a proxy in a visited domain C where the initiator endpoint is currently attached. In this mode, entity A first queries authorization to S with a

[15] This fulfills: $len(K_{AS,M}) = len(TRID_N) + len(K_{AB,N})$.

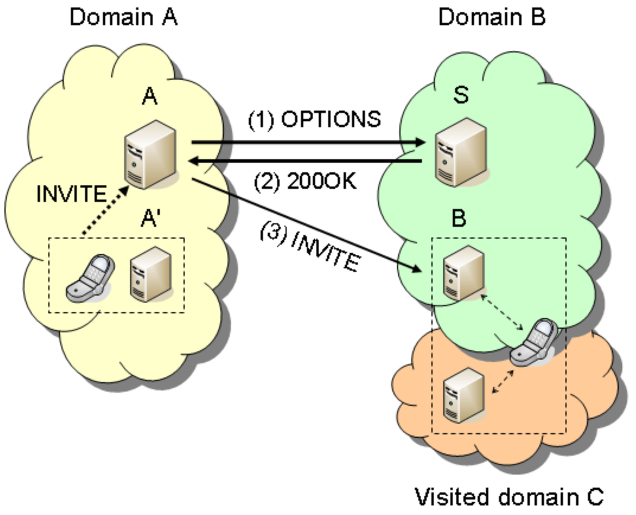


Figure 3: Second mode of implementation.

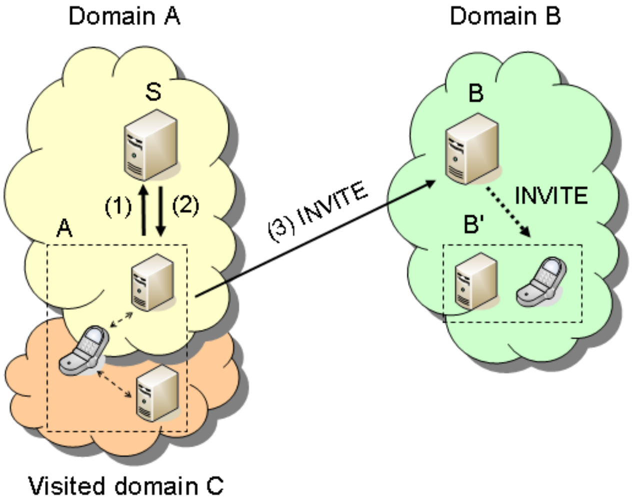


Figure 4: Third mode of implementation.

SIP-OPTIONS/SIP-200OK exchange and S indicates the contact address for B. The final SIP-INVITE request may be sent either by A or by S itself (if it needs to remain in the signalling path). Compared to mode 2, domain B only receives the final SIP-INVITE request. This reduces the number of messages that have to be processed by the called domain, but on the other hand this precludes optimized routing. As in mode 2, a shared-secret is required between domains A and B, which leads to the same comments.

5. DISCUSSION AND CONCLUSIONS

The proposed mechanism is a key exchange protocol which is designed for *open* inter-domain context where interconnection proxies can be reached from anywhere on the public Internet and thus may be the target of (D)DoS attacks. For this reason, we have chosen to use symmetric cryptography and favoured receiver DoS protection by adding a transaction identifier in each message. This identifier is used as a "computation-free" filter value on the responder side and also as a pointer to fetch the (pre-computed) cryptographic material associated to the transaction.

From an heuristic analysis, we also expect the protocol to meet the security requirements of authentication, key freshness, privacy, PFS and anti-replay. Because IDDR-CEP is based on a key-agreement scheme it is resistant to off-line passive attacks. Furthermore, since the protocol information conveyed in each message is short (except the *OP* value), the VoIP signalling transport over UDP can be preserved.

In section 4, we described various implementations of the protocol in a VoIP context, although it can be adapted to other applications. In mode 1, a trusted third party is in charge of authentication, routing and key establishment between domains. With this setting, entity B can receive an authenticated and optionally encrypted message from any other Internet domain without the need for previous round trip with entities A or S. In modes 2 and 3, a shared secret is established directly between a pair of domains thus removing the need for a trusted third party. Various options are still applicable within each mode, especially the choice of the entity which sends the final SIP-INVITE request. The classical VoIP trapezoidal call model can be preserved, whereas more optimized routing schemes can also be supported. Similarly, several options are available for carrying the authorization information between A and S. However, a cleaner implementation would require defining specific SIP information fields for this protocol.

Using a trusted third party in implementation mode 1 raises some operational issues although we believe this is necessary for at least phone number routing and verification, as well as regulatory constraints [11]. Actually, the S entity appears as a single point of failure because it is in charge of authentication and of ensuring key agreement between parties. This limitation may be reduced with load balancing and possibly by using some kind of P2P architecture for implementing the S entity functions. As explained in section 4.3, mode 1 can be mixed with the two other implementation modes: the responder domain has a shared secret with a couple of established domains (modes 2 and 3), whereas untrusted domains must go through a trusted third party (mode 1).

Finally, two issues related to the management and synchronization of the transaction window by entity B are anticipated. The first one occurs in mode 1 if entity S authorizes several transactions towards entity B, but Δ consecutive AM messages are lost, or retained by compromised entities A_i, and thus not received by B. Then the "$\Delta + 1$" AM message will be considered as invalid when received by B, leading to a blocking state. The second limitation occurs in modes 2 or 3, where the responder domain B would have to maintain a transaction window with each other domain, which raises a scalability issue.

In addition to investigating these two issues, future work on IDDR-CEP includes prototyping, performance evaluation and more formal proof of security properties.

6. ACKNOWLEDGEMENTS

The author would like to thank the conference reviewers for their helpful comments along with the following persons: Henri Gilbert, Cyril Delétré, Joaquin Garcia-Alfaro, Nora

Cuppens, Frédéric Cuppens and Sarah Cook for their help in preparing this paper.

7. REFERENCES

[1] 3GPP. IMS Functional Architecture. 3GPP TR33.828, May 2009.

[2] H. Abdelnur, R. State, I. Chrisment, and C. Popi. Assessing the security of voip services. In *IM'07: The 10th IFIP/IEEE Symposium on Integrated Management*, 2007.

[3] W. Aiello, S. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 48–58. ACM New York, NY, USA, 2002.

[4] F. Andreasen, M. Baugher, and D. Wing. Session Description Protocol (SDP) Security Descriptions for Media Streams. RFC 4568 (Proposed Standard), July 2006.

[5] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. RFC 3830 (Proposed Standard), Aug. 2004. Updated by RFC 4738.

[6] E. A. Blake. Network security: Voip security on data network–a guide. In *InfoSecCD '07: Proceedings of the 4th annual conference on Information security curriculum development*, pages 1–7, New York, NY, USA, 2007. ACM.

[7] C. Boyd and A. Mathuria. *Protocols for authentication and key establishment*. Springer Verlag, 2003.

[8] J. Elwell. Connected Identity in the Session Initiation Protocol (SIP). RFC 4916 (Proposed Standard), June 2007.

[9] J. Floroiu and D. Sisalem. A comparative analysis of the security aspects of the multimedia key exchange protocols. In *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, pages 1–10. ACM, 2009.

[10] S. E. Griffin and C. C. Rackley. Vishing. In *InfoSecCD '08: Proceedings of the 5th annual conference on Information security curriculum development*, pages 33–35, New York, NY, USA, 2008. ACM.

[11] J. Hill. The storm ahead: how calea will turn voip on its head. In *InfoSecCD '06: Proceedings of the 3rd annual conference on Information security curriculum development*, pages 147–150, New York, NY, USA, 2006. ACM.

[12] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD) Base Protocol. IETF draft-ietf-p2psip-base-07, February 2010.

[13] A. D. Keromytis. A survey of voice over ip security research. In *ICISS '09: Proceedings of the 5th International Conference on Information Systems Security*, pages 1–17, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] J. Mattsson and T. Tian. MIKEY-TICKET: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY). IETF draft-mattsson-mikey-ticket-00, Oct. 2009.

[15] D. McGrew and E. Rescorla. Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-time Transport Protocol (SRTP). IETF draft-ietf-avt-dtls-srtp-07, Feb. 2009.

[16] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021.

[17] S. Niccolini, E. Chen, J. Seedorf, and H. Scholz. SPEERMINT Security Threats and Suggested Countermeasures. IETF draft-ietf-speermint-voipthreats-01, July 2009.

[18] NIST. Advanced Encryption Standard (AES). FIPS PUB 197, Nov. 2001.

[19] NIST. The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198, Mar. 2002.

[20] K. Ono and H. Schulzrinne. Have I met you before?: using cross-media relations to reduce SPIT. In *Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, pages 1–7. ACM, 2009.

[21] S. Peng and Z. Han. Proxy cryptography for secure inter-domain information exchanges. In *Dependable Computing, 2005. Proceedings. 11th Pacific Rim International Symposium on*, Dec. 2005.

[22] J. Peterson and C. Jennings. Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP). RFC 4474 (Proposed Standard), Aug. 2006.

[23] J. Rosenberg and C. Jennings. Verification Involving PSTN Reachability: Requirements and Architecture Overview. IETF draft-rosenberg-dispatch-vipr-overview-01, November 2009.

[24] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[25] Z. Wan, B. Zhu, R. Deng, F. Bao, and A. Ananda. DoS-resistant access control protocol with identity confidentiality for wireless networks. In *2005 IEEE Wireless Communications and Networking Conference*, volume 3, 2005.

[26] D. Wing. SIP E.164 Return Routability Check (RRC). IETF draft-wing-sip-e164-rrc-01, Feb. 2008.

[27] F. Wong and H. Lim. Identity-Based and Inter-Domain Password Authenticated Key Exchange for Lightweight Clients. In *Proceedings of 3rd IEEE International Symposium on Security in Networks and Distributed Systems*. Citeseer, 2007.

[28] C. Wu, C. Huang, and J. Irwin. Using Identity-Based Privacy-Protected Access Control Filter (IPACF) to against denial of service attacks and protect user privacy. In *Proceedings of the 2007 spring simulation multiconference-Volume 3*, pages 362–369. Society for Computer Simulation International, 2007.

[29] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media Path Key Agreement for Secure RTP. IETF draft-zimmermann-avt-zrtp-17, Jan. 2010.

Work in progress: A secure and lightweight scheme for media keying in the Session Initiation Protocol (SIP)

Vijay K. Gurbani
Acatel-Lucent, Bell Labs
1960 Lucent Lane,
Naperville, IL 60566 (USA)
vkg@research.bell-labs.com

Vladimir Kolesnikov
Alcatel-Lucent, Bell Labs
600-700 Mountain Ave.,
Murray Hill, NJ 07974 (USA)
kolesnikov@research.bell-labs.com

ABSTRACT

Exchanging keys to encrypt media streams in the Session Initiation Protocol (SIP) has proved challenging. The challenge has been to devise a key transmission protocol that preserves the features of SIP while minimizing key exposure to unintended parties and eliminating voice clipping. We first briefly survey the two IETF SIP media keying protocols – SDES and DTLS-SRTP – and evaluate them against a core feature set. We then introduce a novel simple and lightweight scheme to significantly increase the security of SDES SIP keying with minimal overhead costs. Our proposed key exchange involves only one symmetric key operation by sender and receiver and is secure against the Man-in-the-middle attack unless the attacker is able to intercept both the SIP signaling and media plane traffic. Our key exchange scheme is much simpler than DTLS-SRTP; in fact, compared to SDES, it includes only one additional simple step. At the same time, it provides significantly better security than SDES and is only slightly weaker than the non-PKI version of DTLS-SRTP.

Keywords

SIP, key exchange, media, security, SDES, DTLS

1. INTRODUCTION

The Session Initiation Protocol (SIP [17]) is an Internet protocol to set up, maintain, and terminate multimedia sessions. While SIP is used to rendezvous the session participants, the session itself is conducted using separate protocols. The Session Description Protocol (SDP, [11]), which is transported in SIP is used to describe endpoint capabilities, exchange the voice or video codecs and network identifiers — IP addressses and port numbers — where the media will flow.) The media itself, i.e., the actual contents that comprise the voice or video session, use the Real-Time transport Protocol (RTP, [18].)

Because the protocols for initial rendezvous, capability description, and eventual media stream are different, it becomes a challenge to provide security for the system as a whole. As an example of this challenge, consider that signaling in SIP can be protected by hop-by-hop use of Transport Layer Security (TLS [6]), yet the media often flows end-to-end using plaintext RTP. Furthermore, the protection afforded to the signaling messages is such that confidentiality, message authentication and replay protection are ensured on a per hop channel, but the intermediary that forwards the signaling onwards have unhindered access to the plaintext that comprise the signaling messages.

Today, while secure keying techniques (e.g., DTLS-SRTP) are available and standardized, SIP implementations predominantly use (weakly secure) SDES key transmission for securing media-plane communication (see Table 1 for samples collected at SIP interoperability events.) This state of affairs is due to the implementation complexity and increased computation and communication costs associated with the public-key based proposals, such as DTLS-SRTP and ZRTP[24].

Our Contributions and Outline of the Work

We close this security/efficiency gap, by proposing a new media keying protocol that involves only one symmetric key operation by sender and receiver and is secure against man-in-the-middle (MiTM) attack unless the attacker is able to intercept both the SIP signaling and media plane traffic. To match its efficacy against the standardized SIP media keying protocols, we first analyze the two media keying protocols – Security Descriptions (SDES [1]) and DTLS-SRTP [14] for their suitability in a SIP network. We chose to focus on these two protocols primarily because they are standardized by the Internet Engineering Task Force (IETF) and as such will witness large-scale deployment in SIP networks.

To analyze a protocol's ability to successfully key media SIP streams, we list a feature set against which the particular media keying protocols, including our novel contribution, will be evaluated. We will see that our key exchange scheme is much simpler than DTLS-SRTP; in fact, compared to SDES it includes only one additional simple step. At the same time, it provides significantly better security than SDES and is only slightly weaker than the non-PKI version of DTLS-SRTP.

The paper is structured as follows: Section 2 presents the required background on SIP and SRTP. Section 3 identifies the core feature set that the keying protocols should support. Sections 4 and 5 review SDES and DTLS-SRTP protocols, respectively, and evaluate them on the core feature set. We present our novel keying method, analyze its security, and subject it to the same core feature set evaluation in Section

Table 1: Support for SRTP in SIP

SIPit number (date)	Total unique implementations	Number supporting SRTP	Number using SDES	Number using DTLS-SRTP
18 (April 2006)	73	10	7	0
19 (October 2006)	90	12	predominant [a]	0
20 (April 2007)	90	9	4	1
21 (November 2007)	70	17	0	0
22 (April 2008)	80	32	predominant [b]	0
23 (October 2008)	50	8	0	1
24 (May 2009)	43	16	0	1
25 (September 2009)	42	14	4	0
26 (May 2010)	42	23	23	0

[a] Exact number unknown, SIPit 19 archives state ”Keying was predominantly sdes.”

[b] Exact number unknown, SIPit 22 archive states ”Most of the tests established the session using sdes.”

Data for this table gathered from SIPit official website at https://www.sipit.net/SIPitSummaries. A 0 in column 4 or 5 signifies no support for that particular keying protocol. It is not the case that the number of implementations supporting SDES and DTLS-SRTP add up to the number supporting SRTP; in some cases, implementations were using unspecified means to key the SRTP stream.

6. Section 7 provides related work; we conclude in Section 8.

2. THE SESSION INITIATION PROTOCOL

A SIP ecosystem consists of user agents, proxy servers, redirect servers, and registrars. Of special interest to us with respect to this paper are user agents and proxy servers.

2.1 Establishing a SIP Session

There are two types of SIP user agents: a user agent client (UAC) and a user agent server (UAS). A UAC and a UAS are software programs that execute on a computer, an Internet phone, or a personal digital assistant (PDA). A UAC originates requests (i.e. start a multimedia session) and a UAS accepts and acts upon a request. Proxy servers are used to route requests and responses between a UAC and a UAS.

SIP invests a great amount of trust in the proxies, as we will see later in this paper. In the canonical SIP trapezoid [17], Alice wishes to establish a session with Bob. Her SIP request to establish a session traverses through her proxy to Bob's proxy. Bob's proxy performs a lookup service to determine where Bob can be located, and forwards Alice's request to Bob. If Bob responds in the affirmative, the response backtracks the path taken by the request to reach Alice. Note that the media session is established directly between Alice and Bob, and does not go through the intermediary proxies.

2.2 RTP and SRTP

In SIP, the media is transported end-to-end using RTP, which exchanges packets in cleartext. A profile called Secure RTP (or SRTP [4]) was subsequently developed to provide confidentiality, message authentication, and replay protection to the cleartext RTP traffic. Conceptually, SRTP can be viewed as a "bump in the stack" implementation that resides between the RTP layer and the transport layer. SRTP intercepts RTP packets and then forwards an equivalent SRTP packet on the sending side, and intercepts SRTP packets and passes an equivalent RTP packet up the stack on the receiving side [4].

To achieve the goals of confidentiality, message authentication, and replay protection, SRTP defines extensions to the RTP packet format to encrypt the RTP payload. Each SRTP stream requires the sender and receiver to maintain cryptographic state information (the "cryptographic context"). The cryptographic context provides all the necessary parameters such as the chosen cipher, its mode of operation, and the block size; the master key; session keys; etc. SRTP uses two types of keys: session key and a master key. The session key is used directly in a cryptographic transform (i.e., payload encryption or message authentication) and the master key is a random bit string provided by the keying protocol from which session keys are derived in a cryptographically secure manner. The master key, salt, and other parameters in the cryptographic context are provided by keying mechanisms — such as SDES or DTLS-SRTP — external to SRTP. SRTP is increasingly being used in SIP; however, its wide-spread adoption has been slow (see Table 1.)

The cryptographic context itself is selected by a 32-bit numeric field carried in the fixed RTP header called Synchronization source (SSRC), which is used to identify the source of a RTP stream. Some keying protocols provide this to SRTP, while in others the SSRC is obtained dynamically when SRTP packet arrives at a receiver (the SSRC field is part of the fixed RTP header that is used without any change in SRTP; the only difference being that in SRTP the integrity of the RTP header is protected by a message authentication code.) Since SSRC is a random 32-bit number, the chance of independent RTP streams generating the same SSRC, while small, does exist. However, the two keying protocols handle such collisions appropriately.

While DTLS-SRTP is able to agree on the master key, salt and other parameters independently at the peers, some amount of information to tie the media stream to the signaling channel to prevent a third party from inserting false media packet can be provided by the signaling layer. To accomplish this, DTLS-SRTP can transport the fingerprints of the public certificates exchanged between the peers as an *a=fingerprint* attribute in SDP. As we will observe in Section 4, SDES transports the entire cryptographic parameters, including the master key and salt in an *a=crypto* SDP attribute.

3. IDENTIFYING A FEATURE SET

We now establish a core feature set that we consider immutable. That is, when we analyze the key exchange protocols, we will analyze them with a view towards how they support (or do not) this core feature set in a transparent manner (i.e., the feature behavior should not be modified to conform to the machination of the specific media keying protocol.) This core feature set includes features that are intrinsic to how SIP works as a protocol as well as features that use SIP as a service enabler. Some of the features in our set overlap with those outlined in Wing et al. [23], however, we go further by including in our set those features that are deemed out of scope (e.g., shared-key conferencing) or not discussed at all (e.g., legal interception) in Wing et al. [23].

We consider eight features important enough to be supported by a key exchange protocol. Of these eight, six are described in Wing et al. [23]. These are: forking, the Heterogeneous Error Response Forking (HERFP) problem, minimizing media clipping, re-targeting, placing calls from the Internet to the public-switched telephone network (PSTN), and shared-key conferencing. Shared-key conferencing, while described in Wing et al. [23] is deemed out of scope in their analysis; we include it in our analysis. There are an additional two features that are not mentioned in Wing et al. [23]; these are legal intercept and session recording. We define them below.

3.1 Security Model

Before proceeding with the feature set, it is important to understand the guarantees and limitations of the security services provided by the keying techniques. First, we stress that we analyze security against very strong adversary, so-called *Man-in-the-Middle (MiTM)* who fully controls the communication channel between the parties. Such adversaries are standard in cryptographic design and analysis of key exchange protocols [5, 20, 12, 13]. In particular, most adversarial capabilities considered by IETF and other standards communities are special cases of MiTM.

Second, flooding attacks are far easier to mount against the SIP protocol itself than they are against some of the keying techniques [9]. DTLS-SRTP in particular only performs a pair-wise key exchange with the peer that is interested in establishing a session (i.e., responds with a 200 OK response message.) Thus, the only way an attacker can mount a flooding attack at the keying layer is by causing the initial request to fork to many endpoints, each of which returns a 200 OK response to the sender. This will cause the sender to enter a pair-wise key exchange session with multiple endpoints simultaneously. Note that an attacker that simply causes a swarm of manufactured 200 OK responses to be sent to an arbitrary victim does limited harm to the victim because such a response will not match any pending SIP transaction in the victim's transaction state table, causing the victim to simply throw away the response at the cost of a search across the transaction table. Thus we limit our discussion on flooding attacks as well, unless a certain feature requires specific discussion for such an attack.

3.2 Feature: Legal Interception

In order to comply with the legal procedures and regulatory environments pertinent to business practices and country codes, traditional switched networks evolved to support legal interception of the media traffic by law enforcement or by business or enterprise for other reasons (e.g., recording calls at a call center for training or at a financial brokerage firm for non-repudiation.) In an end-to-end key exchange model, this operational requirement becomes harder to enforce because the service provider will not have access to the master key.

3.3 Feature: Session Recording

Session recording is a critical operational requirement in many businesses, especially where voice is used as a medium for commerce and customer support [22]. SIP does not — at the protocol level — provide any explicit support for session recording. In fact, if Alice is talking to Bob, either can decide to record the session on their local endpoints, assuming that the local endpoint is capable of recording and storing media (in the most general case, recording is simply duplicating arriving and departing media packets and storing them in a persistent store while maintaining the temporal ordering between the packets.)

The problem arises when the local endpoint cannot do recording and a specialized entity — a recording server — has to be invited to a session in order to perform recording. Under this model, Alice and Bob have to send their media streams to the recording server. When Alice and Bob use SRTP, the recording server will not have the required key to decrypt the media for a subsequent playback. There are several ways to mitigate this problem.

One way is for Alice or Bob to send the mixed but unencrypted RTP media stream to the recording server. However, this compromises the privacy of the communications between Alice and Bob if the plaintext media is being sent to the recording server over an insecure channel. A second approach is to share the master key with the recording server — the mixed SRTP media stream is directed towards the recording server and when the session ends, the SRTP master key is shared with the recording server. Wing et al. [22] discuss mechanisms by which the key sharing can be performed. One subtlety to be addressed here is that the recording server may be able to interfere with the communication, since it is given the key used to secure it.

A more secure approach at addressing the problem is for Alice or Bob to execute a key exchange with the recording server. Then, the mixed media is sent to the recording server encrypted using the exchanged key. The disadvantage here is the added complexity of this approach and increased processing on the client responsible for the re-encryption of media to the recording server.

4. SECURITY DESCRIPTIONS

4.1 Security Descriptions Overview

Conceptually, SDES is the simpler of the two key management protocols. Simply put, it arranges for the SRTP master key, salt, and other parameters to be transported in the SIP signaling messages (thus pedantically, it is not a *key exchange* protocol as much as a *key transport* protocol.)

SDES defines a new SDP attribute called “crypto” that is used to signal and negotiate cryptographic parameters for SRTP media streams. This attribute transports the encryption and authentication algorithms, master key and salt of the sender (i.e., the receiver should use the said master key and salt to derive session keys for decryption), and the lifetime of the master key (i.e., maximum number of SRTP and

SRTCP packets that use this master key.)

In its simplest form, the UAC inserts this parameter in the SDP of the INVITE request and sends it to the UAS; the UAS inserts this parameter in the 200 OK response and transmits it to the UAC. Consequently, SDES provides distinct keys for each media stream in each direction. The example below shows the "crypto" attribute in an INVITE from Bob to Alice (only pertinent SIP headers shown):

```
INVITE sip:bob@example.com SIP/2.0
To: Robert <sip:bob@example.com>
From: Alice <sip:alice@example.org>;tag=0ij8z
Content-type: application/sdp
[...]

v=0
o=alice 2890844526 2890844526 IN IP4 a.example.org
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49170 RTP/SAVP 0
a=crypto:1 AES_CM_128_HMAC_SHA1_80
 inline:NzB4d1BINUAvLEw6UzF3WSJ+PSdFcGdUJShpX1Zj|2^20
```

The "crypto" attribute above identifies the encryption and authentication algorithm (AES_CM_128_HMAC_SHA1_80) and specifies the master key, salt, and the lifetime of the master key (2^{20}). The master key and salt are concatenated and base 64 encoded (NzB4d1BINUAvLEw6UzF3WSJ+PSdFcGdUJShpX1Zj). The sender of the "crypto" attribute uses the master key to derive the session key for encryption and the receiver uses it to derive the session key for decryption.

Evidently, if the SIP request or response containing the "crypto" attribute is transmitted in the clear, a malicious eavesdropper can gain access to the master key. Thus, the cryptographic keys and other parameters should be secured on a hop-by-hop link using TLS. While this prevents unauthorized eavesdroppers from gathering the cryptographic keys, it does not afford complete privacy or confidentiality to the media session because the intermediaries at the end of the hop-by-hop TLS link will have access to the cleartext cryptographic keys.

MiTM attack remains a problem for SDES — if an adversary is able to inject itself as a next hop in the intermediary chain, it will have complete access to the cryptographic parameters. From this point of view, SDES may be considered the least secure of the keying protocols we consider. Note that the use of TLS-secured channels across the intermediary chain does not guarantee secure and private delivery of session keying material. This is because, as of this writing, guidelines on SIP certificate issuance are in the process of being standardized [10] and until a certificate can be issued specifically for a SIP service, any other certificate (e.g., one issued for the use of web services) may suffice. Thus, an adversary may be able to obtain a legitimate certificate from a certificate authority and then insert itself in the intermediary chain by techniques such as DNS cache poisoning. We discuss additional subtle vulnerabilities of SDES in Section 6.

4.2 Suitability for Feature Set

We now discuss how SDES supports the feature set we outlined in Section 3.

Forking:

In SDES key leakage occurs as a result of forking; the master key from the initiator of the request will be replicated to all of the forked branches. One way to deal with this is to re-key the media stream after the initial session has been successfully established with one forked branch, thereby making obsolete the old key available at the remaining branches.

HERFP remains a problem for SDES because a higher-class response that intends to negotiate the "crypto" parameters gets masked by a lower class response.

Media Clipping:

Media clipping also remains a problem with SDES. Each party selects their own keys for the encryption of the traffic they generate and send these keys to the other party. Consider the case where Bob establishes a session with Alice, and in that session description, he provides his cryptographic keys. Alice accepts the session and provides her cryptographic keys for decryption in the 200 OK and starts speaking, thus causing SRTP packets to go directly from her user agent to Bob's user agent. Due to the hop-by-hop nature of her 200 OK signaling response, the SRTP packets, which take a direct route, may get to Bob's user agent first. However, Bob does not have Alice's cryptographic key to decrypt the packet, causing playout delay or clipping to occur.

Re-targeting:

Re-targeting in SDES suffers from the same key leakage problem of forking. When an intermediary proxy re-targets a request, it cannot, obviously, change the cryptographic keys. Furthermore, the initiator of the request will not know that re-targeting has occurred until he or she establishes a session and exchanges some media packets with the recipient (that is, only when Bob talks to Alice's delegate, Carol, does he know that he is not talking to Alice.)

Conferencing:

SDES is not suitable for general conferencing since the definition of the "crypto" attribute is limited to a two-party unicast media stream where each source has a unique cryptographic key.

Calls to Other Networks:

There is nothing intrinsically prohibitive about supporting calls to other networks in SDES. However, SDES can only secure communications within the portion of the network that supports it. That is, if SDES is negotiated by a UAC and a PSTN gateway, the media is protected using SRTP between the UAC and the PSTN gateway. When sessions continue to the PSTN from the gateway, SDES will be unable to secure the portion of the session that continues to the PSTN (or any other network.)

Legal Interception:

Insofar as legal interception can be supported by provisioning known cryptographic keys in endpoints, SDES will support it. Unlike DTLS-SRTP that negotiate the keys in the media layer, an endpoint that uses SDES can be provisioned with a key known to the operator of the service.

Session Recording:

Because SDES transports the cryptographic keys in signaling, it is conceivable to route the signaling messages through a recording server such that it has access to the SRTP master key of each endpoint in a session.

However, there is a subtelty that comes into play here.

Because the keys are delivered to the recording server in the initial request to establish a session, the recording server can act as a MiTM and inject or modify any encrypted media packets (note that while a SIP proxy also has access to the keys, the difference is that proxies are trusted in SIP whereas a recording server may not be.) A better solution would be to provide the keys to the recording server at the end of the session (through a SIP BYE request), but the SDES specification [1] does not contain any such provisions.

5. DTLS-SRTP

We start with reviewing DTLS-SRTP and then in Section 5.3 discuss its suitability for supporting the basic features described in Section 3.

As we build the presentation from the top down, note that DTLS-SRTP is a DTLS-based extension of SRTP, designed to combine the performance and security flexibility benefits of SRTP with the key and association management of DTLS. DTLS-SRTP can be equivalently viewed as a key management method for SRTP, or as a new RTP-specific data format for DTLS. We now briefly discuss DTLS, to give the necessary background for the discussion of the main aspects of DTLS-SRTP.

5.1 DTLS Overview

DTLS — Datagram TLS [15] — is an adaptation of the established and well-understood TLS to the datagram transport. The design goal of the authors of DTLS was only minimal deviation from TLS, for the simplicity of analysis (in relation to the complex TLS), and minimization of the risks of introducing errors or vulnerabilities. For the purposes of this survey paper, the differences introduced by DTLS can be largely ignored, and a reader familiar with TLS may assume that DTLS is a faithful implementation of TLS executed over datagram transport. For completeness, we give a brief overview of DTLS, and make several comments on its inherited and introduced vulnerabilities.

DTLS message exchange

In this section, we omit some cryptographic details, such as agreement on suites, etc. For concreteness, we show the case with mutual authentication using RSA. (If the Client is not authenticated, "request cert", ClientCertificate and ClientCertificateVerify messages are not sent. Then the Client avoids the expense of the computation, and the Server does not perform corresponding verifications.) Further (not included in the diagram), an optional cookie is exchanged prior to the core execution to mitigate DoS attacks. That is, server only proceeds to the crypto-intensive part of the handshake, if the client is able to replay the cookie sent to the claimed IP address.

DTLS core description

The description below depicts the cryptographic core of the DTLS exchange; it is not a complete description of the DTLS protocol itself.

Client		*Server*	
	C.random →		(1)
	← S.random		(2)
	← $pk_S, cert_S$, "req. cert"		(3)
Verify $cert_S$			
	$pk_C, cert_C$ →		(4)
Choose rand. r	$Enc_{pk_S}(r)$ →		(5)
	$sig_{pk_C}(H($ $prev.msgs))$ →		(6)
		Verify $cert_C$ Verify sig_{pk_C} Decrypt $Enc_{pk_S}(r)$ to obtain r,	

where messages (1)-(6) are:

1. ClientHello
2. ServerHello
3. ServerCertificate
4. ClientCertificate
5. ClientKeyExchange
6. ClientCertificateVerify

The session key is set to $PRF(r,$ "master secret", $ClientHello.random{+}ServerHello.random)$. Note, prev.msgs includes all previously exchanged messages, and, in particular, $Enc_{pk_S}(r)$, C.random and S.random. $Sign_{pk_C}$ is the public key signature.

Here $pk_C, pk_S, cert_C, cert_S$ are public keys and certificates of the client and server respectively. Certificates include the public keys, but we wrote them out separately to be explicit. Note that these parameters are transmitted in the clear and are publicly known.

DTLS security

First, we would like to point out that DTLS-SRTP and its use in SIP are *n*ot vulnerable to variants of Man-in-the-Middle (MitM) attack on TLS derivatives, described in [3], even though, by design, no improvements were introduced in the DTLS derivation. The reason is that the attack is applicable only in a few settings, namely, where TLS is used to establish a tunnel over which a second-factor (e.g. password) authentication is performed.

Given that DTLS-SRTP is run in mutual authentication mode, it provides good protection against active attacks. In addition to TLS DoS attacks, DTLS suffers from the standard resource consumption attack, and an amplification attack. In our opinion, both of these are of mild severity, and are further mitigated by the cookie exchange described above.

DTLS-SRTP depends on a PKI to prevent MiTM attacks. Additionally, to remove/reduce reliance on PKI, DTLS-SRTP endpoints exchange the fingerprint of the certificates in SIP signaling channel; when key exchange is performed in the media channel, each side compares the other sides fingerprint to the received key. A MiTM attack would effectively need to control both the media and signaling to mount a successful attack.

5.2 DTLS-SRTP Overview

While DTLS provides the key to the communicating parties, DTLS-SRTP specifies its usage in the following data exchanges.

DTLS-SRTP is defined for point-to-point media sessions, in which there are exactly two participants. Each DTLS-SRTP session contains a single DTLS association, and either two SRTP contexts (if media traffic is flowing in both directions on the same host/port quartet) or one SRTP context (if media traffic is only flowing in one direction). All SRTP traffic flowing over that pair in a given direction uses a single SRTP context. A single DTLS-SRTP session only protects data carried over a single UDP source and destination port pair *in a single direction*.

The general pattern of DTLS-SRTP is as follows. For each RTP or RTCP flow, the peers do a DTLS handshake on the same source and destination port pair to establish a DTLS association. Which side is the DTLS client and which side is the DTLS server is established via an out of band mechanism (SIP). The keying material from that handshake is fed into the SRTP stack. Once that association is established, RTP packets are protected (becoming SRTP) using that keying material.

Between a single pair of participants, there may be multiple media sessions. There must be a separate DTLS-SRTP session for each distinct pair of source and destination ports used by a media session. However, for efficiency, it is recommended that such sessions share a single DTLS session and hence amortize the initial public key handshake. This is done by deriving separate DTLS-SRTP master keys for each DTLS-SRTP session from the same DTLS output.

Credentials and Authentication

The security of entire data exchange run by DTLS-SRTP is dependent on the integrity of the public key certificates possessed by the communicating parties. Ideally, they will be maintained by a PKI; however this solution has potential high costs associated with it.

An alternative natural approach is to delegate some of the responsibility to the SIP layer. For example, as described in [14], parties may exchange hashes of their public keys in the SIP layer. Then, if the SIP layer is secured, this provides sufficient guarantees; if it is not, this serves merely as an additional hurdle for the attacker, and the combined protocol is still vulnerable to attacks.

More specifically, when Alice wishes to set up a secure media session with Bob, she sends an offer in a SIP message to Bob. This offer includes, as part of the SDP payload, the fingerprint (i.e. secure collision-resistant hash) of Alice's certificate. Alice should utilize existing SIP security mechanism, and send this message to her proxy over an integrity-protected channel. If all the channels on the way to Bob are integrity-protected, a polynomial time adversary will not be able to compromise the security of DTLS-SRTP.

5.3 Suitability for Feature Set

In this section, we go over features discussed in Section 3 and analyze their support by DTLS-SRTP.

Forking:

Key exchange and session establishment occurs in DTLS-SRTP in the media plane. Therefore, each responder would establish independent key with the initiator, and key leakage will not occur. Further, as also noted in [7] (Appendix A.24), since key exchange is executed in the media path, error messages are also communicated along this path, and proxies will not need to take action based on error messages. Thus, Heterogeneous Error Response Forking Problem (HERFP) is not applicable here either. In summary, threats associated with forking, as described in Section 3 are not applicable in DTLS-SRTP.

Media Clipping:

Again, since keying occurs in the media plane, user agent applications are in full control over how to send the data (encrypted or not), depending on whether DTLS has completed and keys were derived. Therefore, the problem of early media clipping, as described in Section 3 is easily avoidable by client applications.

DTLS-SRTP signals its intent such that both peers must support the extension before SRTP media flows between them. In this respect, it does not result in any leak of privacy by first sending plaintext RTP.

Re-targeting:

First, we observe that the keys will not be leaked to unintended recipients since key exchange is executed end-to-end in the media plane. Further, authenticated DTLS-SRTP will always detect an exception in case of re-targeting, since the credentials won't match. Because DTLS-SRTP relies on certificates, the initiator will have received the certificate of the responder and will be able to identify the person to whom the call has been re-targeted.

As an aside, to our knowledge no proposed protocol supports cryptographic delegation of authorization from Bob to Carol. Such an authorization, for example, may be a simple specially formatted message signed by Bob, associating Carol's public key, delegation period, and possibly other relevant information. When Carol answers the call, this message can be attached to her PKI chain to convince Alice that Carol is an authorized representative.

Conferencing:

DTLS-SRTP does not support establishment of a single key shared between more than two endpoints. However, participants can still establish DTLS-SRTP sessions individually with a conference bridge.

When Alice participates in a conference, DTLS-SRTP allows her to establish secure media to the conference bridge or entity acting as the bridge in the case of three-way calling when a participant bridges someone into the call. Alice has no control over whether or not media from her is encrypted as it is sent from the bridge to other participants. Alice also has no control over who the other participants are and therefore to whom the media is sent (aside from being able to choose not to participate herself).

Calls to Other Networks:

As mentioned in the discussions on forking and re-targeting, one endpoint may not be within a VoIP network and the SRTP terminates at a gateway to another network, such as a switched cellular network or PSTN. The same gateway may not be used for every call between the same two endpoints. For such calls, DTLS-SRTP only provides establishment of SRTP keying material between the participant on the VoIP network and an undetermined endpoint.

Legal Interception:

DTLS-SRTP exchanges keys end-to-end in the media stream. Unlike SDES, it does not transport the keys in signaling thus making legal interception (or informed recording of conversation as in the case of call centers or financial transactions) harder to support.

Session Recording:
DTLS-SRTP does not provide a general recording solution since it does not specify the exact means by which the key can be shared with a recording server.

6. OUR MEDIA KEYING SCHEME

As discussed in the introduction, while DTLS-SRTP and ZRTP provide strong security in establishing session keys, they are still not widely deployed due to the complexity of implementation and significant computation and communication costs. For simplicity and efficiency reasons, media key is often chosen and transmitted by Alice to receiver Bob via trusted SIP framework intermediaries. This method provides adequate security for low-value media streams.

In this section, we propose a simple and efficient way to significantly increase the security of SIP key transmission, with minimal additional costs. We do not resort to more expensive public key cryptography. Our proposal involves sending an extra key, and evaluating one Pseudo-Random Permutation (PRP), such as AES. We believe that our protocol presents a desirable trade-off between security, costs, and deployment complexity. It can be built directly from SIP key transmission by adding two simple steps.

We start the presentation with discussion of some of the weaknesses of SIP key transmission.

Key Transmission weaknesses.

We assume that both Alice and Bob are properly authenticated to the SIP network. We mention obvious vulnerabilities resulting from corrupt SIP server node(s) – in this scenario all security is lost since adversary sees the session key.

However, there are subtle attacks by a relatively weak adversary who does not have access to privileged SIP nodes. These attacks are due to forking, which may result in the SIP network sending Alice's key to more than one of Bob's devices.

The first attack may occur in the scenario where adversary is in possession of one of the Bob's devices B_1. Then, adversary is informed of the session key by the SIP network and can interact with Bob pretending to be Alice. (We note that this attack is prevented in our protocol.) Note that this is a different and stronger attack than an (unavoidable) possibility of adversary in possession of B_1 pretending to be Bob to unsuspecting Alice.

In the second attack scenario, adversary does not have access to Bob's devices, but controls a portion of the media-plane network. He is able to redirect the messages between honest Alice's device A_1 and Bob's two devices B_1 and B_2, all of which use the same key k. Even though adversary does not know the shared key all three devices share, adversary may be able to route the messages to create unintended transactions, even if channels are protected with k. For example, Bob's devices may talk to each other thinking they are talking to Alice. Or, outside of VoIP scope, both Bob's devices would initiate a transaction (e.g. a money transfer), and result in duplicate transaction execution[1].

Finally, even if SIP servers are trusted — and it is reasonable to trust the intermediaries not to abuse the knowledge of all session keys — hiding the keys from them, among other advantages, reduces the servers' liability, consequences of compromise, and makes system recovery easier.

DTLS-SRTP is a secure Key Exchange (KE); using SIP with DTLS-SRTP avoids all possibilities of attacks[2], including the above weaknesses. However, this solution involves the use of PKI. While conceptually relatively simple, PKI systems are expensive to deploy and manage, and it is best to avoid them. As suggested in DTLS-SRTP, a natural way to eliminate the logistical complexity of PKI is for the participants to transfer hashes of their certificates in the secured SIP layer. This way, the adversary on the insecure media plane channel would not be able to substitute the certificates, and thus the certificates can be trusted. However, this approach fails to provide security against SIP-layer adversaries, who in fact can substitute the certificates and enable man-in-the-middle attacks. We believe this is a reasonable compromise between security and deployment and running costs. The above non-PKI version of DTLS-SRTP is (ever so slightly) more secure but also still significantly more costly than our key agreement protocol described next.

6.1 Description of our solution

Our proposed solution achieves most of the security goals achieved by the above non-PKI version of DTLS-SRTP, but without the computation and communication complexity associated with its public-key operations.

We present a generic version of the protocol, based on Pseudorandom Permutation Generators (PRPG). Further, we do not fix the domains for the randomly drawn keys, messages, and values. We only require that they are "large enough", according to the current suggested key lengths. Today, we envision using the AES encryption as the PRPG, in which case the domains of k and r may be $k, r \in \{0,1\}^{128}$.

PROTOCOL 1. *(Secure SIP Key Transmission)*
Setup: Initiator Alice wishes to securely connect to responder Bob. Both Alice's and Bob's devices are authenticated to the corresponding SIP servers.

1. *Alice chooses a random key k and transmits string (Alice, Bob, k) to the responder Bob via the SIP framework, as it is done in the SIP key transmission method. We stress that the transmitted key will* not *be the session key that is used for communication.*

2. *Upon the receipt of the key, Bob chooses a* random *nonce r and sends back string (Alice, Bob, r) to the initiator Alice, together with the media stream.*

3. *The session key, which can be immediately used to encrypt the media, is the PRPG F evaluated with the seed k on the data r. Namely, the session key is $sk = F_k(r)$*

This protocol flow is illustrated in Figure 1.

[1] We note that while the session encryption *may* be such that such message manipulation is difficult (e.g. using special counters), key security should not be delegated to the session, but achieved in the key exchange/transmission phase. This would allow for better modularity and more easily understandable protocols. Further, standard proofs of security are done in this modular world.

[2] The crypto core of TLS was formally proven secure [16]. However, the complete protocol suite has not been fully analyzed, and there are possibilities of errors leading to possible attacks, such as the recent TLS renegotiation attack.

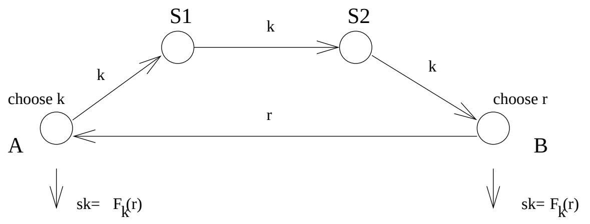


Figure 1: Secure SIP Key Transmission

Note that, in particular, this protocol prevents more than one instance of Bob from obtaining the session key due to forking. This is because each Bob's instance will choose its own nonce, and obtain a corresponding random session key.

As with other solutions, the security of the system can be further improved by maintaining some state between sessions. That is, if honest Alice and Bob establish a key without interference and eavesdropping, they can exchange a long-term pre-shared key, and execute provably secure and efficient key exchange in future sessions.

6.2 Security Analysis

In this section, we analyze security properties of our protocol and conclude that our improved method of key transmission is indeed secure against relatively strong attackers. We achieve standard cryptographic Key Exchange (KE) security properties when the adversary is restricted to operate either in the signaling or the data plane, but not both. More specifically, we protect against general MiTM attacks, where the adversary may arbitrarily interfere on the corresponding channels among the many communicating instances of Alice and Bob. (Such MiTM subsumes all active attacks, e.g., replay.) As we aim for maximal simplicity and efficiency, we do not provide non-essential KE features such as *perfect forward secrecy* (can be easily added at the cost of running a DH exchange), or *mutual authentication* – assurance that a successful termination of KE by Alice (resp. Bob) implies that indeed Bob (resp. Alice) participated in this protocol (can be added at the cost of two additional "key confirmation" flows and refreshing the session key). We further do not worry about DoS attacks, and their equivalents, such as adversary causing players outputting unrelated random keys (this does no more harm than simply cutting the channel, which active adversary can do anyway).

There are several definitions of security for key exchange in the cryptographic literature. While the exact relationship among them is often not investigated, they all guarantee very strong security properties, including the secrecy of the key and very general inability to mismatch players (i.e., forge an unintended communications channel). As there does not appear to be a KE definition for our setting, to formally prove security, we would first need to formally define KE. It seems easiest to do by restricting the powers of the adversary in the definition of Kolesnikov and Rackoff [12]. The full (involved) proof of security of our protocol can then be constructed based on that of [12]. In this work, we sketch the main points of the proof, and state corresponding theorems.

We first show that in the absence of SIP-layer adversaries, our protocol is secure in the strong cryptographic sense. Here, and in Theorem 2, by "secure" we mean the satisfying above (informal, but naturally formalizable) properties.

Theorem 1. *Let F be a PRPG. Assume that the SIP network securely and privately transmits the key k chosen by Alice. Then our protocol is a secure key exchange protocol.*

As noted above, here we only present the main points of why Theorem 1 holds. Indeed, a polynomial-time adversary who observed (or even modfied in transit) r, but had not obtained k (this is the media-plane-only adversary considered in the theorem), will not be able to distinguish sk from a random string of the same length. This follows immediately from the security properties of PRPG F, namely, from the fact that the output of F evaluated with a random and unknown key on any adversarily chosen message, is indistinguishable from a random string. Further, adversary will not be able to mismatch honest players (i.e. forge an unintended communication channel), since honest Alice instance A_i selects random k_i, and each of the honest Bob's devices B_j (who receives some k_i) independently chooses r_j; both k_i and r_j are unique with overwhelming probability. Therefore, even if the same k_i is delivered to several Bob instances (e.g. due to forking) and arbitrary r values are delivered to Alice instances, the keys output by each player instance will be either all independently random, or there may be (at most) two equal keys, which would correspond to a successful completion of the KE protocol. (We note that a media-plane MiTM may "connect", i.e. cause output of the same session key, of a different Bob's device that Alice expects, e.g., based on the IP address. We note that this can be avoided by additional signaling in the SIP layer, but we do not consider this a KE vulnerability. All we guarantee here is that if Alice establishes a channel, it is with a single Bob's device.)

Next, we show that our protocol is secure against a SIP signaling-plane-only adversary. We note that this corresponds to the setting where a SIP server may be corrupted, but the attacker is unable to consistently monitor the general Internet traffic of the parties.

For formal proof of security in one of the attack scenarios covered by the next theorem, we would need to rely on a slightly stronger than PRPG notion of security, *ideal cipher*. (See Footnote 3 for high-level description of its security properties.) We envision using AES as the instantiation of ideal cipher, as its design aims to satisfy the required properties.

Theorem 2. *Let F be an ideal cipher. Assume that adversary is unable to observe or interfere with (only) the protocol message sent in the media plane. Then our protocol is a secure key exchange protocol.*

We give intuition for the proof of Theorem 2. Indeed, a polynomial-time adversary who observed k, but had not obtained r, cannot distinguish sk from random. This is because F_k is a (known to the adversary) permutation, which, applied to a random input, produces random output. We note that a SIP signaling plane attacker (i.e. a rogue server) modifies k in transit does not gain any advantage, if a "good" pseudorandom function (e.g. AES) is used[3].

[3] Strictly speaking, PRPG does not guarantee any security properties if executed on *related* keys. Resilience to related key attacks is modeled by assuming stronger properties on the underlying function, in our case, AES. This assumption

Further, adversary cannot forge an intended connection among the players, since all Bob's devices choose an independently random r, which results in all of them computing independent session keys. Alice receives r generated by one of the Bob's devices, and outputs either the same corresponding session key, or an independently random key, in case k was modified (here we use the ideal cipher assumption). Note that SIP-layer adversary may misrepresent the identity of Alice to Bob, hoping to cause Bob to believe he is talking to Carol, while he is in fact talking to Alice. We address this by including the names of both players, in order (initiator, responder) in both protocol messages. This way, Alice will not accept Bob's response which includes Carol's name.

We stress that we use the ideal cipher assumption only for proving claims related to active adversary in the SIP layer. All other claims are proven only assuming F is a PRPG.

Finally, we caution the reader that colluding SIP servers and the media-stream attackers succeed easily. It is sufficient for the SIP server to leak the key k to the media-stream MiTM to break into the conversation. However, at the same time, attacking a non-PKI DTLS-SRTP version described in this section requires only slightly stronger resources. There, a SIP server simply substitutes the transmitted hash to enable the media-stream MiTM to perform the attack.

In conclusion, we proposed a simple, secure, and very efficient amendment of the protocol for key transmission. In particular, our proposed amendment reduces the trust assumptions on the SIP servers, and prevents instances of the responder sharing the session key due to forking.

6.3 Suitability for Feature Set

We now discuss the applicability of our approach to the features outlined in Section 3.

Forking: In DTLS-SRTP, keying material is exchanged completely in the the media stream. In our proposal, the key exchange is distributed between the signaling stream and the media stream: the random key k is sent in the signaling stream and the nonce r flows in the media stream. When forking occurs, k remains constant for all the forked branches, but each branch contributes a unique r, thus deriving a separate session key and preventing key leakage to parties not part of the session. Similarly, HERFP does not pose a problem since there is no key negotiation done in signaling; the k carried in signaling is not subject to negotiation. If an endpoint does not support the interpretation of k, it will simply ignore it (following the accepted practice of handling unknown headers and attributes in Internet protocols.)

Media Clipping: Media clipping does not pose a problem in our approach. Key derivation is complete when A (see Figure 1) receives nonce r. Since B will send the first media packets, it can encrypt them using the session key (thus, no plaintext RTP packets will be sent.) Furthermore, since the nonce r is different for each endpoint the request forked to, HERFP does not pose a problem for our approach.

Re-targeting: We do not formally address re-targeting. However, we briefly sketch the possibilities to handle its simple forms.

When Alice's UA has a good user interface (e.g. a computer, or a phone with a display), SIP layer may inform Alice that the call was sent to Carol rather than Bob, and that would imply Carol is authorized to receive Bob's calls. Further, Alice's UA may store the pre-shared key she had shared with Bob (if this is a repeat call), and determine by itself that re-targeting has occurred. In either of these cases, Alice is notified of an exception, and may take corresponding action. Finally, as with forking, we note that keys will not be leaked to unintended recipients.

Conferencing: Like the other two approaches, our solution does not allow the establishment of a shared conference key.

Calls to other networks: The security properties of calls to other networks with respect to our approach remain the same as the approach taken by SDES and dtls-srtp.

Legal intercept: Since r is exchanged directly between the peers, our approach like DTLS-SRTP, does not support legal intercept.

Session recording: Our approach, like DTLS-SRTP, does not support session recording.

7. RELATED WORK

MIKEY [2] is another IETF standardized protocol for keying multimedia applications. However, it has largely remained unimplemented for SIP today primarily because it is a signaling- only keying technique. ZRTP [24] is a media-path key exchange protocol that does not use PKI (Floroiu et al. [8] discuss ZRTP in the context of SIP in more detail.) Wang et al. [21] use Identity-based encryption [19] to exchange keys for authenticating the endpoints as well as keying the SRTP media stream.

8. CONCLUSIONS

We have presented and proved secure a novel key exchange method that involves only one symmetric key exchange operation by the sender and receiver. We provide security guarantees which are much stronger than that of SDES, and are nearly as strong as that of DTLS-SRTP. At the same time, our computational costs are comparable to that of SDES, and are *much* less expensive than DTLS-SRTP and ZRTP, which use public-key encryption. Table 2 provides a feature evaluation summary of our key exchange method with the analysis we performed also for DTLS-SRTP and SDES. We note that our method compares well against both SDES and DTLS-SRTP.

9. REFERENCES

[1] F. Andreasen, M. Baugher, and D. Wing. Session Description Protocol (SDP) Security Descriptions for Media Streams. RFC 4568 (Proposed Standard), July 2006.

[2] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. Internet Draft, Aug. 2004.

[3] N. Asokan, V. Niemi, and K. Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *Security Protocols Workshop*, pages 28–41, 2003.

is referred to as the *ideal cipher* assumption. While it is sometimes considered too strong in theoretical cryptography, in our scenario, it is far easier to stage a different class of attack (e.g., intercept r in the media layer) than to exploit the strength of this assumption.

Table 2: Feature Evaluation Summary

	SDES	DTLS-SRTP	Our approach
MiTM	Attacks possible	Mitigated through certifcate fingerprints	Adversary controls media plane: secure; adversary controls signaling plane: secure; adversary controls both planes: attacks possible
Forking	Key leakage occurs	No key leakage	No key leakage
HERFP	Remains problemmatic	Not a problem	Not a problem
Media clipping	Remains problemmatic	Not a problem	Not a problem
Retargeting	Remains problemmatic	Detects retargetting	Detects retargetting
Conferencing	Not supported	Not supported	Not supported
PSTN calling	Supported	Supported	Supported
Legal Intercept	Supported (keys can be provisioned in endpoints)	Not supported (key exchange is end-to-end)	Not supported (key exchange is end-to-end)
Session Recording	Supported (keys available to the recording server)	Not supported (no mechanism in protocol to share keys)	Not supported (no mechanism in protocol to share keys)

[4] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711 (Proposed Standard), Mar. 2004.

[5] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO 93*, volume 773 of *LNCS*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag.

[6] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008.

[7] J. Fischl, H. Tschofenig, and E. Rescorla. Framework for Establishing an SRTP Security Context using DTLS, IETF Internet-Draft, Work in Progress, Mar 2009.

[8] J. Floroiu and D. Sisalem. A Comparitive Analysis of the Security Aspects of the Multimedia Key Exchange Protocols. In *Proceedings of the 3rd international conference on Principles, systems and applications of IP telecommunications (IPTComm)*. ACM, July 2009.

[9] D. Geneiatakis, A. Dagiouklas, S. Ehlert, G. Kambourakis, C. Lambrinoudakis, D. Sisalem, and S. Gritzalis. Survey of Security Vulnerabilities in SIP. *IEEE Communications Tutorials and Surveys*, 8(3), October 2006.

[10] V. Gurbani, S. Lawrence, and A. Jefferey. Domain certificates in the session initiation protocol (SIP), IETF Internet-Draft, Work in Progress, draft-ietf-sip-domain-certs-04, May 2009.

[11] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[12] V. Kolesnikov and C. Rackoff. Key exchange using passwords and long keys. In *Theory of Cryptography, TCC 2006*, volume 3876 of *LNCS*, pages 100–119. Springer, 2006.

[13] V. Kolesnikov and C. Rackoff. Password mistyping in two-factor-authenticated key exchange. In *ICALP (2)*, pages 702–714, 2008.

[14] D. McGrew and E. Rescorla. Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-time Transport Protocol (SRTP). RFC 5764 (Proposed Standard), May 2010.

[15] N. Modadugu and E. Rescorla. The Design and Implementation of Datagram TLS. In *The 11th Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, February 2004.

[16] P. Morrissey, N. P. Smart, and B. Warinschi. A modular security analysis of the tls handshake protocol. In *Advances in Cryptology – ASIACRYPT 2008*, volume 5350, pages 55–73, Berlin, Heidelberg, 2008. Springer-Verlag.

[17] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002.

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[19] A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[20] V. Shoup. On formal models for secure key exchange. Technical Report RZ 3120 (#93166), IBM, 1999.

[21] F. Wang and Y. Zhang. A new provably secure authentication and key agreement for sip using certificateless public-key cryptography. *Computer Communications*, 31(10):2142–2149, June 2008.

[22] D. Wing, F. Audet, S. Fries, H. Tschofenig, and A. Johnston. Secure media recording and transcoding with the session initiation protocol, IETF Internet-Draft, Work in Progress, draft-wing-sipping-srtp-key-04, October 2008.

[23] D. Wing, S. Fries, H. Tschofenig, and F. Audet. Requirements and Analysis of Media Security Management Protocols. RFC 5479 (Informational), 2009.

[24] P. Zimmermann, A. Johnston, and J. Callas. ZRTP media path key agreement for secure RTP, IETF Internet-Draft, Work in Progress, draft-zimmermann-avt-zrtp-17, January 2010.

Reusable Features for VoIP Service Realization

Thomas M. Smith
AT&T Labs Research
180 Park Avenue
Florham Park, NJ 07932
USA
tsmith@research.att.com

ABSTRACT

Telecommunication services vary greatly in their behavior. However they often can be decomposed into tightly-focused components, each designed to accomplish a certain limited function. In some cases, these functions are repeated across many services that seem quite disparate at first glance. We examine some components that have proven to be highly reusable, and demonstrate how they can be composed into a variety of interesting services.

Keywords

Telecommunications, VoIP, features, application composition, patterns

1. INTRODUCTION

A wide variety of telecommunication services provide familiar behavior to users everywhere. Common examples include voicemail systems, conference calling services, IVR "phone-tree" applications commonly employed by businesses, and end-user features such as call waiting and three-way calling. Some services of limited scope may be provided by a standalone software implementation of modest complexity. However, when the number of supported features increases, they may be implemented in highly complex, even byzantine, software systems. Such systems complicate the task of maintaining and extending the service logic.

It has long been considered useful to consider models in which system behavior can be decomposed into standalone modules that can be independently specified, developed, and tested. Such modules can then be combined to provide more functionality without undue complexity [8, 9, 14, 17, 10, 21]. Such a design ethic is evident in currently-emerging standards such as the IP Multimedia Subsystem (IMS) [19] and the SIP Servlet 1.1 specification [11].

This paper draws on experience developing Voice over IP (VoIP) software modules to distill a number of highly useful software components, each performing a limited function. In Section 2, the function of each component is described and illustrated by example. In Section 3, the components are employed to compose some familiar services. Section 4 contains some preliminary thoughts on different software mechanisms that can be used as compositional patterns, and the appropriateness of each in different settings. We conclude in Section 5 with a description of future work.

2. FEATURES

In this section, we will describe a number of telecommunication modules that represent common functionality that can be found in a number of different contexts. In traditional telecommunication parlance, these are designated as *features.* More generally, these features embody common *design patterns* of telecommunication logic. Software design patterns have been studied extensively [7, 16, 12], and there is a body of work concerning the use of design patterns in telecommunications [18, 20]. The features described here exhibit patterns of call-control logic. Note that although these features can be characterized as patterns, they have also been reified as functional, standalone software components.

Note that the increment of functionality embodied in these features is small; the scope of an individual feature is on the order of "call waiting," not "IP-PBX." A system with a scope of the latter could be built out of a large number of features with the scope of the former, however. This set of features is certainly not meant to be comprehensive, but rather illustrative of how small components can be composed to realize a variety of outcomes.

The features described here can be viewed through the lens of *scope, commonality, and variability* [6], though these decompositions are the result of informal iteration rather than systematic procedure. Encounters with repeated call-control patterns have suggested the scope of each feature. The commonality between instances of the features is generally dictated by the call processing message classes being acted upon (either sent or received); and the variability usually applies to parameters used to form the details of the message contents, rather than the message classes. Generally these parameters take the form of addresses, timeout values, etc.

Each of the features is described in isolation below. Then we will consider how to compose these features into useful services using *pipe-and-filter composition* [2, 10, 13]. This style of composition allows all components (features) to remain independent of the others; in fact the components may not even be aware of the existence of other components. The graph that results from the runtime composition of multiple features may be achieved by direct addressing, in which

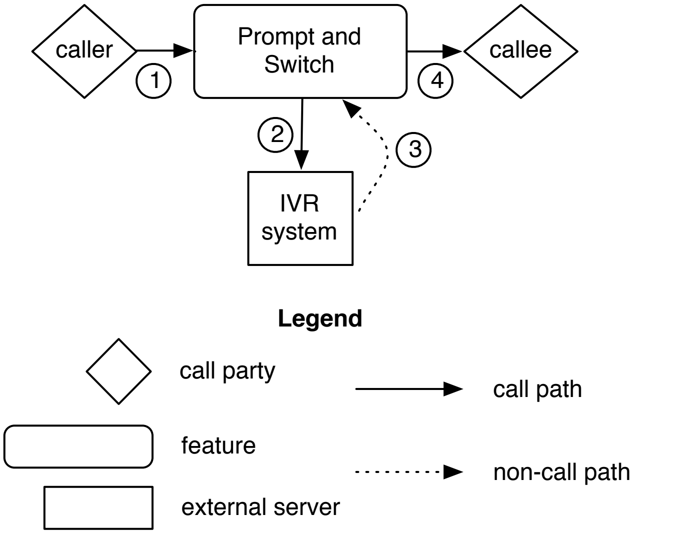


Figure 1: Prompt-and-Switch feature

features translate addresses in order to route calls to other features, or through a more sophisticated subscription mechanism, as in Distributed Feature Composition (DFC) [10]. In either case, the mechanics of composition are achieved through telecommunication protocol means, with no changes required to feature logic. Call-time composition is also used in IMS [19] and the SIP Servlet 1.1 specification [11]. Note than anywhere that a *call party* appears in the representation of a feature call flow, that party may be another feature, not necessarily an end user. From the vantage point of a feature, it does not know or care what entity is placing or receiving a call.

2.1 Prompt and Switch

The *Prompt-and-Switch* (PS) feature, shown in Figure 1, provides the ability to answer an incoming call with an automated audio dialog as commonly found in Interactive Voice Response (IVR) systems. After a period of time during which the calling party interacts with the automated dialog, the logic of the IVR system can specify an address to which the calling party should be switched. This function should be very familiar to anyone who has called the main number of a business, only to encounter an automated menu listing the parties (or departments) that can be reached via an interaction with the menu. This feature takes care of the required signaling to redirect the call to the IVR system, as well as any further signaling required to tear down the IVR call and switch the (connected) caller to a new (unconnected) address.

As presented, this feature could perform a number of familiar standalone functions:

- It can provide an "automated attendant" function, prompting the caller with a menu of departments that can be reached.
- It can be the basis of a prepaid calling service, where calls are placed to an IVR system to collect a prepaid account number and desired number to call. When the called party hangs up, the caller is reconnected to the IVR system and allowed to make more calls.

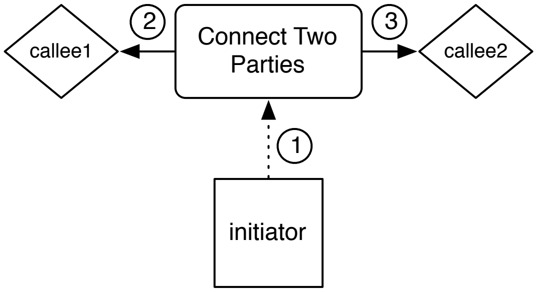


Figure 2: Connect-Two-Parties feature

- It can support "remote authentication," where subscribers call in and authenticate themselves before being allowed access to some protected functions (including placing an outbound call). Such a function is present in AT&T's Callvantage Service [4].

Figure 1 shows a graphical representation of this feature. The incoming call, labeled (1), is directed to the IVR system (2). The IVR system returns a command (3), causing the feature to switch the caller to an address specified by the IVR logic (4).

In terms of [6], the *commonality* is as described above. One element of *variability* in the call-processing logic is that the feature may be optionally configured to detect call termination from the callee and to reconnect the caller to the IVR system for further instructions. This behavior could be used to allow a caller to make multiple sequential calls after entering a prepaid calling card number or performing remote authentication.

2.2 Connect Two Parties

While a great many features are activated by an incoming call, there are occasions where the initial event is something other than a call-related event. An example of this would be a corporate web page with a button that reads, "Click here to be connected to one of our representatives." In this case, the initiating event would be a web click, not a call event. This function can be implemented in a *Connect-Two-Parties* (CTP) feature that is activated by the non-call event, as shown in Figure 2.

The initiating event must specify the addresses of the two parties that should be connected. To reduce confusion, it is desirable that the parties be connected one-at-a-time; that is, the second party will not be called until the first party answers. Accordingly, execution should terminate if the first leg of the call fails.

Figure 2 shows a graphical representation of this feature. A non-call event (1) prompts the CTP feature to place outgoing calls to first one party (2), then the other party (3). These parties are located at addresses that are specified in the initiating event.

Note that the endpoints of this call need not both be live people; rather they can be any callable entity, including media servers and other features.

2.3 Redirect on Failure

The *Redirect-on-Failure* (RF) feature provides a simple but crucial capability: it continues an incoming call toward its destination, but in the event that the call cannot reach its destination for any of a variety of reasons (busy, no answer,

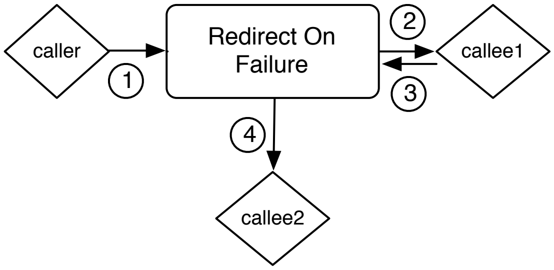


Figure 3: Redirect-On-Failure feature

network error), this feature does not propagate the failure back to the calling party. Instead, it continues the call to a different address, which may be explicitly specified through provisioning, or may be algorithmically determined based on the addresses of the parties in the call. The basic functionality may be enhanced by the optional ability (variability) of only redirecting for certain classes of failure responses, and otherwise propagating the failure back to the calling party.

The canonical use of this feature is to redirect a failed call to a resource that will record a voicemail message for later retrieval by the initial called party. Another use is to provide a "Safe Forwarding Number" in case of failure from an endpoint [4].

Figure 3 shows a graphical representation of this feature. An incoming call (1) is continued toward its destination (2). A failure response from callee1 (3) causes the RF feature to continue the original call to callee2 instead (4).

2.4 No Answer Timeout

The ability to detect call timeouts is important in a variety of contexts. A familiar example is the redirection of an incoming call to a voicemail server after a specified "Ring No Answer" timeout. However the ability to detect and act upon such timeout conditions is more general and can be used in other contexts, as we will see below. Therefore a simple *No-Answer-Timeout* (NATO) feature which detects such a condition and can signal its occurrence to other parties can be valuable. A failure response may be the mechanism it uses to signal the outcome to other components.

No graphical representation should be necessary for the understanding of this feature. It works with a calling party and a called party, and behaves transparently unless the no-answer condition is satisfied, in which case it ends the call to the called party and signals the calling party of the condition.

3. SERVICES

This section discusses *services*, by which we mean a grouping of *features* used to accomplish a task of interest. Informally, a service represents a unit of functionality that could be marketed and sold. The functionality of each example service will be described, and a possible implementation using pipe-and-filter composition of the preceding features will be presented.

These services can be viewed as products in a *software product line* [15]. In this context, the set of features presented in Section 2 comprise the *platform*; customization for individual products occurs through the selection and relative arrangement of the appropriate features from the platform as well as parameter-based specialization of those features.

3.1 Conference Calling Service

A typical business-oriented *Conference Calling Service* operates as follows. A user calls a well-known address (the *bridge number*) and interacts with an IVR system to provide details about the conference that the user wishes to join. This information often takes the form of a personal identification number (PIN) for the desired conference. Upon successful entry of this information, the user is switched into a conference call, in which the media streams from all users are mixed.

The core of such a service can be realized through the use of a PS feature for initial PIN processing in conjunction with a media server to provide the media mixing after successful completion of the IVR dialog. In addition, a capability could be added to enable users to connect to the conference via clicking on a web link instead of dialing the phone. This can be enabled through the introduction of a CTP feature to call out to the user and then connect the user to the IVR system.

Figure 4 shows an architectural diagram of the service, composed of the individual features.

3.2 Voicemail/Do Not Disturb Service

A *Voicemail* service is used to capture incoming calls when the intended recipient is not available. The two common cases that result in voicemail treatment for a call are when the called party is *busy* or when there is *no answer*. Either of those conditions, as well as various network failure conditions, can be viewed a failure, so the use of the RF feature is natural. The NATO feature can be employed to generate a failure response when the no-answer condition is detected.

Do Not Disturb treatment is intended to reduce or eliminate unwanted incoming calls to a user. The simplest implementation rejects all calls while active. A less draconian form may use a *blacklist* (or *whitelist*) approach, where a list of addresses is deemed as undesirable (or desirable), and calls from those addresses are consequently rejected (or accepted), while all others are accepted (or rejected). Rejected calls may be redirected to a voicemail system.

A more sophisticated implementation may include a notion of a *graylist*, indicating that the caller is required to interact with an IVR system before the decision is made as to whether or not the call can "ring through." It is this version of the service that we describe in this section.

Figure 5 shows an architectural diagram of the service, composed of the individual features. The presence of an IVR system with conditional switching based on the outcome of the IVR dialog suggests the presence of a PS feature. Note that the NATO feature is not adjacent to the RF feature, as it might be for a pure voicemail service. In this case, the NATO feature should not be invoked until the IVR system has determined that the caller is authorized to ring through. This dictates the placement of NATO between the PS feature and the callee. Now three different conditions can cause the RF feature to send the caller to voicemail:

- Call rejection from Do Not Disturb logic
- Failure response from callee (e.g., busy condition)
- No answer from callee

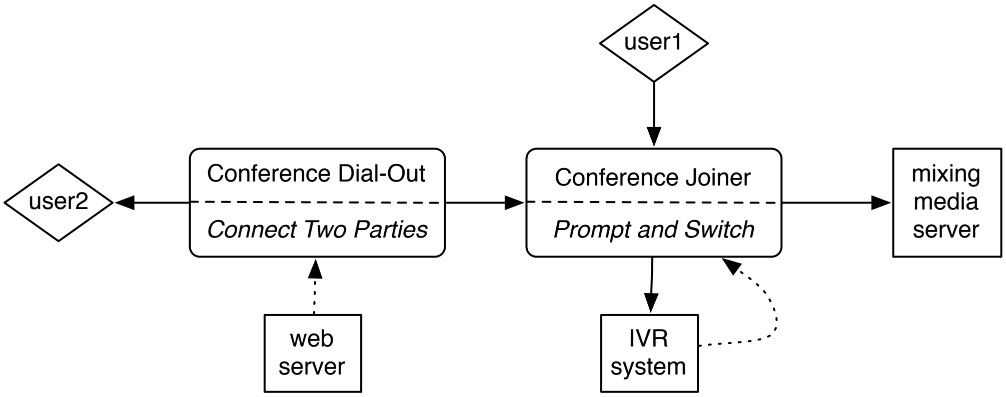


Figure 4: Conference Calling Service Architecture

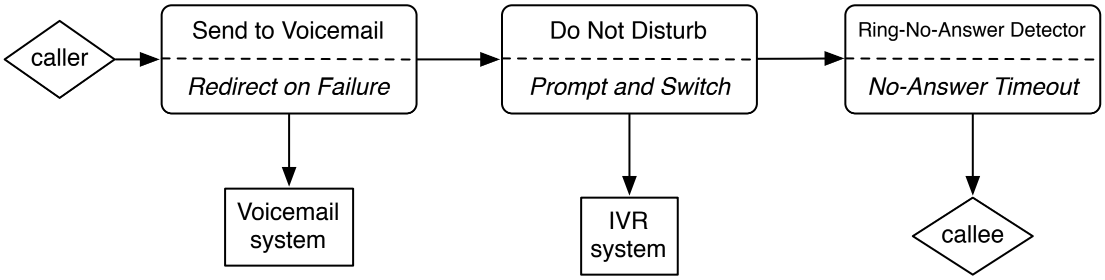


Figure 5: Voicemail/Do Not Disturb Service Architecture

The RF feature, which sends the caller to the voicemail system, is not concerned with which of these conditions has occurred. Its role is the same, regardless of what has happened in the rest of the call path. This is the essence of modularity.

3.3 Record and Send

The *Record and Send* service permits a subscriber to record a message which will then be automatically delivered to a list of addresses via an outbound call. Such a service can be used for event reminders (*"Don't forget to vote in today's school elections"*) as well as timely notifications (*"Today's lacrosse game is cancelled due to poor field conditions"*). The subscriber could possibly set up the service over an IVR session, but the complexity of managing a list of addresses to notify lends itself to a visual medium, such as a web interface. This variation is described in this section.

To use the service, the subscriber logs into a service website. A list of addresses may be maintained between sessions, so that certain addresses may be selected from a list of previously-used addresses (or from an address book); new addresses can be added as required. Once the list is complete, the subscriber indicates whether the notifications should be made right away, or at a scheduled time. Finally, the message must be recorded. When the subscriber chooses to record, the web server can send a request to a CTP feature to connect the subscriber and an IVR dialog which will prompt the user to record the notification message. This is in case the link is clicked in error when the subscriber is not near the telephone, and this prevents the device ringing again and again, when under normal circumstances the subscriber should be able to answer quickly. This can be provided via the NATO feature on the leg of the call going to the subscriber.

Once the outgoing message has been recorded, the service logic can place calls to the specified list of addresses, once again via a CTP feature. Each instance of the CTP feature will connect one notification address with an IVR dialog in order to play out the message. The service could do these all in parallel, or could stagger them in time in order to meet resource constraints. Such a decision has no affect on the logic of the features. A Ring Stopper can be employed if desired on the outgoing calls.

Figure 6 shows an architectural diagram of the service, composed of the individual features.

Note that the initial Connect-To-Recorder function can be reused in other contexts. Voicemail systems typically allow a subscriber to record an *outgoing message* that will be played to callers when the subscriber is unavailable. Sometimes there are multiple outgoing messages, each to be played when certain conditions are met (subscriber does not answer, subscriber is on the phone, etc.). The user interface employed to record these messages is typically an IVR system which is used to manage all aspects of the voicemail service, including message retrieval and management of settings such as the number of rings allowed before redirection. As such, the functions used to record outgoing messages are typically located within a fairly complex IVR dialog.

There is a trend, particularly for VoIP systems, for providing web-based interfaces to voicemail systems in addition to the traditional IVR interface. These web interfaces typically allow access to recorded messages, as well as to configuration options. By converting the serial presentation of an IVR dialog to a visual presentation, usability can be greatly

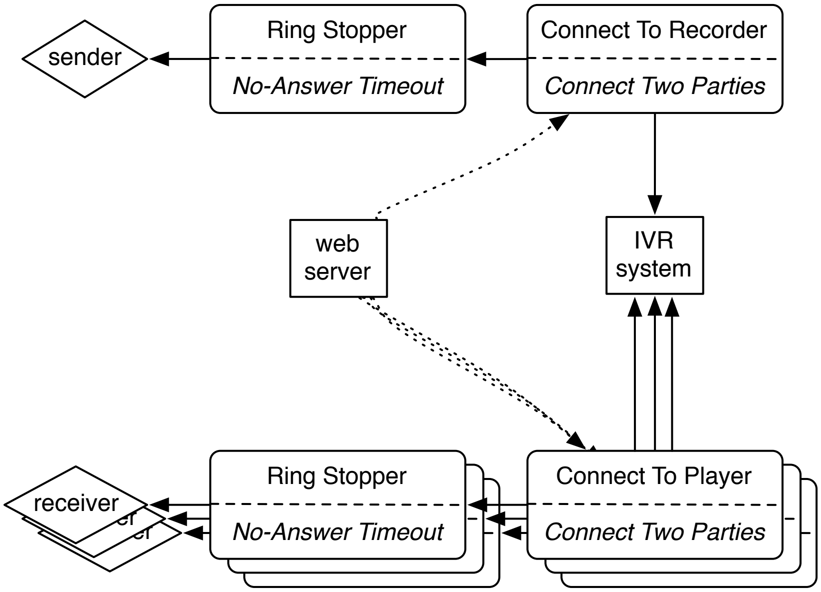


Figure 6: Record and Send Architecture

enhanced. Some systems, such as AT&T's CallVantage Service, allow the user to click a link on this web interface in order to record an outgoing message. When the link is clicked, the subscriber's phone will ring; when the call is answered, the subscriber is placed precisely at the point in the IVR dialog where the greeting is recorded. This allows the use of a web browser for viewing and choosing configuration options, coupled with the use of the telephone as a ubiquitous audio input device.

Since the IVR component is presumed to exist already, the web-based greeting recording function can be added using the same Connect-To-Recorder composition shown in Figure 6.

4. OTHER COMPOSITIONAL PATTERNS

The previous section illustrated the creation of service logic through *pipe-and-filter* composition. This design pattern is well known in software engineering, and can be seen in Unix process pipelines, data processing, and web development frameworks [2, 1]. The Distributed Feature Composition (DFC) architecture [10] employs the pipe-and-filter architecture to great advantage in controlling feature interaction. By proper use of address translation [22], a variety of DFC *usages*, each containing an appropriate feature chain, can be assembled. DFC provides a theoretical basis that can be used for realizing composed services, like those in the previous section. By having a toolbox full of general features, services can be easily composed by manipulating addresses and feature subscription, instead of writing new code.

However, pipes and filters are not the only mechanism for software re-use. Perhaps the most common software re-use mechanism is that of a *software library* made available to applications via a documented API. Software libraries are routinely used during almost any software development process. Some of the functions defined in the preceding sections could also be implemented as library modules.

There are differences in the patterns of re-use between composing complete feature modules, as discussed in Section 3, and using software libraries. One key difference is that feature composition occurs at runtime, and library composition occurs at compile time.

There are tradeoffs associated with these two patterns of re-use. For example, use of a software library may well have less overhead associated with a call to a library routine than with the instantiation of a feature instance and the overhead of having more call protocol instances to manage. Compile-time checking could also ensure that the function is being invoked from an appropriate context. On the other hand, run-time assembly of the software components means that no recompilation is required to change behavior. This means that the software composition can be controlled purely through configuration of the desired call graph.

In practice, it may not be clear which of these compositional patterns is more suitable for a particular implementation. Some experiences indicate that pipe-and-filter composition may be most appropriate when designing a new service for the first time, as this level of composition can be achieved through configuration rather than code changes [5]. When an implementation is more mature and stable, it may be desirable to migrate certain functions to library routines. This can potentially improve performance characteristics of the system as a whole, presuming that calling a library routine is less computationally expensive than performing the incremental call processing that would be required for a standalone feature. A good rule of thumb may be that if a component needs to create or absorb call-path messages in order to perform its function, it should be realized in a feature, not a library routine. There are at least two justifications for this design rule. First, if one component is linked to another at compile time, it prevents run-time interpolation of a third component between the other two and thus constrains flexibility for re-use. Also, inter-feature messages are a critical mechanism for analyzing and controlling feature interaction.

5. FUTURE WORK

Extensive experience building VoIP services has resulted in the extraction of common call-control patterns into the features described in this paper. There is every reason to expect that further experience will result in continuing insights into interesting decompositions. Investigation of systematic techniques such as [6] could yield new insights. The applicability of software product line engineering techniques [15, 3] to this domain should be explored.

Additionally, the thoughts on alternate compositional patterns are in the early stages. Other compositional patterns from the software engineering literature [2] should be explored for possible applicability to this domain. Further reflection and experience should lead to more concrete and rigorous design guidelines.

Finally, the topic of *converged services* has not been directly addressed in this paper. When non-telecommunication protocols (such as HTTP) are included as part of the service logic, the boundary lines of features, and thus compositional patterns, may shift.

6. CONCLUSION

By carefully designing telecommunication features to perform certain common functions, it is possible to realize complex service logic through call-time composition of the constituent features. There is growing industry momentum for such mechanisms as seen in the IMS architecture and the SIP Servlet 1.1 standard.

7. ACKNOWLEDGMENTS

The author gratefully acknowledges his close extended collaboration with Greg Bond, Eric Cheung, Karrie Hanson, Don Henderson, Gerald Karam, Hal Purdy, Venkita Subramonian and Pamela Zave, out of which this work was produced. Alicia Abella consistently provides excellent advice. Laura Dillon also provided valuable suggestions which improved this paper. Finally, the anonymous reviewers provided excellent suggestions about related work.

8. REFERENCES

[1] Apache cocoon site. `http://cocoon.apache.org/`.

[2] V. Ambriola and G. Tortora. *Advances in Software Engineering and Knowledge Engineering.* World Scientific, 1993.

[3] F. Bachmann and L. Bass. Managing variability in software architectures. *SIGSOFT Softw. Eng. Notes*, 26(3):126–132, 2001.

[4] G. W. Bond, E. Cheung, H. H. Goguen, K. J. Hanson, D. Henderson, G. M. Karam, K. H. Purdy, T. M. Smith, and P. Zave. Experience with component-based development of a telecommunication service. In *Proceedings of the Eighth International Symposium on Component-Based Software Engineering*, pages 298–305. Springer-Verlag LNCS 3489, May 2005.

[5] E. Cheung and T. M. Smith. Experience with modularity in an advanced teleconferencing service deployment. In *ICSE Companion*, pages 39–49. IEEE, 2009.

[6] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *IEEE Softw.*, 15(6):37–45, 1998.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[8] J. J. Garrahan, P. A. Russo, K. Kitami, and R. Kung. Intelligent Network overview. *IEEE Communications*, 31(3):30–36, March 1993.

[9] N. D. Griffeth and H. Velthuijsen. The Negotiating Agents approach to runtime feature interaction resolution. In L. G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems*, pages 217–235. IOS Press, Amsterdam, 1994.

[10] M. Jackson and P. Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering*, XXIV(10):831–847, October 1998.

[11] *JSR 289: SIP Servlet Version 1.1.* Java Community Process, 2008. Available from: `http://jcp.org/en/jsr/detail?id=289`.

[12] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[13] D. Mennie and B. Pagurek. An architecture to support dynamic composition of service components, 2000.

[14] M. Plath and M. Ryan. Plug-and-play features. In K. Kimbler and L. G. Bouma, editors, *Feature Interactions in Telecommunications and Software Systems V*, pages 150–164. IOS Press, Amsterdam, 1998.

[15] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[16] W. Pree. *Design patterns for object-oriented software development.* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[17] C. Prehofer. Plug-and-play composition of features and feature interactions with Statechart diagrams. In D. Amyot and L. Logrippo, editors, *Feature Interactions in Telecommunications and Software Systems VII*, pages 43–58. IOS Press, Amsterdam, 2003.

[18] L. Rising, editor. *Design patterns in communications software.* Cambridge University Press, New York, NY, USA, 2001.

[19] F. Salm. Application servers and sip signaling in ims environments.

[20] D. C. Schmidt. Using design patterns to develop reusable object-oriented communication software. *Commun. ACM*, 38(10):65–74, 1995.

[21] R. Steenfeldt and H. Smith. Sip service execution rule language framework and requirements, November 2001.

[22] P. Zave. Address translation in telecommunication features. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 13(1):1–36, January 2004.

Specification and Evaluation of Transparent Behavior for SIP Back-to-Back User Agents

Gregory W. Bond
AT&T Labs—Research
Florham Park, NJ, USA
bond@research.att.com

Eric Cheung
AT&T Labs—Research
Florham Park, NJ, USA
cheung@research.att.com

Thomas M. Smith
AT&T Labs—Research
Florham Park, NJ, USA
tsmith@research.att.com

Pamela Zave
AT&T Labs—Research
Florham Park, NJ, USA
pamela@research.att.com

ABSTRACT

A back-to-back user agent (B2BUA) is a powerful mechanism for realizing complex SIP applications. The ability to create, terminate, and modify SIP dialogs allows the creation of arbitrarily complex services. However, B2BUAs must be designed with care so as not to disrupt service interoperability. A commonly-stated goal is for B2BUAs to be as *transparent* as possible while achieving its design goals. Though the notion of transparency is intuitively appealing, it is difficult to define. To address this issue, this paper proposes a definition of transparency and presents a formal model of a transparent B2BUA to serve as the specification of transparency. From this specification, we identify issues with both the realizability and desirability of this behavior, and suggest modifications to the original model. We evaluate the behavior of a number of public B2BUA implementations via testing, using some novel techniques to create test cases based on the formal models.

1. INTRODUCTION

A back-to-back user agent (B2BUA) is a powerful mechanism for realizing complex SIP applications. The ability to create, terminate, and modify SIP dialogs allows the creation of arbitrarily complex services. However, B2BUAs must be designed with care so as not to disrupt service interoperability.

A commonly-stated goal is for B2BUAs to be as *transparent* as possible while achieving its design goals. However, the notion of transparency is not defined by the SIP specification [13]. This specification defines a back-to-back user agent as "a logical entity that receives a request and processes it as a user agent server (UAS). In order to determine how the request should be answered, it acts as a user agent client (UAC) and generates requests." The specification further states that "Since it is a concatenation of a UAC and UAS, no explicit definitions are needed for its behavior."

To date, the behavior of B2BUAs has not been specified, other than that it must comply with the behavior of a UA on each side. This leads to the perception that B2BUAs break transparency of the network, and therefore hinders innovation at the endpoints. On the other hand, a large number of use cases have arisen in real-world deployments of SIP services that require B2BUAs. For example:

- Hide network topology information. This is often performed by Session Border Controllers (SBCs) that interface the networks of two service providers.
- Terminate an existing session, for example by a prepaid application when calling credit has run out, or by an IMS P-CSCF when it detects that the radio linkage with the mobile device has been disconnected.
- Modify the Session Description Protocol (SDP) information in the message body, for example to work through firewalls.
- Perform third party call control by advanced applications, for example to change a direct two-party call to a three-party call by bringing in a mixing media server.

The conflict between common usage and lack of specification is untenable. It is important that the behavior of B2BUAs is specified such that developers can implement them correctly and service providers can test them for compliance. At the same time, any innovations and extensions to the SIP protocol can be designed to work with transparent B2BUAs as intermediaries. While a transparent B2BUA does not provide any useful service, it can serve as the baseline and various B2BUAs that provide services can be defined as deviations from the transparent B2BUA. For example, a prepaid application B2BUA is transparent except when it terminates the session by sending BYE requests on both SIP dialogs.

In 2007, the IETF SIPPING working group started to work on a Best Current Practices document for a transparent B2BUA. Unfortunately, this work has not been continued to completion. The last draft [11] specifies how the Allow, Required and Supported headers should be handled when a request is received. It also specifies that when the

B2BUA relays a message certain headers should be generated, and the other headers and message body should be copied.

The SIP Servlet API standard [1] provides limited support for B2BUAs by providing methods for creating an outgoing request to be sent out on the UAC side based on an incoming request received on the UAS side. It specifies that the implementation must copy the headers from the incoming request to the outgoing request (with a few exceptions). However, the SIP Servlet API standard does not further specify any transparent B2BUA behavior.

The purpose of this paper is to provide a firmer foundation for B2BUAs in SIP by providing a rigorous and pragmatic specification of transparent behavior. This entails a number of contributions.

First, we show that it is difficult to define what "transparency" means, even on an informal and intuitive basis. After examining the alternatives, we settle on a pragmatic working definition (Section 2).

Second, we formalize our informal definition (Section 3). Message sequencing is formalized in terms of an executable model in Promela, the language of the Spin model checker. Message contents are described in terms of header values. To provide an environment in which the B2BUA model can be analyzed and verified automatically, we also developed new formal models of SIP user agents. Because of the use of model checking, all of the Promela models are guaranteed to be complete, consistent, unambiguous, and correct with respect to well-defined criteria.

Third, we demonstrate the pragmatic use of our specification by evaluating existing B2BUA implementations (Section 4). Because manually generated tests are not sufficient, we generated a suite of 2,408 tests automatically from the formal UA models. We then ran both manually and automatically generated tests, using automated testing tools, on the available implementations. None of the implementations comply fully with the B2BUA specification. This shows that implementing a correct B2BUA is difficult, and that comprehensive testing is necessary to ensure the correctness of implementations.

Overall, this research shows that judicious use of specification, analysis, and testing tools can greatly improve the quality of SIP components and SIP-based applications. Although our research needs to be extended in various ways, further work is amply justified by the initial results.

2. DEFINITION OF TRANSPARENT BEHAVIOR

What does it mean for a B2BUA to behave transparently? Transparency is an appealingly intuitive concept, but it is not easy to give it a rigorous definition.

In general, there are two approaches to definition. An *operational* definition of transparency would focus on the behavior of the B2BUA itself. An *observational* definition would define transparent behavior of a B2BUA as observed by its environment, which consists of the UAs at the far ends of its two dialogs. The advantage of an operational definition is that it is easy to tell whether a specific B2BUA satisfies the definition. One advantage of an observational definition is that it corresponds most closely to the intuitive notion of transparency. Another advantage is that it allows the most freedom in implementing B2BUAs.

In this paper, *pure propagation* refers to the following behavior of a B2BUA: receive a message in one dialog, and send it unchanged in the other dialog. A possible operational definition is, "A transparent B2BUA applies pure propagation to each received message, and does not send any messages that are not propagated."

Pure propagation cannot be correct transparent behavior because a B2BUA must change propagated messages in straightforward ways, such as modifying the `Call-ID` header and `tag` parameters to match the unique identifiers of each dialog. These changes to message content are specified in Section 3.2.

In this paper, *propagation* as a B2BUA behavior is the same as pure propagation, except with necessary header changes. A revised operational definition is, "A transparent B2BUA applies propagation to each received message, and does not send any messages that are not propagated." This definition does not work either, because it sometimes violates the SIP standard in the individual dialogs. Section 3.1.3 describes these situations.

Unfortunately, a rigorous observational definition is even harder to find than an operational definition. It could require that the presence of the B2BUA be undetectable by the far endpoints, but that is not achievable, even when real-time delays and header changes are excepted (see Section 3.1.3).

An observational definition should require that the media sessions between the far endpoints be the same whether there is a B2BUA present or not, because controlling media sessions is the primary purpose of SIP. To formalize this successfully, it would be necessary to define "the same" so that it generalizes over the nondeterministic behavior of the network between the far endpoints, which can affect endpoint behavior and media sessions even when there is no B2BUA. Also, a definition of transparency based on media sessions would be necessary but not sufficient—SIP signaling accomplishes more than just controlling media sessions.

In this paper we use a pragmatic definition of transparency that lies somewhere between the two extremes. A B2BUA is *transparent* if and only if:

- It acts as a standards-compliant UA in both dialogs.
- Its behavior within the two SIP dialogs is to propagate each message, and to not send any messages that are not propagated, except when this behavior would violate the protocol in either dialog.
- When its behavior is an exception to the basic rule, its behavior minimizes the effect of its presence between the far endpoint UAs.

The first two points are operational and precise. The third point is observational and rather vague.

We feel that this definition, despite its flaws, has enabled us to make progress toward understanding transparency. We regard it as an interim result, to be replaced in the future by a more precise observational definition.

3. SPECIFICATION OF TRANSPARENT BEHAVIOR

Our study covers the basic version of SIP defined in RFC 3261 [13], plus *info* [3] requests. *Info* requests allow application-level mid-call signaling without affecting dialog state, and are used extensively for PSTN–SIP interworking and media server control.

3.1 Message Sequencing

3.1.1 Method of Study

Message sequencing is an aspect of behavior. It is concerned with when a user agent can or must send a message, and what messages a user agent might receive at any given time. We study message sequencing by means of formal modeling and analysis.

In the sequencing view a message is identified primarily by a type, which is a member of an enumerated set. The request types within our scope are *invite, ack, cancel, info,* and *bye.* The possible responses to these requests are categorized in an enumerated set according to the level of detail needed. For example, the possible responses to an *info* request are categorized as *infoDVR* or *infoRsp.*

The *infoDVR* category consists of 408 (Request Timeout) and 481 (Call/Transaction Does Not Exist) messages in response to an *info.* The name stands for Dialog Vanished Response, because both of these indicate that the dialog is gone. The *infoRsp* category consists of all other responses, whether successful (200) or failing (3xx-6xx). In the models, there is a need to distinguish between DVR responses and all other responses, because they are handled differently by the models. There is no need to distinguish between successful responses and other failing responses, because (from the perspective of our models, see Section 3.1.2) they are simply passed to the user.

Secondarily, messages that can carry SDP are categorized as carrying *offer, answer,* or *none* in their SDP fields. All other aspects of message content are discussed in Section 3.2, and are not included in the sequencing models.

In a previous study [18], we used formal modeling in the Promela language and verification with the Spin model checker [7] to investigate invite dialogs in SIP. We wrote nondeterministic models documenting all possible behaviors of the two user agents (caller UA and callee UA) during an invite dialog. To validate the models with respect to the RFCs, we included pointers to those documents. We used a suite of formal analysis and verification techniques to ensure that the models were complete and consistent according to specific definitions of those terms. We also wrote a large number of in-line assertions expressing our assumptions and understanding of the protocol, and verified automatically that the model was correct according to those assertions. These validation and verification techniques are described in detail in [18].

Our study of B2BUAs builds on this previous work. First, we improved our UA models in various ways. The endpoint UAs are the environment of a B2BUA, so they must be understood as well as possible. Most importantly, we added UA failures as manifested by 408 and 481 messages.

The new models are described in Section 3.1.2, and are available on the Web [5]. Some readers may be surprised at their complexity—the original intent was for SIP to be a "simple" protocol, but simplicity is long gone, even for the basic version studied here. The important point is that, faced with this unavoidable complexity, we must take advantage of available technology such as model checking to help us deal with it.

Our specification of transparent behavior of a B2BUA also takes the form of a Promela model. Unlike the UA models, it is a deterministic program, prescribing exactly what the B2BUA should do in each circumstance. It has been subjected to all the same analysis and verification activities as the UA models. This means that it is guaranteed to be complete, consistent, and unambiguous. It is also guaranteed to preserve a large number of correctness assertions evaluated at control points within the UA and B2BUA code.

The B2BUA model (in two versions) is described in Sections 3.1.3 and 3.1.4, and is available on the Web [5].

3.1.2 The User-Agent Models

We assume that message delivery is reliable and FIFO in each direction, because without this assumption a number of significant new problems arise [18].

The UA models are more complete with respect to RFC 3261 than our previous models. They include early media, 408 and 481 messages, and timeouts in the callee UA waiting for an *ack* to a successful initial *invite.* In the modeled behavior, failure of one UA is detected when the failed UA does not respond to a request from the live UA. Simultaneous failure of both UAs is not represented, however.

Because our primary goal is to help people program B2BUAs, SIP is modeled from the viewpoint of the *transaction user* in RFC 3261. According to RFC 3261, 100 (Trying) messages, retransmissions, and acknowledgments after *invite* failures are all handled exclusively by a lower-level *transaction* layer of the protocol stack. This means that they need not be present in our models.

As mentioned previously, the UA models are highly nondeterministic. There are four major causes of nondeterminism. First, nondeterminism can reflect user choice. For example, after sending an initial *invite*, a caller UA can choose to send a *cancel* message or wait for the response to the *invite.* Second, nondeterminism can represent the possibility of failure. Whenever a UA is due to respond to a request, the UA model can send the request or else fail. Third, nondeterminism can reflect concurrency. The two UAs and message channels between them are distributed and largely independent, so their events can be interleaved in arbitrary ways. Fourth, nondeterminism can reflect implementation freedom. For example, on receiving a *cancel* message when it has not yet responded to the initial *invite*, a callee UA must send both a 200 response to the *cancel* and a failure response to the *invite.* The order is not specified, however, so the model has a nondeterministic choice between the two orders.

We have made every effort to read RFC 3261 closely and interpret it correctly, but this is difficult to do because the RFC is informal, incomplete, and vague in many places. Our formal models have precise semantics and are guaranteed to be complete; they are organized so that a specific answer to a specific question is always easy to find. With the help of the SIP community they can be improved until they are declared correct by consensus, at which time they can serve as valuable appendices to the RFCs.

In the remainder of this section we discuss some specific aspects of UA behavior that are important for B2BUA behavior.

During a confirmed dialog, either UA can send an *invite* message to alter the session description (specification of the media channels). Because there is only supposed to be one such re-invite transaction at a time, a *re-invite race* occurs if both UAs re-invite at about the same time.

A typical re-invite race is shown in Figure 1. Each UA knows there is a race as soon as it receives *invite* after sending *invite*. Each UA responds with *inv491* (a 491 message in response to an *invite*), so that both re-invite requests fail. Although each UA is free to try again at a later (and different) time, our models do not show any relationship between the earlier and later re-invites.

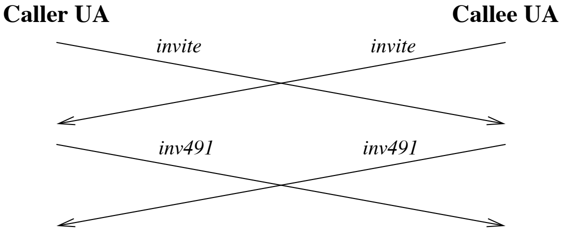


Figure 1: A re-invite race.

On receiving any *invite* (initial or re-invite) message, a UA need not respond immediately. This provides time for the UA to get instructions from a human user if necessary. In the models, a UA receiving an *invite* goes into an *invited* or *reInvited* state. In these states the UA can send or receive other messages. At any time, however, it has the choice to send a final response to the *invite*.

An invite transaction can take two forms with respect to the offer/answer exchange [12]. These two forms are illustrated in Figure 2 by re-invites from the callee UA. On the left, the *invite* message carries an offer and the *inv200* carries an answer. On the right, the *invite* message does not carry an offer, but rather solicits an offer from the other UA. In this form the *inv200* carries the offer, and the *ack* message carries the answer.

On the left, the caller UA leaves the *reInvited* state after sending *inv200*, even though it has not yet received the *ack*. Because the offer/answer exchange is complete, *even before receiving the ack* it can send a new *invite* message to begin a new re-invite transaction [12].

Any time after sending the initial *invite* and before receiving a final response to it, a caller UA can send *cancel* to cancel the transaction and abort the dialog. A *cancel race* occurs if the *cancel* message arrives at the callee UA after the callee UA has sent a final, successful response to the *invite*.

A typical cancel race is shown in Figure 3. The caller UA knows there is a race as soon as it receives *inv200* (a 200 message in response to an *invite*) after sending *cancel*. Having failed to cancel the initial transaction, it ends the dialog by sending a *bye* instead. Later it receives *canc200* sent by the callee UA.

For all requests, a *DVR* response in the model corresponds to either a 408 or 481 response. In the models, a failing UA sends a DVR response and then enters a state in which it no longer communicates except to send additional DVR responses.

This modeled behavior corresponds quite closely to the actual behavior of a UA that fails and restarts, having lost dialog state. The restarted UA will respond to all subsequent requests for that dialog with 481 messages.

The modeled behavior corresponds more loosely to the actual behavior of a UA that fails and does not restart. In this case, obviously, the dead UA does not send any messages. Rather, the transaction layer of the live UA generates a 408 response for the transaction user to see. Thus a UA's sending 408 messages at and after failure is a modeling trick ensuring that one UA gets 408 responses when and only when the other UA has failed.

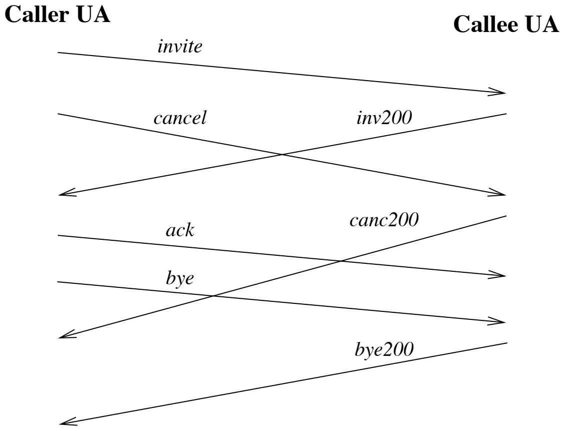


Figure 3: A cancel race.

On receiving a DVR response, a UA that has not already sent a *bye* is supposed to send a *bye*. This causes two difficulties in the callee UA. First, the callee UA can receive a 408 or 481 response to an *info* message when it is still in the *invited* state and cannot legally send a *bye*. In this case we have the callee UA send a failure response to the initial *invite*.

Second, the callee UA can receive these responses to *info* requests when it has already sent an *inv200* for the initial *invite* but has not yet received the corresponding *ack*. It cannot legally send a *bye* in this case, either. It becomes blocked until it receives an *ack* or *ack* timeout, at which time it sends the *bye*.

Modeling reveals that a queue of messages in transit from one UA to the other can grow to size 7 (even though the model allows only one provisional response and only one outstanding *info* request). In this unusual scenario, one UA generates the message sequence *inv200, invite, info, bye* and then is suspended for a long interval. During this interval the other UA receives the 4 messages and processes them to generate the following sequence: *ack, inv200, infoRsp, info, invite, bye, bye200.* Of these 7 messages, 4 are responses to the 4 queued messages, and 3 are new requests.

3.1.3 The Back-to-Back User Agent Model

Our model of a back-to-back user agent is a deterministic Promela program that acts as a callee UA in one dialog and a caller UA in another. It is proposed as a specification of correct transparent behavior.

Whenever possible, the transparent B2BUA reacts to receiving a message from one dialog simply by propagating it. The remainder of this section discusses the situations in which this is not possible, and how the B2BUA can deal with the situation safely.

A typical re-invite race is shown in Figure 4. When the B2BUA receives an *invite* from the right, it cannot forward

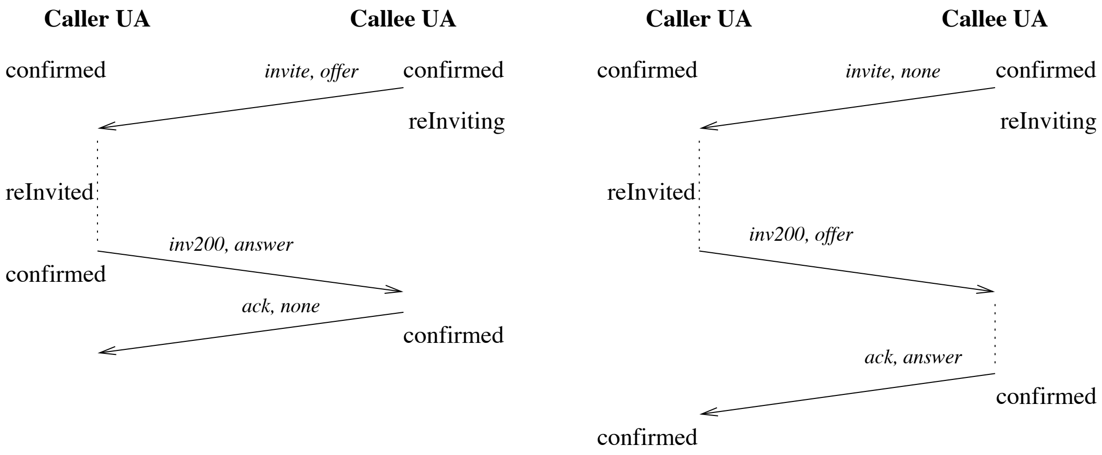


Figure 2: Two ways the callee UA can re-invite, with local states of the UAs shown. These transactions can also be initiated symmetrically by the caller UA.

it to the left, because it would violate the SIP protocol in the leftmost dialog by knowingly creating a re-invite race. Because it knows that there is a race, it generates an *inv491* response instead. After that point it can resume propagation of messages.

As a result of the presence of the B2BUA, the endpoint caller UA on the left receives *inv491* without having received a racing *invite*—something that could never happen in a simple dialog. This illustrates the point that "transparent behavior" cannot be defined as "undetectable behavior."

In Figure 4 the B2BUA absorbs a request (rather than propagating it) and generates its own response to the request. This deviation is benign for two reasons: (1) if the B2BUA had propagated the request, the request would not have changed the state of the UA that received it; and (2) both endpoint UAs receive the same responses to their requests as they would have received without the B2BUA.

A B2BUA can never simply propagate *cancel* requests, because *cancel* requests are "hop-by-hop". On receiving a *cancel* from the left, a B2BUA must immediately generate a response in the leftmost dialog. If the B2BUA also sends a *cancel* to the right and receives a response from the right, then the B2BUA must absorb the response. This behavior is shown in Figure 5.

In one form of cancel race, *cancel* and *inv200* messages cross in the left dialog. This means that the B2BUA receives the *cancel* after it has already propagated *inv200* to the left, so that the dialog on the left of the B2BUA looks like Figure 3. There is no point to propagating *cancel* to the right, and it would also be illegal to do so because the dialog to the right has been confirmed. The B2BUA must simply absorb the *cancel* and generate *canc200* as a response.

Cancel races detected on the right of the B2BUA do not require a transparent B2BUA to behave differently than in Figure 5. If the *cancel* arrives at the callee UA too late, then the calle UA may have already sent *inv200.* As with *invFail* in Figure 5, the B2BUA simply propagates *inv200.*

The B2BUA can receive a request (for example a re-invite) from a dialog after it has received a *bye* from the other dialog and propagated it to the requesting dialog, as shown in Figure 6. If the B2BUA were to propagate the new request (in Figure 6, to the right), it would be sending a new request in a dialog that has already seen a *bye.* In our opinion this is clearly wrong and should be illegal, although we cannot find a specific prohibition in RFC 3261. Instead of propagating the *invite*, the B2BUA should absorb it and generate *invFail.* If the B2BUA had propagated the *invite*, it would have had no effect on the state of the callee UA.

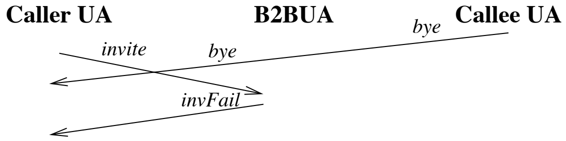


Figure 6: A late request arrives at a B2BUA.

Similarly, the B2BUA cannot propagate a provisional response by sending it in a dialog that has already seen a *bye.* In this case the B2BUA absorbs the message without generating any other message.

3.1.4 A Modified B2BUA

The pure B2BUA described in Section 3.1.3 propagates messages whenever propagation is legal. Unfortunately, it is a specification that cannot be implemented in a SIP Servlet container. The reason is that the SIP Servlet standard [1] mandates handling *cancel* requests in a different and less transparent way.

Because of the importance of the SIP Servlet containers as platforms for SIP applications, we provide a modified model to serve as an alternative specification of a transparent B2BUA. The modified specification is compatible with the SIP Servlet standard.

Figure 7 shows how the modified B2BUA handles a *cancel.* Provided that the *cancel* is not too late in arriving at the B2BUA (see Figure 3), the B2BUA immediately generates *invFail* to the left, ending that dialog. It also sends the *cancel* to the right.

In the scenario shown in Figure 7, there is a cancel race on the right, so that the B2BUA receives *inv200* from the right and generates *bye* to the right. If there were no race, it would receive *invFail* from the right and simply absorb it.

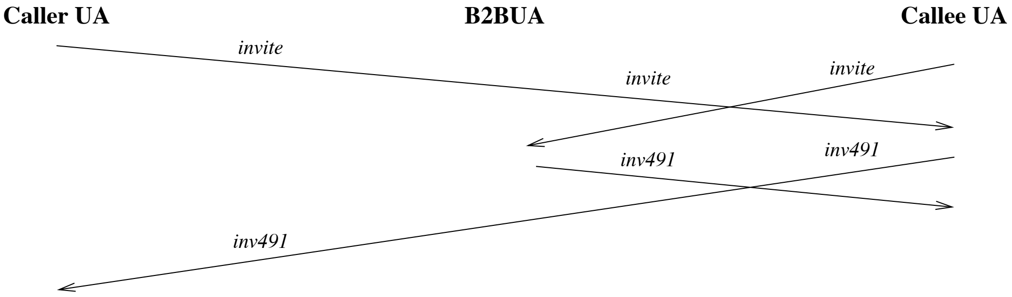


Figure 4: A re-invite race in the presence of a B2BUA.

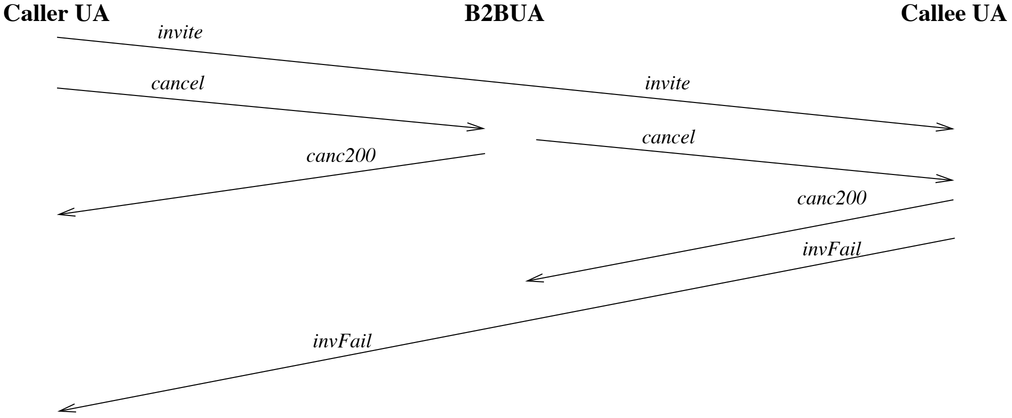


Figure 5: Canceling in the presence of a B2BUA.

When the modified B2BUA generates *invFail*, it creates a situation in which its two dialogs are in different and incompatible states. From that point on, there is *no* message propagation, and the B2BUA handles the two dialogs separately.

Although the modified B2BUA does not satisfy our interim definition of transparency, it has other advantages, such as responding faster overall to a *cancel* request. This points to a potential benefit of finding a better definition of transparency. If the definition were more observational, it would allow more freedom in the implementation of B2BUAs. This would give application and platform developers the room to design better B2BUAs, with improved efficiency and possibly other desirable properties.

3.2 Message Content

In order to achieve transparency, the message *content*, as well as sequence, must be preserved. Message content includes the *headers* of a SIP message as well as the *body*. A minimal number of headers are mandated by RFC 3261; other headers are specified in a variety of RFCs. Finally, it is always possible for a SIP UA to include so-called *private* or *extension headers*. Message bodies convey information in a wide variety of contexts, for example, descriptions of media connectivity, conveyed via SDP; carriage of instant messages (IMs); or carriage of commands and associated responses between a UA and a media server.

As discussed in Section 1, [11] begins to address the issues of transparency with respect to message contents. The recommendations in this section generally accord with that document; however that document also discusses issues that are outside the scope of this paper.

For correct propagation of the message, the body must be copied from the incoming message to the outgoing message. Furthermore, with the exceptions noted below, all headers in the incoming message should be copied to the outgoing message.

Part or all of three headers are used to provide a unique dialog identifier: the value of the `Call-ID` header along with the values of the `tag` parameter of the `From` and `To` headers. Due to requirements for global uniqueness, these values cannot be re-used in a new dialog; the B2BUA must generate its own unique values.

The `Via` and `Contact` headers are used for hop-by-hop message routing, and thus should not be copied. Similarly, if the topmost `Route` header in an incoming request targets the B2BUA, it should not be copied. The `Record-Route` header applies to a dialog; since the B2BUA terminates two dialogs, it is responsible for adhering to any routing requirements of this header in the two dialogs, but the header should not be copied between dialogs.

The B2BUA must inspect `Allow`, `Supported`, and `Required` headers and modify them accordingly to reflect the capabilities of the B2BUA. The `Max-Forwards` header is used to detect routing loops. If the value of the header in an incoming request is greater than 0, the B2BUA should decrement the value of the header by 1 for the propagated request; otherwise the B2BUA should reject the request.

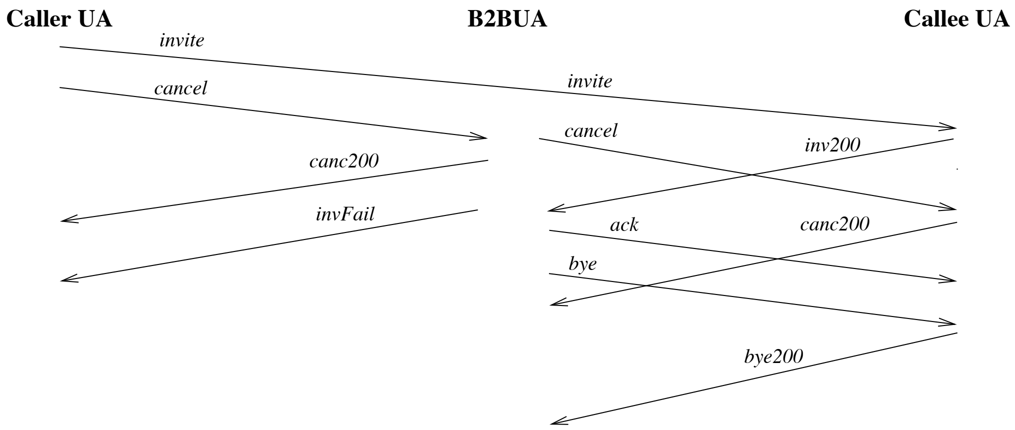


Figure 7: A cancel race in the presence of a modified B2BUA.

4. EVALUATION OF IMPLEMENTATIONS

After we completed the specification of the correct behavior of a transparent B2BUA and wrote a Promela program to formally model the behavior, we undertook the task of testing existing B2BUA implementations to evaluate if they comply with the specification.

4.1 Systems Under Test

Our evaluation is restricted to implementations that (1) are built on the SIP Servlet API, because it is the dominant standard for SIP application development; and (2) have source code freely available for inspection and use, so that users of these implementations may make use of these results to make any necessary correction to the source code if desired.

The SIP Servlet API provides limited support for B2BUA applications in the form of the `B2buaHelper` class with several convenience methods. A B2BUA application can use this class to manage linkage of its two dialogs. As well, upon receiving a request on the first dialog, the application can call one of the convenience methods to create an outgoing request on the second dialog. The SIP Servlet container is responsible for modifying and copying various headers correctly. The SIP Servlet container also handles certain requests on behalf of the application, for example the *cancel* request. Therefore, a B2BUA implemented using the SIP Servlet API relies on both correct application programming and correct container behavior.

The B2BUA implementations we evaluated are listed below:

B2bTerminator (BT) This is a complete example application to illustrate the use of `B2buaHelper` given in [2]. This application behaves transparently except it tears down the call after a certain time. We test this application as a transparent B2BUA by setting a very large timeout value.[1]

ECharts for SIP Servlets (E4SS) E4SS is an open source framework that allows the use of the finite state machine paradigm to program SIP servlets at a higher level of abstraction [16]. It also includes reusable features, amongst them a transparent B2BUA application called `B2buaSafe`. The version tested is SVN version 1578.

SailFin Converged Application Framework (CAFE) This is another open source framework that provides a higher level of programming abstraction for SIP applications [15]. By default, a CAFE application acts as a transparent B2BUA. The programmer can override the transparent behavior at different events to implement the specific logic of the application. The version tested is sailfin-cafe-v1-b24.

The containers we evaluated are SailFin [14] version sailfin-v2-b31g and Oracle Communication Converged Application Server (OCCAS) version 4.0. BT and E4SS can be deployed and tested on both containers. CAFE currently only supports the SailFin container and thus is not tested on OCCAS. Thus in total there are five systems under test (SUTs): BT/SailFin, BT/OCCAS, E4SS/SailFin, E4SS/OCCAS, and CAFE/SailFin.

4.2 Manual Test Generation

We first utilized KitCAT [17] to test the above B2BUA implementations. KitCAT is a test tool for performing functional testing of converged (SIP and HTTP) applications. For this testing, KitCAT acts as both the caller and callee user agents. Drawing on experience at the SIP Interoperability Test events and call flow documents [9, 6], we wrote 12 test cases including race conditions where the two endpoints send messages at the same time (e.g. *cancel* and *inv200*, *bye* and *bye*). The test cases include assertions to check that the SUT sends the correct messages and that the message headers and contents are passed correctly according to Sections 3.1 and 3.2 respectively. The test results are discussed in Section 4.4.1.

However, writing these KitCAT test programs manually proved to be time-consuming. Moreover, KitCAT imposes a call state machine on its test agents which precludes the generation of certain scenarios such as the re-invite race shown in Figure 1. We concluded that we require a lower level test tool for this kind of protocol testing, and also automatically generated tests for better coverage. This approach is

[1] This code change, together with changes necessary for BT to run on OCCAS, is available at [5].

discussed in the following section.

4.3 Model-Based Test Generation

Given the complexity of the SIP protocol, and the possible interactions that may occur between agents and a B2BUA, one might infer that the universe of possible behaviors is very large indeed. Verification confirms this: the Spin model-checker discovers 48,966,575 unique states for our combined agent-B2BUA model. In the context of testing, this immense state space indicates that a hand-crafted test suite of 10, 20, or even 100 tests cannot possibly provide adequate coverage. The testing challenge then is to improve upon what can achieved by hand-crafting a test suite.

The approach we've chosen is to generate tests using the same model we use for verification. One advantage of this approach is that generated tests are guaranteed to conform to behaviors specified by the model. Another advantage is that it is possible, in principle, to generate a test suite that satisfies a notion of complete coverage. However, as we have seen, the tremendous size of the state space makes this latter goal impractical for any obvious notion of completeness. For this reason we've identified a series of test criteria that allow us to intuitively partition the universe of possible tests into practically sized test suites. For each test we identify:

- the length: the total number of messages sent or received by the user agents – the greater the number of messages sent, the more complex the interaction is between agents;
- the maximum queue size: the maximum number of messages that are enqueued at any time on the agent and B2BUA queues – more enqueued messages corresponds to greater channel latency or scarcity of processing resources
- the "weather profile" which is determined by the messages present in the test:
    - a "sunny day" test excludes *invFail*, *cancel*, *DVR*, and *ackTimeout* messages;
    - a "cloudy day" test includes at least one *invFail* or *cancel* message but no *DVR* or *ackTimeout* messages;
    - a "stormy day" test includes at least one *DVR* or *ackTimeout* message.

We can now identify tests that meet a particular criteria, for example: "all" sunny day tests with queue size 1 and length less than or equal to 12 (the meaning of the "all" will be qualified shortly). This test suite would correspond to moderately complex but normal behavioral interaction between agents via a B2BUA.

We use a two phase approach to automatically generate tests as shown in Figure 8. The first phase generates "test traces" from the model. A test trace is a high-level symbolic representation of a sequence of messages sent or received by the user agents via the B2BUA. The second phase translates a set of test traces to an executable test suite. Each executable test ensures that messages are sent and received in a timely fashion and in the expected order.

An example of a test trace is:

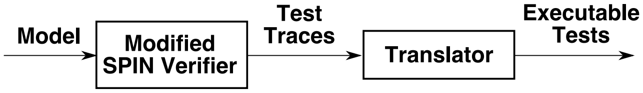


Figure 8: Two phase model-based test generation.

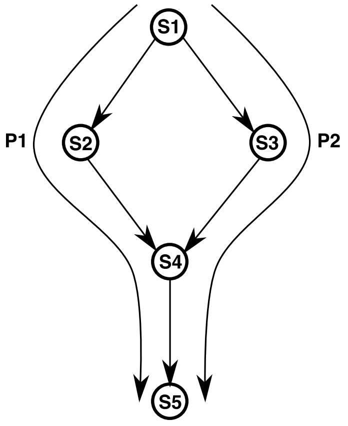


Figure 9: A model's states and paths.

```
8
(caller)[out!invite,none]:(callee)[in?invite,sdp]:
(callee)[out!inv200,offer]:(caller)[in?inv200,sdp]:
(caller)[out!bye,none]:(callee)[in?bye,sdp]:
(callee)[out!bye200,none]:(caller)[in?bye200,sdp]:
1:1:1:ALL:1
```

where the first number indicates the test length, the final five colon-delimited fields indicate the individual and overall maximum queue lengths for the agents and the B2BUA (here "ALL" indicates that both agents and the B2BUA had the same maximum queue size of 1) and the remaining colon-delimited fields represent messages sent or received by the agents. Thus, a test trace contains all the information required for selecting a test that meets the criteria described in the previous section.

Since the Spin model checker parses our model and traverses its state space to perform verification we chose to harness this machinery in order to generate tests. However, trace-based test generation involves recording the paths interconnecting a model's states but, being a model checker, Spin endeavors only to visit all of a model's states. Considering the example model shown in Figure 9, Spin would visit states S1 through S5 of the model shown. However, utilizing its depth-first search algorithm, Spin would completely traverse path P1 or P2 but it wouldn't completely traverse both. This is because state S4 is common to both paths so the second path would be truncated when the depth first algorithm arrives at state S4 a second time. To bridge the divide between state and path traversal we augmented our model and Spin's verifier. The agent model is augmented to maintain a record of messages sent and received by the agents. This way, each reachable state of the augmented model will include a record of the path traversed to reach the state. Spin's depth-first verifier algorithm is augmented to output a complete path (a path from the initial state to a valid end state) when it encounters one.

It was also necessary to refine the modified B2BUA model presented in Section 3.1.4 in order to exclude tests that reflected unachievable container behavior. Containers normally serve requests in FIFO order but, using a B2BUA model that dedicates a separate request queue to each agent, tests were generated that required a container to serve one agent's requests while unfairly neglecting the other agent's requests. To eliminate this unfair behavior we replaced the B2BUA's two input queues with a single queue that is shared by the two agents. This modification ensures that the B2BUA serves requests in FIFO order the same way a container does.

Spin supports limiting a search to a pre-defined depth, where depth is defined in terms of the number of transitions traversed between model states. We use this depth limiting facility to limit the number of generated tests. For example, for a depth limit of 71 we generate 361,737 unique tests ranging from length 4 to 17, and overall maximum queue size of 5. Generating a test set this way is memory and CPU intensive. For example, generating the aforementioned tests required 105 GB of preallocated RAM and 40 CPU minutes of a single 2.40 GHz Intel Xeon processor core.

Given the enormous number of generated tests, one might expect reasonable coverage of system behavior. To confirm this intuition we inspected the tests for instances of example call flows. We confirmed that there were many representative test cases corresponding to the hand-crafted test suite described in Section 4.2. We also identified the four call flows from the "Example Call Flows of Race Conditions" RFC [6] that conform to the scope of our model and confirmed that representative tests existed for each.

Although inspection of the generated tests cases reveals excellent test coverage, inspection also reveals that the set of generated tests is not complete for the specified depth limit. For example, inspection of our tests reveals that not all message interleavings are present for all call flows. We assume that the underlying reason is that Spin's verification algorithm is not designed for traversing all paths of a model, rather it is designed for traversing all states of the model. Determining which particular aspect of the verification algorithm is responsible for compromising completeness is something we are currently investigating.

Figure 10 shows the test application architecture. The test driver is responsible for administering the generated tests and recording test results. We use the JUnit unit test framework [10] for executing a test suite and reporting test results. In addition to JUnit the tests also use the ECharts for JAIN-SIP (E4JS) API for sending and receiving SIP messages. JAIN-SIP [8] provides a transaction-user API for SIP and E4JS [4] is an abstraction layer on top of JAIN-SIP that provides facilities for managing multiple agents, in our case a caller and callee agent, sharing a SIP stack instance. The system under test is a B2BUA SIP Servlet application running in a SIP Servlet container.

4.4 Test Evaluation

The following presents the results of applying the manually and automatically generated tests to the systems under test.

4.4.1 Results of Manually Generated Tests

Table 1 shows the results of using KitCAT and handcrafted test cases to test the five SUTs listed in Section 4.1.

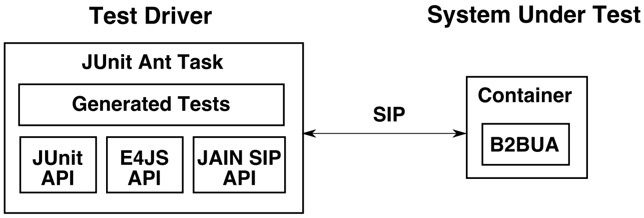


Figure 10: The test application architecture.

The results reveal two problems related to message sequencing. First, when faced with the cancel race shown in Figure 7, BT and CAFE do not send *bye* to terminate the right dialog even though the left dialog has been terminated.

Second, in the scenario where callee receives re-*invite*, sends *inv200*, but before receiving *ack* the callee sends *bye*, all three SUTs on SailFin fail. This reveals a bug in the SailFin implementation where it throws an exception if the application attempts to send a mid-dialog request before receiving the *ack* request, even though this is allowed by [13].

In terms of message content transparency, BT and CAFE rely on `B2buaHelper` class to create outgoing requests based on incoming requests. OCCAS copies unknown extension headers in this operation, but SailFin does not. However, in forwarding responses the application must copy unknown headers. E4SS uses its own code to copy headers from incoming to outgoing messages. However, this testing reveals a bug in the E4SS implementation where headers in the *ack* request are not copied.

4.4.2 Results of Automatically Generated Tests

We used only the OCCAS container for evaluating B2BUAs with automatically generated tests. This is because of the SailFin bug uncovered using manual testing described in the previous section. Since we did not use SailFin then we could not test CAFE. Thus in total there are two SUTs for testing with automatically generated tests: BT/OCCAS, E4SS/OCCAS. Table 2 shows the results of our testing.

Of the over 360,000 tests generated, we used the criteria described in Section 4.3 to select a manageable test suite of 2,408 tests: 257 sunny day, 1,335 cloudy day and 816 stormy day. We confirmed that these tests included the scenarios covered by our manually generated tests. Furthermore, we made sure that the cloudy day tests included *cancel*/*invite200* races, re-*invite* races, and common failure scenarios. In general, we selected tests with a maximum queue size of 1 except for cases that required queue sizes of 2, such as in some *cancel* scenarios where two response messages can be sent in a row by an agent. Using a small queue size represents normal environmental conditions, with minimal channel latency and minimal competition for processing resources. Test length for *cancel* scenarios and stormy day scenarios were limited to length 11 and 13, respectively.

The test results, shown in Table 2, reveal failures in both the B2BUA applications and in the underlying OCCAS container (container failures are indicated by a * superscript).

For the cloudy day tests, neither SUT was capable of negotiating the complexities of certain re-*invite* races. Testing also uncovered the same problem with BT that we uncovered using manual testing, namely the inability to properly handle a *cancel*/*inv200* race. The E4SS B2BUA does not

SUT	Message Sequence		Message Content
	Tests passed	Failed cases	
BT/OCCAS	11/12	*cancel* and *inv200* race	Partially fail: responses
BT/SailFin	10/12	*cancel* and *inv200* race Callee sends *bye* before receiving *ack*	Fail: requests, responses
E4SS/OCCAS	12/12		Partially fail: *ack*
E4SS/SailFin	11/12	Callee sends *bye* before receiving *ack*	Partially fail: *ack*
CAFE/SailFin	10/12	*cancel* and *inv200* race Callee sends *bye* before receiving *ack*	Fail - requests, responses

Table 1: Results of Manually Generated Tests

Category	SUT			
	BT/OCCAS		E4SS/OCCAS	
	Tests passed	Failed cases	Tests passed	Failed cases
Sunny Day	257/257		257/257	
Cloudy Day	830/1,335	56 re-*invite* race 98 *cancel*/*inv200* race 217 request after *bye* 134 504s after dialog terminated*	888/1,335	56 re-*invite* race 40 outstanding requests after *invFail* 217 request after *bye* 134 504s after dialog terminated*
Stormy Day	568/816	32 *canc200* instead of *cancDVR** 196 *cancel*/*inv200* race 15 *bye* after DVR* 5 create final response after DVR	760/816	32 *canc200* instead of *cancDVR** 5 outstanding requests after *invFail* 15 *bye* after DVR* 4 *bye* after *ackTimeout*

Table 2: Results of Automatic Testing

propagate responses to outstanding requests after receving an *invFail*. Both B2BUAs continue to propagate requests after receiving a bye. Finally, the tests revealed an OCCAS container bug, where the container sends 504 responses to outstanding requests after a dialog has terminated.

For the stormy day tests we discovered that OCCAS prevents sending a *bye* after receiving a *DVR* response, even though sending a *bye* is specified by RFC 3261. As for the cloudy day tests, E4SS failed to propagate responses to outstanding requests after receiving an *invFail* and BT failed to handle *cancel*/*inv200* races. Another OCCAS container problem is that it would sometimes send a *canc200* instead of the expected *cancDVR* in some DVR scenarios. BT fails to propagate a message because one of BT's SipSessions no longer exists. It isn't clear if this is due to a bug in BT, the SIP Servlet specification or the OCCAS container. Finally, E4SS failed to propagate *bye* messages after an *ackTimeout* event.

4.4.3 Discussion of Results

Our testing, using both manually and automatically generated tests, reveals problems with every application and container we looked at. From this we conclude that implementing a correct B2BUA is difficult and, moreover, that comprehensive testing is necessary to validate B2BUA behavior. Our results support efforts like SailFin CAFE and E4SS whose goals include providing a reusable, correctly implemented B2BUA that hides the inherent complexity from the programmer. Furthermore, our results indicate that comprehensive application-level testing supports validating container behavior and reveals ambiguities in the SIP Servlet specification. Finally, our results support our approach to model-based test generation. Not only do our automatically generated tests uncover the same B2BUA failures that our hand-crafted tests do, but they also uncover new failures resulting from unusual stormy day call flows.

5. DISCUSSION AND FUTURE WORK

SIP is becoming increasingly important as the dominant protocol for IP-based telecommunications and multimedia systems. The specification of SIP is informal and in some places incomplete, inconsistent, or ambiguous. SIP is complex already and its complexity is increasing, as the protocol is extended for a variety of reasons.

This study and our previous work [18] show that this situation is both dangerous and unnecessary. With judicious use of formal specification and automated analysis, the SIP protocol can be documented in a way that is guaranteed complete, consistent, unambiguous, and correct with respect to a variety of assertions. Critical SIP components such as B2BUAs can be defined with an equivalent level of quality. These models can be exploited to generate large, comprehensive test suites for real implementations. Given the number of bugs and other problems that our work has uncovered, it is safe to say that important goals such as interoperability and reliability cannot be achieved without formal methods.

Important future work is to continue to extend the scope of the model such that commonly used extensions to the SIP protocol are included.

The B2BUA models presented here prescribe deterministic behavior. However, in some cases, we made a choice from multiple legal alternatives. For example, in the cancel race of Figure 7, the B2BUA sends *ack* before *bye*, even though it is legal to send the *bye* without sending a previous *ack*. Further study is required in order to determine the criteria used to resolve such ambiguous situations.

It is our intention to expand the scope of our testing to include tests with greater lengths and maximum queue sizes. The machine we use for generating traces has 128 GB of RAM which limits us to a Spin verification depth limit of 71. Using the current model this results in a maximum test length of 17. By simplifying the model, for example, by constraining certain transition sequences to execute atomically, we should be able to greatly reduce the state space without compromising completeness, thereby permitting deeper searches and generation of longer test traces.

Another challenge we faced was analyzing test results. While our test platform unambigiously indicates how many tests pass and how many fail, it does not provide any insight into why tests fail. To do this we resorted to manually inspecting failure signatures extracted from log files, their associated test cases and the associated application code. Naturally, this becomes tedious and error-prone as the number of failure cases increases. This process would benefit from automated post-processing where similar failure signatures could be grouped thereby reducing the number of failures requiring investigation.

A goal of the SIP Servlet specification is to simplify life for the application developer. To this end, a SIP Servlet container presents an abstraction of the SIP stack to the programmer. This abstraction intersects with that of a SIP transaction-user but, in some cases, also presents a higher-level abstraction. The problem, as revealed by our testing, is that this abstraction is incompletely specified and has led container vendors to make their own, independent implementation decisions without fully understanding their implications. The result is that the SIP Servlet standard, whose goal is to support interoperability of applications across containers, does not achieve that goal. To address this situation, a topic for future research is to formally specify SIP Servlet container behavior and integrate the resulting model with our B2BUA models.

6. ACKNOWLEDGMENTS

We gratefully acknowledge Kristoffer Gronowski for supplying the BT example code, as well as the SailFin CAFE team for their publicly-available B2BUA implementation. Finally, we acknowledge, with gratitude and fondness, the great contributions and lasting memories of our late colleague Venkita Subramonian.

7. REFERENCES

[1] BEA. SIP Servlet API version 1.1, 2008. Java Community Process JSR 289. http://jcp.org/en/jsr/detail?id=289.

[2] C. Boulton and K. Gronowski. *Understanding SIP Servlets 1.1.* Artech House, April 2009.

[3] S. Donovan. The SIP INFO method, October 2000. IETF RFC 2976.

[4] ECharts for JAIN SIP (E4JS). http://echarts.org/.

[5] Formal models of SIP, 2010. `http://www.research.att.com/~pamela/sip.html`.

[6] M. Hasebe, J. Koshiko, Y. Suzuki, T. Yoshikawa, and P. Kyzivat. Example call flows of race conditions in the session initiation protocol (SIP). IETF RFC 5407, December 2008.

[7] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

[8] *JAIN(tm) SIP Specification*. Java Community Process, 2003. Available from: `http://jcp.org/aboutJava/communityprocess/final/jsr032/`.

[9] A. Johnston, S. Donovan, R. Sparks, C. Cunningham, and K. Summers. Session Initiation Protocol (SIP) basic call flow examples. IETF RFC 3665, December 2003.

[10] JUnit. http://www.junit.org/.

[11] X. Marjou, I. Elz, and P. Musgrave. Best current practices for a session initiation protocol (SIP) transparent back-to-back user-agent (B2BUA). IETF Internet-Draft draft-marjou-sipping-b2bua-01, July 2007.

[12] J. Rosenberg and H. Schulzrinne. An offer/answer model with the session description protocol (SDP), June 2002. IETF RFC 3264.

[13] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol, June 2002. IETF RFC 3261.

[14] Project SailFin. https://sailfin.dev.java.net/.

[15] SailFin CAFE project. https://sailfin-cafe.dev.java.net/.

[16] T. M. Smith and G. W. Bond. ECharts for SIP Servlets: a state-machine programming environment for VoIP applications. In *IPTComm '07: Proceedings of the 1st International Conference on Principles, Systems and Applications of IP telecommunications*, pages 89–98. ACM, 2007.

[17] V. Subramonian. Towards automated functional testing of converged applications. In *IPTComm '09: Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, pages 1–12, New York, NY, USA, 2009. ACM.

[18] P. Zave. Understanding SIP through model-checking. In *Proceedings of the Second International Conference on Principles, Systems and Applications of IP Telecommunications*, pages 256–279. Springer-Verlag LNCS 5310, 2008.

The Impact of TLS on SIP Server Performance

Charles Shen† Erich Nahum‡ Henning Schulzrinne† Charles Wright‡

†Department of Computer Science, Columbia University
New York, NY 10027, USA
{charles,hgs}@cs.columbia.edu

‡IBM T.J. Watson Research Center
Hawthorne, NY 10532, USA
{nahum,cpwright}@us.ibm.com

ABSTRACT

Securing VoIP is a crucial requirement for its successful adoption. A key component of this is securing the signaling path, which is performed by SIP. Securing SIP is accomplished by using TLS instead of UDP as the transport protocol. However, using TLS for SIP is not yet widespread, perhaps due to concerns about the performance overhead.

This paper studies the performance impact of using TLS as a transport protocol for SIP servers. We evaluate the cost of TLS experimentally using a testbed with OpenSIPS, OpenSSL, and Linux running on an Intel-based server. We analyze TLS costs using application, library, and kernel profiling, and use the profiles to illustrate when and how different costs are incurred, such as bulk data encryption, public key encryption, private key decryption, and MAC-based verification.

We show that using TLS can reduce performance by up to a factor of 17 compared to the typical case of SIP-over-UDP. The primary factor in determining performance is whether and how TLS connection establishment is performed, due to the heavy costs of RSA operations used for session negotiation. This depends both on how the SIP proxy is deployed (e.g., as an inbound or outbound proxy) and what TLS options are used (e.g., mutual authentication, session reuse). The cost of symmetric key operations such as AES, in contrast, tends to be small.

1. INTRODUCTION

Securing Voice over IP (VoIP) is a necessary requirement for enabling its stable, long-term adoption. A key aspect of VoIP security is securing the signalling path, typically provided by the Session Initiation Protocol (SIP) [35]. SIP is an application layer signaling protocol for creating, modifying, and terminating media sessions in the Internet. Major standards bodies including 3GPP, ITU-T, and ETSI have all adopted SIP as the core signaling protocol for services such as VoIP, conferencing, Video on Demand (VoD), presence, and Instant Messaging (IM). Like other Internet services, SIP-based services may be susceptible to a wide variety of security threats including social threats, traffic attacks, denial of services and service abuse [3, 7, 22]. One of the main reasons that enable these threats is the common use of insecure SIP signaling such as SIP-over-UDP, which provides no signaling confidentiality, integrity, or authenticity. Given a trace of SIP traffic, one can see who is communicating with whom, when, for how long, and sometimes even what is being said (e.g., in SIMPLE [8]). It has also been shown that even commercial VoIP services may be prone to large-scale voice pharming [41], where victims are directed to fake interactive voice response systems or human representatives for revealing sensitive information.

Transport Layer Security (TLS) [15] is a widely used Internet security protocol occupying a layer between the application and a reliable transport like TCP. There is also a Datagram TLS (DTLS) [33] protocol that provides similar security functionalities but runs over an unreliable transport like UDP. The SIP specification [35] lists TLS as a standard method to secure SIP signaling. Various other organizations and industrial consortiums have also recommended or mandated the use of TLS for SIP signaling. For example, the SIP Forum [2] mandated TLS for interconnecting enterprise and service provider SIP networks in its specification document.

However, while interest in securing SIP is growing [31], actual large scale deployment of SIP-over-TLS has not yet occurred. One important reason is the common perception that running an application over TLS is costly compared to running it directly over TCP (or UDP in the case of SIP). VoIP providers will be hesitant to deploy TLS until they understand the resource provisioning and capacity planning required. Thus we need to understand how much using TLS with SIP actually costs.

This paper makes the following contributions:

- We present an experimental performance study of the impact of using TLS on SIP servers. Our study is conducted using OpenSIPS [27] with OpenSSL [28] on Linux on an Intel-based server. We evaluate the CPU cost of TLS under four SIP proxy usage scenarios: proxy chain, outbound proxy, inbound proxy, and local proxy. We show that using TLS can reduce performance by up to a factor of 17 compared to the typical case of SIP-over-UDP.

- We use application, library, and kernel profiles to examine, analyze, and explain performance differences. The profiles illustrate how costs are incurred under different scenarios (e.g., extra Rivest, Shamir and Adleman (RSA) overheads when mutual authentication is used) and how the costs can be reduced (e.g., RSA costs disappear when session reuse is performed). They also show some results that distinguish SIP server from other server scenarios (e.g., bulk crypto costs of Advanced Encryption Standard (AES) or Triple Data Encryption Standard (3DES) are small) and how some overheads are due to mechanisms (e.g., kernel overhead, Secure Sockets Layer (SSL) state management) rather than simply crypto algorithms such as RSA or AES.

- We identify and solve two performance problems in OpenSIPS. Each is related to connection management with large numbers of connections under high loads. The fixes improve performance in some cases from a few tims up to an order of magnitude.

Previous studies on TLS performance have either focused on TLS for Web servers [5, 10, 21, 44] or policy-based network management [43]. SIP protocol behavior is different from these protocols in several ways. SIP messages tend to be small, whereas Web downloads can be quite large. SIP proxy servers can act as clients to other servers and therefore can incur large client-side TLS costs. Finally, SIP servers have a much wider range of connection management behavior than other servers, and this connection management is the primary issue in determining TLS overheads, due to the heavy costs of RSA operations used for session negotiation. Symmetric key operations such as AES or 3DES are trivial in comparison.

The net result is that the performance cost of deploying SIP over TLS instead of UDP can be significant, depending on how the SIP proxy server is deployed (e.g., as an inbound or outbound proxy) and how TLS is configured (e.g., with or without mutual authentication or session reuse). Network operators can minimize this cost by attempting to maximize the persistence of secure sessions, but still need to be aware of the overhead of utilizing TLS.

The remainder of this paper is structured as follows. Section 2 provides some background on TLS and SIP. Section 3 describes the testbed used and how we determine our test cases. Section 4 presents our experimental results. Section 5 describes related work.

2. BACKGROUND

2.1 TLS Operation Overview

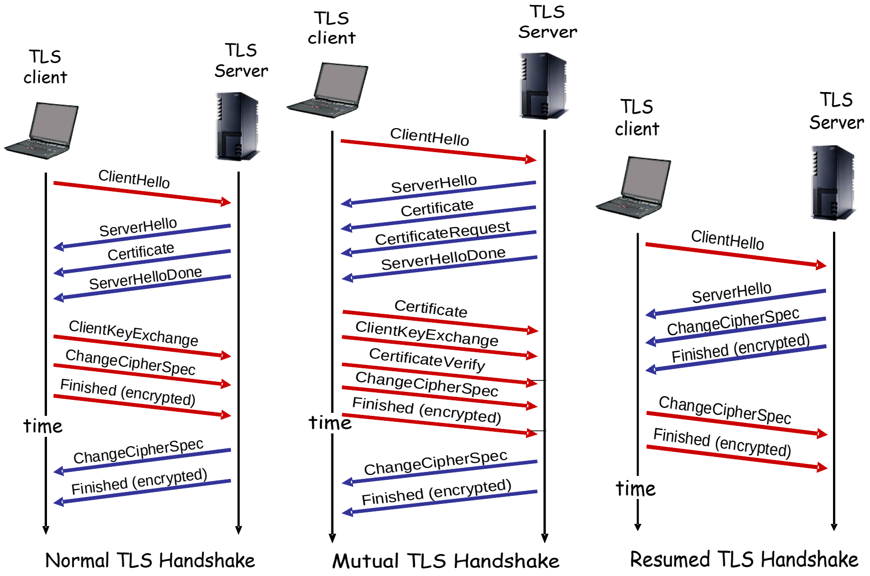
We provide a brief description of the TLS protocol. For more details, please see [15, 32, 37]. We focus on the aspects relevant to our study, namely session establishment and its corresponding RSA public key costs.

TLS operation consists of two phases: the handshake phase and the bulk data encryption phase. The handshake phase allows the parties to negotiate the algorithms to be used during this TLS session, authenticate the other party and prepare the shared secret for the bulk data encryption phase.

The normal message flow in the TLS handshake phase is illustrated in Figure 1(a). The key performance aspects of this handshake are that the server sends its certificate to the client, which then verifies the certificate using a certificate authority's public key. Depending on the key exchange mode, the client may then generate a `pre_master_secret`, and encrypt it using the server's public key obtained from the server's certificate. The server decrypts the `pre_master_secret` using its own private key. Both the server and client then compute a `master_secret` they share based on the same `pre_master_secret`. The `master_secret` is further used to generate the shared symmetric keys for bulk data encryption and message authentication.

In normal TLS handshake, only the client authenticates the server. In situations where the server also wishes to authenticate the client, TLS provides a mutual authentication mode as shown in Figure 1(b). In the mutual authentication mode, after the server sends its own certificate to the client, the server sends an additional `CertificateRequest` message to request the client's certificate. The client responds with two additional messages, a `Certificate` message containing the client certificate with the client public key, and a `CertificateVerify` message containing a digest signature of the handshake messages signed by the client's private key. Since only a client holding the correct private key can sign the message, the server can authenticate the client using the client's public key.

In general, public key cryptographic operations such as RSA are much more expensive than shared key cryptography. This is why TLS uses public key cryptography to establish the shared secret key in the handshake phase, and then uses symmetric key cryptography with the negotiated shared secret as the data encryption key. TLS offers a session reuse mode where the two parties can avoid negotiating the `pre_master_secret` if it has been done previously within some time threshold. It is important to distinguish the notion of a *connection* versus a *session* in TLS. A TLS *connection* corresponds to one specific communication channel which is typically a TCP connection; while a TLS *session* is associated with a negotiated set of algorithms and the established `master_secret` based on the `pre_master_secret`. Multiple connections may be mapped to the same session, all sharing the same set of algorithms and the `master_secret`, but each with different symmetric keys for bulk data encryption. The notion of session reuse indicates the reuse of a previously negotiated set of cryptographic algorithms and the `master_secret`. The handshake message flow for TLS session reuse is shown in Figure 1(c). The first time the client and server communicate, they establish a new connection and a new session. The server stores the session information including the algorithm choice and the `master_secret` for later reference. The session is identified by a `session_id` which is conveyed to the client during the initial handshake in the `ServerHello` message. The next time the client needs to establish a connection, it can include the previous `session_id` in the `ClientHello` message. The server agrees to session reuse by specifying the same `session_id` in the responding `ServerHello` message. The TLS handshake will then proceed to `ChangeCipherSpec` message and `Finished` message directly, avoiding the re-computation of a `pre_master_secret`. The session reuse timeout is configurable based on the security assumptions of how long it takes to break the key by brute-force.



(a) normal (b) with mutual authentication (c) with session reuse

Figure 1: TLS Handshake Message Flows

2.2 SIP Overview

SIP defines two basic types of entities: User Agents (UAs) and servers. UAs represent SIP end points. SIP servers consist of registrar servers for location management, and proxy servers for message forwarding. SIP messages are divided into requests (e.g., `INVITE` and `BYE` to create and terminate a SIP session, respectively) and responses (e.g., `200 OK` for confirming a session setup). The set of messages including a request and its associated responses is called a SIP transaction.

SIP message forwarding, known as proxying, is a critical function of the SIP infrastructure. This forwarding process is provided by proxy servers and can be either stateless or stateful. Stateless proxy servers do not maintain state information about the SIP session and therefore tend to be more scalable. However, many standard application functionalities, such as authentication, authorization, accounting, and call forking require the proxy server to operate in a stateful mode by keeping different levels of session state information. Therefore, we focus on stateful SIP proxying in this paper.

Figure 2 shows a typical message flow of stateful SIP proxying with authentication enabled. Two SIP UAs, designated as User Agent Client (UAC) and User Agent Server (UAS) represent the caller and callee of a multimedia session. The hashed circle around the proxy indicates that this is the server that we are measuring ("system under test"). In this example, the UAC wishes to establish a session with the UAS and sends an `INVITE` message to the proxy. The proxy server enforces proxy authentication and responds with a `407 Proxy Authentication Required` message, challenging the UAC to provide credentials that verify its claimed identity (e.g., based on MD5 [34] digest algorithm). The UAC then retransmits the `INVITE` message with the generated credentials in the `Authorization` header. After receiving and verifying the UAC credential, the proxy sends a `100 TRY-`

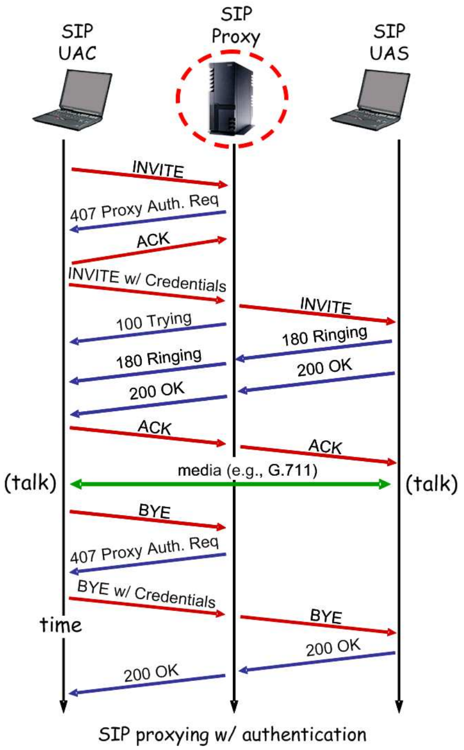


Figure 2: SIP Stateful Proxying with Authentication

ING message to inform the UAC that the message has been received and that it needs not worry about hop-by-hop retransmissions. The proxy then looks up the contact address for the SIP URI of the UAS and, assuming it is available, forwards the message. The UAS, in turn, acknowledges receipt of the INVITE message with a 180 RINGING message and rings the callee's phone. When the callee actually picks up the phone, the UAS sends out a 200 OK. Both the 180 RINGING and 200 OK messages make their way back to the UAC through the proxy. The UAC then generates an ACK message for the 200 OK message. Having established the session, the two endpoints communicate directly, peer-to-peer, using a media protocol such as RTP [39]. However, this media session does not traverse the proxy, by design. When the conversation is finished, the UAC "hangs up" and generates a BYE message that the proxy forwards to the UAS. The UAS then responds with a 200 OK which is forwarded back to the UAC.

SIP proxy authentication is an optional operation, typically done between a UA and its first hop SIP proxy server. While the example above shows a single SIP proxy along the path, in practice it is common to have multiple proxy servers in the signaling path. The message flow with multiple proxy servers is similar, except that the proxy authentication is usually only applicable to the first hop.

2.3 SIP Connection Management over TLS

SIP can operate over different transport protocols, both reliable and unreliable. Since TLS requires a reliable transport, all our evaluations for TLS use TCP transport. In general, a TCP connection is first established between endpoints, and then a TLS handshake occurs to negotiate the TLS session. Once the TLS session is established, the SIP signaling messages will be passed to the TLS layer and encrypted.

When a connection oriented transport such as TCP is used, the connection management policy needs to be defined. In a multi-hop SIP server network scenario, it is usually preferable to maintain a single long-lasting connection between two interconnected proxy servers. All SIP messages between the two proxy servers that go through the same existing connection can avoid the per-session connection handshake overhead. In contrast, if the proxy server is connected with a SIP UAC or UAS directly, the proxy typically has to establish separate connections with each of them since they are located on separate hosts.

3. TESTBED AND METHODOLOGY

3.1 OpenSIPS SIP Server

The SIP server we evaluated is Open SIP Server (OpenSIPS) version 1.4.2 [27], a freely-available, open source SIP proxy server. OpenSIPS is a fork of OpenSER, which in turn was a fork of SIP Express Router (SER) [20]. All these proxy servers are written in the C language, use standard process-based concurrency with shared memory segments for sharing state, and are considered to be highly efficient. These sets of server implementations represent the de facto open source version of SIP server, occupying a role similar to that of Apache for web server [4, 6, 13, 14, 16, 17, 24, 30, 42].

We made several modifications to OpenSIPS in order to support all of our test cases. In particular, we added a connection mode where OpenSIPS will establish a new connection to a UAS upon a new call, even if the UAS has the same IP address. This is needed to test the multiple connection mode between the proxy server and UAS using a limited number of UAS machines. We also added OpenSIPS options to to request TLS session reuse when it is acting as the TLS client, and OpenSIPS options to request for TLS mutual authentication when it is acting as the TLS server.

One unexpected parameter that initially prevented us from running high load tests with SIP proxy authentication is the "nonce index" value in OpenSIPS. It turns out that the default MAX_NONCE_INDEX value used to create nonce for proxy authentication is too small and could exhaust easily at high load. When the nonce could no longer be generated, authentication cannot proceed and the server will simply reject calls. We increased the default MAX_NONCE_INDEX value from $100,000$ to $10,000,000$. This change alone increased the throughput results dramatically, e.g., in the proxy chain mode the peak throughput with SIP proxy authentication is increased by close to an order of magnitude.

In configurations involving proxy authentication where a user database is required, we use MySQL-5.0.67 [26], which we populated with $10,000$ unique user names and passwords. The MySQL server runs on the same machine as the OpenSIPS server.

3.2 SIPp Client Load Generator

We use another freely available open-source tool, SIPp [19] to generate SIP traffic. SIPp allows a wide range of SIP scenarios to be tested, such as UAC, UAS and Third-Party Call Control (3PCC). We use the SIPp release from August 26th, 2008. We also added additional functionality to SIPp to accommodate all our test cases. Specifically, we added SIPp options to request TLS session reuse when it is acting as the TLS client and SIPp options to request TLS mutual authentication when it is acting as the TLS server. The TLS support library for SIPp is a statically-compiled version based on OpenSSL [28] release 0.9.8i (which is the latest release at the time of the compilation).

3.3 Hardware and Connectivity

The server hardware we use has 2 Intel Xeon 3.06 GHz processors with 4 GB RAM and 34 GB disk drives. However, for our experiments, we only use one processor because SIP performance under multiple processors or a multi-core processor is itself a topic that requires separate attention [42]. We use 10 client machines, six of which have 2 Intel Pentium 4 3.00 GHz processors with 1 GB RAM and 80 GB hard drives. The other four have 2 Intel Xeon 3.06 GHz processors with 4 GB RAM and 36 GB hard drives. The server and client machines communicate over copper Gigabit or 100Mbit Ethernet. The round trip time measured by the ping command from the client to the server is around 0.15 ms.

3.4 Software Platform

The server uses Ubuntu 8.04 with Linux kernel 2.6.24-19, OpenSSL 0.9.8.g, and oprofile 0.9.3. The clients use Ubuntu with either a 2.6.22 kernel or a 2.6.24 kernel. We encountered an SSL library failure at the SIPp load generator side when generating high loads. After examining the OpenSSL error queue in more detail, the ERR_error_string is found to be error:1409F07F:SSL routines:SSL3_WRITE_PENDING: bad write retry. A bug fix is found at [18]. This fix was submitted in 2003 but had not yet been incorporated into the OpenSSL release. We therefore recompile SIPp using OpenSSL version 0.9.8i source with this fix included. The OpenSIPS server machine uses the existing OpenSSL version 0.9.8g. The bug does not manifest itself there and keeping the original OpenSSL on the server makes profiling more convenient.

3.5 Workload and Performance Metrics

The workload is a standard SIP call flow provided by SIPp illustrated in Figure 2. There is no call hold time. Our main metrics are server throughput as reported by SIPp and server profile CPU events as reported by oprofile [29]. We also measure server CPU utilization. All our test runs last for 120 seconds after a 30-second warm-up time. All metrics are the average of three consecutive test runs.

3.6 Test Matrix and Evaluated Test Cases

We first group possible SIP server connection management configurations into four different deployment modes as shown in Figure 3.6.

1. Figure 3(a) shows the *proxy chain* mode, where the proxy server interconnects two other proxy servers in a chain fashion. This is intended to model, e.g., how two core SIP proxy servers of different service providers communicate. Only one connection is needed for each neighboring proxy server in this case.

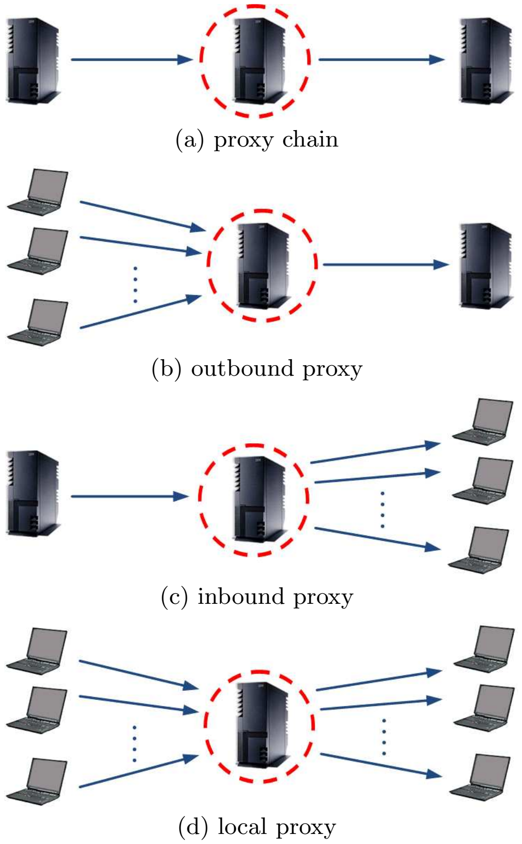
2. Figure 3(b) shows the *outbound proxy* mode, where the proxy accepts multiple connections from UACs but only establishes a single outgoing connection with another proxy server. This configuration models how phones in an enterprise VoIP deployment would make calls external to the organization.

3. Figure 3(c) is the *inbound proxy* mode, where the proxy server under test accepts a single connection from an upstream proxy server and establishes multiple connections to individual UASes. This is the mirror of the outbound proxy configuration above, where incoming SIP traffic is routed to phones.

4. Figure 3(d), is the *local proxy* mode, where the proxy server under test connects UACs and UASes directly, and therefore accepts both incoming connections and creates outgoing connections simultaneously. This configuration is intended to model how phones in an enterprise deployment would communicate with each other.

SIP proxy servers usually support all these four modes of operation, thus this characterization is somewhat logical rather than physical. While in practice real proxy behavior will lie somewhere in the middle of these four extremes, the characterization lets us explore the design space fully.

For example, a SIP proxy operating in the proxy chain mode could well connect a number of different proxy pairs. It does not necessarily interconnect only a single pair of proxy servers. Similarly, an outbound proxy might connect to more than one upstream proxy. The four modes thus describe the full range of connection management behavior for SIP proxy servers, from completely persistent connections to a small set of nodes (the proxy chain mode) to non-persistent connections where each call requires a connection setup and teardown (the local proxy mode). In addition, the inbound and outbound cases distinguish where connections are passively accepted (the inbound proxy mode) vs. those that are created (the outbound proxy mode). To explore the applicable test matrix, we characterize five main configuration variables in our SIP-over-TLS tests: TLS connection management, TLS session reuse, TLS mutual authentication, TLS cipher suite and SIP proxy authentication. Note that the connection mangagement configuration options also applies to TCP.


(a) proxy chain

(b) outbound proxy

(c) inbound proxy

(d) local proxy

Figure 3: SIP Proxy Operation Modes

To relate connection management with other configuration parameters, we draw a unified logical component graph of the testbed as in Figure 4. The proxy server in the middle represents the server under test. The whole testbed is split into the left path and the right path, which consists of the left pair and the right pair of the logical UAC and UAS components, respectively. The applicable configuration options in each of the four connection management modes can then all be mapped into Table 1, where N/A indicates "Not Applicable".

Directly expanding the whole test space in Table 1 results in numerous configuration scenarios which are both intractable and unnecessary. We make the following decisions to narrow down the cases towards a workable test set. First, for TLS cipher suite, since the SIP standard [35] already specifies the mandatory `TLS_RSA_WITH_AES_128_CBC_SHA` cipher suite (abbreviated as TLS-AES) and a recommended `TLS_RSA_WITH_3DES_EDE_CBC_SHA` cipher suite (abbreviated as TLS-3DES), we focus on these two cipher suites only. In particular, since the impact differences between these two ci-

Configuration	TCP/TLS Multiple Connections		TLS Session Reuse		TLS Mutual Authentication		TLS Cipher Suite	SIP Proxy Auth.
	Left Path	Right Path	Left Path	Right Path	Left Path	Right Path		
Proxy Chain	N/A	N/A	N/A	N/A	N/A	N/A	any	Yes/No
Outbound Proxy	Yes	N/A	Yes/No	N/A	Yes/No	N/A	any	Yes/No
Inbound Proxy	N/A	Yes	N/A	Yes/No	N/A	Yes/No	any	Yes/No
Local Proxy	Yes	Yes	Yes/No	Yes/No	Yes/No	Yes/No	any	Yes/No

Table 1: Overall Test Matrix

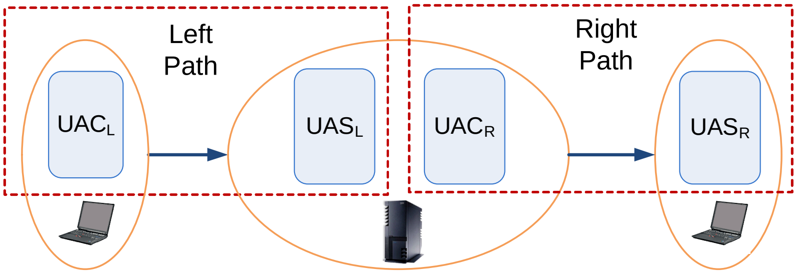


Figure 4: Logical Component Graph of SIP Testbed

pher suites are mainly on the bulk data encryption phase, we test both cipher suites only in the proxy chain mode which is specifically meant to examine the impact of TLS bulk data encryption. For all other three proxy modes, we test TLS-AES only. Second, we enable SIP proxy authentication only in the outbound proxy and local proxy modes, which is a common setting. Third, we test the TLS session reuse and TLS mutual authentication separately to understand each of their impacts. We configure appropriate certificates on both servers and clients in experiments which require them. Fourth, when both the left path and the right path can apply TLS session reuse or TLS mutual authentication, both paths have the same setting. These decisions reduce our test space for TCP and TLS to 16 configurations. Adding the two UDP Auth and UDP NoAuth settings, we have a total of 18 test configurations.

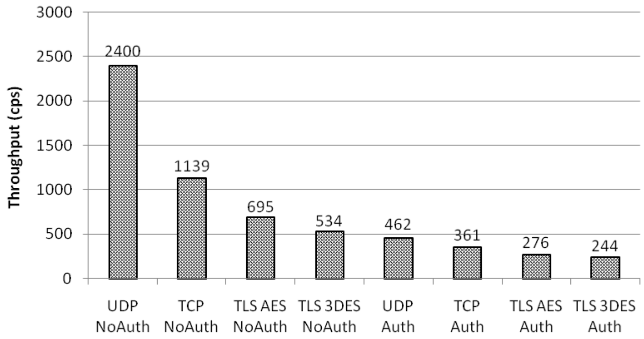


Figure 5: Peak Throughput: Proxy Chain

4. RESULTS AND ANALYSIS

Different proxy modes and configuration scenarios can incur significantly different overheads and result in very different limits on throughput. We start with the relatively simple proxy chain mode and then examine the more complex modes of outbound proxy, inbound proxy, and local proxy. For each of the 18 scenarios, we measure peak throughput and then use CPU profiles to understand and explain the performance costs.

4.1 Proxy Chain

Figure 5 shows the peak throughput in calls per second (cps) for the proxy chain mode using several configurations. Each bar shows the performance for a different configuration. The first four bars have SIP proxy authentication disabled and the next four have SIP proxy authentication enabled. The tests include UDP, TCP only, TLS with the TLS-AES cipher suite, and TLS with the TLS-3DES cipher suite. Recall that in this mode, no connection setup overheads are incurred. The average CPU utilization ranges from 95% to 100% in all the peak test cases except for the UDP and TCP without authentication cases, which is about 70% and 85%, respectively. Note that not all the tests could reach full CPU utilization because there is not always quite enough client machines to fully load the testbed.

We see from Figure 5 that the peak throughput using TCP achieves about 47% of the throughput using UDP, when SIP proxy authentication is not used. When authentication is enabled, TCP provides 78% of the corresponding UDP performance. Adding TLS to the scenario results in even more substantial performance reductions. When SIP proxy authentication is not enabled, TLS-AES achieves 60% of the corresponding TCP throughput, and TLS-3DES achieves 47% of the TCP throughput. When proxy authentication is enabled, TLS-AES achieves 76% of the corresponding TCP throughput and TLS-3DES achieves 68% of the TCP throughput.

While it would be convenient to simply attribute the extra overheads to the corresponding encryption algorithms, it turns out the reality is more complex. To better understand the overheads, we turn to the CPU profiles generated by oprofile. Our approach is to obtain a CPU profile of each configuration run at the same load level of 50 calls per second so that results across configurations can be compared meaningfully. As components are added (e.g., TLS vs. no TLS) or changed (AES vs. 3DES), the attendant CPU costs will manifest themselves in the profiles. This assumes costs scale relatively linearly with load and exhibit the same proportions at the peak as they do at 50 cps, which is not always the case. To test the accuracy of this assumption, we compare the observed peak throughputs with the ones extrapolated based on the CPU cycle costs observed. On average, the estimates match the observed peaks within 15 percent.

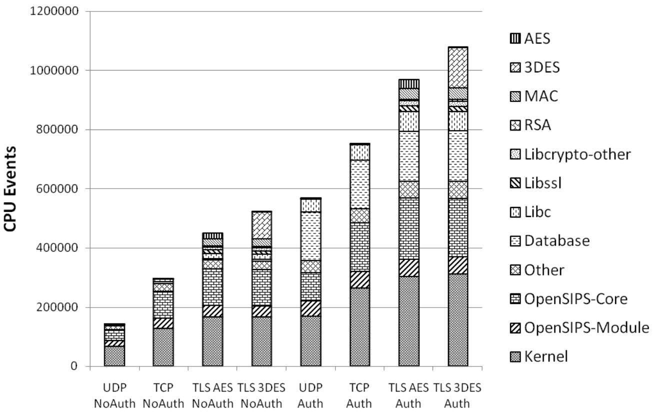


Figure 6: CPU Profile Cycle Costs: Proxy Chain (50 cps)

Figure 6 shows the number of non-idle CPU cycles consumed by the server in the proxy chain mode for each configuration during the test. We see that the total cost of the baseline UDP case without SIP authentication is about 144K CPU cycles. The most significant cost components are kernel (68K) which accounts for 47%, and the sum of OpenSIPS-Core and OpenSIPS-Model (54K), which contributes another 38% of the total cost. When TCP is used instead of UDP, the total costs increase 152K cycles or over 100%. Again most of the increase belongs to Kernel (60K) and the sum of OpenSIPS-Core and OpenSIPS-Module (71K).

We see that adding TLS-AES introduces another 50% of additional overhead, roughly 450K cycles vs. 300K cycles for the TCP case. TLS-3DES is similar, with roughly 525K cycles, and as would be expected, the differences in total cost between TLS-AES and TLS-3DES are almost solely contributed by the cost difference in cryptographic operations.

Half of the 150K increase from TCP to TLS-AES is directly contributed by TLS operations, and most of the remainder is relatively evenly shared by increases in Kernel and OpenSIPS-Core. Since 128 bits AES is less expensive than SHA-1, AES itself only adds about 19K cycles in cost; Message Authentication Code (MAC) overheads are higher at 25K cycles. MAC overheads are incurred by the bulk encryption algorithm, since each message is verified for authenticity using the MAC algorithms. MAC overheads are roughly equivalent regardless of the choice of AES or 3DES since we use SHA-1 in both cases. While 3DES is over 4X as expensive as AES (93K vs. 19K cycles), the relative difference between the two complete software stacks is only about 17% (525K vs. 450K). We expect AES to be faster since it is a more recent cipher than 3DES and was designed for performance. Other TLS overheads come from other components in the OpenSSL library. For example, in the TLS-AES case, there are other libcrypto functions (10K) and libssl (11K). Thus a non-trivial component of SSL overheads is from using the SSL mechanisms, such as allocating, freeing, maintaining, and looking up SSL session state.

Comparing the TCP case and the two TLS cases, the CPU profiles do not show the increases in kernel and OpenSIPS-Core centering on any specific functions. Between the two TLS cases themselves, the cost of Kernel and OpenSIPS-Core are quite similar.

The major difference when SIP proxy authentication is enabled is the additional database cost, which ranges from $16 - 29\%$ of the total cost in each case. When the database overhead is included, TCP will introduce 32% overhead over UDP. TLS-AES and TLS-3DES will incur an additional 30% and 44% over TCP, respectively. The rest of the cost contributions are similar to when SIP authentication is not enabled, because the authentication database functions are orthogonal to the TLS functions.

4.2 Outbound Proxy

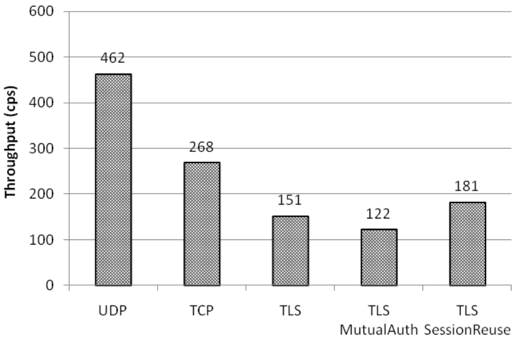


Figure 7: Peak Throughput: Outbound Proxy

Figure 7 shows the peak throughputs of the outbound proxy mode for several configurations. Recall that in the TCP or TLS cases of this mode, each call results in a new connection being established with the server, as opposed to the proxy chain mode above. Each bar again indicates a different configuration, namely UDP, TCP, TLS, TLS with mutual authentication, and TLS where session reuse occurs on each TLS connection. Each configuration has SIP authentication enabled. Since the choice of AES or 3DES only affects the bulk data encryption overheads, which we examined in Section 4.1, for simplicity we restrict our experiments with TLS to use only AES for the remainder of this paper. The average CPU utilization in each case is around 90%. We see that the peak throughput in the TCP case is around 58% of the baseline UDP case. The TLS case is approximately 56% of the TCP case. Within the TLS cases, adding TLS mutual authentication reduces throughput about 20%, while enabling session reuse increases throughput about 20%.

Figure 8 shows the CPU profiles for the different outbound proxy configurations, again at the 50 calls per second load level. Using TCP introduces about 47% more or 271K of overheads compared to using UDP. Within this in-

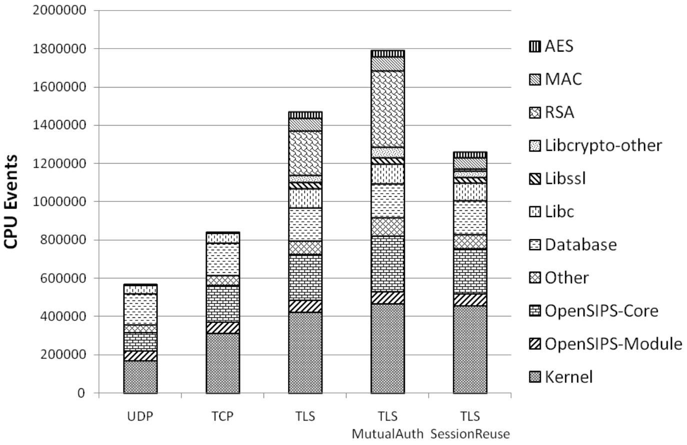


Figure 8: CPU Profile Cycle Costs: Outbound Proxy (50 cps)

crease, Kernel costs contribute 144K, while OpenSIPS-Core and OpenSIP-Module contribute 102K. The remaining 25K is contributed by libc and other functions.

The use of TLS introduces 75% of additional overhead compared to the TCP case. TCP consumes about 840K cycles whereas TLS costs about 1,470K cycles. Much of this increase comes from RSA (233K cycles) because in this configuration the proxy needs to perform one of the most costly operations in the TLS handshake: RSA decryption of the `pre_master_secret` using its private key. Another major component of the increase is from MAC processing (65K cycles), which is not only used to verify the encrypted messages but also to verify the server certificate and construct the `master_secret`. Other OpenSSL overheads such as libssl (34K) and other libcrypto functions (36K) also contribute.

Enabling TLS mutual authentication incurs about 1,790K cycles or an additional 330K over the baseline TLS, most of which comes from increased RSA costs (160K). Recall in this case the server requests the client's certificate which the server verifies using RSA public key decryption. In addition, the server performs another RSA public key decryption for the client's certificate verification message and also verifies the certificate using the MAC algorithm. Indeed, we see MAC costs increase by 10K cycles when mutual authentication is used. Kernel costs also increase by 45K cycles, presumably due to additional network packets transmitted and context switches between user and kernel space.

However, enabling TLS session reuse reduces the overhead by 15% compared to the baseline TLS case, or by about 200K cycles. Most of this overhead is explained by the reduction in RSA costs, which shrink from 233K cycles to only 10K cycles. This is because in the session reuse case, no key exchange and certificate verification is required. MAC costs remain, however, since new cryptographic keys are still computed for data encryption.

It is worth noting that the TLS mutual authentication test above also includes SIP proxy authentication. While TLS mutual authentication is used to authenticate and authorize "client systems", SIP proxy authentication is used to authenticate and authorize "users".

4.3 Inbound Proxy

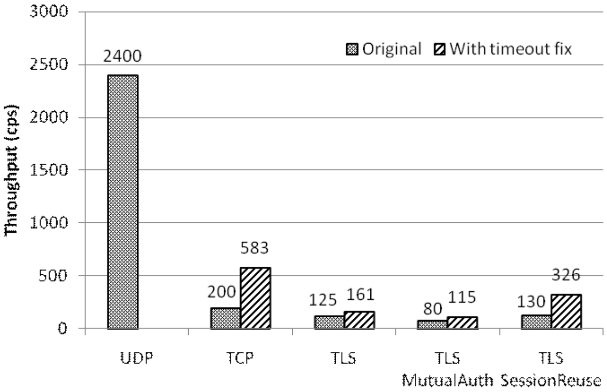


Figure 9: Peak Throughput: Inbound Proxy

Figure 9 shows the peak throughput of the inbound proxy mode, where SIP proxy authentication is not enabled. The figure shows two versions of OpenSIPS: the original version and one with a modification we developed, denoted "with timeout fix" in the graph. We start by explaining the performance problem we discovered and how we solved it.

We examined the original OpenSIPS CPU profile under the peak throughput for TCP and TLS. Surprisingly, we found that 50% of the CPU cycles in the TCP case and 20% percent of the CPU cycles in the TLS case are spent in two functions, `tcp_main_loop` and `tcp_receive_loop`. More detailed profiling reveals that the overhead in the two functions are primarily the cost of two timeout mechanisms used to close the TCP connections which are no longer in use. In the inbound proxy case, the timeout mechanism becomes prominent because the UAS in our tests does not close the TCP/TLS connection when the call is over. There can be thousands of simultaneous TCP connections existing in the TCP connection table. The current server code calls a `timeout` function every time the `epoll` mechanism returns when events are detected. Since the connection expiration time is not linked to the corresponding hash key, during each call to the `timeout` function, the entire TCP connection hash table is traversed . Therefore, at high loads when the hash table has thousands of entries, the time spent in the timeout function becomes much larger than that of the case under lower load.

We applied a fix to the existing OpenSIPS TCP connection timeout mechanism. Observing that the timeout is based on a time tick with one second resolution, it makes no sense to enter the timeout function more than once per second. We therefore added a time tick check before calling the timeout function. If the program has called the timeout function during the current time tick value already, then it will not enter the timeout function until the time tick value is advanced. This simple fix turned out to have a drastic effect on performance involving TCP and TLS, as shown in Figure 9.

As can be seen, the TCP case and the TLS with session reuse scenario enjoy the most obvious boosts in throughput, by about 200% and 150% respectively. For example, in the TCP inbound proxy test case, the contribution of the two timeout functions to the total overhead reduces from 50%

to a negligible 0.6%, and the total cost drops by 73%. In addition, kernel costs shrink by 43%. CPU utilization at the 200 calls per second load level reduces from 95% to 20%. The CPU utilizations at the peak throughput values with the timer fix are in the range of 80% to 90%.

The other two scenarios, TLS and TLS with mutual authentication, also see performance increases but the differences are less dramatic. The reason is that in the latter two scenarios, the proportion of cryptographic overheads take a greater part of the total cost, so reducing the OpenSIPS and kernel overheads has a smaller impact.

From Figure 9, we see that the peak throughput with TCP is about 24% of the UDP case. The peak throughput of TLS is approximately 28% of the TCP case. Within the TLS cases, adding TLS mutual authentication reduces throughput by 29%, while enabling session reuse increases throughput by 100%.

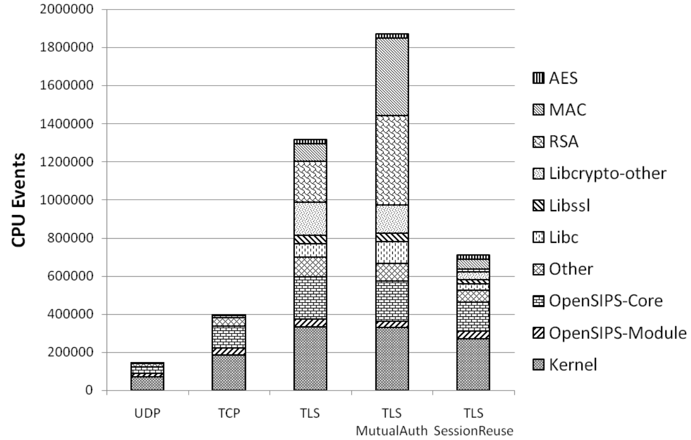


Figure 10: CPU Profile Cycle Costs: Inbound Proxy (with Timeout Fix)

Figure 10 shows the CPU profiles for the several inbound proxy configurations where the timeout fix has been applied. In general, using TCP incurs 174% (250K) of additional overhead compared to using UDP, 118K of which comes from increase in Kernel and 94K from increases in OpenSIPS-Model and OpenSIPS-Core. The remainder comes from libc (8K) and other functions (30K). The use of TLS introduces over 233% of additional overhead compared to the TCP case (1,315K cycles vs. 394K). 212K cycles are contributed by RSA, 173K by other libcrypto processing, 93K by MAC processing, 44K by libssl, and 23K by AES. Kernel overheads increase by 150K and OpenSIPS-Core by 110K.

Enabling mutual authentication incurs an additional 42% overhead (550K cycles) over the baseline TLS. The majority of that increase comes from RSA (260K). MAC processing is also increased by 310K.

Enabling TLS session reuse reduces costs by 46% compared to the base TLS case, with total costs falling from 1,315K to 710K or about 600K cycles. Reduced RSA processing contributes 200K of those reductions; other libcrypto costs drop by 135K; MAC overheads are reduced by 40K; libssl costs shrink by 20K.

In this configuration, the main RSA costs in the normal TLS case come from the proxy verifying the UAS's certificate and the proxy encrypting the `pre_master_secret` to be sent to the UAS. The additional increase in RSA overheads in the mutual TLS configuration is mainly because the proxy needs to sign the client authentication message using its private key.

An interesting observation from this figure is the cost of MAC functions, which are substantially higher than in the previous proxy scenarios. This is because the proxy in the inbound mode acts as TLS client and needs to verify the certificates presented by the UAS, which was not present in the outbound mode. In addition, in the mutual TLS case, the inbound proxy needs to perform RSA encryption using its own private key and to sign the certificates using the MAC algorithm. These overheads are exhibited in the profiles. Furthermore, in the TLS with session reuse case, the MAC costs are significantly reduced, indicating that a large amount of the MAC cost is associated with the RSA key exchange phase, rather than during the bulk data encryption.

4.4 Local Proxy

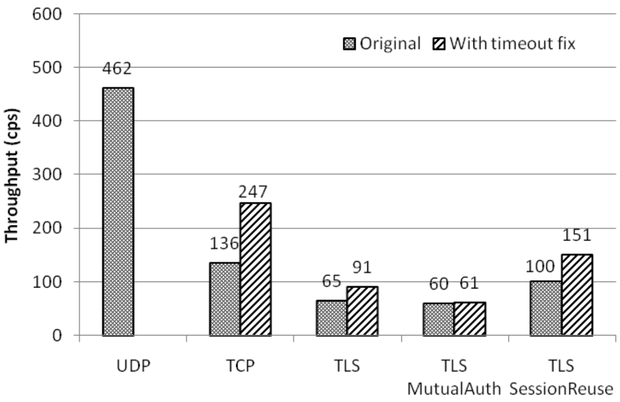


Figure 11: Peak Throughput: Local Proxy

Figure 11 shows the peak throughputs of various configurations in the local proxy mode, both with and without the timeout fix described in Section 4.3, and with SIP authentication enabled. We see the timeout fix has a substantial impact on performance for both the baseline TCP case and for TLS when session reuse is enabled, where TCP overheads are significant. The timeout fix makes less of an impact on the other TLS cases because in those cases the TLS overheads account for a larger proportion of the total cost. For the remainder of this Section, we focus our analysis on the configurations where the timeout fix is applied.

The average CPU utilizations in the four configurations with the timeout fix are between 80% to 90%. We see that the peak throughput with TCP is around 53% of the UDP case, while the peak throughput with TLS is approximately 37% of the TCP case. Within the TLS cases, adding TLS mutual authentication reduces throughput by 33%, while enabling session reuse increases throughput by 66%.

Figure 12 shows the CPU profile results for the local proxy mode with the timeout fix. In general, the use of TCP incurs 58% of additional overhead compared to the baseline UDP case. 186K of this is contributed by Kernel, 108K by OpenSIPS-Core and OpenSIPS-Module, 10K by libc and

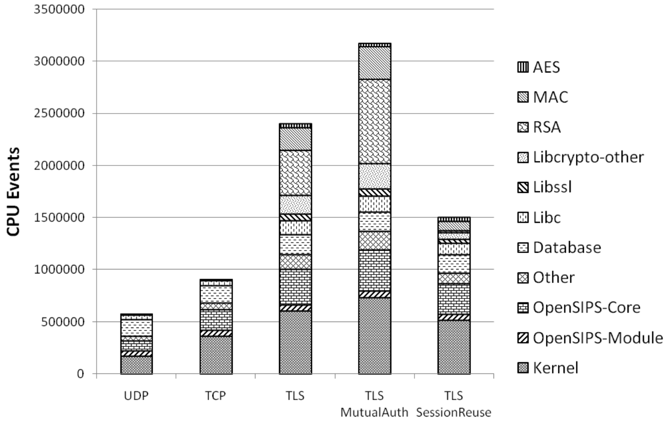


Figure 12: CPU Profile Cycle Costs: Local Proxy (with Timeout Fix)

30K by other functions. Using TLS introduces over 166% of additional overhead compared to the TCP case. Total cycles increase by 1,500K from 900K to 2,400K. RSA contributes 434K to that increase, followed by kernel overheads 240K, MAC processing 219K, other libcrypto functions 174K, OpenSIPS-Core 140K, libssl 67K, and AES 36K.

Enabling TLS mutual authentication incurs an additional 32% overhead over the baseline TLS, increasing total costs about 800K from 2,400K to 3,170K. Additional RSA overheads contribute 375K of the increase, 125K from kernel, 100K from MAC, 70K from libcrypto, 45K from OpenSIPS-Core, and 5K from libssl.

Enabling TLS session reuse reduces the cost relative to the baseline TLS case by 38%. Cycles shrink by 900K from 2,400K to 1,500K. RSA savings contribute 415K to the difference, followed by MAC 130K, other libcrypto functions 110K, kernel 80K, OpenSIPS 50k, libssl 25k.

The MAC cost is significantly reduced in the TLS with session reuse case, indicating that a large amount of the MAC cost is associated with the RSA public key exchange phase, as discussed in the inbound proxy case in Section 4.3.

5. RELATED WORK

SSL/TLS performance has been studied by a number of researchers. However, almost all these studies are based on SSL/TLS Web servers. Apostolopoulos et al. [5] found that the overhead due to TLS can reduce the number of HTTP transactions handled by up to two orders of magnitude. Kant et al. [21] investigated the architectural impact of SSL, and concluded that the use of SSL increases the compositional cost of transactions by a factor of 5 − 7. Zhao et al. [44] provided an oprofile-based anatomy of SSL processing for an SSL Web server. They found that about 70% of the total processing time of an HTTP over SSL transaction is spent in SSL processing. Coarfa et al. [10] measured the difference of TLS server throughput by selectively replacing TLS operations with no-ops, instead of using a CPU profiler. Their results show that RSA computations are the single most expensive operation in TLS, which accounts for 13 − 58% of the total time spent under different available server CPU cycles and workload conditions.

Zeng and Cherkaoui [43] studied the performance of TLS on a Common Open Policy Service (COPS) over TLS environment. The results of their study showed that establishing a COPS over TLS session took about a thousand times as much as needed for a pure COPS connection without TLS.

Many researchers have studied SIP server performance, albeit without TLS. Schulzrinne et al. presented SIPstone [40], a suite of SIP benchmarks for measuring SIP server performance on common tasks. Cortes [12] measured the performance of four different stateful SIP proxy server implementations over UDP and reported throughput results from 90 − 700 cps. Nahum et al. [16, 24] showed experimental performance results of the OpenSER SIP server under different scenarios including stateful and stateless proxying, TCP and UDP transport, with and without SIP proxy authentication. Their results indicate that any of these configurations can affect performance by a factor of 2 − 4. Their evaluated SIP-over-TCP scenario corresponds to the TCP single connection or the proxy chain mode in this paper. Oho and Schulzrinne [25] studied the performance of the SIPd [38] SIP server over the UDP and TCP transports. Their TCP tests include the multiple connection mode between the SIP proxy and the UA similar to the local proxy mode of this paper. Ram et al. [30] provided more understanding of the impact of TCP on SIP server performance using OpenSER. They show that a substantial component of the performance loss from using TCP is due to the process architecture in OpenSER and provide improvements. Wright et al. [42] studied the performance of SIP servers on multi-core systems. They proposed and evaluated several optimizations to improve scalability up to eight cores.

Kim et al. [23] described a study of SIP with TLS, DTLS and IPSec over TCP, UDP and SCTP. However, the work is based on ns-2 [1] simulation and the scope of the evaluation is on call setup delay in a two-hop SIP proxy scenario with background traffic. Thus the focus is on delay as a function of packet exchanges rather than server CPU overheads. Cha et al. [9] also measured the call setup delay (along with voice quality metrics such as mean option score) of a SIP-based VoIP system implementation which contains both TLS and S-MIME. But it is not clear what the software and hardware used are, or what the call request rate during the measurement is.

The most relevant work we found is from Salsano et al. [36] who measured the throuhgput performance and processing cost of SIP proxy server over UDP, TCP and also TLS. Their test cases for stateful SIP proxy servers represent four of the 18 scenarios that we look at, essentially the UDP NoAuth, UDP Auth, TCP Auth, and TLS Auth configurations, all in the proxy chain mode. The total cost ratios of these four scenarios in their work are 1:1.44:1.52:1.54, while the corresponding ratios from our results are 1:4:5.2:6.7. These numbers are not directly comparable because of the different software and hardware platforms used in the two sets of experiments. Salsano et al. used their own open source SIP server implemented in Java using a 300 MHz Pentium machines running either Linux or Windows 98/2000. We use contemporary hardware and standard open-source software implemented in C. As a result, the peak performance of the two testbeds are also dramatically different. For example, in the basic UDP NoAuth scenario, the peak throughput on their testbed is 21 cps, compared to 2,400 cps on ours, a factor of 100 difference in performance.

One approach to reducing security overheads is to use a hardware crypto accelerator, e.g., Sun's Crypto 6000 card [11]. While this can improve performance (e.g., the card claims 13,000 1024-bit RSA operations per second), the cards tend to be expensive (e.g., the list price for the board was $1350 at the time of this writing). More importantly, in many cases, much of the overhead we observed was in the OpenSSL software libaries themselves (e.g., libssl, libssl-other), rather than the crypto algorithms (libcrypto). Crypto acceleration hardware will not help with these overheads.

6. CONCLUSIONS

Insecure UDP-based signaling is one major reason that exposes SIP-based services to many common security threats. We have evaluated and analyzed the impact of using TLS as a transport on SIP server performance versus the standard approach of SIP-over-UDP. Using an experimental testbed with the OpenSIPS server, OpenSSL, Linux, and an Intel-based server, we show that the performance with TLS can be reduced significantly. We use application, library, and kernel profiling to illustrate where different costs are incurred (e.g., extra RSA overheads when mutual authentication is used) and how they can be avoided (i.e., RSA costs are nearly eliminated when session reuse is effective).

In the best case, the baseline UDP performance is about three times that with TLS (the proxy chain mode); in the worst case, UDP is 17 times the performance than with TLS (the local proxy with TLS and mutual authentication). The performance results depend primarily on whether and how frequent TLS connection establishment is performed, since TLS session negotiation incurs expensive RSA public key operations. In turn, session negotiation depends on how the SIP proxy is deployed (as an inbound, outbound, or local proxy) and how TLS is configured (with mutual authentication or session reuse). Bulk encryption costs such as 3DES or AES, in contrast, are minimal, typically no more than seven percent.

Implementation plays a role as well. We found several performance bugs in OpenSIPS and OpenSSL, despite the fact that they have mature code bases and large numbers of users. When fixed, performance improved in some cases from a few times up to an order of magnitude.

Network operators considering deploying SIP over TLS will need to consider the extra resources required to provide the same service quality as would be the case with UDP. Costs can be reduced by maximizing the potential for persistent TLS sessions, which avoid heavy connection setup costs. These lessons may be appropriate for other protocols that use TLS, especially if they tend to have short messages.

Acknowledgments

The authors would like to thank the anonymous reviewers for insightful and detailed comments which helped improve this paper. Charles Shen would like to acknowledge Dr. Arata Koike of NTT for useful discussions.

7. REFERENCES

[1] ns-2 simulator. `http://www.isi.edu/nsnam/ns/`.

[2] SIP forum. `http://www.sipforum.org`.

[3] VoIP security alliance. `http://www.voipsa.org`.

[4] Arup Acharya, Xiping Wang, and Charles Wright. A programmable message classification engine for session initiation protocol (sip). In *ANCS '07: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, pages 185–194, Orlando, FL, December 2007.

[5] G. Apostolopoulos, V. Peris, and D. Saha. Transport layer security: How much does it really cost? In *IEEE InfoCom '99: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, New York, NY, March 1999.

[6] Vijay A. Balasubramaniyan, Arup Acharya, Mustaque Ahamad, Mudhakar Srivatsa, Italo Dacosta, and Charles P. Wright. Servartuka: Dynamic distribution of state to improve SIP server scalability. In *ICDCS '08: Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, pages 562–572, Beijing, China, June 2008.

[7] D. Butcher, X. Li, and J. Guo. Security challenge and defense in VoIP infrastructures. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(6):1152–1162, November 2007.

[8] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitima, and D. Gurle. Session Initiation Protocol (SIP) extension for Instant Messaging. RFC 3428 (Standard), December 2002.

[9] E. Cha, H. Choi, and S. Cho. Evaluation of security protocols for the Session Initiation Protocol. In *Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, HI, August 2007.

[10] C. Coarfa, P. Druschel, and D. Wallach. Performance analysis of TLS Web servers. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security (NDSS)*, San Diego, CA, February 2002.

[11] Oracle Corporation. Sun crypto accelerator 6000 PCIe card.

[12] M. Cortes, J. Ensor, and J. Esteban. On SIP performance. *IEEE Network*, 9(3):155–172, Nov 2004.

[13] Italo Dacosta, Vijay Balasubramaniyan, Mustaque Ahamad, and Patrick Traynor. Improving authentication performance of distributed SIP proxies. In *IPTComm '09: Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications*, pages 1–11, Atlanta, GA, July 2009.

[14] Italo Dacosta and Patrick Traynor. Proxychain: Developing a robust and efficient authentication infrastructure for carrier-scale VoIP networks. In *USENIX '10: Proceedings of the USENIX Annual Technical Conference*, Boston, MA, June 2010.

[15] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), 2008.

[16] E. Nahum and J. Tracey and C. Wright. Evaluating SIP server performance. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):349–350, June 2007.

[17] Joachim Fabini, Norbert Jordan, Peter Reichl, Alexander Poropatich, and Rainer Huber. "IMS in a bottle": Initial experiences from an OpenSER-based prototype implementation of the 3GPP IP multimedia subsystem. In *ICMB '06: Proceedings of the International Conference on Mobile Business*, page 13, Copenhagen, Denmark, June 2006.

[18] RT for openssl.org. Ticket no. 598. `http://rt.openssl.org/Ticket/Display.html?id=598\&user=guest\&pass=guest`.

[19] R. Gayraud and O. Jacques. SIPp. `http://sipp.sourceforge.net`.

[20] IPTel.org. SIP express router (SER). `http://www.iptel.org/ser`.

[21] K. Kent, R. Iyer, and P. Mohapatra. Architectural impact of secure socket layer on Internet servers. In *International Conference on Computer Design (ICCD)*, pages 7–14, Austin, TX, October 2000.

[22] A. Keromytis. Voice over IP: Risks, threats and

vulnerabilities. In *Proceedings of the Cyber Infrastructure Protection (CIP) Conference*, New York, NY, June 2009.

[23] J. Kim, S. Yoon, H. Jeong, and Y. Won. Implementation and evaluation of SIP-based secure VoIP communication system. In *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, Shanghai, China, December 2008.

[24] E. Nahum, J. Tracey, and C. Wright. Evaluating SIP proxy server performance. In *17th International Workshop on Networking and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Urbana-Champaign, IL, June 2007.

[25] K. Ono and H. Schulzrinne. One server per city: Using TCP for very large SIP servers. In *IPTComm '08: Proceedings of the 2nd International Conference on Principles, Systems and Applications of IP Telecommunications*, volume 5310/2008, pages 133–148, Heidelberg, Germany, October 2008.

[26] The MySQL Project. MySQL database server. `http://www.mysql.org`.

[27] The OpenSIPS Project. The open SIP server (OpenSIPS). `http://www.opensips.org`.

[28] The OpenSSL Project. The OpenSSL library. `http://www.openssl.org`.

[29] The OProfile Project. OProfile. `http://oprofile.sourceforge.net`.

[30] K. Kumar Ram, I. Fedeli, A. Cox, and S. Rixner. Explaining the impact of network transport protocols on SIP proxy performance. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 75–84, Austin, TX, April 2008.

[31] Light Reading. VoIP security: Vendors prepare for the inevitable. *VoIP Services Insider*, 5(1), January 2009.

[32] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison Wesley, 2000.

[33] E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347 (Proposed Standard), April 2006.

[34] R. Rivest. The MD5 Message-Digest Algorithm . RFC 1321 (Informational), April 1992.

[35] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320.

[36] S. Salsano, L. Veltri, and D. Papalilo. SIP security issues: the SIP authentication procedure and its processing load. *IEEE Network*, 16(6):38–44, Nov/Dec 2002.

[37] B. Schneier. *Applied Cryptography (2nd Edition)*. John Wiley and Sons, Inc., New York, NY, 1996.

[38] H. Schulzrinne. SIPd. `http://www.cs.columbia.edu/IRT/cinema`.

[39] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[40] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle. SIPstone-benchmarking SIP server performance. April 2002. `http://www.sipstone.com`.

[41] X. Wang, R. Zhang, X. Yang, X. Jiang, and D. Wijesekera. Voice pharming attack and the trust of VoIP. In *SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–11, Istanbul, Turkey, September 2008.

[42] C.P. Wright, E Nahum, D. Wood, J. Tracey, and E. Hu. SIP server performance on multicore systems. *IBM Journal of Research and Developement*, 54(1), February 2010.

[43] Y. Zeng and O. Cherkaoui. Performance study of COPS over TLS and IPsec secure session. In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, pages 133–144, Montreal, Canada, October 2002.

[44] L. Zhao, R. Iyer, S. Makineni, and L. Bhuyan. Anatomy and performance of SSL processing. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 197–206, Austin, TX, March 2005.

On TCP-based SIP Server Overload Control

Charles Shen and Henning Schulzrinne
Department of Computer Science, Columbia University
New York, NY 10027
{charles,hgs}@cs.columbia.edu

ABSTRACT

The Session Initiation Protocol (SIP) server overload management has attracted interest since SIP is being widely deployed in the Next Generation Networks (NGN) as a core signaling protocol. Yet all existing SIP overload control work is focused on SIP-over-UDP, despite the fact that TCP is increasingly seen as the more viable choice of SIP transport. This paper answers the following questions: is the existing TCP flow control capable of handling the SIP overload problem? If not, why and how can we make it work? We provide a comprehensive explanation of the default SIP-over-TCP overload behavior through server instrumentation. We also propose and implement novel but simple overload control algorithms without any kernel or protocol level modification. Experimental evaluation shows that with our mechanism the overload performance improves from its original zero throughput to nearly full capacity. Our work leads to the important general insight that the traditional notion of TCP flow control alone is incapable of managing overload for time-critical session-based applications, which would be applicable not only to SIP, but also to a wide range of other common applications such as database servers.

1. INTRODUCTION

The Session Initiation Protocol (SIP) [34] is an application layer signaling protocol for creating, modifying, and terminating media sessions in the Internet. SIP has been adopted by major standardization bodies including 3GPP, ITU-T, and ETSI as the core signaling protocol of Next Generation Networks (NGN) for services such as Voice over IP (VoIP), conferencing, Video on Demand (VoD), presence, and Instant Messaging (IM). The increasingly wide deployment of SIP has raised a requirement for SIP server overload management solutions [33]. SIP server can be overloaded for many reasons such as emergency-induced call volume, flash crowds generated by TV programs (e.g., American Idol), special events such as "free tickets to the third caller", or denial of service attacks.

Although a SIP server is an application server, the SIP server overload problem is distinct from other well-known application server such as HTTP overload because in the SIP architecture, multiple server hops are common. There are also many SIP application level retransmission timers, and there is a time-critical session completion requirement. SIP's built-in session rejection mechanism is known to be unable to manage overload [33] because it could cause the server to spend all cycles rejecting messages and result in congestion collapse. If, as often recommended, the rejected sessions are sent to a load-sharing SIP server, the alternative server will soon also be generating nothing but rejection messages, leading to a cascading failure. Hilt *et al.* [40, 41] articulate a SIP overload control framework based on augmenting the current SIP specification with application level feedback between SIP proxy servers. The feedback, which may be rate-based or window-based, pushes the burden of rejecting excessive sessions from the target server to its upstream servers and thus prevents the overload. Detailed SIP application level feedback algorithms and their effectiveness have been demonstrated by a number of researchers, e.g., Noel [27], Shen [37] and Hilt [19].

As far as we know, all existing SIP overload control design and evaluation focus on SIP-over-UDP, presumably because UDP is still the common choice for today's SIP operational environment. However, SIP-over-TCP is getting increasingly popular and seen as a more viable SIP transport choice for a number of reasons, such as the need for securing SIP signaling over TLS/TCP [1, 32, 34, 36] (There is also a newer TLS version - Datagram TLS, which runs over UDP, but its deployment popularity is not clear), support for message sizes exceeding the maximum UDP datagram size [34], facilitation of firewall and NATs traversal [28], and potentially overload control.

The SIP-over-TCP overload control problem differs in two main aspects from the SIP-over-UDP overload control problem. One is TCP's built-in flow control mechanism which provides an inherent, existing channel for feedback-based overload control. The other is the removal of many application layer retransmission timers that exacerbates the overload condition in SIP-over-UDP. Nahum *et al.* [9] have experimentally studied SIP performance and found that overload leads to congestion collapse for both SIP-over-TCP and SIP-over-UDP. Their focus, however, is not on overload control so they do not discuss why SIP-over-TCP congestion collapse happens or how to prevent it. Hilt *et al.* [19] have shown simulation results by applying application level feedback control to SIP servers with TCP-specific SIP timers but without including a TCP transport stack in the simulation.

This paper systematically addresses the SIP-over-TCP overload control problem through an experimental study and analysis. To the authors' knowledge, our paper is the first to provide a comprehensive answer to the following questions: why is there still congestion collapse in SIP-over-TCP despite the presence of the well-known TCP flow control mechanism and much fewer SIP retransmission timers? Is there a way we can utilize the existing TCP infrastructure to solve the overload problem without changing the SIP protocol specification as is needed for the UDP-based application level feedback mechanisms?

We find that the key reasons why TCP flow control feedback does *not* prevent SIP congestion collapse has to do with the session-based SIP load characteristics and the fact that the session needs to be established within the timeout threshold. Different messages in the message flow of the same SIP session arrive at different times from upstream and downstream SIP entities; start-of-session requests trigger all the remaining in-session messages and are therefore especially expensive. The transport level connection-based TCP flow control, without knowing the causal relationship among the messages, will admit too many start-of-session requests and result in a continued accumulation of in-progress sessions in the system, leading to large queuing delays. When that happens, the TCP flow control creates back pressure propagating to the session originators, adversely affecting their ability to generate messages that could complete existing sessions. In the meantime, SIP response retransmission still kicks in. The combined delayed message generation and processing as well as response retransmission lead to SIP-over-TCP congestion collapse.

Based on our observations, we propose a novel SIP overload control mechanisms within the existing TCP flow control infrastructure. To respect the distinction between start-of-session requests and other messages, we introduce the concept of *connection split*. To meet the delay requirements and prevent retransmission, we develop *smart forwarding* algorithms combined with *buffer minimization*. Our mechanisms contain only a single tunable parameter for which we provide a recommended value. Implementation of our mechanisms exploits existing Linux socket API calls and is extremely simple. It does not require any modifications at the kernel level, nor changes to the SIP or TCP specification.

We evaluate throughput, delay and fairness results of our mechanisms on a common Intel-based Linux testbed using the popular open source OpenSIPS server with up to ten upstream servers overloading the target server at over ten times the server capacity.

Our mechanism is best suited for the common case where the number of upstream servers overloading the target server at the same time is not excessively large, such as servers in the core networks of big service providers. But we also point out possible solutions when a large number of upstream servers overload a single target server, such as when numerous enterprise servers connect to the same server from a big service provider.

Our research leads to the important insight that the traditional notion of TCP flow control alone is insufficient in preventing congestion collapse for time-sensitive session-based loads, which cover a broad range of applications, e.g., from SIP servers to data center systems [42].

The remainder of this paper is structured as follows. Section 2 describes related work. Section 3 provides some background on SIP and TCP flow and congestion control. Section 4 describes the experimental testbed used for our experiments. Section 5 explains the SIP-over-TCP congestion collapse behavior. Section 6 and Section 7 develop and evaluate our overload control mechanism.

2. RELATED WORK

SIP overload falls into the broader category of application server overload where, in particular, web server overload control [7, 12, 48] has been studied extensively. Although most of the work on web server overload control uses a request-based workload model, Cherkasova and Phaal [6] presented a study using session-based workload, which is closer to our SIP overload study. However their mechanism uses the overloaded server to reject excessive loads, which is known to be insufficient for SIP [33].

A number of authors [9, 28, 31, 36] have measured SIP server performance over TCP, without discussing overload. The SIP server overload problem itself has received intensive attention only recently. Ejzak *et al.* [10] provided a qualitative comparison of the overload in PSTN SS7 signaling networks and SIP networks. Whitehead [45] described a protocol-independent overload control framework called GOCAP but its mapping to SIP is still being defined. Ohta [24] explored the approach of using a priority queueing and bang-bang type of overload control through simulation. Noel and Johnson [27] presented initial results of a rate-based SIP overload control mechanism. Sun *et al.* [39] proposed adding a front end SIP flow management system to conduct overload control including message scheduling, admission control and retransmission removal. Sengar [35] combined the SIP built-in backoff retransmission mechanism with a selective admittance method to provide server-side pushback for overload prevention. Hilt *et al.* [19] provided a side-by-side comparison of a number of overload control algorithms for a network of SIP servers, and also examined different overload control paradigms such as local, hop-by-hop and end-to-end overload control. Shen *et al.* [37] proposed three window-based SIP feedback control algorithms and compared them with rate-control algorithms. Except for [19], all of the above work on SIP overload control assumes UDP as the transport. Hilt *et al.* [19] present simulation of application level feedback overload control for SIP server with only TCP-specific SIP timers enabled, but their simulation does not include a TCP transport stack.

The basic TCP flow and congestion control mechanisms are documented in [22, 29]. Modifications to the basic TCP algorithm have been proposed to improve various aspects of TCP performance, such as start-up behavior [20], retransmission fast recovery [13], packet loss recovery efficiency [15, 25], or overall congestion control [2, 5]. There are also research efforts to optimize the TCP algorithm for more recent network architecture such as mobile and wireless networks [11, 47] and high-speed networks [17, 23], as well as additional work that focuses not on modifying TCP flow and congestion control algorithm itself, but on using dynamic socket buffer tunning methods to improve performance [8, 18]. Another category of related work focuses on routers, e.g., active buffer management [14, 26] and router buffer sizing [43]. Our work differs from all the above in that our metric is not the direct TCP throughput, but the application level throughput. Our goal is to explore the existing

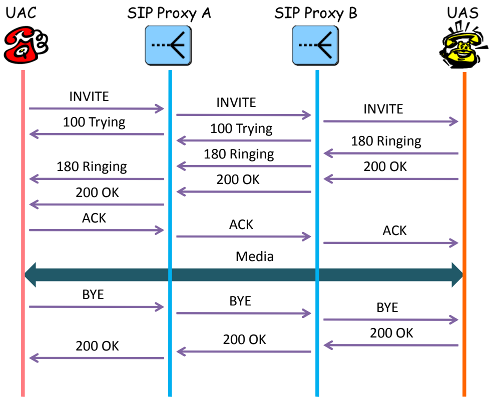


Figure 1: Basic SIP call flow

TCP flow control mechanism for application level overload management, without introducing TCP or kernel modifications.

There are also studies on TCP performance for real-time media, e.g., [3, 4, 44]. Our work, however, addresses the session establishment phase for real-time services, which has very different load characteristics.

3. BACKGROUND

3.1 SIP Overview

SIP defines two basic types of entities: User Agents (UAs) and servers. UAs represent SIP end points. SIP servers can be either registrar servers for location management, or proxy servers for message forwarding. SIP messages are divided into requests (e.g., INVITE and BYE to create and terminate a SIP session, respectively) and responses (e.g., 200 OK for confirming a session setup).

SIP message forwarding, known as proxying, is a critical function of the SIP infrastructure. Fig. 1 shows a typical message flow of stateful SIP proxying where all SIP messages are routed through the proxy with the SIP Record-Route option enabled. Two SIP UAs, designated as User Agent Client (UAC) and User Agent Server (UAS), represent the caller and callee of a multimedia session. The UAC wishes to establish a session with the UAS and sends an INVITE request to proxy A. Proxy A looks up the contact address for the SIP URI of the UAS and, assuming it is available, forwards the message to proxy B, where the UAS can be reached. Both proxy servers also send 100 Trying response to inform the upstream SIP entities that the message has been received. After proxy B forwards the message to the UAS. The UAS acknowledges receipt of the INVITE with a 180 Ringing response and rings the callee's phone. When the callee actually picks up the phone, the UAS sends out a 200 OK response. Both the 180 Ringing and 200 OK make their way back to the UAC. The UAC then generates an ACK request for the 200 OK. Having established the session, the media flows directly between the two endpoints. When the conversation is finished, the UAC "hangs up" and generates a BYE request that the proxy servers forward to the UAS. The UAS then responds with a 200 OK response which is forwarded back to the UAC.

SIP is an application level protocol on top of the transport layer. It can run over any common transport layer protocols, such as UDP, TCP and SCTP [38]. SIP defines quite a number of timers. One group of timers is for hop-to-hop message retransmissions in case a message is lost. These retransmission timers are not used when TCP is the transport because TCP already provides a reliable transfer. There is however a retransmission timer for the end-to-end 200 OK responses which is enabled even when using TCP transport, in order to accommodate circumstances where not all links in the path are using reliable transport. The 200 OK retransmission timer is shown in Fig. 2. The timer starts with $T_1 = 500\,ms$ and doubles until it reaches $T_2 = 4\,s$. From then on the timer value remains at T_2 until the total timeout period exceeds 32 s, when the session is considered to have failed. The UAC should generate an ACK upon receiving a 200 OK. The UAS cancels the 200 OK retransmission timer when it receives a corresponding ACK.

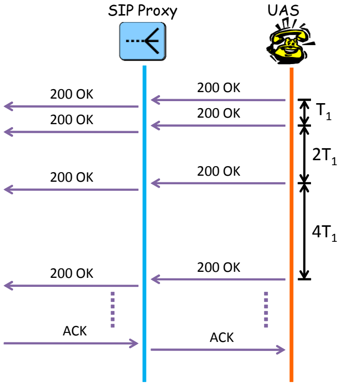
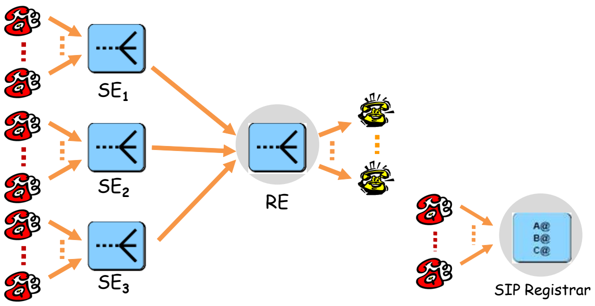


Figure 2: 200 OK retransmission

3.2 Types of SIP Server Overload

There are many causes to SIP overload, but the resulting SIP overload cases can be grouped into either of the two types: proxy-to-proxy overload or UA-to-registrar overload.



(a) proxy-to-proxy overload (b) UA-to-registrar overload

Figure 3: Types of SIP server overload

A typical proxy-to-proxy overload topology is illustrated in Fig. 3(a), where the overloaded proxy server is connected to a relatively small number of upstream proxy servers. The overloaded server in Fig. 3(a) is also referred to as a Receiving Entity (RE) and its upstream servers are also referred to as Sending Entities (SEs) [41]. One example of the proxy-to-proxy overload is a special event like "free tickets to the

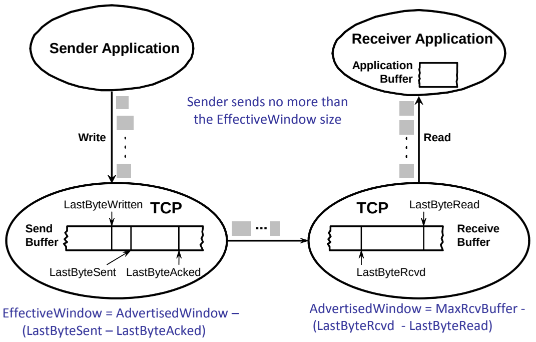

Sender Application
Receiver Application
Application Buffer
Sender sends no more than the EffectiveWindow size
Write
Read
LastByteWritten
TCP
TCP
LastByteRead
Send Buffer
Receive Buffer
LastByteSent
LastByteAcked
LastByteRcvd
EffectiveWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)
AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - LastByteRead)

Figure 4: TCP flow control

third caller", also known as flash crowds. Suppose RE is the service provider for a hotline. SE_1, SE_2 and SE_3 are three service providers that reach the hotline through RE. When the hotline is activated, RE is expected to receive a large call volume to the hotline from SE_1, SE_2 and SE_3 that far exceeds its usual call volume, potentially putting RE into overload.

The second type of overload, known as UA-to-registrar overload, occurs when a large number of UAs overload their next hop server. A typical example is avalanche restart, which happens when power is just restored after a mass power failure in a large metropolitan area and a huge number of SIP devices boot up trying to perform registration simultaneously. This paper only discusses the proxy-to-proxy overload problem.

3.3 TCP Window-based Flow Control Mechanism

TCP is a reliable transport protocol with its built-in flow and congestion control mechanisms. Flow control is exercised between two TCP end points. The purpose of TCP flow control is to keep a sender from sending so much data that overflows the receiver's socket buffer. Flow control is achieved by having the TCP receiver impose a receive window on the sender side indicating how much data the receiver is willing to accept at that moment; on the other hand, congestion control is the process of a TCP sender imposing a congestion window by itself to avoid congestion inside the network. Thus, a TCP sender is governed by both the receiver flow control window and sender congestion control window during its operation.

The focus of our work is on using TCP flow control since we are interested in the receiving end point being able to deliver transport layer feedback to the sending end point and we want to see how it could facilitate higher layer overload control. We illustrate the TCP flow control architecture in Fig. 4. A socket level TCP connection usually maintains a send buffer and a receive buffer at the two connection end points. The receiver application reads data from the receive buffer to its application buffer. The TCP receiver computes its current receive buffer availability as its advertised window to the TCP sender. The TCP sender never sends more data than an effective window size derived based on the receiver advertised window and data that has been sent but not yet acknowledged.

4. EXPERIMENTAL TESTBED AND METRICS

4.1 Server and Client Software

We evaluated the Open SIP Server (OpenSIPS) version 1.4.2 [30], a freely-available, open source SIP proxy server. OpenSIPS is a fork of OpenSER, which in turn is a fork of the SIP Express Router (SER) [21]. These sets of servers represent the *de facto* open source version of SIP server, occupying a role similar to that of Apache for web servers. We also implemented our overload control mechanisms on the OpenSIPS server.

We choose the widely used open source tool, SIPp [16] (May 28th 2009 release) to generate SIP traffic. We also make corrections to SIPp for our test cases. For example, we found that the SIPp does not trigger the 200 OK retransmission timer over TCP as required by the SIP specification, and therefore we added it.

4.2 Hardware, Connectivity and OS

The overloaded SIP RE server has 2 Intel Xeon 3.06 GHz processors with 4 GB RAM. However, for our experiments, we only use one processor because SIP performance under multiple processors or a multi-core processor is itself a topic that requires separate attention [46]. We use up to 10 machines for SEs, and up to 10 machines for UACs. All the SE and UAC machines either have 2 Intel Pentium 4 3.00 GHz processors with 1 GB memory or 2 Intel Xeon 3.06 GHz processors and 4 GB RAM. The server and client machines communicate over copper Gigabit or 100 Mbit Ethernet. The round trip time measured by the `ping` command between the machines is around 0.2 ms. More constrained link transmission conditions such as longer delays or explicit packet losses may be considered in future experiments.

All of our testbed machines run Ubuntu 8.04 with Linux kernel 2.6.24. The default TCP send buffer size is 16 KB and the default TCP receive buffer size is 85 KB. Since the Linux operating system uses about 1/4 of the socket receive buffer size for bookkeeping overhead, the estimated effective default receive buffer size is about 64 KB. In the rest of the paper we use the effective value to refer to receive buffer sizes. The SIP server application that we use allocates a default 64 KB application buffer.

Linux provides the `setsockopt` API call to allow applications to manipulate connection-specific send and receive socket buffer sizes. Linux also supports API calls that enable the applications to retrieve real-time status information about the underlying TCP connection. For example, using the `ioctl` call, the application can learn about the amount of unsent data currently in the socket send buffer.

4.3 Test Suite, Load Pattern and Performance Metrics

We wrote a suite of Perl and Bash scripts to automate running the experiments and analyzing results. Our test load pattern is the same as in Fig 1. For simplicity but without loss of generality, we do not include call holding time and media. That means, the UAC sends a BYE request immediately after sending an ACK request. In addition, we do not consider the time between the ringing and the actual pick-up of the phone. Therefore, the UAS sends a 200 OK response immediately after sending a 180 Ringing response. In order to facilitate the load generation for overload tests, we also introduced extra cryptographic functions to the authentication operations in the SIP sessions to contrain the default server capacity.

Our main performance metrics is the server throughput, i.e., number of sessions successfully set up per-second by receiving the ACK to 200 OK at UAS. We also examine a delay metrics similar to the Post Dial Delay (PDD) in PSTN networks, which roughly corresponds to the time from sending the first INVITE to receiving the 200 OK response. The combination of both throughput and delay metrics actually gives us the system goodput. A number of other metrics such as CPU utilization and server internal message processing rate are also used in explaining the results.

5. DEFAULT SIP OVER TCP OVERLOAD PERFORMANCE

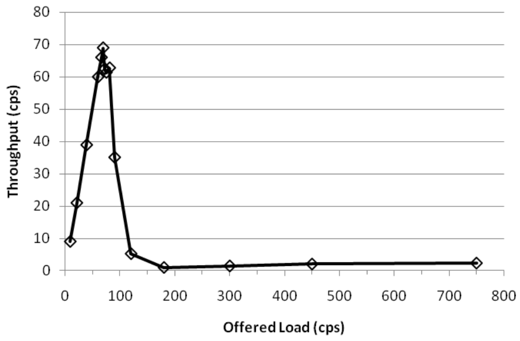


Figure 5: Default SIP-over-TCP throughput

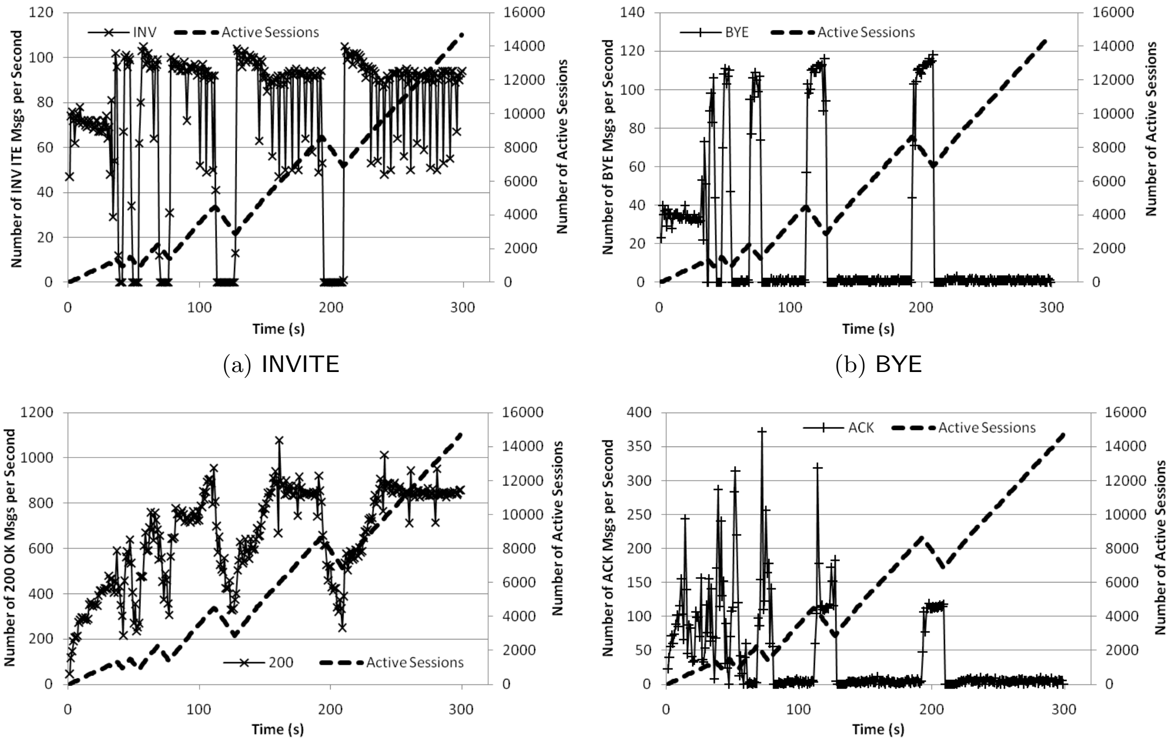
We start our investigation with a single SE - single RE testbed with all out-of-the-box configurations. The SE is connected to a machine acting as many UACs that generate the desired rate of SIP requests; the RE is connected to a machine acting as many UASes that receive and process SIP requests. The throughput results in calls per second (cps) of this testbed are shown in Fig. 5. It can be seen that the throughput immediately collapses as the load approaches and exceeds the server capacity at around 65 to 70 cps. In this section, we explore the detailed causes of this behavior through server instrumentation.

We examine a particular run at a load of 150 cps which is about 2.5 times the server capacity. Fig. 6 depicts the per second message processing rate. The four figures show INVITE, BYE, 200 OK and ACK, respectively. It should be noted that the number of 180 Ringings, not shown in these figures, basically follows the number of INVITEs processed, because the UAS is not overloaded and can always deliver responses to RE. For the same reason, the number of 200 OKs to BYEs which are also not shown, follows the number of BYEs. Along with the individual message processing rates, Fig. 6 also includes the current number of active sessions in the RE. The active sessions are those sessions that have been started by an INVITE but have not yet received a BYE. Since the call holding time is zero, in an ideal situation, any started sessions should be terminated immediately, leaving no session outstanding in the system. In a real system, the number of active sessions could be greater than zero. The larger the number of such in-progress sessions, the longer the delay that those sessions will experience.

Fig. 6 indicates that 200 OK retransmission happens almost immediately as the test starts, which means the end-to-end round trip delay immediately exceeds 500 ms. This is caused by the large buffers at the different stages of the network system, which allow too many sessions to be accepted. The SIP session load is not atomic. The INVITE request is always first introduced into the system and then come the responses and follow-up ACK and BYE requests. When too many INVITEs are admitted to the system, the BYE generation rate cannot keep up with the INVITEs, resulting in a large number of active sessions in the system and also a large number of messages queued in various stages of the buffers. These situations translate to prolonged delays in getting the ACK to 200 OK to the UAS. More specifically, assuming the server's capacity is 65 cps, if the sessions are indeed atomic, each session will take a processing time of 15.4 ms. In order to avoid 200 OK retransmission, the end-to-end one-way delay cannot exceed 250 ms, corresponding to a maximum of about 16 active sessions in the system. Factoring in the non-atomic nature of the session load, this maximum limit could be roughly doubled to 32. But with the default system configuration, we have a 16 KB TCP socket send buffer, and 64 KB socket receive buffer, as well as 64 KB SIP server application buffer. Considering an INVITE size of around 1 KB, this configuration means the RE can be filled with up to 130 INVITEs at one time, much larger than the threshold of 32. All these INVITEs contribute to active sessions once admitted. In the experiment, we see the number of active sessions reaches 49 at second 2, immediately causing 200 OK retransmissions. 200 OK retransmissions also trigger re-generated ACKs, adding more traffic to the network. This is why during the first half of the time period in Fig. 6, the number of ACKs processed is higher than the number of INVITEs and BYEs processed. Eventually the RE has accumulated too many INVITEs both in its receive buffer and application buffer. So its flow control mechanism starts to advertise a zero window to the SE, blocking the SE from sending additional INVITE requests. Subsequently the SE stops processing INVITE requests because of the send block to the RE. This causes SE's own TCP socket receive buffer and send buffer to get full as well. The SE's flow control mechanism then starts to advertise a zero window to UAC. This back pressure on UAC prevents the UAC from sending anything out to the SE. Specifically, the UAC can neither generate new INVITE requests, nor generate more ACK and BYEs, but it could still receive responses. When this situation happens, retransmitted 200 OKs received can no longer trigger retransmitted ACKs. Therefore, the number of ACKs processed in the later half of the graph does not exceed the number of INVITEs or BYEs. The number of ACKs actually becomes similar to the number of BYEs because BYEs and ACKs are generated together at the same time in our workload.

It can further be seen that under the default settings, the INVITE and BYE processing tends to alternate with gradually increasing periods as the test proceeds. During each period, the INVITE portion is increasingly larger than the BYE portion. Since the number of active sessions always increases with INVITE processing, and decreases with BYE processing, those processing patterns lead to the continued growth of the number of active sessions in the RE and exacerbate the situation.

In addition to observing the per-second message process-



(a) INVITE

(b) BYE

(c) 200 OK

(d) ACK

Figure 6: RE message processing rates and number of active sessions in default SIP-over-TCP test

ing rate at RE, we also confirm the behavior from the total number of messages processed at the UAS, along with the number of active sessions at RE as in Fig. 7. Note that the numbers of INVITEs received, 180 Ringing and initial 200 OK (not retransmissions) messages sent are the same, because 180 Ringing and 200 OK are generated by UAS immediately upon receiving an INVITE. Similarly the number of ACK, BYE, and 200 OK to BYEs are the same, because ACK and BYE are generated at the same time at the UAC and 200 OK to BYE is immediately generated upon receiving BYE at the UAS. In Fig. 7, initially between 0 and the 38th second, the numbers of ACKs and BYEs received are roughly half of the total INVITEs received. Therefore, the number of active sessions in the RE and the number of ACKs received at the UAS are roughly the same. Then RE enters the abnormal INVITE processing and BYE processing alternating cycle. During the period when RE is processing ACKs and BYEs, the number of active sessions decreases. During the period when RE is processing INVITEs, no ACKs are forwarded, so the number of ACKs remains constant.

200 OK retransmission starts at second 2. The total period of 200 OK retransmission lasts 32 seconds for each individual session, therefore the expiration of the first session that has exhausted all its 200 OK retransmissions without receiving an ACK happens at the 34th second. The actual 200 OK retransmission timeout we see from Fig. 7 is at the 66th second. The difference between the 66th and 34th second is 32 seconds, which is a configured maximum period UAS waits to receive the next message in sequence, in this case the ACK corresponding to the 200 OK.

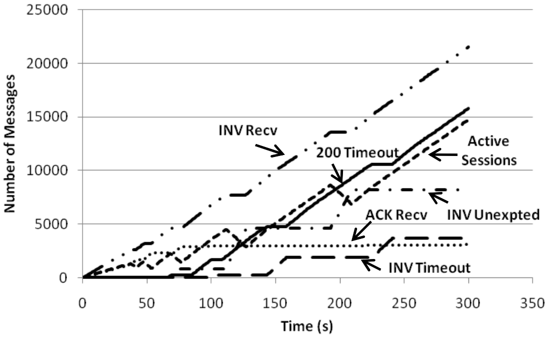
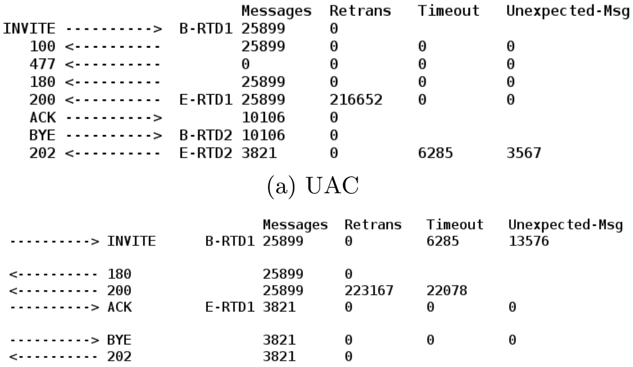


Figure 7: Total number of messages processed at UAS and number of active sessions at RE

Starting from the 69th second, we see a category of messages called *INVITE Unexpected*. These are ACKs and BYEs that arrive after the admitted sessions have already timed out at the UAS. These ACKs and BYEs without a matching session also create session states at the SIPp UAS, which normally expect a session message sequence beginning with an INVITE. Since those session states will not receive other normal in-session messages, at the 101th second, or after 32 seconds of UAS receive timeout period, those session states start to time out, reflected in the figure as the *INVITE Timeout* curve. Finally, a very important overall observation from Fig. 7 is that at a certain point, the 77th second, the number of timely received ACKs virtually stopped growing, causing the throughput to drop to zero.



```
                              Messages  Retrans   Timeout   Unexpected-Msg
INVITE ----------> B-RTD1 25899     0
   100 <----------             25899     0         0         0
   477 <----------             0         0         0         0
   180 <----------             25899     0         0         0
   200 <---------- E-RTD1 25899     216652    0         0
   ACK ---------->             10106     0
   BYE ----------> B-RTD2 10106     0
   202 <---------- E-RTD2 3821      0         6285      3567
```

(a) UAC

```
                                 Messages  Retrans   Timeout   Unexpected-Msg
----------> INVITE     B-RTD1 25899     0         6285      13576

<---------- 180                25899     0
<---------- 200                25899     223167    22078
----------> ACK        E-RTD1 3821      0         0         0

----------> BYE                3821      0         0         0
<---------- 202                3821      0
```

(b) UAS

Figure 8: Screen logs in default SIP-over-TCP test

We also show the final screen logs at the UAC and UAS sides for the test with default configurations in Fig. 8, where status code 202 is used instead of 200 to differentiate the 200 OK to BYE from the 200 OK to INVITE. Earlier in this section we have explained the 200 OK retransmissions, 200 OK timeouts, INVITE timeouts, and INVITEs unexpected messages. We can see that among the 25,899 INVITEs received at the UAS side, 22,078 eventually time out and only 3,821 receive the final ACK. The UAC actually sends out a total of 10,106 ACKs and BYEs. The remaining 6,285 ACKs and BYEs are eventually delivered to UAS but are too late when they arrive, therefore those BYEs do not trigger 202 OK and we see 6,285 202 OK timeouts at the UAC. At the UAS side, those 6,285 ACKs and BYEs establish abnormal session states and eventually time out after the 32 s receive timeout for INVITE. The unexpected messages at the UAC side are 408 Send Timeout messages triggered at the SIP servers for the BYEs that do not hear a 202 OK back. Note that the number of those messages (3,567) is smaller than the exact number of BYEs that do not receive 202 OK (6,285). This is because the remaining 2,718 408 Send Timeout messages arrive after the 202 OK receive timeout and therefore those messages were simply discarded and not counted in the screen log.

Finally, we also measure the PDD and find that even without considering whether ACKs are delivered successfully, 73% of the INVITEs have PDDs between 8 and 16 seconds, which are most likely beyond the human interface acceptability limit. Another 24% have PDDs between 4 to 8 seconds, which might be close to the acceptable limit.

6. SIP-OVER-TCP OVERLOAD CONTROL MECHANISM DESIGN

From the SIP-over-TCP congestion collapse, we learned a key lesson that we must limit the number of INVITEs we can admit to avoid too many active sessions accumulating in the system. For all admitted INVITEs, we need to make sure the rest of the session messages complete within finite delay. In this section, we propose specific approaches to address these issues, namely *connection split*, *buffer minimization*, and *smart forwarding*.

6.1 Connection Split and Buffer Minimization

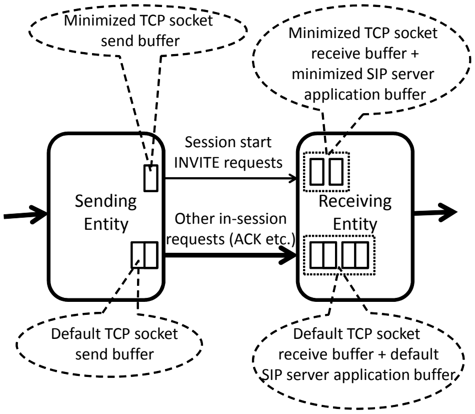

Minimized TCP socket send buffer
Minimized TCP socket receive buffer + minimized SIP server application buffer
Session start INVITE requests
Sending Entity
Other in-session requests (ACK etc.)
Receiving Entity
Default TCP socket send buffer
Default TCP socket receive buffer + default SIP server application buffer

Figure 9: ECS + BM

First, it is clear that we only want to limit INVITEs but not non-INVITEs because we do not want to drop messages for sessions already accepted. In order to have a separate control of INVITEs and non-INVITE messages, we split the TCP connection from SE to RE into two, one for INVITE requests, and the other for all other requests. In other words, the RE will listen on two TCP connections, and the SE makes sure that it will send all INVITEs to one connection and all non-INVITEs to the other connection. Second, in order to limit the number of INVITEs in the system and minimize delay, we minimize the total system buffer size between the SE and the RE for the INVITE connection, which includes three parts: the SE TCP socket send buffer, the RE TCP socket receive buffer and the RE SIP server application buffer. We call the resulting mechanism *Explicit Connection Split + Buffer Minimization* (ECS+BM) and illustrate it in Fig. 9.

We find, however, although ECS+BM effectively limits the number of INVITEs that could accumulate at the RE, the resulting throughput differs not much from that of the default configuration. The reason is that, since the number of INVITEs SE receives from UAC remains the same and the INVITE buffer sizes between SE and RE are minimized, the INVITE pressure merely moves a stage back and accumulates at the UAC-facing buffers of the SE. Once those buffers, including the SE receive buffer and SE SIP server application buffer, have been quickly filled up, the system delay dramatically increases. Furthermore, the UAC is then blocked from sending to SE and unable to generate ACKs and BYEs, causing the number of active sessions in the RE to skyrocket. In conclusion, ECS+BM by itself is insufficient in preventing overload.

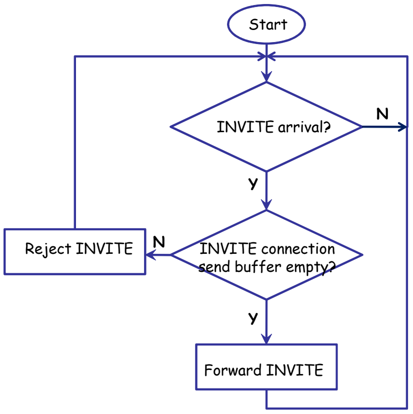


Figure 10: Smart forwarding for ECS

6.2 Smart Forwarding

In order to release, rather than pushing back the excessive load pressure present in the ECS+BM mechanism, we introduce the *Smart Forwarding* (SF) algorithm as shown in Fig. 10. This algorithm is enforced only for the INVITE connection. When an INVITE arrives, the system checks whether the current INVITE connection send buffer is empty. If yes, the INVITE is forwarded; otherwise the INVITE is rejected with an explicit SIP rejection message. This algorithm has two advantages: first, although we can choose any send buffer length threshold value for rejecting an INVITE, the decision to use the emptiness criterion makes the algorithm parameter-free; second, implementation of this algorithm is especially easy in Linux systems because the current send buffer occupancy can be retrieved by a simple standard `ioctl` call.

Our resulting mechanism is then ECS+BM+SF. We evaluate its performance on our testbed from light to heavy overload and find it achieving nearly full system capacity all the time. Due to space limitation, we do not present the results of the ECS+BM+SF here, but discuss in more detail an even simpler mechanism developed based on it called ICS+BM+SF.

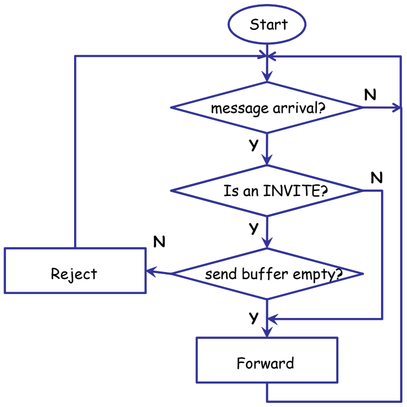


Figure 11: Smart forwarding for ICS

6.3 Implicit Connection Split, Buffer Minimization and Smart Forwarding (ICS+BM+SF)

Our results show that the ECS+BM+SF mechanism is very effective. Even in high overload, the RE contains only a few active sessions all the time, and achieves full capacity. The only inconvenience is that it requires to establish two separate connections for INVITEs and non-INVITEs. But if the server is never backlogged, the queue size for both INVITE and non-INVITE request connections should be close to zero. In that case, the dedicated connection for non-INVITE requests does not require the default large buffer setting either. We therefore decide to merge the two split connections back into one but still keep the minimized SE send buffer, RE receive buffer and application buffer settings. We also need to revise our *smart forwarding* algorithm accordingly, as in Fig. 11. Since there is only a single request connection now, the algorithm performs an additional check for INVITE requests and rejects it if the send buffer is non-empty. Otherwise, the INVITE is forwarded. All non-INVITE requests are always forwarded. Although the revised mechanism no longer requires a dedicated connection for INVITEs, it treats INVITEs and non-INVITEs differently. Therefore, we call this revised mechanism *Implicit Connection Split* (ICS) as opposed to the previous ECS mechanism.

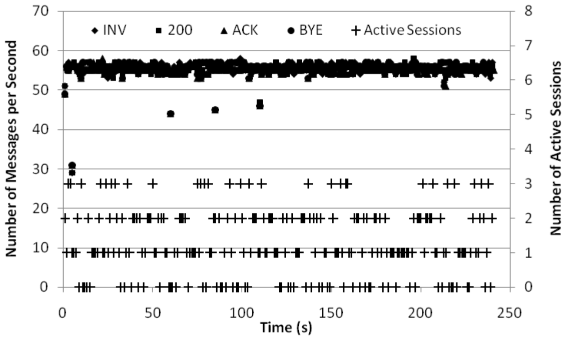
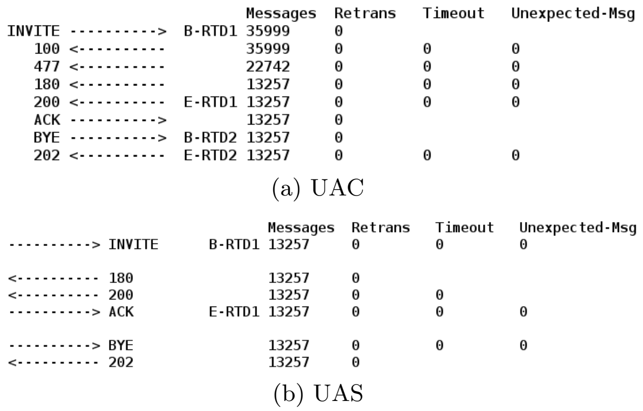


Figure 12: RE message processing rates with ICS+MB+SF

We evaluate the resulting ICS+BM+SF mechanism and compare its performance with the default configuration in the same scenario as in Section 5 with one SE overloading an RE at an offered load of 2.5 times the server capacity. Fig. 12 shows the average message processing rate and the number of active sessions in the RE. We can see how this figure differs dramatically from Fig. 6. Here, the values of INVITE, 200 OK, ACK, and BYE processing rate overlap most of the time, which explains why the number of active sessions remains extremely low, between 0 and 3, all the time. Furthermore, from the overall UAC and UAS screen logs in Fig. 13, we see that among the 35,999 INVITEs that are generated, 22,742 of them are rejected by the *smart forwarding* algorithm. The remaining 13,257 sessions all successfully get through, without triggering any retransmission or unexpected messages - a sharp contrast to Fig. 8. The good performance is also shown by the PDDs. We find that over 99.8% of the sessions have a delay value smaller than 30 ms, far smaller than the 500 ms 200 OK retransmission threshold. Finally, the system achieves full capacity as confirmed by the full CPU utilization observed at the RE.

6.4 Parameter Tuning

Our ICS+BM+SF mechanism in section 6.3 contains three



```
                           Messages  Retrans   Timeout   Unexpected-Msg
INVITE ---------->  B-RTD1 35999     0
   100 <----------         35999     0         0         0
   477 <----------         22742     0         0         0
   180 <----------         13257     0         0         0
   200 <----------  E-RTD1 13257     0         0         0
   ACK ---------->         13257     0
   BYE ---------->  B-RTD2 13257     0
   202 <----------  E-RTD2 13257     0         0         0
```

(a) UAC

```
                                Messages  Retrans   Timeout   Unexpected-Msg
----------> INVITE      B-RTD1 13257     0         0         0

<---------- 180                 13257     0
<---------- 200                 13257     0         0
----------> ACK         E-RTD1 13257     0         0         0

----------> BYE                 13257     0         0         0
<---------- 202                 13257     0
```

(b) UAS

Figure 13: Screen logs with ICS+MB+SF

minimized buffer sizes: the SE send buffer at 2 KB, RE receive buffer at 1 KB and RE application buffer at 1,200 bytes. We conducted extensive tests to explore the impact of tuning these three buffer sizes, and we summarize the results in this section.

First, we find that since the RE receive buffer and RE application buffer are connected in series, they do not have to be minimized at the same time. Minimizing either one of them achieves similar near-capacity throughput. However, recall that enlarging either RE buffer size could hold messages in the RE and increase queuing delay. For example, we plot the PDD distribution for four test cases in Fig. 14. Two of those cases compare the delay when the RE application buffer is set to 2 KB vs. the default 64 KB, while the RE receive buffer is at its default value of 64 KB. Most of the delays in the small application buffer case are below 375 ms, and as a result we observe no 200 OK retransmissions at the UAS side. In the large application buffer case, however, nearly 70% of the sessions experience a PDD between 8 seconds and 32 seconds, which will most likely be hung up by the caller even if the session setup messages could ultimately complete. Not surprisingly, we also see a large number of 200 OK retransmissions in this case.

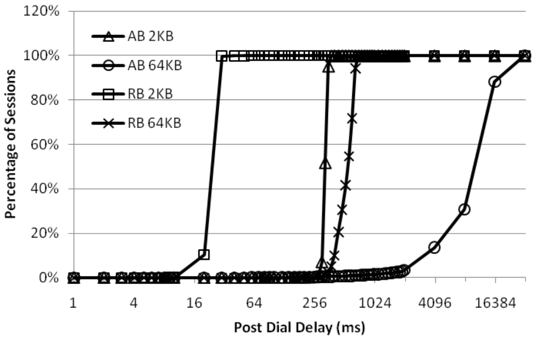


Figure 14: PDD comparison for RE side buffer tuning (AB: Application Buffer; RB: Receive Buffer)

The other two cases in Fig. 14 compare the PDD when the receive buffer is set to 2 KB vs. the default 64 KB, while the application buffer is at its default value of 64 KB. In the small receive buffer case, over 99.7% of the sessions have a PDD below 30 ms, and there is certainly no 200 OK retransmissions at the UAS side. In the larger receive buffer case, about 30% of the sessions have a PDD below 480 ms, and the remaining 70% between 480 ms and 700 ms. Since a large number of sessions experienced a round trip delay exceeding 500 ms, we see quite a number of 200 OK retransmissions at the UAS side, too. Therefore, tuning the receive buffer is preferable over tuning the application buffer, which matches the intuition: the receive buffer is closer to the SE and produces more timely transport feedback than the application buffer does.

Second, we find that the SE send buffer size actually does not have to be minimized. This can be attributed to our *smart forwarding* algorithm which already prevents excessive non-INVITE messages from building up in the system. Combined with minimized buffers at the RE, our mechanism minimizes the number of active sessions in the system, which means there will always be only a small number of messages in the SE send buffer.

In summary, our investigation confirms that the only essential tunable parameter of the ICS+BM+SF mechanism is the RE receive buffer size. Therefore, we finally obtain our extremely simple ICS+BM+SF mechanism as illustrated in Fig. 15.

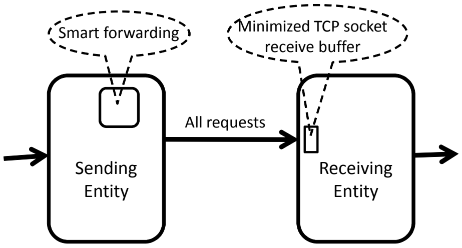


Figure 15: ICS+BM+SF

7. OVERALL PERFORMANCE OF OUR SIP-OVER-TCP OVERLOAD CONTROL MECHANISMS

In this section we evaluate the overall performance of our ICS+BM+SF mechanism as shown in Fig. 15. To demonstrate scalability, we test on three scenarios with 1 SE, 3 SEs and 10 SEs, respectively.

7.1 Overall Throughput and PDD

Fig. 16 illustrates the throughput with and without our control mechanism in the three test scenarios with varying number of SEs and an offered load up to over 10 times the capacity. The RE receive buffer was set to 2 KB and the SE send buffer and RE application buffer remain at their default values. As we can see, in all test runs with our control mechanisms, the overload throughput maintains at close to the server capacity, even in the most constrained case with 10 SEs and a load of 750 cps. Moreover, we observe no single 200 OK retransmissions in any of those tests.

We further compare the tests with different number of SEs. Fig. 17 shows that the numbers of active sessions in RE for the three scenarios roughly correspond to the ratio of

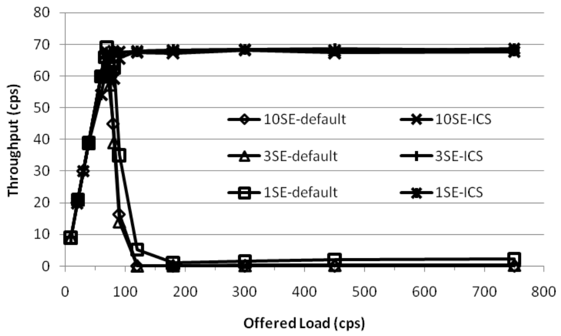


Figure 16: Overall throughput of SIP-over-TCP: with and without our overload control mechanism

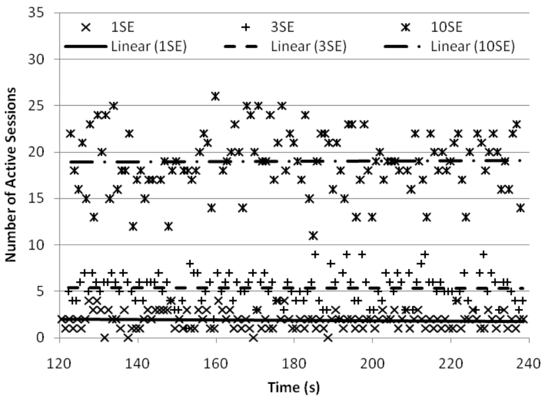


Figure 17: Number of active sessions in RE in scenarios with varying number of SEs

the numbers of SEs (1:3:10), as would be expected because in our testbed configuration each SE creates a new connection to the RE and is allocated a new set of RE buffers. Increased number of active sessions causes longer PDDs, as demonstrated in Fig. 18, where the overall trend and the 50 percentile values match the 1:3:10 ratio pretty well.

Fig. 17 and Fig. 18 also imply that if the number of SEs keeps increasing, the system will eventually still accumulate an undesirably large number of active sessions. The PDD will also exceed the response retransmission timer value to cause 200 OK retransmissions.

Thus, our mechanism is most applicable to cases where the number of SEs is reasonably small, which however, does cover a fairly common set of realistic SIP server overload scenarios. For example, there are typical national service providers deploying in total hundreds of core proxy and edge proxy servers in a hierarchical manner. The resulting server connection architecture leaves each server with a few to dozens of upstream servers.

7.2 RE Receive Buffer Tuning

The only tunable parameter in our mechanism is the RE receive buffer size. We explore the impact of this parameter under the most constrained case where there are 10 SEs with a total load of 750 cps in Fig. 19. It is not surprising that the receive buffer size cannot be too small because that will cause a single message to be sent and read in multiple

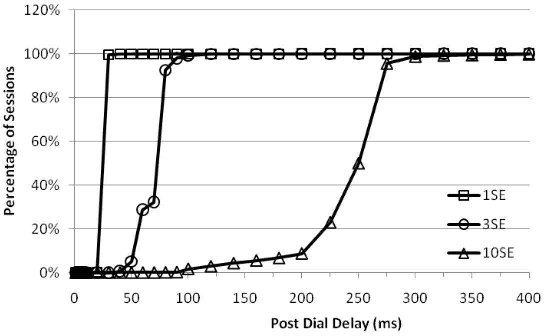


Figure 18: PDD in scenarios with varying number of SEs

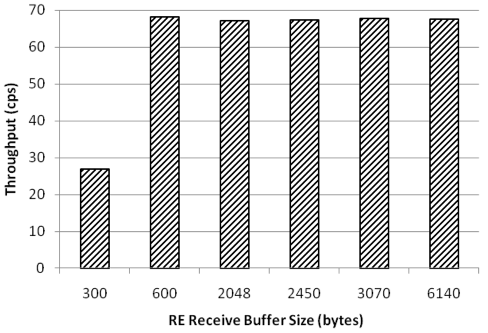


Figure 19: Impact of RE receive buffer size on Throughput

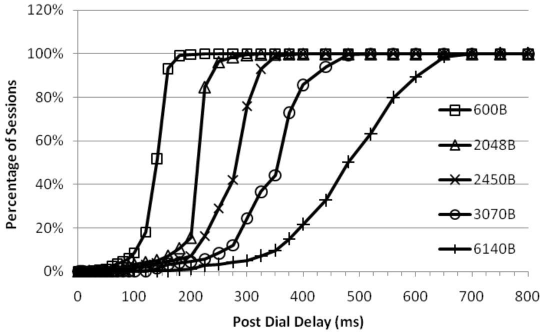


Figure 20: Impact of RE receive buffer size on PDD

segments. After exceeding a certain threshold, the receive buffer does not make difference in overload throughput, but the smaller the buffer is, the lower the PDD, as shown in Fig. 20. The PDD is roughly the same as round trip delay. If the round trip delay exceeds 500 ms, we will start to see 200 OK retransmissions, as in the cases where the receive buffer is larger than 3,070 bytes.

Overload control algorithms are meant to kick in when overload occurs. In practice, a desirable feature is to require no explicit threshold detection about when the overload control algorithm should be activated, because that always introduces additional complexity, delay and inaccuracy. If we keep our overload control mechanism on regardless of the load, then we should also consider how our mechanism could affect the system *underload* performance. We find that in general our mechanisms have a pretty satisfactory underload performance, meaning the throughput matches closely with a below-capacity offered load as shown in Fig. 16, although in some corner cases ICS's underload performance is not as good as ECS because ICS tends to be more conservative and reject more sessions.

Overall, in order to scale to as many SEs as possible yet minimizing the PDD, we recommend an RE receive buffer size that holds roughly a couple of INVITEs.

7.3 Fairness

All our above tests with multiple SEs assume each SE receiving the same request rate from respective UACs, in which case the throughput for each UAC is the same. Now we look at the situation where each SE receives different request rates, and measure the fairness property of the achieved throughput.

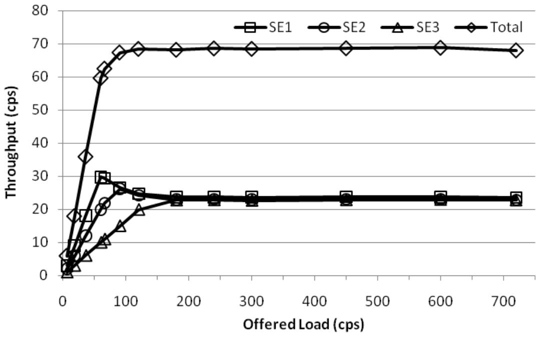


Figure 21: Throughput: three SEs with incoming load ratio 3:2:1

Fig. 21 shows the throughput of a 3 SE configuration with the incoming offered load to the three SEs distributed at a 3:2:1 ratio. As we can see, when the load is below total system capacity, the individual throughputs via each SE follow the offered load at the same 3:2:1 ratio closely. At light to moderate overload until 300 cps, the higher load sources have some advantages in competing RE resources. At higher overload above 300 cps, each SE receives a load that is close to or higher than the server capacity. The advantages of the relatively higher load SEs are diminishing, and the three SEs basically deliver the same throughputs to their corresponding UACs.

Shen *et al.* [37] define two types of fairness for SIP server overload: *service provider-centric* fairness and *end user-centric* fairness. The former allocates the same portion of the overloaded server capacity to each upstream server; the latter allocates the overloaded server capacity in proportion to the upstream servers' original incoming load. Our results show that the system achieves *service provider-centric* fairness at heavy overload. Obtaining *end user-centric* fairness during overload is usually more complicated; some related techniques are discussed in [37].

7.4 Additional Discussions

During our work with OpenSIPS, we also discover subtle software implementation flaws or configuration guidelines. For example, an SE could block on sending to an overloaded RE. Thus, if there are new requests coming from the same server at the upstream of the SE but are destined to other REs that are not overloaded, those new requests cannot be accepted either. This head-of-line blocking effect is clearly a flaw that is hardly noticeable unless we conduct systematic TCP overload tests.

Another issue is related to the OpenSIPS process configuration. OpenSIPS employs a multi-process architecture and the number of child processes is configurable. Earlier work [36] with OpenSIPS has found that configuring one child process yields an equal or higher maximum throughput than configuring multiple child processes. However, in this study we find that when overloaded, the existing OpenSIPS implementation running over TCP with a single child process configuration could lead to a deadlock between the SE and RE servers. Therefore, we use multiple child processes for this study.

8. CONCLUSIONS

We experimentally evaluated default SIP-over-TCP overload performance using a popular open source SIP server implementation on a typical Intel-based Linux testbed. Through server instrumentation, we found that TCP flow control feedback cannot prevent SIP overload congestion collapse because of lack of application context awareness at the transport layer for session-based load with real-time requirements. We develop novel mechanisms that effectively use existing TCP flow control to aid SIP application level overload control. Our mechanism has three components: the first is *connection split* which brings a degree of application level awareness to the transport layer; the second is a parameter-free *smart forwarding* algorithm to release the excessive load at the sending server before they reach the receiving server; the third is minimization of the essential TCP flow control buffer - the socket receive buffer, to both enable timely feedback and avoid long queueing delay. Implementation of our mechanisms is extremely simple without requiring any kernel or protocol level modification. Our mechanisms work best for the SIP overload scenarios commonly seen in core networks, where a small to moderate number of SEs may simultaneously overload an RE. For other scenarios where a large number of SEs overload the RE, deploying our mechanism will still improve performance, but the degree of effectiveness is inherently constrained by the per-connection TCP flow control mechanism itself. Since each SE adds to the number of connections and subsequently to the total size of allocated connection buffers at the RE, as the buffer size accumulates, so does the delay. Indeed, the solution to this numerous-SE-single-RE overload problem may ultimately require a shift from the current push-based model to a poll-based model. Specifically, instead of allowing all the SEs to send, the RE may advertise a zero TCP window to most of the SEs and open the windows only for those SEs that the RE is currently polling to accept loads. Future work is needed in this area.

Our study sheds light both at software level and conceptual level. At the software level, we discover implementation flaws for overload management that would not be noticed without conducting a systematic overload study, even though our evaluated SIP server is a mature open source server. At the conceptual level, our results suggest an augmentation to the long-held notion of TCP flow control: the traditional TCP flow-control alone is incapable of handling SIP-like time-sensitive session-based application overload. The conclusion may be generalized to a much broader application space that share similar load characteristics, such as database systems. Our proposed combined techniques including *connection split*, *smart forwarding* and *buffer minimization* are key elements to make TCP flow control actually work for managing overload of such applications.

9. ACKNOWLEDGEMENT

The authors would like to acknowledge NTT for funding this project and Dr. Arata Koike for useful discussions. We would also like to thank the anonymous reviewers for the helpful comments.

10. REFERENCES

[1] SIP forum. `http://www.sipforum.org`.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, Apr. 1999. Updated by RFC 3390.

[3] A. Argyriou. Real-time and rate-distortion optimized video streaming with TCP. *Image Commun.*, 22(4):374–388, 2007.

[4] S. Baset, E. Brosh, V. Misra, D. Rubenstein, and H. Schulzrinne. Understanding the behavior of TCP for real-time CBR workloads. In *Proc. ACM CoNEXT '06*, pages 1–2, Lisboa, Portugal, Dec. 2006.

[5] L.S. Brakmo and L.L. Peterson. TCP vegas: end to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, Oct. 1995.

[6] L. Cherkasova and P. Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, 51(6):669–685, 2002.

[7] M. Colajanni, V. Cardellini, and P. Yu. Dynamic load balancing in geographically distributed heterogeneous web servers. In *ICDCS '98: Proceedings of the 18th International Conference on Distributed Computing Systems*, page 295, Amsterdam, The Netherlands, May 1998.

[8] T. Dunigan, M. Mathis, and B. Tierney. A TCP tuning daemon. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–16, Baltimore, Maryland, Nov. 2002.

[9] E. Nahum and J. Tracey and C. Wright. Evaluating SIP server performance. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 349–350, San Diego, California, Jun. 2007.

[10] R. Ejzak, C. Florkey, and R. Hemmeter. Network overload and congestion: A comparison of ISUP and SIP. *Bell Labs Technical Journal*, 9(3):173–182, Nov. 2004.

[11] H. Elaarag. Improving TCP performance over mobile networks. *ACM Comput. Surv.*, 34(3):357–374, 2002.

[12] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *Proceedings of the 13th international conference on World Wide Web*, pages 276–286, New York, New York, May 2004.

[13] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, Apr. 2004.

[14] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug. 1993.

[15] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. An Extension to the Selective Acknowledgement (SACK) Option for TCP. RFC 2883, Jul. 2000.

[16] R. Gayraud and O. Jacques. SIPp. `http://sipp.sourceforge.net`.

[17] S. Ha, I. Rhee, and L. Xu. Cubic: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, 2008.

[18] G. Hasegawa, T. Terai, T. Okamoto, and M. Murata. Scalable socket buffer tuning for high-performance web servers. In *Ninth International Conference on Network Protocols*, pages 281–289, Riverside, California, Nov. 2001.

[19] V. Hilt and I. Widjaja. Controlling overload in networks of SIP servers. In *IEEE International Conference on Network Protocols (ICNP)*, pages 83–93, Orlando, Florida, Oct. 2008.

[20] J. Hoe. Improving the start-up behavior of a congestion control scheme for TCP. In *SIGCOMM' 96*, pages 270–280, Palo Alto, California, 1996.

[21] IPTel.org. SIP express router (SER). `http://www.iptel.org/ser`.

[22] V. Jacobson. Congestion avoidance and control. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 314–329, Stanford, California, Aug. 1988.

[23] D. Kliazovich, F. Granelli, and D. Miorandi. Logarithmic window increase for TCP westwood+ for improvement in high speed, long distance networks. *Computer Networks*, 52(12):2395–2410, 2008.

[24] M. Ohta. Overload Protection in a SIP Signaling Network. In *International Conference on Internet Surveillance and Protection*, Cote d'Azur, France, Aug. 2006.

[25] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. RFC 2018, Oct. 1996.

[26] R. Morris. Scalable TCP congestion control. In *Proc. IEEE INFOCOM 2000*, pages 1176–1183, Tel-Aviv, Israel, Mar. 2000.

[27] E. Noel and C. Johnson. Initial simulation results that analyze SIP based VoIP networks under overload. In *ITC*, pages 54–64, Ottawa, Canada, Jun. 2007.

[28] K. Ono and H. Schulzrinne. One server per city: Using TCP for very large SIP servers. In *IPTComm '08: Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, volume 5310/2008, pages 133–148, Oct. 2008.

[29] J. Postel. Transmission Control Protocol. RFC 793, Sep. 1981. Updated by RFC 3168.

[30] The OpenSIPS Project. `http://www.opensips.org`.

[31] K. Kumar Ram, I. Fedeli, A. Cox, and S. Rixner. Explaining the impact of network transport protocols on SIP proxy performance. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 75–84, Austin, Texas, Apr. 2008.

[32] Light Reading. VoIP security: Vendors prepare for the inevitable. *VoIP Services Insider*, 5(1), Jan. 2009.

[33] J. Rosenberg. Requirements for Management of Overload in the Session Initiation Protocol. RFC 5390, Dec. 2008.

[34] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Jun. 2002.

[35] H. Sengar. Overloading vulnerability of VoIP networks. In *IEEE/IFIP International Conference on Dependable*

Systems & Networks, pages 419–428, Lisbon, Portugal, Jul. 2009.

[36] C. Shen, E. Nahum, H. Schulzrinne, and C.P. Wright. The impact of TLS on SIP server performance. Technical Report CUCS-022-09, Columbia University Department of Computer Science, May 2009.

[37] C. Shen, H. Schulzrinne, and E. Nahum. Session Initiation Protocol (SIP) server overload control: Design and evaluation. In *IPTComm '08: Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, volume 5310/2008, pages 149–173, Heidelberg, Germany, Oct. 2008.

[38] R. Stewart. Stream Control Transmission Protocol. RFC 4960, Sep. 2007.

[39] J. Sun, J. Hu, R. Tian, and B. Yang. Flow management for SIP application servers. In *Proc. IEEE ICC '07*, pages 646–652, Glasgow, Scotland, Jun. 2007.

[40] V. Gurbani, V. Hilt, and H. Schulzrinne. Session Initiation Protocol (SIP) Overload Control. Internet draft, Jun. 2010. Work in progress.

[41] V. Hilt, E. Noel, C. Shen, and A. Abdelal. Design Considerations for Session Initiation Protocol (SIP) Overload Control. Internet draft, Jun. 2010. Work in progress.

[42] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. Andersen, G. Ganger, G. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Proc. of SIGCOMM '09*, pages 303–314, Barcelona, Spain, Aug. 2009.

[43] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on router buffer sizing: recent results and open problems. *SIGCOMM Comput. Commun. Rev.*, 39(2):34–39, 2009.

[44] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: an analytic performance study. In *Proc. ACM MULTIMEDIA '04*, pages 908–915, New York, New York, Oct. 2004.

[45] M. Whitehead. GOCAP - one standardised overload control for next generation networks. *BT Technology Journal*, 23(1):144–153, 2005.

[46] C.P. Wright, E Nahum, D. Wood, J. Tracey, and E. Hu. SIP server performance on multicore systems. *IBM Journal of Research and Developement*, 54(1), Feb. 2010.

[47] X. Wu, M. Chan, and A. Ananda. Improving TCP performance in heterogeneous mobile environments by exploiting the explicit cooperation between server and mobile host. *Computer Networks*, 52(16):3062–3074, 2008.

[48] W. Zhao and H. Schulzrinne. Enabling on-demand query result caching in dotslash for handling web hotspots effectively. In *1st IEEE Workshop on HOTWEB*, pages 1–12, Boston, Massachusetts, Nov. 2006.

A Novel Implementation of Very Large Teleconferences

Eric Cheung
AT&T Labs—Research
Florham Park, NJ, USA
cheung@research.att.com

Gerald Karam
AT&T Labs—Research
Florham Park, NJ, USA
karam@research.att.com

ABSTRACT

Certain teleconferencing applications must host very large number of participants that exceeds the capacity of a single mixing media server. Traditionally multiple media servers are connected in a cascading arrangement to meet the capacity requirement. This paper discusses the shortcomings of the traditional approach such as lower audio quality and unfairness of speaker selection. It then presents a novel approach that exploits the flexibility of Voice-over-IP and the Session Initiation Protocol to move participants between a main conference mixer and one or more media-distributing replicators as their role change between active talk-listen and listen-only. Benchmarking of an implementation of the replicators on general-purpose computers shows that large capacity can be achieved without specialized hardware. Moreover, it is shown how this approach can augment an already deployed teleconferencing system without modifying the existing telephony features, thereby illustrating the power of modularity and application composition in the SIP servlet environment.

1. INTRODUCTION

With the increased cost and security concerns of traveling, and environmental concerns favoring telecommuting, teleconferencing involve multiple users located in distant locations participating in audio, video and application sharing sessions are becoming more prevalent both in the business and consumer sectors. Teleconferences vary greatly in sizes, i.e. number of participants, and the size influences implementation choices. For voice over IP (VoIP) teleconferences, the main computational requirements are decoding of the audio streams, detecting the loudest streams, mixing the selected streams, encoding and packetize the mixed output, and eventually transmitting to each participant. Small conferences in the order of 10 participants can be supported by present day computing devices without specialized dedicated hardware. For example, the popular Skype service performs audio mixing at the initiator's computer and can support up to 25 participants. The limit is constrained by the processing power of the initiator's endpoint as well as the available bandwidth in this approach. Larger conferences are better implemented using a centralized mixing media server. However, even purpose-built high-density mixers have limits on the size of conferences, typically of the order of 1000 participants.

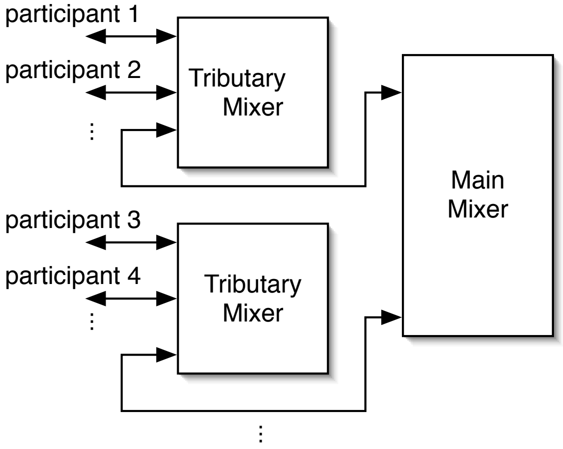


Figure 1: A Two-Layer Cascading Conference

There are certain applications that need to support even larger number of participants, and cannot be supported readily by a single media server. For example large enterprises often host 'all-hands town hall meetings'. Very large conferences are also useful for training sessions. The difference between very large conference and broadcast is that in the former any participant can speak, for example to ask a question.

To support these very large conferences, traditionally a cascading conferences approach is used. This approach and its drawbacks are discussed in the next section. They motivate a novel approach that is proposed in Section 3. Section 4 discusses an implementation of the novel approach, and how it can be integrated with an existing conferencing system. Related work is presented in Section 5. Section 6 discusses some proposed future work.

2. CASCADING CONFERENCES

To extend beyond the port limit of a single media server, multiple media servers can be arranged in a hierarchical manner [10, 18, 11]. Figure 1 shows a typical arrangement with two layers in the hierarchy, where a number of 'tributary' mixers in turn act as participants in a main mixer.

As an illustrating example, the CMS-9000 product from Radisys Corporation supports a maximum of 1800 ports on one media server (a Media Processing Card or MPC-4) [12]. A conference may encompass all 1800 ports, but only a maximum 125 talk-listen participants are allowed, and the remaining 1675 participants are listen-only. An application server can send control signal to the media server to change a participant from talk-listen to listen-only and vice versa. At any given time, the n loudest participants are added to the media mix, where n is settable to 1 to 16.

If it is desired that all participants may speak at any time without any user action, then without cascading the maximum conference size is 125. With cascading, there can be at most 16 tributary mixers, and each tributary mixer may have $125 - 1 = 124$ participants. (One port on each tributary mixer is required to connect to the main mixer.) Thus a maximum of 1984 participants may be supported, utilizing two MPC-4 cards.

If it is acceptable the participants must perform some action before speaking, larger conferences can be accommodated. When a participant requests to speak, and perhaps after a moderator exercising floor control has given permission, an application server promotes the participant to talk-listen status. Without cascading the maximum conference size is 1800. With cascading, each tributary mixer may have $1800 - 1 = 1799$ participants. Still there can only be at most 16 tributary mixers, as certain participants on any of the tributary mixers must be able to talk. This gives a maximum conference size of 28784 participants, utilizing 17 MPC-4 cards.

There are however a number of drawbacks to cascading conferences:

Increased latency between speaking parties In order to achieve acceptable VoIP audio quality, the one-way latency should ideally be kept below 150ms [9]. If the latency is too high, two users in a conversation will begin to talk over each other. In a cascading arrangement, if two participants in a dialogue are on two different tributary mixers, then the audio packets must travel through three audio mixers to reach the listener. Depending on the network conditions, jitter buffer configuration, codecs in use, and processing delay in the mixing media server, the one-way audio latency may exceed the recommended limits. This may be mitigated to some extent by (1) locate the tributary mixers and main mixer in a high-speed local area network (2) select codec and packetization size to minimize jitter and processing delay for the legs between the tributary mixers and the main mixer.

Inconsistent mix Within each tributary mixer, the main mixer competes with the participants for N-loudest selection. If the main mixer is drowned out, the participants on this tributary mixer will hear a different mix from participants on other tributary mixers. This may be mitigated if the media server supports a preferred port that is always included in the mix.

If the main mixer is not including all tributary mixers in the output, then it is also possible that a participant hear another participant on the same tributary mixer, but participants on other tributary mixers do not.

Unfair active speakers selection Within each tributary mixer, the participants compete for N-loudest selection. A participant may not be selected in the mix because there are N louder participants on the same tributary mixer, but another participant on a quieter tributary mixer may be selected although he is not as loud.

Increased noise level Typically N is set to 3 or 4 to allow some simultaneous talking on each tributary mixer. When multiplied by the number of tributary mixers there are many audio streams added to the mix, leading to high background noise level.

Inaccurate reporting of active speakers Media servers typically report the list of active speakers in the N-loudest selection sorted by loudness. With multiple tributary mixers reporting active speakers, it is not possible to obtain a sorted list unless the signal level is also reported.

As discussed above, there are ways to mitigate some of the issues caused by cascading conferences. However, they add complexity to both the media servers and the application. Most of the features required are not provided by today's media servers.

3. PROPOSED SOLUTION

3.1 Motivations

From the discussion in the last section, it becomes apparent that the shortcomings of cascading conferences stem from the fact that speakers are scattered across multiple mixers. If all talk-listen participants are connected to the same mixer, the audio quality and other properties would be identical to a regular conference that can be accommodated on a single mixer.

In analyzing the characteristic of very large conference applications, the authors have observed that compared to regular conference where all participants take turn speaking, the majority of participants in very large conferences only listen and do not speak at all. Moreover, large conferences are usually more structured with a hand-raising protocol to request to speak and a moderator who performs floor control and grants permission to speak. These suggest that the media server does not need to constantly monitor the signal levels of all the participants for the N-loudest selection. Furthermore, the same mix is distributed to a large number of participants requiring minimal processing.

Another observation is that, unlike in legacy circuit-switched TDM systems, in VoIP the media is transmitted as IP packets. Session Initiation Protocol (SIP) [14] is the dominant VoIP signaling protocol. SIP provides a lot of flexibility in modifying the topology of connections between the endpoint devices and the mixing media servers, which can be controlled by application servers programmable by standard APIs. However, in the literature where cascading conferences are discussed, a participant is connected to a tributary mixer for the lifetime of the participation. If this flexibility is exploited, more efficient design becomes possible.

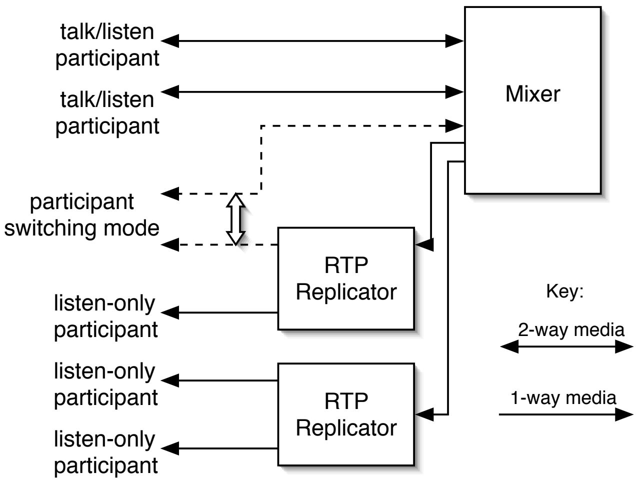


Figure 2: Proposed media architecture showing one very large conference

3.2 Architecture

With the above observations, we propose a novel design for very large conferences. Figure 2 shows the media architecture of our design.

All talk-listen participants are connected to a standard mixing media server, in the same manner as in a regular size conference. However, listen-only participants are connected to a new component called the *RTP replicator*, or replicator for short. The replicator is connected to the mixer as a listen-only participant. The mixer sends the conference mix to the replicator. Each time when the replicator receives a RTP packet, it distributes the packet to the listen-only participants it is serving. The same packet content is sent to all the different destinations. In the other direction, the replicator does not send media to the mixer, and the mixer does not need to monitor the port to the replicator. There may be multiple replicators involved in a single very large conference.

With this approach, the number of talk-listen participants is limited by the mixing media server, for example in the Radisys CMS-9000 the limit is 125. However, because the replicators do not send audio to the main mixer, there can be up to $1800 - 125 = 1675$ replicators. If the replicators can support in the order of 1000 listen-only participants, the capacity of this design should satisfy all practical applications.

Of course, there is a tradeoff. Participants must perform some action to request to speak. This may be granted automatically, or a moderator exercising floor control may grant the permission at a suitable time. In the system that will be discussed in Section 4.2, the participants and moderator use a Web interface to control these plus other call control features.

When a participant is promoted from listen-only to talk-listen role or demoted from talk-listen to listen-only, the media connection switches from the replicator to the main mixer or vice versa. This is achieved by SIP signaling. The mixing media server and the replicators are SIP user agents. The participants' endpoints are also SIP user agents, for example SIP softphones running on desktop computers, hardware SIP phones, or PSTN gateways. Therefore, an SIP application server can act as a third-party call controller to modify how the media streams are connected. In Figure 3, the SIP application we call *Large Conference Controller (LCC)* is inserted in the signaling path between the participant endpoint and the mixing media server or replicator. Note that we have shown one separate instance of LCC for each participant, as it operates independent of the other instances. Note also that there is a SIP dialog between the replicator and mixing media server to establish the one-way media session from the mixer to the replicator. Consider participant B who is being promoted to talk-listen, LCC uses the call flow shown in Figure 4 to change the media connectivity to the mixer.

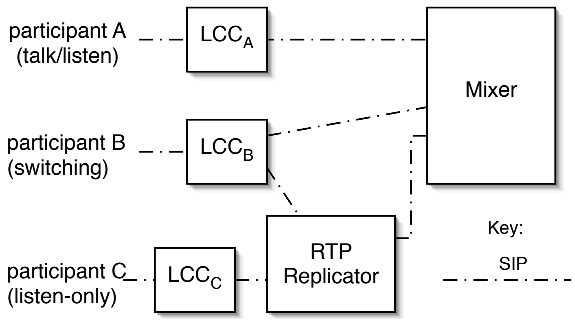

participant A (talk/listen)
participant B (switching)
participant C (listen-only)
LCC$_A$
LCC$_B$
LCC$_C$
Mixer
RTP Replicator
Key:
SIP

Figure 3: Large Conference Controller application in SIP signaling path

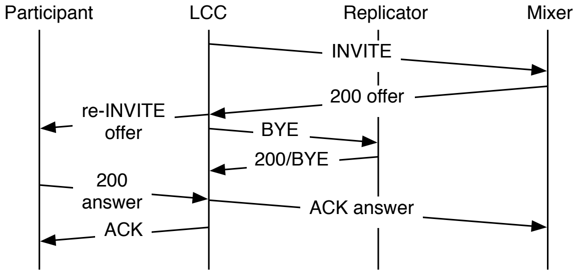


Figure 4: Third party call control signaling flow to switch a participant from replicator to mixer

4. IMPLEMENTATION AND EVALUATION

4.1 RTP Replication

Compared to the tasks of a mixing media server, the operation of the RTP replicator is much simpler. It only has to distribute packets to multiple destinations immediately upon receipt. Therefore we investigated the feasibility of implementing it in software on general-purpose computers.

The first consideration is whether general-purpose, commodity computer servers have sufficient networking bandwidth to handle sending a large number of media streams. At present gigabit Ethernet is most common in computer servers, while some high-end servers also support 10-gigabit Ethernet. In our production environment most servers use full-duplex gigabit Ethernet, i.e. 10^9 bits per second in each direction.

Number of Destinations	CPU%		Jitter (ms)
	`java.net`	`java.nio`	
2000	38	43	1.28
4000	77	76	0.02
6000	93	100 (fail)	0.02
7000	112 (fail)	–	–

Table 1: Performance of RTP Replicator Implementation

Assuming G.711 codec at 20ms packetization, there are 50 packets of 160 bytes of audio data per second, giving 64 kbps. However, when RTP header and lower layer headers are added the data bandwidth is close to 86 kbps. Thus in theory a gigabit Ethernet interface can support distributing media to about 11,600 destinations. This is considerably more than most high-density specialized media servers.

Next we consider the number of UDP ports required. It is desirable to receive RTP packets from the mixer on a dedicated port, so that the replicator does not need to check the source of each packet. It is however possible to send packets to all destinations from the same port. Therefore two ports are required for RTP traffic, and another two if RTCP is supported (our current implementation does not generate RTCP reports).

The RTP replicators are SIP user agents, and our development and deployment environment is the SIP Servlet API [2]. SIP Servlet is currently the dominant standard for SIP application development, and provides a Java API for programming SIP user agents, back-to-back user agents and proxies to be executed on a container. Therefore, we implemented the RTP replication in the Java programming language. An execution thread is dedicated to each replication. The pseudocode below describes its operation:

```
Blocks to receive packet on UDP datagram socket.
Update set of destinations if there has been
    addition or deletion.
Send packet to all destinations.
Repeat.
```

Java provides two options for sending and receiving datagram packets: `DatagramSocket` and related classes in the `java.net` package, or `DatagramChannel` and related classes in the `java.nio` package. The latter stands for 'new I/O' and was introduced in the 1.4 version of Java Standard Edition. We conducted performance benchmark on a standard Red Hat Enterprise Linux Server (release 5.3) with one quad-core Intel Xeon CPU at 2.5GHz and 16GB RAM. A gigabit Ethernet interface was used for both receiving and sending RTP packets. CPU utilization is measured averaged over 20 second periods with `top`. Note that because of the quad-core processor 400% corresponds to the peak utilization. We also measured peak interarrival jitter as defined in [16] when using `java.net` package by running the Wireshark packet analyzer on one of the destination. The results are shown in Table 1.

It is perhaps not surprising that the older `java.net` package provides slightly better performance, supporting up to 6000 destinations. The `java.nio` package provides advanced features such as non-blocking poll and select on multiple sockets, but at some performance cost. The simple operation performed by the replicator does not make use of these advanced features.

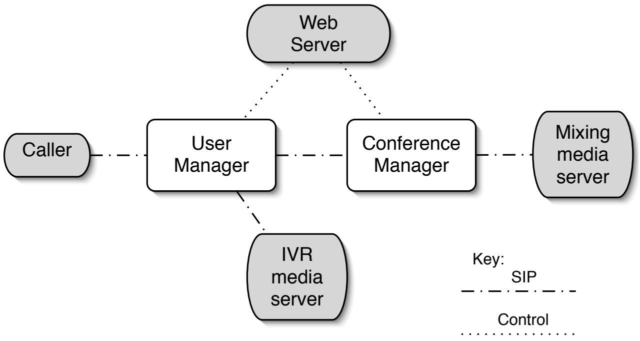


Figure 5: Architecture of Standard Conferencing

The interarrival jitter is low in all cases, and the higher value with 2000 destinations is not significant as it occurred only for one packet that arrived late during a test run. Overall the packets arrive on the 20ms schedule closely even at 6000 destinations. It should also be noted that in any case for the listen-only participants a higher one-way latency due to larger jitter buffer size is acceptable because there is no possibility of over-talk.

The performance testing results show that an inexpensive general-purpose computer can support distribution to a large number of listen-only participants without significant degradation to the audio quality.

4.2 SIP Servlet Application Design

The authors are part of a team who had previously designed and developed an advanced teleconferencing service described in [5]. This is a production-grade deployment that serves our company's standard teleconferencing (but not very large conferencing) needs, and currently handles tens of thousands of simultaneous calls and millions of usage minutes on a typical workday. Users of the service can either use a Web interface or touch-tone key presses to control various features. In [5], we discussed the design consideration that led to a modular and compositional design consisting of two SIP Servlet applications:

User manager (UM) Handles interaction with a user, beginning with connecting the user to an interactive voice response (IVR) media server to collect credential and conference access code to identify the conference to join. Subsequently, the user may participate in multiple conferences and UM is responsible for switching the user between conferences.

Conference Manager (CM) Handles all participants on one conference and interacts with the mixing media server, for example to mute and unmute a participant, play prompts, and manage recordings.

Figure 5 shows how these two applications are composed in an application chain, and their relationship to other components of the system.

After completing the design for very large conferencing, we are now in the process of adding the capability to our service deployment. The same platform would support standard and very large conferences. Besides reduced capital and operational expenses, users can also benefit from a consistent user interface and integrated experience, for example the ability to switch between a regular conference and a very large conference.

This represents a good opportunity to evaluate how our modular design and application composition techniques can support incremental feature enhancement. Ideally, the existing software would require no or minimal changes.

To support very large conferencing, we developed two new SIP servlet applications:

Large Conference Controller (LCC) Manages one participant on one very large conference, switching the participant between the mixer and the replicator as the participant changes role.

RTP Replicator (RTPR) Performs the media replication to all listen-only participants on a very large conference. When the first listen-only participant connects to RTPR, it first connects to the mixer to start receiving the conference mix. It then adds this first and any subsequent participants to the set of destinations.

In the SIP Servlet environment, application selection is performed by an Application Router (AR). Specifically, we use the Distributed Feature Composition (DFC) AR[4] which performs application selection based on caller and callee addresses and subscription to applications. In this case, when a user calls in, UM is selected first. After the IVR identifies the conference the user is joining, and if the conference is a regular conference and not a very large conference, UM sends INVITE request and DFC-AR selects CM as the next application, resulting in the original application chain shown in Figure 5.

However, if the conference is a very large conference, the DFC-AR selects LCC next instead. When LCC in turn sends INVITE request, if the initial role of the participant is listen-only, RTPR is selected next. If the initial role of the participant is talk-listen, CM is selected next to connect the user to the main mixer.

LCC receives command from the Web server when a participant changes role, and tears down the SIP dialog with RTPR and switches to CM and vice versa using the call flow shown in Figure 4.

In Figure 6, three participants on the large conference are shown. Caller 1 is talk-listen, and caller 2 and 3 are listen-only. Two ports on the mixing media server are used, one for Caller 1 and one for the replicator.

It should be noted that UM and CM are not affected by this new feature. The changes are confined to (1) the subscription configuration of the DFC-AR, and (2) the Web application where user interface and business logic are modified to handle the hand-raising and floor control functionalities, and the interaction with LCC.

Our production system consists of a number of servers hosting SIP Servlet containers and the UM, CM, LCC and RTPR applications. A SIP load balancer is responsible for distributing incoming calls evenly to the servers. In order to balance the load of the replicators, LCC may simply invoke a RTPR executing in the same server. The listen-only participants in a very large conference would then be distributed close to evenly across RTPRs executing on all the servers.

4.3 Touchtone Control

While a participant can use the Web interface to request to speak, it is necessary to also provide a telephone-only means as not all participants access the Web site. In our current regular conferencing service, participants already have the ability to use touchtone key presses to mute or unmute their lines. We need to provide the same capability for participants to request to speak.

However, this presents a problem. In the current system, the key presses are detected as dual-tone multi-frequency (DTMF) audio by the mixing media server, which then reports the detected tones to CM in the SIP signaling path. However, in a large conference the listen-only participants are not connected to the media server but to the replicator.

One option is to rely on Key Press Markup Language (KPML) events with which a participant's endpoint device reports key presses in the signaling path [3]. However, KPML is not widely supported. Another option is to enhance the replicator to also receive media packets from the listen-only participants, and detect touchtone key presses. Fortunately most user agent implementations support transmitting key presses as special RTP payload packets [17]. Therefore it is not necessary to analyze the audio packets to detect DTMF tones.

To further reduce the computational load required for the task of detecting key presses, when the participant endpoint and the replicator negotiate media streams and codecs using the offer-answer procedure, the replicator can indicate that it wishes to send audio only and receive key press event only. We have implemented touchtone detection in the RTP replicator. The computation load is much smaller than that imposed by audio packet replication.

4.4 Switching Delay

When a participant is switched from listen-only to talk-listen, signaling is required to establish a new call to CM and the mixing media server. Similarly, when a participant is switched from talk-listen to listen-only, signaling is required to establish a new call to RTPR. If the time taken to perform the switching operations is too long, the participant will hear break in the audio from the conference.

In order to evaluate the switching delay, we designed an experiment with one talk-listen participant and two listen-only participants on a conference. The talk-listen participant constantly sent audio to the mixer. One of the other participants was switched from listen-only to talk-listen and back while received RTP packets were captured. The switching delay was measured as the time gap between the last packet from the RTPR to the first packet from the mixing media server for the listen-only to talk-listen switch, and vice versa.

Averaged over three test, the listen-only to talk-listen switching delay was 35ms, and the talk-listen to listen-only switching delay was 24ms. The small delay indicated that the participant should not hear noticeable break or artifact, and this was confirmed by listen tests.

5. RELATED WORK

Extensive work on conferencing has been conducted in the Internet Engineering Task Force (IETF). Our proposed solution fits into the models of 'tightly coupled conference' [13] and 'centralized conference' [1]. A number of conferencing scenarios are presented in [7], and among them 'lecture mode

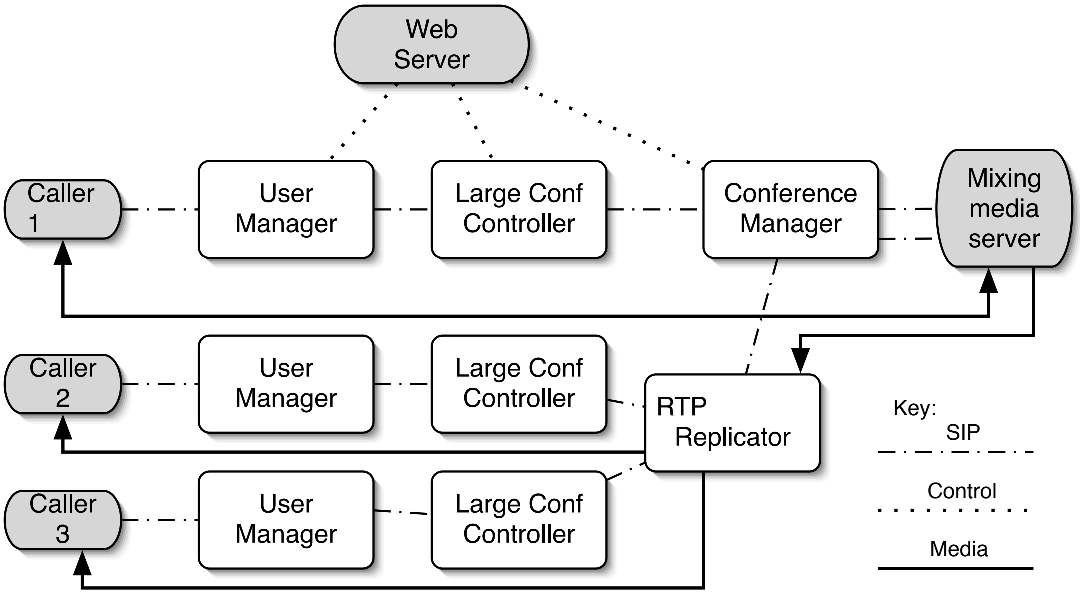


Figure 6: Architecture of Very Large Conferencing

conferences' and 'presentation and Q & A sessions' can be supported readily by the proposal in this paper. However, the IETF has not proposed a solution that switches participants between a mixer and a replicator or multicast.

As noted in Section 2, cascading conferences have been proposed to support conferences with very large number of participants in [10, 18, 11]. In these work, the assignment of participants to mixers are static, and does not take into account the status of the participants.

A scalable group communication system is proposed in [19]. A tree-based control topology is used to improve scalability and responsiveness, with reconfiguration mechanisms aimed to optimize performance. However, this work is primarily concerned with control and signaling. The transmission and mixing of audio and video data rely on existing transport facilities, and no new mechanisms are proposed.

IP multicast can also be used to support large scale audio and video conferencing [6, 8]. Each participant broadcasts his media packets that are then received by all other participants. Alternatively application-layer multicast can be used to remove the requirement of network-layer multicast [15]. However, in these loosely coupled conferences it is difficult to exert floor control. Nonetheless, we intend to investigate a hybrid system that uses multicast to distribute media to the listen-only participants.

6. SUMMARY AND FUTURE WORK

In this paper, we have described a novel design for very large teleconferencing systems. We have completed an implementation using the SIP Servlet Java API, and showed that high capacity can be achieved on general-purpose computers. We have also showed how this new capability can be integrated into our existing teleconference system because of modularity and application composition techniques. At the present time, this very large teleconferencing system is ready to begin user trial and will go into production soon afterwards.

There are several areas of future work that we intend to pursue. First, our current service only uses the G.711 codec. In order to support multiple codecs, a RTP replicator can be assigned to each codec. The replicator would then connect to the mixing media server using the relevant codec. The task of selecting the correct replicator can reside with the Large Conference Controller, which can inspect the SDP in the media offer received from the participant endpoint to make the determination.

For scenarios where participants are divided into several geographically distant sites, the cascading conferences arrangement can offer a way to reduce the traffic on the distant links by having participants connect to their local tributary conference. Similarly, in this proposal a number of RTP replicators may be distributed geographically to close to groups of participants. The talk-listen participants would still need to connect directly to the mixing media server in a single location. However, one possible optimization is to again exploit the flexibility of VoIP signaling and media control to dynamically move to a media server that is close to the majority of the current talk-listen participants.

Currently we use a form of multi-unicast for the distribution of RTP packets to the listen-only participants. We intend to investigate using multicast techniques for the very large teleconference application, taking into consideration SIP endpoint support, multicast through wide area networks, latency, and security issues.

7. ACKNOWLEDGMENTS

We are grateful for our colleagues on the advanced teleconferencing service team: Mike Bamert, Jessie Chen, Adam Combs, Tom Everling, Vince Hadap, and Tom Smith, who constantly provide inspirations, insights and assistance for our work. The anonymous reviewers also provided valuable comments and suggestions.

8. REFERENCES

[1] M. Barnes, C. Boulton, and O. Levin. A framework for centralized conferencing, June 2008. IETF RFC 5239.

[2] BEA. SIP servlet API version 1.1, 2008. Java Community Process JSR 289. http://jcp.org/en/jsr/detail?id=289.

[3] E. Burger and M. Dolly. A Session Initiation Protocol (SIP) event package for key press stimulus (KPML), November 2006. IETF RFC 4730.

[4] E. Cheung and K. Purdy. An application router for SIP servlet application composition. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 1802 –1806, May 2008.

[5] E. Cheung and T. M. Smith. Experience with modularity in an advanced teleconferencing service deployment. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, pages 39–49, 2009.

[6] J. Eriksson. MBONE: The multicast backbone, August 1994.

[7] R. Even and N. Ismail. Conferencing scenarios, July 2006. IETF RFC 4597.

[8] V. Hardman, M. A. Sasse, and I. Kouvelas. Successful multiparty audio communication over the internet. *Commun. ACM*, 41(5):74–80, 1998.

[9] ITU-T. One-way transmission time, May 2003. ITU-T Recommendation G.114.

[10] D. Ozone. Proposal for MC+MP cascading in H.323, 1997. ITU-T SG 16 Document AVC-1108.

[11] M. Radenkovic, C. Greenhalgh, and S. Benford. A scaleable and adaptive audio service to support large scale collaborative work and entertainment. In *Proceedings of International Conference on Advances in Infrastructures for Electronic Business, Education, Science and Medicine on the Internet (SSGRR 2002)*, 2002.

[12] Radisys corporation. http://radisys.com/.

[13] J. Rosenberg. A framework for conferencing with the Session Initiation Protocol (SIP), February 2006. IETF RFC 4353.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol, June 2002. IETF RFC 3261.

[15] T. C. Schmidt and M. Wahlisch. *Group Conference Management with SIP*, chapter 6, pages 123–158. CRC Press, 2008. In SIP Handbook: Services, Technologies, and Security of Session Initiation Protocol.

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, July 2003. IETF RFC 3550.

[17] H. Schulzrinne and T. Taylor. RTP payload for DTMF digits, telephony tones, and telephony signals, December 2006. IETF RFC 4733.

[18] K. Singh, G. Nair, and H. Schulzrinne. Centralized conferencing using SIP. In *Proceedings of the 2nd IP-Telephony Workshop (IPTel'2001)*, April 2001.

[19] D. Trossen. *Scalable Group Communication in Tightly Coupled Environments*. PhD thesis, University of Technology North-Rhine Westfalia Aachen, Germany, 2000.

CCMP: a novel standard protocol for Conference Management in the XCON Framework

Mary Barnes
Nortel
mary.barnes@nortel.com

Lorenzo Miniero
Meetecho srl
lorenzo@meetecho.com

Roberta Presta
University of Napoli Federico II
roberta.presta@unina.it

Simon Pietro Romano
Univeristy of Napoli Federico II
spromano@unina.it

Henning Shulzrinne
Columbia University
hgs+xcon@cs.columbia.edu

ABSTRACT

This paper presents the design and implementation of CCMP, a conference management protocol currently under standardization within the IETF, conceived at the outset as a lightweight protocol allowing conferencing clients to access and manipulate objects describing a centralized conference. The CCMP is a state-less, XML-based, client-server protocol carrying in its request and response messages conference information in the form of XML documents and fragments conforming to the centralized conferencing data model schema. It represents a powerful means to control basic and advanced conference features such as conference state and capabilities, participants and relative roles and details. We first focus on the design of the protocol and then discuss how it has been integrated in the Meetecho collaborative framework developed at the University of Napoli as an active playground for IETF standardization activities in the field of real-time applications and infrastructure.

Keywords

Conferencing, Conference Control and Manipulation, Protocol Design, Protocol Integration

1. INTRODUCTION

In the latest years, the IETF *(Internet Engineering Task Force)* has devoted many efforts to the definition of standard conferencing solutions. Among such solutions, the Framework for Centralized Conferencing [2] (XCON Framework) defines a signaling-agnostic architecture, naming conventions and logical entities required for building advanced conferencing systems. The XCON Framework introduces the *conference object* as a logical representation of a conference instance, representing the current state and capabilities of a conference. The Centralized Conferencing Manipulation Protocol (CCMP) illustrated in this paper is the latest output to be produced by the XCON working group. It is currently undergoing review from the international research community and it is heading towards completion and publication as an RFC (Request For Comments) standard document.

CCMP allows authenticated and authorized users to create, manipulate and delete conference objects. Operations on conferences include adding and removing participants, changing their roles, as well as adding and removing media streams and associated end points. CCMP is based on a client-server paradigm and is specifically suited to serve as a conference manipulation protocol within the XCON framework, with the Conference Control Client and Conference Control Server acting as client and server, respectively. The CCMP uses HTTP as the protocol to transfer requests and responses, which contain the domain-specific XML-encoded data objects defined in [7].

This paper is structured in 8 sections. We first briefly introduce, in section 2, the general architecture for centralized conferencing defined by the XCON working group in the IETF. We then present, in section 3, a bird's eye view of the Centralized Conferencing Manipulation Protocol. The same section also provides some insights on the history of the overall specification process. Section 4 drills down on the specific messages that can be carried inside the body of the CCMP protocol, while section 5 concludes the part associated with our standardization work by depicting a typical call flow related to a CCMP-based interaction between a conferencing client and an XCON Conferencing server. The second part of the paper is entirely devoted to the implementation of the CCMP specification. Such part is based on the work ongoing at the University of Napoli "Federico II", which is since long involved in the IETF activities falling in the area of real-time applications and infrastructures. The University of Napoli has contributed to the activities in the XCON working group, by also providing timely prototype implementations of most of the protocols therein involved and/or specified. As far as the CCMP protocol is concerned, we have worked both on the specification of the protocol and on its implementation, during the various phases of its long-lived design history. Information about this activity is hence provided in section 6. Section 7 reports information about the history of the CCMP specification within the IETF community. Finally, section 8 provides some concluding remarks, as well as information about our future work related to the protocol.

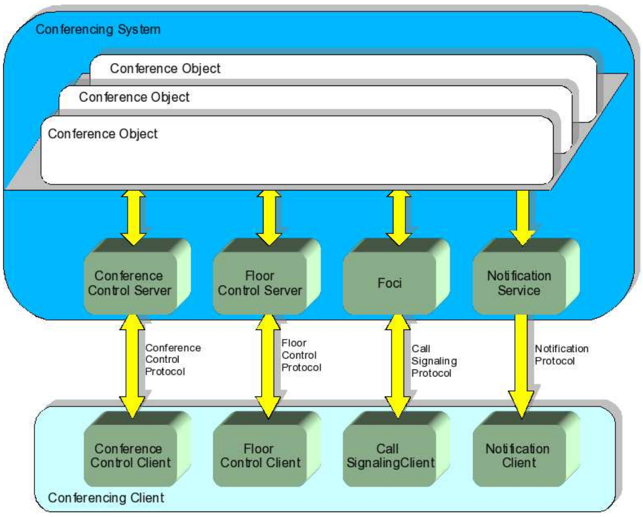


Figure 1: The XCON framework: protocols

2. XCON CONFERENCE CONTROL SYSTEM ARCHITECTURE

RFC5239 defines an architecture for centralized conferencing, and the associated protocol interactons. Such relations are depicted in Fig. 1.

As it can be seen in the figure, several protocols are involved in an XCON-compliant framework architecture. While all the protocols implicitly interact with conference objects somehow, the generically called Conference Control Protocol is probably the most important of them in that regard, as it directly manipulates the conference objects themselves.

Fig. 2 illustrates the typical life cycle of a conference object in the XCON framework. At each instant in time, a conference object is associated with an XML representation compliant with the XCON data model specification. Without digging into the details of the data model, we nevertheless recall that it basically describes all of the features of a conference, starting from its general description (purpose, hosting entity, status, etc.) and arriving at much more detailed information like participants and available media, as well as potential *sidebars* associated with it (i.e. sub-conferences involving part of the users participating in the main conference).

Creation of such an object is usually performed through a cloning operation, i.e. by replicating the structure of one of the blueprints (also known as conference object templates) available at the server.

A newly created conference object is typically marked as "registered" until the first user joins the conference and it will stay "active" until either the last user leaves the conference (in which case it comes back to the "registered" state) or a user (holding the right to do so) deletes it.

CCMP is the protocol used to manipulate conference objects during the above described lifetime. The next section will present a protocol overview in more detail.

3. PROTOCOL OVERVIEW

CCMP is a client-server, XML-based protocol, which has been specifically conceived to provide users with the necessary means for the creation, retrieval, modification and deletion of conference objects. CCMP is also state-less, which means implementations can safely handle transactions independently from each other. Conference-related information is encapsulated into CCMP messages in the form of XML documents or XML document fragments compliant with the XCON data model representation.

The core set of objects manipulated in the CCMP protocol includes conference blueprints, conference objects, users, and sidebars. CCMP is completely independent from underlying protocols, which means that there can be different ways to carry CCMP messages across the network, from a conferencing client to a conferencing server. Indeed, there have been a number of different proposals as to the most suitable transport solution for the CCMP. It was soon recognized that operations on conference objects can be implemented in many different ways, including remote procedure calls based on SOAP [6] and by defining resources following a RESTful [5] architecture. In both approaches, servers will have to recreate their internal state representation of the object with each update request, checking parameters and triggering function invocations. In the SOAP approach, it would be possible to describe a separate operation for each atomic element, but that would greatly increase the complexity of the protocol. A coarser-grained approach to the CCMP does require that the server process XML elements in updates that have not changed and that there can be multiple changes in one update. For CCMP, the resource (REST) model might appear more attractive, since the conference operations nicely fit the so-called *CRUD (Create-Retrieve-Update-Delete)* approach. Neither of these approaches was finally selected. SOAP was not considered to be general purpose enough for use in a broad range of operational environments. Similarly, it was deemed quite awkward to apply a RESTful approach since CCMP requires a more complex request/response protocol in order to maintain the data both in the server and at the client. This doesn't map very elegantly to the basic request/response model, whereby a response typically indicates whether the request was successful or not, rather than providing additional data to maintain the synchronization between the client and server views. Apart from this, the RESTful approach was considered too restrictive, since it strictly couples the application-level protocol to HTTP messages and semantics. Even though the current implementation of the CCMP relies on HTTP as the preferred transport means, its specification has been kept completely independent of such a choice. Just as an example, work is in full swing at our laboratory related to both an XMPP-based and a UDP-based implementation of the protocol.

The solution for the CCMP at which we arrived can be viewed as a good compromise amongst the above mentioned candidates and is referred to as "HTTP single-verb transport plus CCMP body". With this approach, CCMP is able to take advantage of existing HTTP functionality. As with SOAP, it uses a "single HTTP verb" for transport (i.e. a single transaction type for each request/response pair); this allows decoupling CCMP messages from HTTP messages. Similarly, as with any RESTful approach, CCMP messages are inserted directly in the body of HTTP messages, thus avoiding any unnecessary processing and communication burden associated with further intermediaries. This said, we nonetheless remark once again that with this approach no modification to the CCMP messages/operations is

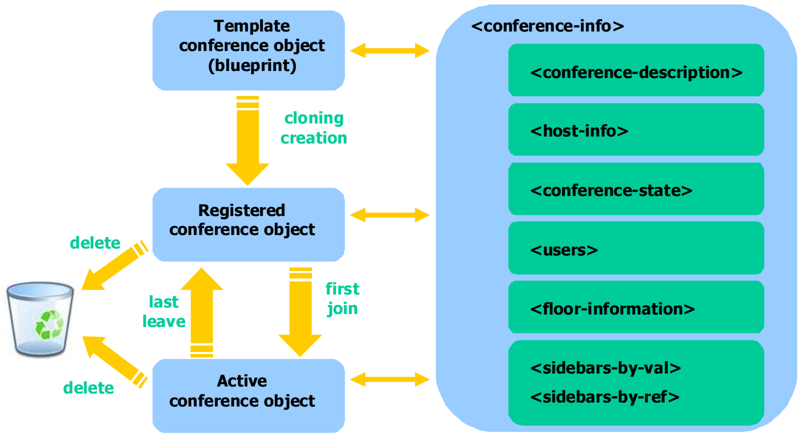


Figure 2: Conference Object Life Cycle

required to use a different transport protocol. The remainder of this paper focuses on the selected approach. We will show how the CCMP protocol inserts XML-based CCMP requests into the body of HTTP POST operations and retrieves responses from the body of HTTP "200 OK" messages. CCMP messages will have a MIME-type of "application/ccmp+xml", which appears inside both the "Content-Type" and "Accept" fields of HTTP requests and responses.

3.1 Protocol Operations

The main operations provided by CCMP belong in four general categories:

- create: for the creation of a conference, a conference user, a sidebar, or a blueprint;
- retrieve: to get information about the current state of either a conference object (be it an actual conference or a blueprint, or a sidebar) or a conference user. A retrieve operation can also be used to obtain the XCON-URIs of the current conferences (active or registered) handled by the conferencing server and/or the available blueprints;
- update: to modify the current features of a specified conference or conference user;
- delete: to remove from the system a conference object or a conference user.

Thus, the main targets of CCMP operations are: (i) conference objects associated with either active or registered conferences; (ii) conference objects associated with blueprints; (iii) conference objects associated with sidebars, both embedded in the main conference (i.e. `<entry>` elements in `<sidebars-by-value>`) and external to it (i.e. whose XCON-URIs are included in the `<entry>` elements of <sidebars-by-ref>); (iv) `<user>` elements associated with conference users; (v) the list of XCON-URIs related to conferences and blueprints available at the server, for which only retrieval operations are allowed.

Each operation in the protocol model is atomic and either succeeds or fails as a whole. The conference server must ensure that the operations are atomic in that the operation invoked by a specific conference client completes prior to another client's operation on the same conference object. The details for this data locking functionality are out of scope for the CCMP protocol specification and are implementation specific for a conference server. Thus, the conference server first checks all the parameters, before making any changes to the internal representation of the conference object.

Also, since multiple clients can modify the same conference objects, conference clients should first obtain the current object from the conference server and then update the relevant data elements in the conference object prior to invoking a specific operation on the conference server. In order to effectively manage modifications to conference data, a versioning approach is exploited in the CCMP. More precisely, each conference object is associated with a version number indicating the most up to date view of the conference at the server's side. Such version number is reported to the clients when answering their requests. A client willing to make modifications to a conference object has to send an update message to the server. In case the modifications are all successfully applied, the server sends back to the client a "success" response which also carries information about the current server-side version of the modified object. With such approach, a client which is working on version "X" of a conference object and finds inside a "success" response a version number which is "X+1" can be sure that the version it was aware of was the most up to date. On the other hand, if the "success" response carries back a version which is at least "X+2", the client can detect that the object that has been modified at the server's side was more up to date than the one it was working upon. This is clearly due to the effect of concurrent modification requests issued by independent clients. Hence, for the sake of having available the latest version of the modified object, the client can send to the conference server a further "retrieve" request. In no case a copy of the conference object available at the server is returned to the client as part of the update response message. Such a copy can always be obtained through an ad-hoc "retrieve" message. Based on the above considerations, all CCMP response messages carrying in their body a conference document (or a fragment of it) must contain a "version" parameter. This does not hold for request messages, for which the "version" parameter is not at all required, since it represents useless information for the server: as long as the required modifications can be

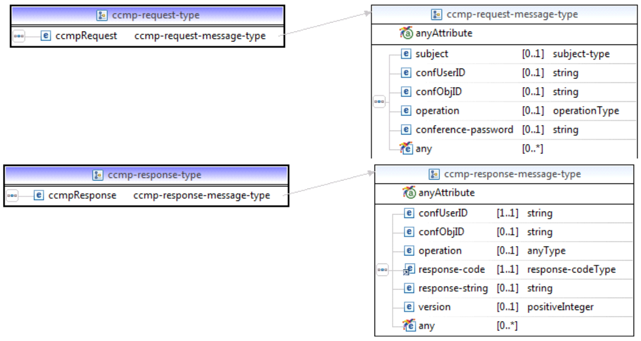

ccmp-request-type
ccmpRequest
ccmp-request-message-type
ccmp-request-message-type
anyAttribute
subject [0..1] subject-type
confUserID [0..1] string
confObjID [0..1] string
operation [0..1] operationType
conference-password [0..1] string
any [0..*]
ccmp-response-type
ccmpResponse
ccmp-response-message-type
ccmp-response-message-type
anyAttribute
confUserID [1..1] string
confObjID [0..1] string
operation [0..1] anyType
response-code [1..1] response-codeType
response-string [0..1] string
version [0..1] positiveInteger
any [0..*]

Figure 3: CCMP Request and Response messages

applied to the target conference object with no conflicts, the server does not care whether or not the client had an up to date view of the information stored at its side. This said, it stands clear that a client which has subscribed at the server, through the XCON event package [4], to notifications about conference object modifications, will always have the most up to date version of that object available at his side.

A final consideration concerns the relation between the CCMP and the main entities it manages, i.e. conference objects. Such objects have to be compliant with the XCON data-model, which identifies some elements and attributes as mandatory. From the CCMP standpoint this can become a problem in cases of client-initiated operations, like either the creation or the update of conference objects. In such cases, not all of the mandatory data can be known in advance to the client issuing a CCMP request. As an example, a client has no means to know, at the time it issues a conference creation request, the XCON-URI that the server will assign to the yet-to-be-created conference and hence it is not able to appropriately fill with that value the mandatory 'entity' attribute of the conference document contained in the request. To solve this kind of issues, the CCMP will fill all mandatory data model fields, for which no value is available at the client at the time the request is constructed, with fake values in the form of wildcard strings (e.g. AUTO_GENERATE_X, with X being an incremental index initialized to a value of 1). Upon reception of the mentioned kinds of requests, the server will: (i) generate the proper identifier(s); (ii) produce a response in which the received fake identifier(s) carried in the request has (have) been replaced by the newly created one(s). With this approach we maintain compatibility with the data model requirements, at the same time allowing for client-initiated manipulation of conference objects at the server's side (which is, by the way, one of the main goals for which the CCMP protocol has been conceived at the outset).

4. CCMP MESSAGES

As anticipated, CCMP is a request/response protocol. Besides, it is completely stateless, which explains why HTTP has been chosen as the perfect transport candidate for it.

For what concerns the protocol by itself, both requests and responses are formatted basically in the same way, as depicted in Fig. 3. In fact, they both have a series of heading parameters, followed by a specialized message indicating the particular request/response (e.g., a request for a specific blueprint). This makes it quite easy to handle a transaction in the proper way and map requests and related responses accordingly.

For what concerns the shared parameters:

- `confUserID` indicates the participant making the request;
- `confObjID` indicates the conference the request is associated with;
- operation specifies what has to be done, according to the specialized message that follows.

Other parameters are defined which are more strictly related to either requests or responses. There is, for instance, a 'password' parameter participants may need to provide in CCMP requests for password-protected conferences, as well as a 'response-code' parameter (which is carried just by responses) providing information about the result of a requested operation.

That said, the core of a CCMP message is actually the specialized part. In fact, as stated in the previous section, the CCMP specification describes several different operations that can be made on a conference object, namely: (i) blueprints retrieval, (ii) conference creation and manipulation, (iii) users management, (iv) sidebar-related operations. All these operations have one or more specialized message formats, instead of a generalized syntax, in order to best suit the specific needs each operation may have.

Indeed, requesting a blueprint and adding a new user to a conference have very different requirements for what concerns the associated semantics level, and as such they need different modes of operation. This is reflected in what is carried in the specialized message body, which will always contain information (compliant with the XCON common data model specification) strictly related to the operation it is associated with. The specialization of the message then allows for an easier and faster management at the implementation level.

To better highlight the considerations above, we show in Fig. 4 the structure of a CCMP `confRequest` message, which is used in all operations concerning the manipulation and control of an entire conference object. As described in the picture, each such message is a specialization of the general CCMP request message, specifically conceived to transport, through the `confInfo` element, an XCON-compliant conference object (i. e. an object whose representation conforms to the common data model specification) towards the CCMP server.

5. CCMP SAMPLE CALL FLOW

To better clarify how a CCMP transaction can occur, this section presents a sample call flow. This example comes from a real implementation deployment, as it will be explained in section 6.

For the sake of conciseness, we chose a very simple example, which nevertheless provides the reader with a general overview of both CCMP requests and responses. As mentioned previously, HTTP is suggested by the CCMP specification as a transport for the protocol messages, and Fig. 5 shows the typical request/response paradigm involved in that case.

As it can be seen, the CCMP request (in this case, a 'blueprintRequest') is sent by an interested participant to

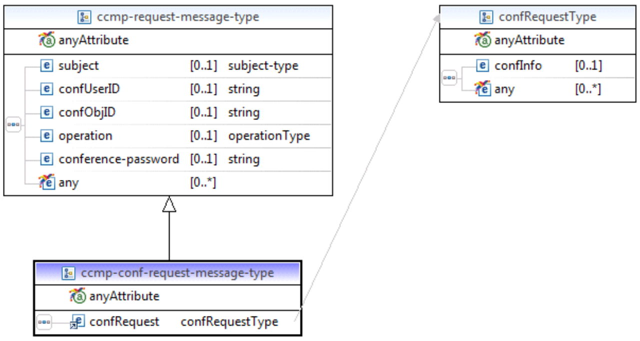

ccmp-request-message-type
anyAttribute
subject [0..1] subject-type
confUserID [0..1] string
confObjID [0..1] string
operation [0..1] operationType
conference-password [0..1] string
any [0..*]
ccmp-conf-request-message-type
anyAttribute
confRequest confRequestType
confRequestType
anyAttribute
confInfo [0..1]
any [0..*]

Figure 4: CCMP confRequest message

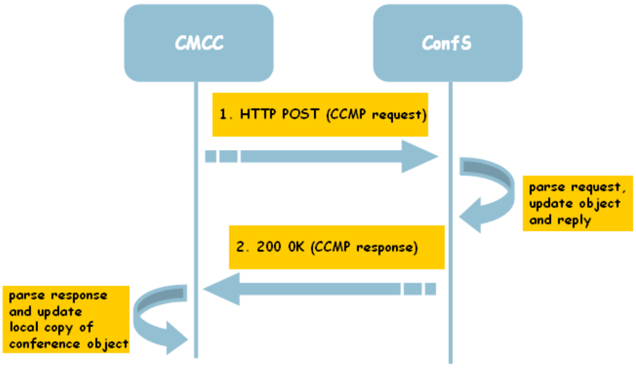

CMCC
ConfS
1. HTTP POST (CCMP request)
parse request, update object and reply
2. 200 OK (CCMP response)
parse response and update local copy of conference object

Figure 5: CCMP transported in HTTP

the conference server. This request is carried as payload of an HTTP POST message:

```
POST /Xcon/Ccmp HTTP/1.1
Content-Length: 657
Content-Type: application/ccmp+xml
Host: example.com:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.0.1 (java 1.5)

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
  <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="ccmp:ccmp-blueprint-request-message-type">
      <confUserID>xcon-userid:Alice@meetecho.com</confUserID>
      <confObjID>xcon:MeetechoRoom@meetecho.com</confObjID>
      <operation>retrieve</operation>
      <ccmp:blueprintRequest/>
  </ccmpRequest>
</ccmp:ccmpRequest>
```

The Content-Type header instructs the receiver that the content of the message is a CCMP message (application/ ccmp+xml). For what concerns the request itself, as mentioned, it is a ‘blueprintRequest’: this means that the participant is interested in the details of a specific blueprint available at the server. This is reflected by the specialized part of the message, i.e., the <ccmp:blueprintRequest> element. The generic parameters introduced in the previous section are also provided as part of the request: ‘confUserID’ refers to the requestor (Alice’s XCON URI), ‘confObjID’ in this case relates to the blueprint to be retrieved (as an XCON conference URI), while ‘operation’ clarifies what needs to be done according to the request (retrieve the blueprint).

The CCMP response, in turn, is carried as payload of an HTTP 200 OK reply to the previous POST:

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun GlassFish Communications Server 1.5
Content-Type: application/ccmp+xml;charset=ISO-8859-1
Content-Length: 1652
Date: Thu, 04 Feb 2010 14:47:56 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp">
  <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:type="ccmp:ccmp-blueprint-response-message-type">
    <confUserID>xcon-userid:Alice@meetecho.com</confUserID>
    <confObjID>xcon:MeetechoRoom@meetecho.com</confObjID>
    <operation>retrieve</operation>
    <response-code>200</response-code>
    <response-string>Success</response-string>
    <ccmp:blueprintResponse>
      <blueprintInfo entity="xcon:MeetechoRoom@meetecho.com">
        <info:conference-description>
           <info:display-text>MeetechoRoom</info:display-text>
           <info:available-media>
             <info:entry label="audioLabel">
                 <info:type>audio</info:type>
             </info:entry>
             <info:entry label="videoLabel">
                 <info:type>video</info:type>
             </info:entry>
             <info:entry label="jSummitLabel">
                 <info:type>whiteboard</info:type>
             </info:entry>
           </info:available-media>
        </info:conference-description>
        <info:users>
           <xcon:join-handling>
                 allow
           </xcon:join-handling>
        </info:users>
        <xcon:floor-information>
          <xcon:floor-request-handling>
                confirm
          </xcon:floor-request-handling>
          <xcon:conference-floor-policy>
                <xcon:floor id="audioFloor">
                  <xcon:media-label>
                        audioLabel
                  </xcon:mediaLabel>
                </xcon:floor>
                <xcon:floor id="videoFloor">
                  <xcon:media-label>
                        videoLabel
                  </xcon:mediaLabel>
                </xcon:floor>
                <xcon:floor id="jSummitFloor">
                  <xcon:media-label>
                        jSummitLabel
                  </xcon:mediaLabel>
                </xcon:floor>
          </xcon:conference-floor-policy>
        </xcon:floor-information>
      </blueprintInfo>
    </ccmp:blueprintResponse>
  </ccmpResponse>
</ccmp:ccmpResponse>
```

As a reply to a ‘blueprintRequest’ message, the CCMP response includes a ‘blueprintResponse’ specialized message in its body: this element includes the whole conference object (compliant with the XCON common data model specification) associated with the requested blueprint, as part of a <blueprintInfo> container. Besides containing some of the parameters provided in the request (confUserID, confObjID, operation), the response also carries back an additional piece of information related to the result of the request, namely, a ‘response-code’ parameter telling the participant that the request was successfully taken care of (‘200’), which is also reflected in the related ‘response-string’ (‘Success’).

The next section will provide further details on our implementation experience with the protocol. Specifically, we will address the way we designed the process according to the specification (both from the client and the server perspective), and the related implementation choices.

6. CCMP WORK AT UNINA

This section deals with our prototype implementation of the CCMP protocol. The reference scenario is the one depicted in Fig. 6.

As the figure shows, in order to have a working instance of the CCMP protocol which could be used as a playground for testing and validation of the specification in progress, as a

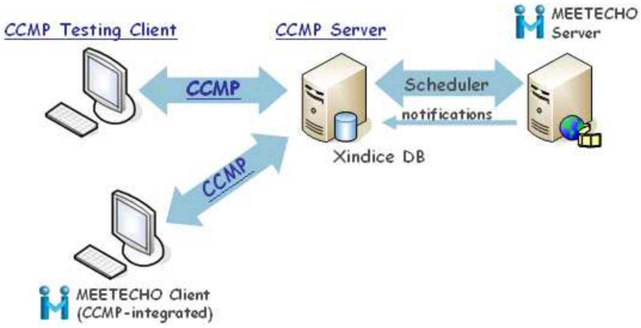


Figure 6: Reference scenario @ unina

first step we have realized a stand-alone Java-based CCMP client and a Java-based CCMP server.

The CCMP testing client presents a very simple graphical user interface through which it is possible to create and send to the CCMP Server the desired CCMP request. All CCMP messages sent and received by the client are logged onto a debugging window which allows to easily visualize the entire call flow associated with client-server interactions.

As to the server, it has been integrated into our Meetecho conferencing platform [1]. Since Meetecho already makes use of a “proprietary” protocol[1] for conference creation, manipulation and scheduling (which is herein called ‘Scheduler’), we had to implement the CCMP server as a proxy towards it. The CCMP server receives CCMP requests from the testing client, converts them into Scheduler requests and forwards them to the Meetecho server by using the Meetecho Scheduler protocol, which is a simple, text-based protocol based on TCP. When the Meetecho server is done with the forwarded request, it sends back to the CCMP server a Scheduler-compliant answer, which is then converted into a CCMP-compliant response and forwarded to the CCMP testing client. The CCMP server takes care of the correct mapping between CCMP- and Scheduler-compliant messages. We also remark that synchronization between the Meetecho server and the CCMP proxy server can be achieved through asynchronous notifications. As soon as something worth communicating happens at the Meetecho server, a notification can be sent to the CCMP proxy (which subscribes to the events associated with conference management and manipulation) in order to let it always have an up-to-date view of the actual situation inside the conferencing server.

Indeed, the notification mechanism described above, allows us to improve the overall performance of the integrated server made of the CCMP proxy combined with the Meetecho server. In fact, provided that the CCMP proxy is always kept aligned with the Meetecho server for all what concerns conference-related information, we can let it respond to CCMP client requests directly, thus skipping the complex operations associated with the needed ‘CCMP-Scheduler’ mapping procedures, along both directions.

Upon activation, the CCMP Server retrieves, through specific Scheduler requests, all the Meetecho blueprints and conferences and loads them into a native XML database. Conference objects hence take the form of XML conference documents compliant with the XCON data model. As stated above, the CCMP Server is also a subscriber to the Meetecho Notification Service, and is thus aware of all modifications taking place on the conferences managed by the Meetecho server (modifications which might also be due to actions undertaken by non-CCMP aware Meetecho conferencing clients). Accordingly to the received notifications, the database is updated. In such a way, the CCMP Server has always available an aligned image of the conference information set managed by the Meetecho platform. This allows the CCMP Server to immediately answer to CCMP retrieve requests, without forwarding the corresponding Scheduler request to the Meetecho server each time this kind of message arrives. Unlike the retrieve case, the CCMP requests associated with an operation of either create, or update, or delete must be translated into the equivalent Scheduler messages to be sent to Meetecho, in order to have an actual effect on the Meetecho Server side. The Scheduler responses are then interpreted and converted into the appropriate database updates, as well as translated into the equivalent CCMP responses to be returned to the CCMP Client.

Having transposed the Meetecho Conference Control plane to the CCMP world, we have integrated a library of CCMP APIs into our Meetecho client, thus allowing it to make use of CCMP (instead of the legacy Meetecho Scheduler protocol) as the Conference Control Protocol, in such way completing the scenario we presented in Fig. 6.

In the following subsections, we delve somehow into the details of the main actors involved in the Conferencing Control environment we have introduced, namely the CCMP client, the CCMP proxy server and the native XML database. We will discuss both the design and the implementation choices associated with the above mentioned components. Some notes and considerations about the way CCMP has been integrated into the Meetecho client are also reported.

6.1 Managing XML CCMP messages and conference information

The CCMP server and client components have both been implemented in Java. Since CCMP messages, as well as the conference-related information they carry, are formatted as XML documents, we faced the need of generating, parsing and handling such items in Java. Besides, as it will be explained in the following section, we also needed a proper way to handle an XML-aware database, which could manage the manipulation of conference objects.

In order to facilitate these operations, we chose to exploit the JAXB API (Java Architecture for XML Binding API) 2.1, which is the last API version at the time of this writing. This API allows to represent XML documents (that also have to be validated against a given XML Schema) in a Java format, i.e. through Java objects representing their different composing parts. The binding indeed represents the correspondence between XML document elements and the Java objects created with JAXB. Accessing XML contents by means of JAXB presents several advantages in terms of both efficiency and easiness with respect to SAX and DOM parsing. In fact, just like DOM, the output analysis can be saved at once and then consulted at any time without having to re-parse the whole document again, while the concerning memory occupation turns out to be lower than the one of

[1] This is due to the fact that, when the Meetecho XCON-compliant conferencing platform has been conceived, inside the XCON Working Group there was no consensus yet as to the standard conference control protocol to be adopted.

the DOM tree; like SAX, on the other hand, it is possible to access specific document parts without performing a further complete document parsing and without traversing the XML tree until the leaf to be examined is reached.

JAXB not only allows for easy access to XML documents, but also for a seamless creation of XML documents from the representative Java counterparts. This operation is called 'marshalling'. The inverse operation, from XML to Java objects, is instead called 'unmarshalling'.

Each JAXB generated class, corresponding to a specific type of XML element or attribute described in the schema file, is equipped with get and set methods that make it very easy to both extract information values and set them.

The JAXB architecture is composed of a set of APIs (contained in the `javax.xml.bind` extension package) and of a binding compiler, called XJC, which generates, starting from an XML Schema, the set of Java classes representing the element types embedded in XML documents compliant with it. In this context we have used the XJC Eclipse plug-in and produced the package of Java classes related to the XML Schema files collected from the data model documents [8, 7], as well as from the most up-to-date CCMP draft [3].

6.2 Managing HTTP

Considering the suggested transport for CCMP messages is HTTP (precisely, POST and 200 messages for requests and responses, respectively), we also had to cope with the issue of handling HTTP messages both at the client and at the server sides.

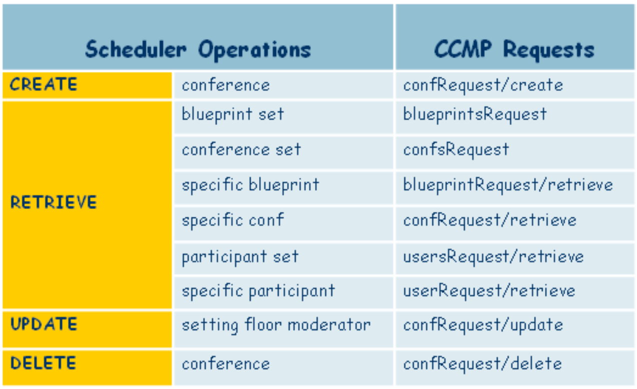
For the CCMP server implementation, we made use of the Apache open-source servlet engine Tomcat. The CCMP server business logic is realized through a servlet which, in the `doPost()` method, extracts the CCMP body from the HTTP POST request and, once the proper CCMP request type has been detected, starts the specific management thread accordingly.

On the client side, instead, we made use of the HTTP open source package provided by Apache, Apache Commons HTTP Client 3.1. This package is widely deployed in several projects, and allowed us to easily create and send to the CCMP server HTTP POST requests containing the CCMP message inside their payload, as well as handle the associated HTTP response accordingly.

6.3 Xindice database

We previously mentioned the need for an XML-aware database. In fact, CCMP handles the manipulation of conference objects compliant with the XCON common data model specification. Such conference objects are XML documents, and so, having an XML-aware database to store and manipulate them instead of relying on a relational databases relieved us form the burden of taking care of the transformation from tables to XML documents and viceversa whenever needed.

To cope with this requirement we chose Xindice, an open source Apache server handling an XML-native database specifically conceived for storing XML documents. Just as we needed, it allows to simply insert the XML data as it is when writing to the database, as well as to return the data in the same format when accessing the database. This feature is very useful when having to deal with complex XML documents like XCON conference objects, which might become very difficult or even impossible to be effectively stored in structured databases.



Scheduler Operations		CCMP Requests
CREATE	conference	confRequest/create
RETRIEVE	blueprint set	blueprintsRequest
	conference set	confsRequest
	specific blueprint	blueprintRequest/retrieve
	specific conf	confRequest/retrieve
	participant set	usersRequest/retrieve
	specific participant	userRequest/retrieve
UPDATE	setting floor moderator	confRequest/update
DELETE	conference	confRequest/delete

Figure 8: CCMP-Scheduler mapping

Xindice is installed as a Tomcat web application, and as such it was seamlessly integrated into our CCMP server prototype implementation. The `XML:DB` Java API is used to access the XML database. Such API is vendor-neutral, meaning that they are independent of the specific native XML database implementation, and operate on XML document collections, allowing the user to perform, on the collected XML documents, XPath queries as well as XUpdate modifications. Document collections are created and accessed through Xindice-specific Java APIs (Xindice Collection Manager Service).

In this context, we generated two main collections: (i) *confs* – the set of active and registered conferences hosted on the Meetecho server, reported in the form of conference documents compliant with the XCON data model; (ii) *blueprints* - the set of the Meetecho conference templates, in the XML XCON data model compliant format as well. A snapshot of the database content is showed in Fig. 7.

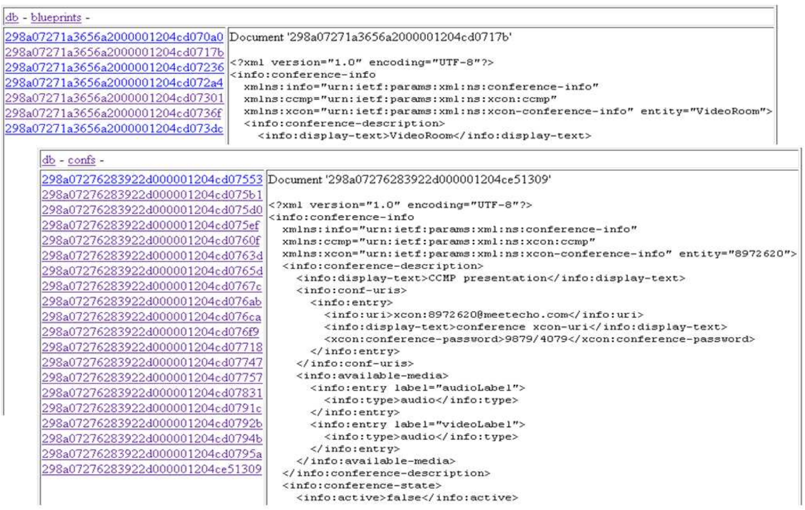
XPath queries are then executed by the CCMP server whenever needed, for instance to select the conference object referred to by the confObjID in CCMP requests, or to retrieve specific conference information from the XML conference documents grouped in the database collections.

XUpdate queries are instead performed to update the conference documents according to received Meetecho notifications (generated, for example, as a consequence of a new user join or leave event) and CCMP client requests (e.g. when a client sets via CCMP a participant as chair of a certain floor).

6.4 CCMP-Meetecho integration

As anticipated, our reference conferencing platform, Meetecho, does not support CCMP natively. It instead currently relies on a proprietary protocol, called Scheduler, to handle conference objects and their manipulation. This protocol has a limited set of functionality available, which nevertheless can be logically mapped in a quite straightforward way to a subset of CCMP operations. This motivated us into integrating CCMP in our platform by handling at first CCMP as a simple wrapper to the operations made available by the Scheduler. This mapping is presented in Fig. 8.

Specifically, the Scheduler protocol allows a participant to: (i) create a new conference; (ii) delete existing conferences; (iii) retrieve the list of available blueprints; (iv) retrieve a specific blueprint; (v) setting a participant as floor chair of



db - blueprints -

298a07271a3656a2000001204cd070a0
298a07271a3656a2000001204cd0717b
298a07271a3656a2000001204cd07236
298a07271a3656a2000001204cd072a4
298a07271a3656a2000001204cd07301
298a07271a3656a2000001204cd0736f
298a07271a3656a2000001204cd073dc

Document '298a07271a3656a2000001204cd0717b'

```
<?xml version="1.0" encoding="UTF-8"?>
<info:conference-info
  xmlns:info="urn:ietf:params:xml:ns:conference-info"
  xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
  xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info" entity="VideoRoom">
  <info:conference-description>
    <info:display-text>VideoRoom</info:display-text>
```

db - confs -

298a07276283922d000001204cd07553
298a07276283922d000001204cd075b1
298a07276283922d000001204cd075d0
298a07276283922d000001204cd075ef
298a07276283922d000001204cd0760f
298a07276283922d000001204cd0763d
298a07276283922d000001204cd0765d
298a07276283922d000001204cd0767c
298a07276283922d000001204cd076ab
298a07276283922d000001204cd076ca
298a07276283922d000001204cd076f9
298a07276283922d000001204cd07718
298a07276283922d000001204cd07747
298a07276283922d000001204cd07757
298a07276283922d000001204cd07831
298a07276283922d000001204cd0791c
298a07276283922d000001204cd0792b
298a07276283922d000001204cd0794b
298a07276283922d000001204cd0795a
298a07276283922d000001204ce51309

Document '298a07276283922d000001204ce51309'

```
<?xml version="1.0" encoding="UTF-8"?>
<info:conference-info
  xmlns:info="urn:ietf:params:xml:ns:conference-info"
  xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
  xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info" entity="8972620">
  <info:conference-description>
    <info:display-text>CCMP presentation</info:display-text>
    <info:conf-uris>
      <info:entry>
        <info:uri>xcon:8972620@meetecho.com</info:uri>
        <info:display-text>conference xcon-uri</info:display-text>
        <xcon:conference-password>9879/4079</xcon:conference-password>
      </info:entry>
    </info:conf-uris>
    <info:available-media>
      <info:entry label="audioLabel">
        <info:type>audio</info:type>
      </info:entry>
      <info:entry label="videoLabel">
        <info:type>audio</info:type>
      </info:entry>
    </info:available-media>
  </info:conference-description>
  <info:conference-state>
    <info:active>false</info:active>
```

Figure 7: An image of the Xindice native XML database used in the prototype

a media; (vi) retrieving the list of users in a conference. All these operations are made available by CCMP as well, and so this allowed us to test our prototype CCMP implementation in realistic scenarios.

The integration was realized by implementing a wrapper on the server side. We deployed our CCMP server (Tomcat, JAXB and Xindice) by putting it side by side with the existing Meetecho server. We then added to the already implemented CCMP server logic a wrapping functionality, in order to handle incoming CCMP requests and translate them into Scheduler directives accordingly, where applicable, and viceversa. On the client side, we replaced the Scheduler client module with our CCMP client implementation and logic.

The mode of operation is quite straightforward. Any time a participant issues a CCMP request, it is handled by the CCMP server. The CCMP server maps the request to the Scheduler counterpart, translating the message. Such a message is then forwarded to the legacy Meetecho Scheduler server, where it is handled and enforced. According to the Scheduler reply that is received as a consequence, the CCMP server takes the related action, e.g. updating the XML conference object on the Xindice database if needed, and providing the participant with a coherent CCMP response.

An example is provided in Fig. 9.

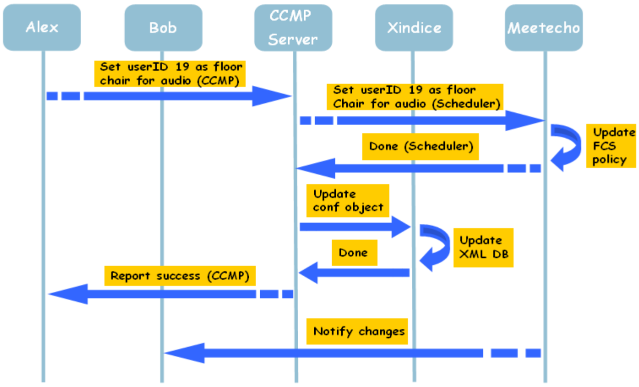


Figure 9: A sample CCMP-based interaction involving protocol mapping

A dump of the CCMP messages exchanged follows:

```
ccmpRequest message sent:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpRequest
        xmlns:info="urn:ietf:params:xml:ns:conference-info"
        xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
        xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
     <ccmpRequest
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:type="ccmp:ccmp-conf-request-message-type">
        <confUserID>xcon-userid:alex@meetecho.com</confUserID>
        <confObjID>xcon:8977777@meetecho.com</confObjID>
        <operation>update</operation>
      <conference-password>1377</conference-password>
        <ccmp:confRequest>
           <confInfo entity="xcon:8977777@meetecho.com">
          <xcon:floor-information>
                <xcon:conference-floor-policy>
                 <xcon:floor id="11">
              <xcon:moderator-id>19</xcon:moderator-id>
                 </xcon:floor>
                </xcon:conference-floor-policy>
            </xcon:floor-information>
           </confInfo>
        </ccmp:confRequest>
     </ccmpRequest>
  </ccmp:ccmpRequest>

ccmpResponse message received:

  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpResponse
       xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
       xmlns:info="urn:ietf:params:xml:ns:conference-info"
       xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp">
    <ccmpResponse
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="ccmp:ccmp-conf-response-message-type">
       <confUserID>xcon-userid:alex@meetecho.com</confUserID>
       <confObjID>xcon:8977777@meetecho.com</confObjID>
       <operation>update</operation>
       <response-code>200</response-code>
       <response-string>Success</response-string>
       <version>10</version>
    <ccmp:confResponse/>
  </ccmpResponse>
</ccmp:ccmpResponse>
```

This simple example shows the process for a typical scenario. In an active conference (identified by its ID 8977777), a participant, Alex (who happens to be administrator of the conference), decides to assign a floor chair for the audio resource, in order to have it properly moderated by means of BFCP. In CCMP, this is achieved by issuing a 'confRequest' with an 'update' operation: the body of the specialized 'confRequest' element contains the part of the conference object that needs manipulation, in this case the floor information associated with the existing audio resource. This audio resource is identified by means of a floor id (11 in the example), which in the conference object itself is explicitly mapped to the label assigned to the audio medium. Since Alex is interested in assigning a floor chair to take care of this medium, he specifies a 'moderator-id' (19) which refers to a specific userID in the Floor Control Server. A password is also provided (1377) since this operation requires special permissions.

This request is sent to the CCMP server which, since a mapping with the Scheduler functionality exists, translates the message accordingly to the Scheduler format, and sends the newly created message to the legacy Meetecho Server. The server handles the request and enforces it, updating the Floor Control Server policy accordingly. The successful result of the operation is reported by means of a Scheduler reply to the CCMP server, which in turn updates the Xindice database coherently with the request. This means that the XML conference object associated with conference 8977777 is updated. A success is finally returned to the participant by means of a CCMP response.

7. CCMP HISTORY AND RELATED WORK

Every time a framework for conferencing has been proposed, the need for a proper Conference Control mechanism has arisen as a consequence. For this reason, such mechanism has been the subject of a lot of efforts. Nevertheless, the proprietary nature of most of the conferencing solutions currently available paved the way to numerous heterogeneous and incompatible solutions for such a functionality. For the sake of conciseness, we don't provide in this section a list of such solutions, considering it would be quite incomplete. We instead focus on the related work carried within the standardization bodies. In fact, since the XCON architecture has been introduced within the IETF, several different candidates have been proposed to play the role of the Conference Control Protocol. Such candidates differed in many aspects, which reflected the discussion within the standardization fora with respect to the approach that should be taken in that sense. An interesting debate took place, for instance, about whether a semantic or a syntactic approach would be better as a basis for a Conference Control Protocol. Besides, the best transport means to be adopted has also been the subject of investigation.

In this spirit, at least three candidates were proposed in the XCON WG before CCMP was chosen as the official protocol.

The first proposal, at the end of 2004, was the "Centralized Conference Control Protocol" (`draft-levin-xcon-cccp`) by O. Levin and G. Kimchi. This protocol, as CCMP, was XML-based and had a client-server organization, but unlike CCMP it was designed using SOAP as a reference model. It likely reflected the implementation work carried out within Microsoft at the time. Despite being in a quite advanced state (four updates were submitted), the proposal was eventually put aside.

Shortly after the first individual submission of the CCCP, another candidate came to the light, "COMP: Conference Object Manipulation Protocol" (`draft-schulzrinne-xcon-comp-00`) by H. Schulzrinne. Like its predecessors, it heavily relied on Web Services as a reference, while stressing the use of SIP for notification purposes. Unlike CCCP, COMP had a strong semantic approach for what concerned the protocol specification. No updated versions of the draft were submitted; this work nevertheless paved the ground to a stimulating discussion that eventually led to CCMP.

One month later, another candidate was proposed, the "Conference State Change Protocol (CSCP)" by C. Jennings and A. Roach. Unlike both its predecessors, CSCP took a completely different approach towards the protocol. In fact, CSCP was basically a proposal to extend the already defined Binary Floor Control Protocol in order to allow it to also deal with conference manipulation functionality. CSCP motivated such an approach stressing the fact that binary messages would be smaller and easier to handle, especially for mobile devices. Besides, it was the authors' opinion that every XCON-compliant entity would likely support BFCP already, and as such CSCP would prove a trivial addition. Nevertheless, the proposal was eventually abandoned, and a text-, possibly XML-based solution was decided to be a preferred approach.

Finally, a last proposal saw the light at the end of 2005, the individual submission that would subsequently become the official CCMP draft. Such a draft has seen many revisions and efforts since then, which have resulted in the work presented in this paper.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the design and implementation of the Centralized Conferencing manipulation Protocol (CCMP), currently on the way towards its steady-state as a standard IETF protocol for conference objects management in the XCON framework.

We highlighted the main motivations behind such a work and illustrated the complex path that has been followed within the IETF community along the many phases of the overall standardization process.

We first described the general structure of the protocol, as well as its main functionality. Then, we focused on the work carried out at the University of Napoli during these last years and centered around a running prototype acting as a major playground for all the activities associated with ongoing standardization work inside some of the key working groups of the RAI (Real-time Applications and Infrastructure) area of the IETF.

At the time of this writing, the specification of the CCMP protocol is close to completion. Its implementation has been heavily used to both test its behavior and to provide invaluable feedback to the authors of the CCMP document. Furthermore, to aid implementors, a specific draft focusing on call flows has been written in order to provide the Internet community with guidelines in the form of Best Common Practices.

Our future work related to CCMP will definitely concern the final refinement of the specification, with the goal of arriving at a well-assessed RFC document. As to the implementation, we are currently working on the integration of the CCMP server within the Meetecho platform as a 'native' component, in such a way as to avoid the unavoidable burden associated with proxying CCMP requests and mapping them onto the legacy scheduler protocol.

When done with such integration, we will also focus on carrying out a thorough experimental campaign aimed at assessing the performance achievable by our protocol implementation, as well as identifying its potential bottlenecks.

9. REFERENCES

[1] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano. Meetecho: A standard multimedia conferencing architecture. In *FMN '09: Proceedings of the 2nd International Workshop on Future Multimedia Networking*, pages 218–223, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] M. Barnes, C. Boulton, and O. Levin. RFC 5239 - A Framework for Centralized Conferencing. Request for comments, IETF, June 2008.

[3] M. Barnes, C. Boulton, S. Romano, and H. Schulzrinne. Centralized Conferencing Manipulation Protocol (work in progress). Internet draft, IETF, June 2010.

[4] G. Camarillo, S. Srinivasan, R. Even, and J. Urpalainen. Conference event package data format extension for centralized conferencing (xcon) (work in progress). Internet draft, IETF, September 2008.

[5] Fielding. Architectural styles and the design of network-based software architectures. Technical report, 2000.

[6] M. Gudgin, N. Mendelsohn, M. Hadley, J. Moreau, and H. Nielsen. Soap version 1.2 part 1: Messaging framework. World wide web consortium first edition rec-soap12-part1-20030624, W3C, June 2003.

[7] O. Novo, G. Camarillo, D. Morgan, and J. Urpalainen. Conference information data model for centralized conferencing (xcon) (work in progress). Internet draft, IETF, February 2010.

[8] J. Rosenberg, H. Shulzrinne, and O. Levin. RFC 4575 - A Session Initiation Protocol (SIP) Event Package for Conference State. Request for comments, IETF, August 2006.

Work in Progress: Black-Box Approach for Testing Quality of Service in Case of Security Incidents on the Example of a SIP-based VoIP Service

Peter Steinbacher, Florian Fankhauser, Christian Schanes, Thomas Grechenig

Vienna University of Technology
Industrial Software (INSO)
1040 Vienna, Austria
{peter.steinbacher, florian.fankhauser, christian.schanes, thomas.grechenig}@inso.tuwien.ac.at

ABSTRACT

One of the main security objectives for systems connected to the Internet which provide services like Voice over Internet Protocol (VoIP) is to ensure robustness against security attacks to fulfill Quality of Service (QoS). To avoid system failures during attacks service providers have to integrate countermeasures which have to be tested. This work evaluates a test approach to determine the efficiency of countermeasures to fulfill QoS for Session Initiation Protocol (SIP) based VoIP systems even under attack. The main objective of the approach is the evaluation of service availability of a System Under Test (SUT) during security attacks, e.g., Denial of Service (DoS) attacks. Therefore, a simulated system load based on QoS requirements is combined with different security attacks. The observation of the system is based on black-box testing. By monitoring quality metrics of SIP transactions the behavior of the system is measurable. The concept was realized as a prototype and was evaluated using different VoIP systems. For this, multiple security attacks are integrated to the testing scenarios. The outcome showed that the concept provides sound test results, which reflect the behavior of SIP systems availability under various attacks. Thus, security problems can be found and QoS for SIP-based VoIP communication under attack can be predicted.

Categories and Subject Descriptors

C.2.0 [**Computer Communications Networks**]: General—*Security and protection, Data communications*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms

Security, Verification, Reliability, Performance

1. INTRODUCTION

Voice over Internet Protocol (VoIP) is steadily gaining a larger user base. Since VoIP communication is based on the Internet Protocol (IP), new challenges concerning the security of telephony arise. Information security objectives like confidentiality, integrity, robustness and availability must be achieved [4]. The most prevalent threats to VoIP systems today are known from data networks [9]. For example, many threats are known for IP based protocols like SIP and Real Time Protocol (RTP) for media transport in VoIP systems. VoIP specific protocols like Session Initiation Protocol (SIP) for call signaling increase the attack surface of such systems which requires new countermeasures for additional vulnerabilities in VoIP-telephony. The four largest threats faced by service providers today are Denial of Service (DoS), Service Abuse and Fraud, Spam over Internet Telephony (SPIT), and Eavesdropping (see Abdelnur et al. [1]). Keromytis mentioned that Service Abuse and DoS need more research [15].

The common similarity of different VoIP solutions is that all data is fully transmitted over IP. This includes media data (e.g., speech) and signaling data. This is not the case in traditional telecommunication systems which use Signaling System 7 (SS7)[1] for signaling. Data transfer is circuit switched and not packet oriented. These may transport IP traffic, but are not based on it. Today, VoIP and traditional telephony have an architectural split of signaling and transmission of media in common. For VoIP signaling, two protocol standards are commonly used: H.323[2] and SIP [22].

In this paper, the focus is on SIP which is defined by the Internet Engineering Task Force[3] and is designed to setup bidirectional communication sessions, not limited to only VoIP calls. If used for VoIP, it is recommended to use it with Session Description Protocol[4] (SDP) and Real Time Protocol[5] (RTP).

The focus of availability [4] is to assure information and

[1] `http://www.itu.int/rec/T-REC-Q`
[2] `http://www.itu.int/rec/T-REC-H.323`
[3] `http://www.ietf.org`
[4] RFC 4566
[5] RFC 3550

communication services will be ready for use when expected. Attempts to block or reduce availability are called Denial of Service attacks or Intentional Interruption of Service. DoS attacks on SIP systems cause loss of service to the users of that system [6]. When talking about DoS attacks in this paper we refer to network based attacks which are executed remotely over the network against services. This may be via the public Internet or on a local network, but without direct access to the operating system or the hardware of the target system.

Depending on the software used and the architecture of the system, different countermeasures are possible for protecting the VoIP system against availability failures. Regardless of the methods used, they should be evaluated on their effectiveness by testing. When evaluating the security of a given SIP based solution a software/system tester should be able to answer the question under which attack scenarios a certain Quality of Service (QoS) requirement or a Service Level Agreement (SLA) can not be met anymore.

We will present a security test concept which provides basic mechanisms to realize and execute such a test of availability against a SIP system. Thus, the evaluation of availability of SIP based VoIP systems will be examined by executing load and performance tests together with different types of DoS attacks, i.e., multiple test techniques are combined. The central idea for the security test concept is to simulate user load on a server as for example defined by QoS requirements while at the same time also performing DoS attacks. From the measured change of real time behavior effects of DoS attacks can be measured. Therefore, the results of a normal load without attacks can be compared with the results made during attacks. The concept is implemented as a black-box approach. This way, the internals of the system are not relevant for testing. Our traffic simulation only verifies and records output and response time of the interactions with the SUT. The test criteria for the defined concept for each imaginary user is service availability, basic functional correctness, and performance. We suggest a concept and describe a proof of concept test environment that makes it possible to define, execute, and evaluate such tests. However, a final set of test cases for SIP based VoIP systems will not be defined. The concept can be integrated into an existing comprehensive test process consisting of, for example, functional tests, performance tests, penetration tests etc. Such test processes already exist in most large IT infrastructures.

The remainder of this paper is structured as follows: Section 2 lists related work. Section 3 introduces the used security verification concept and points out the specific requirements for testing availability aspects during security attacks. Section 4 gives details about implementation of the test concept. The test approach and the implemented prototype were evaluated by doing several security tests which are presented in Section 5. The paper finishes with a conclusion and ideas for future work in Section 6.

2. RELATED WORK

An overview of current VoIP Security research is given by Keromytis in [15]. He identified two specific areas (DoS and service abuse) as being underrepresented in terms of research efforts. Peng et al. publish basic classification and description of DoS and Distributed Denial of Services (DDoS) prevention methods in [19], where countermeasures are distinguished in Attack Prevention, Attack Detection, Attack Source Identification and Attack Reaction. To ensure availability for SIP based services, a time critical method distinguishing between valid and malicious requests is required. In [3], Akbar et al. present an approach focusing on evolutionary algorithms to detect flooding attacks against SIP based VoIP systems. Countermeasures can also simply entail ensuring that enough resources like bandwidth, CPU time, memory, etc. are available and good implementations and configurations are chosen. Keromytis calls for additional effort at securing implementations and configurations [15] and also states in [16] that complex systems with default configurations remain a problem. Sisalem et al. show that the first line of protection starts by deploying a high-performance infrastructure [24]. This work points out that different approaches to protect systems against security attacks are available and after implementation a test of effectiveness is required.

Testing VoIP systems is an important method for revealing DoS vulnerabilities independent from their origins. Security testing approaches for SIP implementations focus testing in order to discover implementation vulnerabilities. For example, PROTOS test suite[6] finds DoS vulnerabilities by malicious INVITE packets, but does not mention flooding attacks. Although focusing on automated penetration tests, in [2] by Abdelnur et al. DoS vulnerabilities are mentioned, but flood attacks are not considered. These approaches could not verify how the behavior of SIP systems may change under system load during flooding attacks. Some SIP flooding attack examples are briefly described and evaluated by Endler and Collier in [8]. A systematic concept is missing in this work, though. A comprehensive testing platform is presented by Srinivasan in [25] which provides security tests. It additionally briefly considers simulated flooding attacks. Ming et al. demonstrate SIP vulnerabilities by CPU based DoS attacks in [17]. In [20] Rafique et al. evaluated DoS attacks against SIP proxy implementations and showed that those are vulnerable against different DoS attacks. They based the analysis on black- and white-box tests. The approach is similar to ours but we focus on integrating simulated security attacks into the test process and combine it with other test techniques of the test process, e.g., performance or penetration tests.

Performance testing of SIP is described from Montagna et al. in [18] and Schulzrinne et al. in [23]. Performance testing is one part of our approach to generate required load.

We suggest a more comprehensive security verification approach towards an iterative test process. Foundations, principles and terminology of general software testing are taken from [6, 12, 13, 14].

General requirements for testing QoS in case of security attacks are gained. Attack types should not be limited. Instead various attack types are used and using threat modeling is suggested. Black-box testing detaches from the tested systems, generalizes the test environment and includes con-

[6] `http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip`

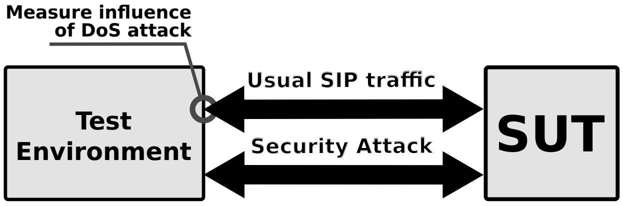


Figure 1: Basic Test Environment of the Security Test Concept

figuration weaknesses. Additionally, aspects of QoS, test coverage, and traffic modeling are introduced to the test approach. Most common SIP scenario examples are given in RFC3665 [7], which can be used to create traffic models. According to Endler et al. (see [6]) traffic is defined as the flow between any collection of network traffic nodes. For the security test concept it has to be decided on a traffic model which should be an approximation to real user interaction with the SUT. The traffic model has to reflect the number of users and their use case scenarios. Jung-Shyr et al. propose a VoIP Traffic Model for SIP signaling in [26]. For the prototype of the framework presented in this paper some attack implementations are based on examples from [8]. Hassan et al. introduce advanced traffic modeling for SIP in [10].

3. VERIFYING AVAILABILITY ASPECTS DURING SIMULATED SECURITY INCIDENTS

When providing services for users, availability is one of the main objectives. In many large IT infrastructures, Service Level Agreements (SLAs) exist that define availability requirements. If those requirements cannot be fulfilled, most of the time punitive damages occur. An attacker may find a method to make service unavailable or to disturb it. If an attack is successful, call setup may not be possible, it could take too long or a call may not be released. In all cases, the attack has an impact on the users and the performance, the QoS requirements are not met and possible SLAs are violated. Measuring the influence an attack has on the user of a service is the basic idea of our approach of a security test concept for VoIP.

3.1 Security Attacks

A network based DoS attack can be achieved in two basic attack methods (see [5, 24]): (1) Consume available resources and (2) bring the system into a faulty state. A third option is a combination of both: Consume the available resources *by* bringing a lot of system processes into a (faulty) state which needs a lot of resources. Consuming resources can be achieved with attack methods like Flooding, which is possible on various protocol layers. Limiting the availability by bringing the system into a faulty state requires to compromise either the whole system or only particular processes. Therefore, possible exploitation of known vulnerabilities and design weaknesses have to be tested [5]. This is achieved by sending malformed messages or by disturbing an ongoing valid communication by spoofing.

3.2 Basic Verification Concept

Valid SIP traffic and simulated attacks are combined in a test case. Figure 1 illustrates the basics of the security test concept. The tested system providing the services is referred to as System Under Test (SUT). Hereby, it is important to clearly define a border where SUT ends and the *Test Environment* begins. For example, it can simply be the SIP implementation, the whole server or a whole server infrastructure including attack detection and prevention systems, etc. *Usual SIP traffic* represents the simulation of a certain number of valid users of the SIP service, for example a SIP proxy/registrar. A point to measure this traffic is observed, various metrics are recorded and change of QoS is recognized here. The simulated load should be at a normal range, e.g., requirements on QoS should be met, because the presented security test concept is not a common performance test or a stress test. In addition to the simulation of valid users using the SUT, every security test simultaneously includes a simulated security *Attack*. To evaluate the influence of a certain conducted security attack, only the change of user traffic is controlled. The success of an attack is defined by the extent of influence on user interaction during attack time. Test results are gained by analyzing the interactions of simulated user scenarios with and without an attack. The main capabilities of the *Test Environment* have to be (1) control simulation of valid and malicious interaction with the SUT and (2) record expressive metrics.

3.3 Used Availability Metrics

As a black-box test, we only measured calls at the User Agents' (UA) side. In this context a *call* is an execution of one single SIP scenario. It might be a telephone call (starting with the SIP INVITE message, transmitting voice data, ending with the SIP BYE message), but can also be a SIP registration, a message, etc. A call is composed of one or more SIP transaction(s), which starts with a SIP request and ends with a response. We count the absolute number of calls c during a certain time interval $[t_i, t_j]$. We distinguish between *call attempts* c_a, calls which are completed *successful* c_s, and calls which are not completed and *fail* c_f. The criteria when a call fails must be defined by test criterion (see Section 3.6) and implemented into the call scenario. Additionally the number of all *SIP retransmissions* r is counted.

A *rate* (RE) is an average value, which expresses a certain number of events per unit of time. Without a mean value, the number of calls is not expressive and comparable in cases of various length of time intervals. For example, *Call Rate* is calculated by $CRE = \frac{c}{t_j - t_i}$. Accordingly, we use the *Call Attempt Rate* as $CRE_a = \frac{c_a}{t_j - t_i}$, *Call Success Rate* as $CRE_s = \frac{c_s}{t_j - t_i}$, and *Call Failure Rate* as $CRE_f = \frac{c_f}{t_j - t_i}$. The unit used is *calls per second (cps)*. We will concentrate on CRE_a and CRE_s. The reason for that is that the start of a call can be in a different time interval than the end (see Figure 2 and Section 3.4). The decision that a call has failed can only be made after a certain timeout. Therefore, both rates are not complements within a time interval ($CRE_{s[t_i,t_j]} + CRE_{f[t_i,t_j]} \sim CRE_{a[t_i,t_j]}$). Only for *all* calls in an interval from the beginning of the simulation to the end of it ($[t_0, t_5]$, see Figure 2) the equations $c_a = c_s + c_f$ must be true. This implies waiting for the last timeout of every session before analysis can start.

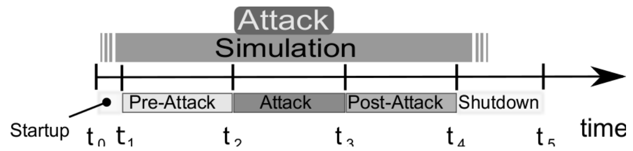


Figure 2: Phases of the Executed Test Presented as Time Line

Further relevant rates are the *Retransmission Rate* ($RRE = \frac{r}{t_j - t_i}$) for SIP UDP retransmissions, UDP *Attack Rate* for *attack packets per second* sent to SUT and *Data Rate* for the number of bits that are transmitted per unit of time with usual units kbit/s and Mbit/s. In this context, the total size of the packets, which are sent over the physical layer, is relevant.

Ratios (RO), different values, especially rates, are related to each other. For example, the *Call Success Ratio* ($CRO_s = \frac{CRE_s}{CRE_a}$). A further useful ratio is the average of *Retransmissions per call* ($RPC = \frac{RRE}{CRE_a}$) during a period of time. During an attack, RPC is assumed to increase. It also reflects the additional traffic caused by the SIP retransmission mechanism. Another expressive metric is the *Call Duration Time* t_{c_s} of successful calls or the average of them ($\bar{t}_c$) during an interval. In our test example in Section 5 we use both CRO_s and RPC.

3.4 Measurements During Test Execution

The proposed security test concept must distinguish between different phases of a test case to evaluate the influence of an attack. This is done by dividing the time of testing into sections as it is shown in Figure 2. Simulated user interaction begins at t_0 and lasts until t_4. During that time, an attack is started from t_2 to t_3. Recording and measurement of the test case begins at t_1. These five points in time mark the following phases: Startup phase (t_0 to t_1), Pre-Attack phase (t_1 to t_2), Attack phase (t_2 to t_3), Post-Attack phase (t_3 to t_4), and Shutdown phase (t_4 to t_5).

The *Startup* phase is the beginning of the test run. Its purpose is to bring the user load to a defined number. Its length depends on the traffic model and duration of single SIP calls and transactions. Scenarios should be started in parallel (see Section 4.1) and it may take some time until the full load is set up. From the view point of the SUT, this means that the allocation and release of resources is at the valid and intended level. At t_1, the full level of load is reached for sure, so measurement starts. The *Pre-Attack* phase measures timeouts and failures before attack. It is necessary to ensure that simulated traffic is running with normal load and failure rate. Results from the *Pre-Attack* phase can be compared to results from other test phases occurring at later steps. This way the relative impact of an attack can be measured and reported. In the *Attack* phase, timeouts and failures which appear during the various conducted security attacks are measured. At t_2, the security attack defined by the test case starts. If a security attack is successful, starting from t_2, effects should be measurable at the testing framework simulating the user interaction. The duration of this phase strongly depends on the type of attack. A flooding attack has to be long enough to predict behavior of SUT on longer attacks. For example, a flood attack with a duration of 10 seconds would increase retransmissions by UAs, but no call would fail. Only after a transaction timeout the failure of a call can be decided. Timeouts and call duration of the used scenarios have to be considered when attack duration is calculated. On the other hand, for a vulnerability exploit attack which consists of only one single request, the duration of the Attack phase will be relatively short. In the *Post-Attack* phase timeouts and failures, occurring after the attack has stopped, are measured. In this phase, user load results are recorded in the same way it is done in the *Attack* and *Pre-Attack* phases. The *Shutdown* phase ends the test case and stops the UA simulation. For verification of whether the measurement mechanisms work correctly ($c_a = c_s + c_f$), availability metrics (see Section 3.3) are used.

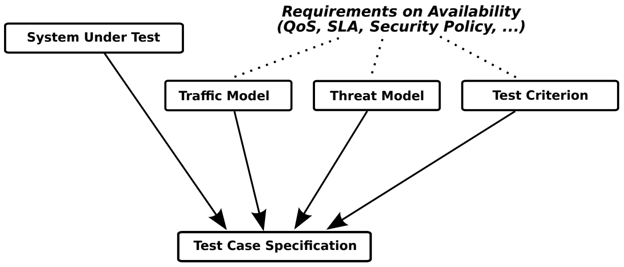


Figure 3: Model of Used Test Case Specification

After testing, only the recorded metrics of the main phases (*Pre-Attack*, *Attack*, and *Post-Attack*) are used for further analyses. By comparing test results from different phases of the test case the efficiency of security measures can be determined. To distinguish result metrics from different phases we add a postfix to the identifiers and refer to them as CRE_{PRE}, CRE_{ATT}, CRE_{POST}, RPC_{ATT}, etc.

During implementation we found that one of the most interesting comparisons are *Pre-Attack* to *Attack* and *Pre-Attack* to *Post-Attack*. The former will reveal influences of the attack whilst it is performed. It can be defined as $CRO_{sATT/PRE} = \frac{CRE_{sATT}}{CRE_{sPRE}}$. CRO expresses the degree of successful calls whilst in the Attack phase, in comparison to the Pre-Attack phase. The latter comparison reveals whether or not the SUT undergoes a lasting influence after the exection of security attacks. This is the case when, for example, an attack brings the system to a state of permanent failure or the attack initiats a long lasting recovery procedure. It can be expressed as $CRO_{sPOST/PRE} = \frac{CRE_{sPOST}}{CRE_{sPRE}}$.

3.5 Requirements For Test Case Specification

One critical part of a security test concept is the test case specification as mentioned by Kapfhammer in [14]. The first step must specify which system has to be tested and against which requirements. These two issues are illustrated in Figure 3 as *System Under Test* and *Requirements on Availability*. The requirements on availability have various input sources like SLAs, QoS definitions, security requirements, etc. Representativeness of attack scenarios and user workload has to be given.

The *Traffic Model* defines all call scenarios which are expected to work. Additionally, it defines traffic load at different levels, e.g., it may even stress the SUT. Security requirements must consider possible attacks by a *Threat Model*. The model should contain possible security attacks, the plausibility of occurrence and the damage they would cause in such a case. The *Test Criterion* defines which behavior is expected under various conditions.

3.6 Test Criteria

In functional software testing the term *pass/fail criteria* is used. The IEEE Standard for Software Test Documentation [11] defines the term as "Decision rules used to determine whether a software item or a software feature passes or fails a test." With a minor adaption this definition is suitable for security tests regarding availability, too. Since non functional requirements are tested, *features* are replaced by *various availability requirements*. QoS can be defined as a minimum on call success ratio CRO_s. For example: "Call setup with INVITE has to succeed in 99.7% of all attempts ($CRO_s = 0.997$)"

Usage of fail/pass criteria is required for two different levels: *scenario level*, which represents single transactions, and *availability level*, which is the aggregation of all transactions. The *fail/pass criteria on scenario level* must be defined for a single scenario (see Figure 4). Correctness is checked at the scenario level and given when a communication is completed successfully within a defined time. A scenario fails if the response time is too long or from an unexpected result from the SUT. A *fail/pass criteria on availability level* is set for a traffic model. It executes a number of different scenarios which have to be successful to a certain degree. If more than a specified number of scenarios fail, the availability is not given and the security test fails.

A further general post condition for every test case can be that *Pre-Attack* results and *Post-Attack* results should not differ significantly. If they are not equal, the system suffers ongoing damage from the security attack. This means that the security measures taken are not sufficient to protect the SUT from the conducted security attacks.

4. IMPLEMENTATION OF A PROTOTYPE

One of the principles of our security test concept is that it should be based on established evaluation and test methods. Moreover, our security test concept is a test of *system performance* during a DoS attack. The objective is to stress the SUT while at the same time including further security attacks. The test results, figures and statistics are similar to performance tests. Also, the simulation of UAs is alike. Existing simulations of user interaction, tools, techniques, and statistical analyzing methods can be reused.

Concerning security attacks these simulations may range from simple exploit scripts up to more complex penetration tests or fuzzing tools. New ideas for testing and attacking can be implemented quickly by scripting. With respect to the simulation of UAs various load generation tools are usable.

Based on the implementation of a prototype we verified our security test concept for testing QoS in case of security incidents for SIP-based VoIP services. The results of using the developed prototype are shown in Section 5.

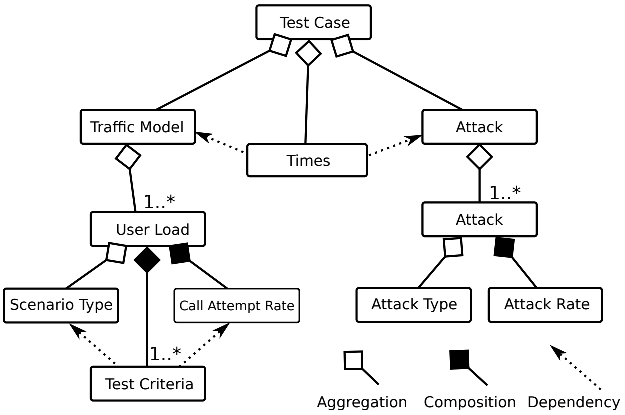


Figure 4: Model of Test Case Definition

The implemented prototype supports a tool chain which enables the generation of test cases out of defined attacks and traffic models. Therefore, the main focus for the security testers can be the definition of attacks and the interpretation of the test results obtained by the prototype.

4.1 Test Case Description

A test case in our security test concept is an aggregation of a *Traffic Model*, one or more *Security Attacks* and a set of *Times* (t_0, t_1, t_2, t_3 and t_4). A model of items a test case can be composed of is shown in Figure 4.

The *Traffic Model* is an aggregation of one or more *User Load(s)*. A *User Load* is a single type of scenario (e.g., registration or call setup through a SIP proxy) with a certain Call Attempt Rate CRE_a. This is a simplified concept of a *Traffic Model* adapted for the SIPp[7] traffic generator. Different scenarios are executed in parallel. This represents a simple approximation of real users interactions. In more complex setups, generators with advanced functions and models closer to reality can be used instead.

As mentioned in Section 3.5 an important issue in this security testing concept is that such a *User Load* must also define test fail/pass test criteria (*Test Criteria* in the model) for a single scenario. It could be implemented as a timer into the scenario, so that a scenario fails because of a timeout defined by the *test case*.

Often, test results of single scenarios are not meaningful in themselves, expressive results can only be calculated after the execution of all scenarios of a test case, e.g., the mean time of all responses has to be calculated at the end of a test case. This way, the impact of an attack on a certain kind of user traffic and therefore QoS can be seen.

4.2 Design of the Test Environment

The main task of the security test prototype which executes test cases is to start and stop processes at a defined time. A further requirement is to be able to track all relevant events

[7] SIPp: Open Source test tool/traffic generator for the SIP protocol (`http://sipp.sourceforge.net`)

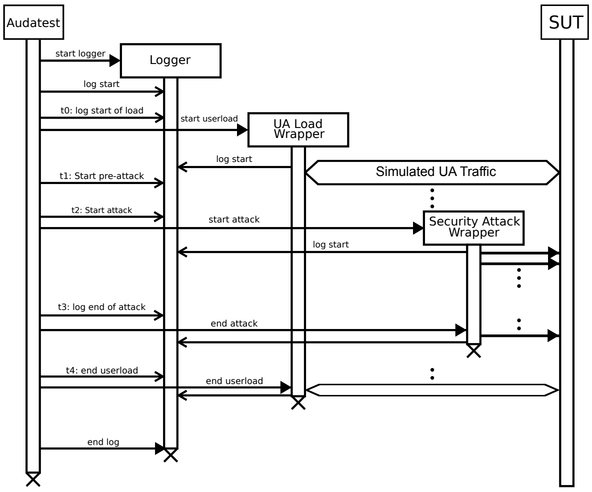


Figure 5: Sequence Diagram of Test Execution

during a security test by logging. In Figure 5 the implementation of the prototype is shown as a sequence diagram.

For our approach we used the SIPp framework for simulating the UAs, for measurement and for the statistics. The framework was not modified and is only started and stopped by a wrapper. This architectural decision makes it possible to easily replace the traffic simulation with other load generating tools. The complete interface to load generating tools is encapsulated into a wrapper. For example, the stopping of SIPp needs special handling, which the implemented SIPp wrapper considers. The same wrapper principle is used for the attack execution. It can start and stop various attack processes concurrently and supplies starting parameters (see aggregation of *attack* in Figure 4). Security attacks which are able to manipulate packets at the IP level (e.g., IP addresses to simulate DDoS attacks) need raw socket access and respective system privileges. In large IT infrastructures the setup can be distributed. This enables even greater scalability.

5. RESULTS

In order to evaluate the security test concept a test was defined and executed within a test environment. The first part of this section will present a definition of the test used for getting the presented results. The test scope is simplified in various aspects due to illustration purposes. The second part describes test results and illustrates them by diagrams. First, a performance test is analyzed which determines general system performance. Afterwards, results of a single test of a DoS attack by malformed SIP MESSAGE requests are presented. Next, the same attack type is illustrated with different call and attack rates. As a comparison an INVITE flood attack is shown in the same manner thereafter. Finally, various flooding attack types, e.g., UDP Flood and SIP REGISTER Flood, are compared.

Based on the results conclusions about behavior of the SIP service can be made. A general DoS vulnerability of UDP based SIP systems could be confirmed during the testing of the security test concept. Moreover, multiple impacts of DoS attacks of VoIP services could be shown with our prototype.

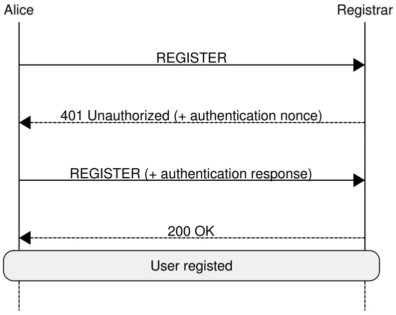

Alice
Registrar
REGISTER
401 Unauthorized (+ authentication nonce)
REGISTER (+ authentication response)
200 OK
User registed

Figure 6: Simple Traffic Model of SIP Registration Scenario With Authentication [7]

5.1 Definition of a Test Task Example

As the SIP server implementation, Kamailio (OpenSER)[8] version 1.5.3 was chosen. It is connected to a MySQL database version 5.0.51. The operating system is an Ubuntu Server 8.04 (Linux 2.6.24) with default settings, i.e., no hardening was done.

Before executing a black-box security test regarding availability it has to be ensured that the dimensions of the test environment are sufficient and the test environment works correctly. Test results must not be distorted by a lack of system resources of the test environment, e.g., the performance of client systems or routers, which are not the subject of the test, has to be checked.

For demonstration purposes our example test scenario includes a simplified traffic model with one scenario type (SIP Registration with Authentication) with five different Call Rate Attempts CRE_a (see Figure 6 and [7]). This represents one scenario which is mandatory for every usual SIP communication. If an attacker achieves disruption of the registration service, users could not be reached any longer. The threat model has a scope of five attack types with six attack rates (30 different attacks). Testing all possible combinations results in 150 test cases (30 attacks * 5 traffic models). To evaluate the influence of a workload we define different possible call rates (105, 210, 315, 420 and 525 cps). This test example is about finding the right dimensioning of a system to ensure QoS under attack. Instead of testing various call rates, different SIP software, SIP servers, countermeasures, configurations, etc. could be tested and compared the same way as it is done here on call rates. Furthermore, for a detailed definition of a test case, testing times are needed and user load and attack call rates have to be selected (see Figure 4).

In the case of a successful DoS attack, users cannot register and are not able to be reached via SIP. As an average registration expiration time, 300 seconds are expected. This causes a certain amount of valid traffic which depends on users per registrar as shown in Table 1.

The workload is related to a maximal load, which was empirically evaluated by a performance test.

[8] Kamailio (OpenSER): The Open Source SIP Server (`http://www.kamailio.org`)

Table 1: Equivalence of Registration CRE_a [cps], Currently Registered Users, Workload, and Data Rate.

Registration CRE_a [cps]	Currently Registered Users	Workload (% of max.)	Data Rate [kbit/s]
105	31500	15%	761
210	63000	30%	1520
315	94500	45%	2310
420	126000	60%	3028
525	157500	75%	3800

Table 2: Attacks Used for Test Case Specification

Flood Type	Package Size [Byte]	Target Layer, Target Component
UDP	1414	syntax and encoding, consuming bandwidth
INVITE	1102	transaction, transaction user, consuming system resources
MESSAGE malformed	402	syntax and encoding, consuming system resources
REGISTER	328	transaction user, database, consuming CPU
REGISTER malformed	1422	syntax and encoding, consuming CPU

In accordance with the description of *Test Criteria* in Section 3.6, two level criteria must be defined: For the fail/pass criteria at the scenario level, the default SIP timeout is used: After 32 seconds without response to a SIP request the call fails. For the fail/pass test criteria at the service level we define two criteria: The first is deduced from QoS requirements for operation without security attack aspects: “A minimum of 99.9% registrations must be successful.” This means the Call Success Ratio has to be $CRO_s > 0.999$. The second criteria takes a security attack into account and in such a case a tolerant QoS is expected: “During a security attack a minimum of 80% of UA registrations should be successful within 32 seconds (default SIP timeout).” In this case retransmissions are allowed, nevertheless, $CRO_{sATT} > 0.8$ should be fulfilled. Therefore, a successful DoS attack is given if more than 20% of registering UAs could not register and would not be reachable.

The *Threat Model* in our test case example consists of a set of five flooding attack approaches and types, e.g., UDP floods, INVITE floods, malformed MESSAGE floods, REGISTER floods, and malformed REGISTER floods by using a big but valid header. This way, the attacks represent flooding on various protocol layers. Further attacks are just flooding and flooding with malicious malformed packages. An overview is given in Table 2. The attack rate is 1000, 3000, 5000, 7000, 9000 and 11000 packets per second. The first approach is to flood by big sized, meaningless non SIP UDP packets on the default SIP port 5060. The second flood type is sending SIP UDP requests with a malformed and malicious header information (MESSAGE request). Both flooding approaches target the SIP parsing layer. Next, two attacks target the SIP application layer by sending the valid requests INVITE and REGISTER are executed. Another REGISTER flood has a valid header, but is malformed by having a lot of redundant information.

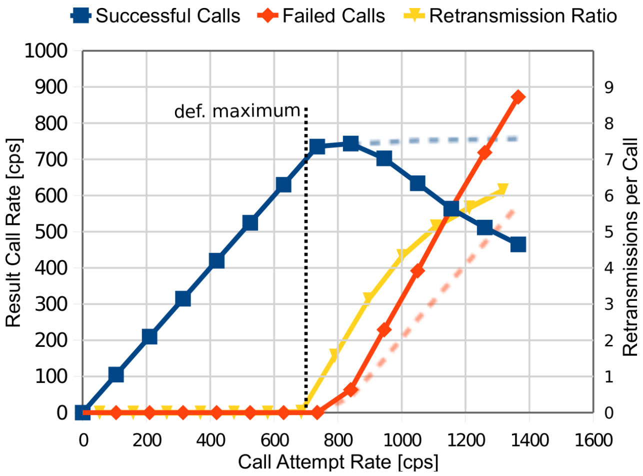


Figure 7: Result of a Performance Test to Evaluate the Maximum Workload Under Normal Conditions

For the *Phase Duration Times* it has to be considered that for the load test tool used the logging time must be set. This is the time the result statistic will be summarized and stored. If it is too short, the resolution is very high and too much data for further analysis will be produced. Therefore, we set the logging time to 15 seconds for SIPp and Phase Duration Times accordingly ($t_0 = 15, t_1 = 120, t_2 = 180, t_3 = 120, t_4 = 15$ seconds). One test case lasts 450 seconds. Since we have 150 possible test cases in our test example, the minimum testing duration for all test cases is 18 hours and 45 minutes.

5.2 Performance Test Results

First a performance test was made with the same user load and test criteria as used for security tests later. The result shows system performance under normal conditions. The performance test was done by increasing load in steps by 105 calls per second. Figure 7 shows the results. The register scenario (see Figure 6) consists of two SIP requests. In Figure 7 it can be seen that up to 735 cps, the system has no failing calls. Failures start slightly at 840 cps, but increase steeply. Successful calls start to decrease at 945 cps. To have a little buffer, the maximum workload is set to $CRE_{aMAX} = 700$ for further tests.

Additionally, the dotted lines in Figure 7 show what normally would be expected from a network service: A certain amount of requests can be handled and the rest is rejected and failures increase by the same amount as the request rate does. However, in our case call success decrease quickly with increasing load. The reason for this behavior is the SIP retransmission mechanism for UDP based transport [22]: For every unanswered SIP, a non-INVITE request retransmission is done after a time. With default SIP timers SIP UAs send up to 9 UDP packets per request (see Figure 8). As a pass/fail criteria at the scenario level, the default RFC3261

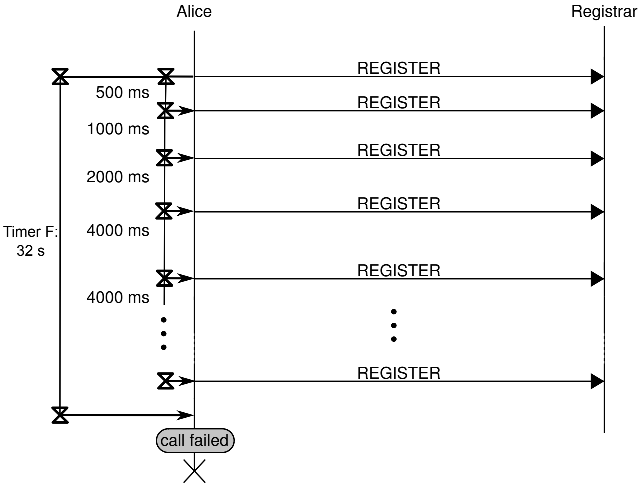


Figure 8: Registration Scenario With UDP Transport When Registrar is not Answering

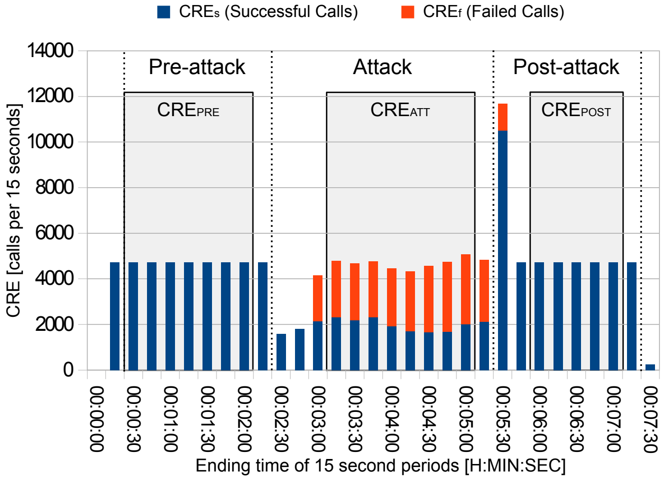


Figure 9: Flood With Malformed MESSAGE Request With 3000 UDP Packets per Second and 315 cps Call Rate

timer F is used (32 seconds). After timer expiration, the call is failed by timeout. This means, if a service does not answer, because it is overloaded, UAs start to increase the load. This way UAs amplify the traffic and in a way support DoS attackers. This is an architectural SIP problem and is discussed by Rosenberg in [21].

5.3 Results With Simulated Security Attacks

Figure 9 shows the result of a test case execution with a call rate at 315 registrations per second. SIPp dumps results every 15 seconds into the results file, that is 4725 calls during a 15 seconds period. During Pre-Attack phase no single call failure occurred $CRE_{sPRE} = 315[cps]$. After 135 seconds, a security attack (DoS) starts and the number of successful calls decreases under 2000 calls within the first 15 seconds. There are no failing calls, because of the retransmission mechanism for UDP transport. By default, retransmissions are done for 32 seconds until Timer F (see [22]) fires a timeout (see Figure 8). That is why after 30 seconds of the attack the first calls fail. The rest of the attack phase only about 40% of call attempts result in successful completed calls. The peak at the beginning of the Post-Attack phase is also a consequence of retransmissions which can now be handled because the attack has stopped. Call failures in this phase are caused by the large number of retransmissions.

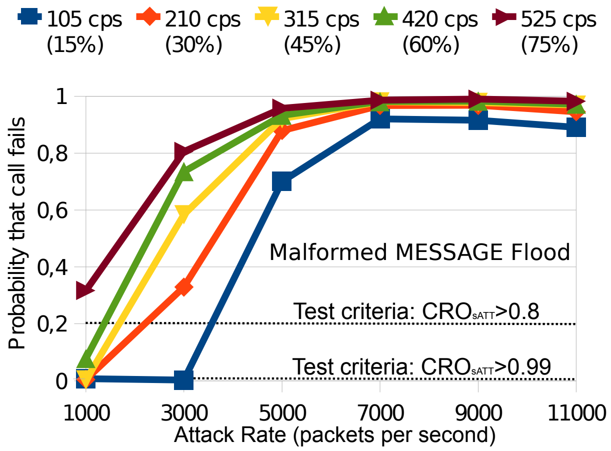


Figure 10: Effect of Malformed MESSAGE Flood Illustrated by Different Attack and Call Rates

The Post-Attack phase has 100% successful calls and so no enduring effect was caused by the attack. To reduce test case results to a single figure, $CRO_{sATT/PRE}$ is calculated as follows: Within the time interval of $[180, 300]$ seconds 37802 call attempts were executed ($CRE_{aATT} = 315.02$), 21618 call failures happened ($CRE_{fATT} = 180.15$), and 15747 calls were completed successfully ($CRE_{sATT} = 131.23$). As mentioned in Section 3.3, failed and succeeded calls are not complements ($CRE_{sATT} + CRE_{fATT} \sim CRE_{aATT}$). To reduce complexity, CRE_{fATT} is withdrawn and $CRE_{aPRE} = CRE_{aATT}$ is assumed.

$$CRO_{sATT/PRE} = \frac{CRE_{sATT}}{CRE_{sPRE}} = \frac{131.23}{315} = 0.42$$

This means that only 42% of the expected successful calls actually were successfully completed. None of the test criteria $CRO_{sATT} > 0.99; CRO_{sATT} > 0.8$ is fulfilled and the SUT failed the security test and thus the QoS requirements. $CRO_{sATT/PRE}$ can also be referred to as the probability that a single call is completed under attack. To illustrate the effect of the attack, in Figure 11 the probability that a single call fails is calculated ($P_f = 0.58$).

Every data point in Figures 10, 11, 12, and 13 represents a single test case result. The influence of different workload on a SIP server instance is visualized by different lines in Figure 10 for a malformed MESSAGE flood and in Figure 11 for an INVITE flood. In Figure 12 and Figure 13 the different lines represent types of attacks and their influence on the same workload of 315 cps.

Tests are missing the test criteria when they are above the criteria lines. For example, it can be seen in Figure 10, which shows a malformed MESSAGE flood, that nearly all

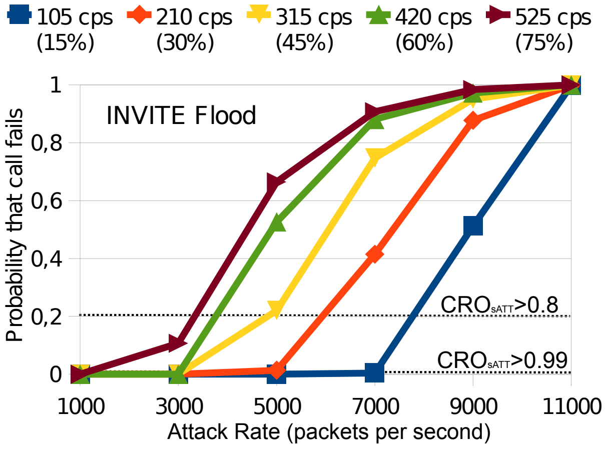


Figure 11: Effect of INVITE Flood Illustrated by Different Attack and Call Rates

tests pass a DoS attack at 1000 packets per second. The single exception is at a call rate of 525 cps. In this case 1000 packets per second are enough that the defined QoS cannot be met. DoS attacks with 3000 packages per second have a much greater impact on QoS requirements. In this case, the QoS requirements can only be met with a maximum workload of 105 cps. All tests with 5000 pps and above fail.

This result can be compared to the INVITE flood shown in Figure 11 where at 1000 pps no test fails, at 3000 one test fails against the $CRO_{sATT} > 0.99$ criteria, and so forth. Finally, at 9000 pps all tests fail.

A further metric used for flooding attacks is the network load. In Figure 13 the results of all five tested attacks are set in relation to the amount of data sent over the network. In this scenario, a *Malformed MESSAGE Flood* attack has the most influence: At 15 Mbit/s over 90% of all calls fail. All other attacks need over 79 Mbit/s to reach this value of CRE_f. For *UDP Flood* and *Malformed REGISTER Flood* attack the failures start to increase at about 10 Mbits/s. The *INVITE Flood* starts at about 35 Mbit/s, but the effect of the DoS attack increases in a fast way. The *REGISTER Flood* was tested with up to 11000 packets per second, which equalizes 29 Mbit/s but it has no visible effect on the users.

6. CONCLUSION AND FURTHER WORK

The suggested black-box test approach, test process and test implementation is able to predict the behavior of SIP communication under various attacks. The influence on various types and amount of user interaction is considered. It is possible to determine attacks and conditions where certain QoS requirements are not met. Results showed that a certain attack type has greater impact at the same attack strength than others, so it is essential to test various attacks to get more comprehensive results for security tests and thus predictions of compliance with the QoS requirements. Even though a small set of attacks was chosen, unexpected system vulnerabilities became obvious. A wider spectrum of tests could certainly identify further security problems.

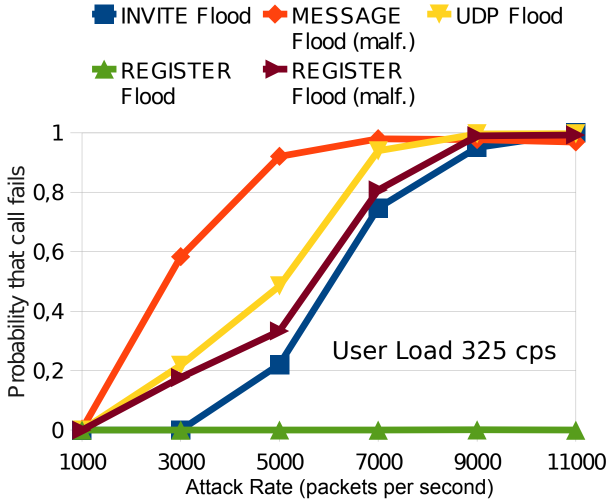


Figure 12: Attacks Compared by Different Attack Rate (Packets per Second)

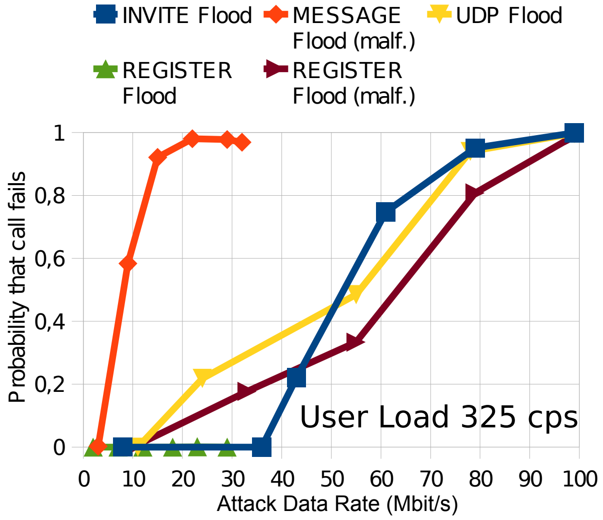


Figure 13: Attacks Compared by Different Attack Rate (Attack Data Load in Mbit/s)

A potential improvement is to shorten execution time of test cases. A further area of investigation is to establish models for both the attack scenarios and the user workload. For choosing representative attacks, existing threat models of SIP service could be consulted. Additionally, possible DoS attacks against RTP payload could be taken into account. The traffic model should be based on statistics of real VoIP systems, for more realistic simulation of user interaction. Another kind of research may be done on iterative cyclic test processes hardening VoIP systems using this concept.

Functional tests, performance tests and penetration tests are often executed in large IT infrastructures. In this paper we showed that this is not enough for building systems with high availability requirements like SIP-based VoIP services. Our results indicate that by combining multiple test techniques like performance and penetration tests it is possible to evaluate quality of service requirements and greatly enhance the security level of critical services.

7. REFERENCES

[1] H. Abdelnur, V. Cridlig, R. State, and O. Festor. Voip security assessment: methods and tools. *VoIP Management and Security, 2006. 1st IEEE Workshop on*, 0(0):29–34, April 2006.

[2] H. Abdelnur, R. State, I. Chrisment, and C. Popi. Assessing the security of voip services. -, 2007.

[3] M. A. Akbar and M. Farooq. Application of evolutionary algorithms in detection of sip based flooding attacks. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1419–1426, New York, NY, USA, 2009. ACM.

[4] M. Bishop. *Introduction to Computer Security.* Addison-Wesley Professional, 2004.

[5] R. Chang. Defending against flooding-based distributed denial-of-service attacks: a tutorial. *Communications Magazine, IEEE*, 40(10):42 – 51, oct 2002.

[6] E. David, D. Ghosal, R. Jafari, A. Karlcut, M. Kolenko, N. Nguyen, W. Walkoe, and J. Zar. Voip security and privacy threat taxonomy. Technical report, VOIPSA, 2005.

[7] S. Donovan, R. Sparks, C. C., and K. Summers. Rfc3665: Session initiation protocol (sip) basic call flow examples. http://www.ietf.org/rfc/rfc3665.txt.

[8] D. Endler and M. Collier. *Hacking Exposed VoIP: Voice Over IP Security Secrets & Solutions (Hacking Exposed).* McGraw-Hill Osborne Media, 2006.

[9] D. Endler and M. Collier. Hacking voip exposed, 2006. BlackHat Conference 2006 USA.

[10] H. Hassan, J. Garcia, and C. Bockstal. Aggregate traffic models for voip applications. In *Digital Telecommunications, , 2006. ICDT '06. International Conference on*, volume 0, pages 70–70, Aug. 2006.

[11] IEEE. Ieee standard for software test documentation. *IEEE Std 829-1998*, Dec 1998.

[12] IEEE. Ieee std 1012 - 2004 ieee standard for software verification and validation. *IEEE Std 1012-2004. Revision of IEEE Std 1012-1998.*, 0(0):1 to 110, 1 2005.

[13] C. Kaner, J. Falk, and H. Q. Hguyen. *Testing Computer Software.* International Thompson Computer Press, London, UK, 1993.

[14] G. M. Kapfhammer. Software testing. In *The Computer ScienceHandbook.* CRC Press, 2004.

[15] A. D. Keromytis. A survey of voice over ip security research. In A. Prakash and I. Gupta, editors, *ICISS*, volume 5905 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009.

[16] A. D. Keromytis. Voice over ip: Risks, threats, and vulnerabilities. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*, pages 557–563. IEEE, 2009.

[17] M. Luo, T. Peng, and C. Leckie. Cpu-based dos attacks against sip servers. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, volume 0, pages 41 –48, april 2008.

[18] S. Montagna and M. Pignolo. Performance evaluation of load control techniques in sip signaling servers. In *Systems, 2008. ICONS 08. Third International Conference on*, pages 51 –56, april 2008.

[19] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Comput. Surv.*, 39(1):3, 2007.

[20] M. Z. Rafique, M. A. Akbar, and M. Farooq. Evaluating dos attacks against sip-based voip systems. In *GLOBECOM*, pages 1–6. IEEE, 2009.

[21] J. Rosenberg. Rfc5390: Requirements for management of overload in the session initiation protocol. http://www.ietf.org/rfc/rfc5390.txt.

[22] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, H. M., and E. Schooler. Rfc 3261: Sip - session initiation protocol. http://www.ietf.org/rfc/rfc3261.txt.

[23] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle. Sipstone - benchmarking sip server performance. http://www.sipstone.com, april 2002.

[24] D. Sisalem, J. Kuthan, and S. Ehlert. Denial of service attacks targeting a sip voip infrastructure: attack scenarios and prevention mechanisms. *Network, IEEE*, 20(5):26–31, Sept.-Oct. 2006.

[25] H. Srinivasan and K. Sarac. A sip security testing framework. In *CCNC'09: Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, pages 1056–1060, Piscataway, NJ, USA, 2009. IEEE Press.

[26] J.-S. Wu and P.-Y. Wang. The performance analysis of sip-t signaling system in carrier class voip network. In *Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on*, pages 39 – 44, march 2003.

Reliability and Relay Selection in Peer-to-Peer Communication Systems

Salman Abdul Baset and Henning Schulzrinne
Department of Computer Science
Columbia University, New York, NY, USA
{salman,hgs}@cs.columbia.edu

ABSTRACT

The presence of restrictive network address translators (NATs) and firewalls prevent nodes from directly exchanging packets and thereby pose a problem for peer-to-peer (p2p) communication systems. Skype, a popular p2p VoIP application, addresses this problem by using another Skype client (relay) with unrestricted connectivity to relay the signaling and media traffic between session endpoints. This distributed technique for addressing connectivity issues raises challenging questions about the reliability and latency of relayed calls, relay selection techniques, and the interference of relayed calls with the applications running on relays – a phenomena we refer to as user annoyance.

We devise a framework to analyze reliability in peer-to-peer communication systems and present a simple model to estimate the number of relays needed for maintaining the desired reliability for the media sessions. We then analyze two techniques for improving the reliability of relayed calls. We present a distributed relay selection technique that leverages a two level hierarchical p2p network to find a relay in $O(1)$ hop. We augment our distributed relay selection technique to find a relay that minimizes call latency and user annoyance. Our results indicate that for Skype node lifetimes, at least three relays are needed to achieve a 99.9% success rate for call duration of 60 mins (95^{th} percentile of Skype call durations).

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Reliability, availability, and serviceability]

General Terms

Design, Reliability, Measurement

Keywords

Reliability, P2P, VoIP, Relay

1. INTRODUCTION

Restrictive network address translators (NATs) and firewalls prevent hosts from directly exchanging packets. A recent survey of 1,787 NAT devices indicates that hosts behind approximately 30% of these devices cannot traverse the NATs using UDP or TCP [3] implying that hosts behind two different such devices are not likely to directly exchange packets without an intermediary. Moreover, corporations are increasingly deploying firewalls to protect their networks from malicious traffic that originates both outside and inside their networks. The restrictive NATs and firewalls pose a problem for IP communication systems because they prevent the user agents from directly exchanging signaling and media traffic.

In a client-server (c/s) communication system, the caller user agent discovers the current network address of a callee user agent through a managed server and exchanges signaling information with the callee user agent to establish a media session. The media traffic flows directly between the user agents. To address the connectivity constraints due to restrictive NATs and firewalls, c/s systems such as Vonage [8] use managed servers for relaying the media traffic between user agents with restrictive connectivity. In contrast, in a peer-to-peer (p2p) communication system, there are minimal or no servers. The user agents collaborate to discover the network address of the callee user agent and then directly exchange signaling and media traffic to establish a media session. When the user agents behind restrictive NATs and firewalls cannot directly exchange packets, they rely on user agents (or peers) with unrestricted connectivity for exchanging signaling and media traffic. Skype is an example of a peer-to-peer communication system that uses this technique [9]. Suh *et al.* [29] report hundreds of calls being relayed by a Skype relay.

The above characteristics of a p2p communication system pose unique challenges for a system designer. First, the lookup performance in p2p systems must at least be as effective as the lookup performance of client-server systems. Additionally, a media session may be prematurely terminated because a relay peer goes offline. This issue necessitates a formal analysis of the reliability of p2p communication systems and techniques to prevent dropped sessions. Moreover, since media sessions such as voice and video have a tight playout requirement, the network latency of a media session involving a relay peer should satisfy these tight requirements. Further, the relaying of a media session may interfere with other user applications and impair their performance. A system designer must either provide incentives for users to run relay peers or design techniques that minimize the interference of relayed session with other user applications.

In this paper, we present a framework to analyze the reliability of peer-to-peer communication systems (Section 3). We then devise a simple analytical model that predicts the smallest number of relays needed to achieve the desired reliability for relayed media sessions (Section 4.1) and evaluate it on exponential, pareto, and Skype node lifetimes. For a given node lifetime and call duration distribution, the model allows determining the minimum number of relays so that the percentage of successful relayed calls does not fall below a desired threshold (e.g., 99.9%). Such an analysis can help characterize the resources (relays) needed for improving the reliability of relayed calls. We then devise two techniques to prevent dropped sessions, selecting k relay peers at the beginning of a call with no-replacement and with-replacement and predict their reliability improvement using reliability theory in Section 4.2 and 4.3. In Section 4.4, we analyze the reliability improvement scheme used by Skype. Section 4.5 presents the experimental evaluation of the model and discussion.

In Section 5, we present a distributed technique to find a relay peer in $O(1)$ hop and compare the performance of this technique to a relay selection scheme that has global knowledge of all the relays in the p2p network. Instead of designing incentives for users to allow relaying of media sessions through their user agents, we aim to minimize the interference of relayed session with the user applications. To capture the impact of the relayed media sessions on the user applications, we introduce the notion of *user annoyance* (Section 5.2). We augment our distributed search technique to select a relay that minimizes delay, user annoyance, or both within a threshold. To the best of our knowledge, we are the first to address the reliability issues in p2p communication systems, and to devise techniques for finding a relay that optimizes the latency of a relayed call and user annoyance. Our analysis and results are also applicable to media translation and conferencing in p2p communication systems.

2. PROBLEM SETTING

We consider a peer-to-peer communication system that has N participating nodes. A node is a machine with CPU, memory and disk and is connected to the Internet through a dialup, DSL, cable, fiber, or a wireless connection. Typically, a human user is associated with each node or a machine and runs a peer-to-peer communication application(s) (also referred to as user agents) and other applications. The p2p applications use any peer-to-peer protocol to form a p2p network. There are two types of nodes in a peer-to-peer communication network, peers and free-riders. In the literature, they are also referred to as super nodes and ordinary nodes [9, 21] or peers and clients [12]. A peer fully participates in the p2p network, collaborates with other peers to discover the reachable network address of the callee user agent, and can relay one or more media sessions. A free-rider does not collaborate in the discovery of the callee user agent and does not relay any media sessions. However, this collaboration is not always purposefully avoided. The presence of restrictive NATs and firewalls may hinder the participation of a node in the overlay, thereby forcing it to act as a free-rider. The need for relaying media sessions between caller and callee user agents arises precisely due to this reason. For ease of exposition, we refer to the caller and callee user agent as caller and callee, relay peer as relay, and voice session as a call. Unless stated otherwise, we refer to the p2p communication application as a p2p application.

3. RELIABILITY OF A P2P COMMUNICATION SYSTEM

Availability is the classical metric for modeling the reliability of a communication system and is typically measured by the number of nines after a decimal point. For example, a "3 nines" (99.9%) reliability means that the system is down only 0.1% of the time. In a p2p communication system, availability implies the ability of the system to find the network address of the callee, and also to find a relay for establishing the relayed call. However, this notion does not fully capture the reliability of relayed calls because in addition to relay search failure, calls can also fail due to relay churn since there is no guarantee about the uptime of relays. Thus, a more accurate metric to capture the reliability of calls in a p2p communication is the number of successfully completed calls.

$$P_{succ} = P_{ss}F_{norelay} + P_{ss}\overline{F}_{norelay}P_{rs}P(R > D) \qquad (1)$$

Equation (1) formalizes the notion of reliability or percentage of successful calls in a p2p communication system. The term to the immediate left of plus sign is the probability of successfully finding the network address of the callee user agent, P_{ss}, times the proportion of calls that do not need a relay, $F_{norelay}$. The term to the immediate right of plus sign is the probability of successfully finding the relay, P_{rs}, times the proportion of calls that need a relay, $\overline{F}_{norelay}$, times the probability that the residual lifetime of a relay, R, is greater than the call duration distribution D. This equation indicates that the proportion of successful calls can be increased by enhancing the performance of lookup schemes using techniques similar to [24,27], by designing schemes that establish a media session between user agents in the presence of NATs and firewalls without requiring a relay [15], and by improving the success rate of distributed relay search and relay calls. We focus our attention on analyzing the reliability of relayed calls and relay search since other areas have seen related work [25].

4. MODELING RELIABILITY OF RELAYED CALLS

We present a simple model to calculate the minimum number of relays per call, k so that the success rate of relayed calls is above a desired reliability criteria such as 99.9% (Section 4.1), analyze two reliability improvement schemes, namely, *no-replacement* (Section 4.2) and *with-replacement* (Section 4.3), and present an evaluation of the model and reliability improvement schemes (Section 4.5). Our analysis assume that the nodes that need a relay to establish a call (ordinary nodes) can randomly select it from the set of all relays, that relays are plenty, and the system has reached stationarity. In Section 5.1, we discuss a distributed scheme to find a relay.

4.1 Number of Relays

Let X_i be a random variable (r.v) that denotes the lifetime of relay i, F_{X_i} be its CDF, and X_i be independent and identically distributed (i.i.d). Let R_i be a random variable that denotes the residual lifetime of relay i when it starts relaying the call and D denote the distribution of call duration. When a relay fails, the call it is relaying is immediately switched to a new relay j, having residual lifetime R_j. Since the new relay is selected immediately when the old relay fails, the residual lifetime of the relays used are also i.i.d. For simplicity, we assume that calls are not dropped during switch over to a new relay. Leonard *et al.* [20] note that if the system has reached stationarity, the CDF of residual lifetimes is given as:

$$F_R(x) = P(R_i < x) = \frac{1}{E[X_i]} \int_0^x (1 - F(z))dz \qquad (2)$$

We are interested in determining the minimum relays per call k, so that the number of successfully completed relayed calls is above a desired criteria such as 99.9%, i.e.,

$$\text{Desired reliability} \leq P(\sum_{i=1}^{k} R_i > D) \qquad (3)$$

LEMMA 1. *When X and D are exponentially distributed with parameters λ and ν, the r.h.s of* (3) *has a closed form solution:*

$$P(\sum_{i=1}^{k} R_i > D) = 1 - (\frac{\lambda}{\lambda + \nu})^k \qquad (4)$$

Proof:
For exponential distribution, (2) can be solved to obtain $F_R(x)$ and its probability distribution function (pdf) $f_R(x)$, which are $1 - e^{-\lambda x}$ and $\lambda e^{-\lambda x}$, respectively. Using conditioning:

$$P(D < \sum_{i=1}^{k} R_i) = \int_0^\infty F(D < m) \times f(\sum_{i=1}^{k} R_i = m)dm$$

$f(\sum_{i=1}^{k} R_i = m)$ is a k-fold convolution of exponential r.v's which have a gamma pdf.

$$= \int_0^\infty (1 - e^{\nu m}) \times \frac{\lambda e^{-\lambda m}(\lambda m)^{k-1}}{(k-1)!} dm$$

$$= \int_0^\infty \frac{\lambda e^{-\lambda m}(\lambda m)^{k-1}}{(k-1)!} - \frac{\lambda e^{-(\lambda+\nu)m}(\lambda m)^{k-1}}{(k-1)!} dm \qquad (5)$$

The left term of (5) is 1 since it is an integral of gamma pdf. Multiple and divide the right term by $(\lambda + \nu)^k$ and using $\Gamma(n) = \int_0^\infty e^{-x}x^{k-1}dx = (k-1)!$

$$= 1 - (\frac{\lambda}{\lambda + \nu})^k \qquad (6)$$

For arbitrary lifetime and call distribution, the r.h.s of (3) is difficult to solve as convolution of k i.i.d random variables is non-trivial. Instead, we use the following approximation which replaces the sum of k r.v's with their maximum.

LEMMA 2. *The sum of* k *i.i.d r.v's R_i being greater than another r.v D is greater than or equal to one minus the k^{th} exponentiation of the probability of R being less than D.*

$$P(\sum_{i=1}^{k} R_i > D) \geq 1 - P(R < D)^k \quad (R_i \text{ are i.i.d}) \quad (7)$$

Proof:

$$P(\sum_{i=1}^{k} R_i < D) \leq P(\max(R_1, \ldots, R_k) < D)$$

$$P(\max R_i < D) = P(R_1 < D, \ldots, R_k < D)$$
$$= P(R < D)^k \quad \text{since } R_i \text{ are i.i.d}$$

$$P(\sum_{i=1}^{k} R_i > D) \geq 1 - P(R < D)^k$$

Observe that if node lifetimes are exponentially distributed, the equality holds in (7) holds and (4) is obtained. For non-exponential node lifetimes, the k^{th} exponentiation decreases much faster than the sum and intuitively, the bound is loose for large values of k. However, the relative error of the bound depends on the lifetime and call duration distributions. Next, we examine the relative error of (7) for pareto distribution since the measurement studies of Skype node lifetimes suggest using heavy tailed distributions as an approximation [17] and pareto is the most natural choice for such an approximation.

4.1.1 Pareto node lifetimes

The CDF of pareto lifetimes is $F(x) = 1 - (\frac{x}{b})^{-a}$, where a is the shape parameter and b is the scale parameter. For our analysis, we use the shifted pareto distribution $F(x) = 1 - (1 + \frac{x}{b})^{-a}$ with mean $\frac{b}{a-1}$ [20], because without the shift, a node is guaranteed to be up for b units of time. Clearly, the mean of this distribution is only defined for $a > 1$ where as variance is only defined for $a > 2$ which prevents the calculation of an an exact analytical formula for sum of k pareto i.i.d r.v's. Zaliapin *et al.* [32] describe methods for approximating the upper quantile (0.98), lower quantile (0.02), and median of sum of k i.i.d r.v's. Their results indicate that although replacing the sum with the maximum can reasonably approximate the quantiles around median, such an approximation is poor for the lower and upper quantiles and for large values of k (e.g., > 10). The CDF of residuals of pareto lifetimes is $F(x) = 1 - (1 + \frac{x}{b})^{1-a}$ [20]. Although, the approximation results by Zaliapin can be extended to the sum of pareto residuals for arbitrary values of a, b, and k, such an effort is beyond the scope of this paper. Further, the utility of precise approximation may be limited due to the difficulty in estimating the pareto parameters. Also, real node lifetimes do not follow a strict pareto distribution and incorporate effects such as diurnal variations [17,19]. Therefore, to obtain a bound on the minimum number of relays to achieve desired reliability, we approximate the sum of k pareto residuals with their maximum, but note that in doing so, it is necessary to get an estimate of the relative error of such an approximation to determine its usefulness.

In Table 1, we show the simulated values of the sum of two and four pareto residual R_i being less than exponentially distributed call holding times D and the relative error of the approximation (maximum of R_i being less than D) with respect to the simulated values. The simulated results are an average over 10^7 runs. The parameters a and b were chosen so that the mean of the distribution was five and one hour, respectively. The choice of mean uptime of five hours approximately reflects the median of the observed Skype node lifetimes [17,19], where as mean node lifetime of one hour is for a relatively less stable system. The top two values in the fourth column are zero because sum of four R_i was

	a=2,b=5 (mean lifetime=5 hours)				a=3,b=2 (mean lifetime=1 hour)			
	k=2		k=4		k=2		k=4	
call duration	sim (%)	rel-e (%)	sim (%)	rel-e (%)	sim (%)	rel-e (%)	sim (%)	rel-e (%)
2.5	0.0074	8.5755	0	0.00	0.1544	0.2171	0.0003	21.3205
5	0.0251	3.6121	0	0.00	0.5517	0.2131	0.0027	6.9055
10	0.0961	1.7193	8e-5	20.0925	1.8179	0.1980	0.0319	3.8110
20	0.3553	1.3791	0.0011	14.7570	5.2869	0.1456	0.2772	0.2958
30	0.7171	0.4476	0.0053	1.9231	9.0853	0.0737	0.8292	0.2894
40	1.1567	0.4465	0.0137	1.7594	12.867	0.0233	1.6608	0.2589
50	1.6537	0.4349	0.0265	1.1979	16.464	0.0061	2.7106	0.0885
60	2.1895	0.1096	0.0482	0.8299	19.836	0.0303	3.9368	0.0585

Table 1: Simulated values of $P(\sum_{i=1}^{k=2} R_i < D)$ and $P(\sum_{i=1}^{k=4} R_i < D)$ for pareto lifetimes are shown in the 'sim' column. The values indicate the percentage of dropped relay calls in 10^7 runs. The relative error of the approximation $P(R < D)^{k=2}$ and $P(R < D)^{k=4}$ with respect to the simulated values is shown in the 'rel-e' column. Call duration is exponentially distributed.

never observed to be smaller than D (with mean of 2.5 and 5 minutes) in 10^7 runs. For relayed calls, these values are interpreted as observing no call failure in 10^7 runs. Observe that the relative error is low ($< 0.2\%$) when the value of the simulated sum of R_i r.v's being less than D is above 2% where as the relative error increases for simulated values smaller than 2%. This result is consistent with [32] which notes that using the maximum of k pareto r.v's instead of their sum is not a good approximation for lower quantiles (< 0.02). However, note that although the relative error increases as call holding times, D, decrease relative to node lifetimes and the number of summands k increase, we are only interested in the smallest value of k for which the call success rate is just above the desired reliability such as 99.9% and not an arbitrary large value of k. In general, the approximation can be applied to determine the smallest value of k that meets the desired reliability criteria, as long as the relative error remains low (e.g., $< 1\%$).

Next, we present two schemes for preventing the failure of relayed media sessions due to relay churn.

4.2 No-replacement Scheme

In the no-replacement scheme, k relays are selected at the beginning of the call with one relay acting as primary and $k-1$ acting as backup. If the primary relay fails, the call is switched to a backup relay. We assume that calls are not dropped during switch over. A call fails when all k relays fail. Let R_i be a random variable that denotes the residual lifetime of the relay i when it is drafted as a relay and D be a random variable that denotes call duration. We are interested in the probability that at least one of the relay, that were selected when call was established, is online before the call completes:

$$
\begin{aligned}
&P(\max(R_1, \ldots, R_k) > D) \\
&= 1 - \int_0^\infty P(R < z)^k P(D = z) dz \quad (R_i \text{ are i.i.d}) \quad (8)
\end{aligned}
$$

We solved (8) to determine the proportion of successful relay calls using two or three relays when node lifetimes are exponentially distributed, and the corresponding expressions are $1 - \frac{2\nu}{\lambda+\nu} + \frac{\nu}{2\lambda+\nu}$ and $(\frac{1}{2\lambda+\nu} - \frac{1}{3\lambda+\nu})\frac{6\lambda^2}{\lambda+\nu}$, respectively. For pareto node lifetimes, we numerically solved (8) to obtain the proportion of successful calls using two or three relays.

How many relays? A question can be asked, how many relays should be selected at the time of establishing the relayed calls in order to improve their reliability. As might be expected, the proportional increase in the reliability diminishes with selecting more relays at the start of the call. For example, when node lifetimes are exponentially distributed, the MTTF of a 2-relay, 3-relay, and 4-relay schemes are $\frac{3}{2\lambda}$, $\frac{11}{6\lambda}$, and $\frac{25}{12\lambda}$, respectively. The proportional increase in the MTTF is 50%, 22%, and 13%, respectively. Clearly, this is a case of diminishing returns. Further, maintaining numerous backup relays exclusively for every call when relays are not plenty is likely to result in a poor performance from the perspective of successful call establishment for relayed calls. Moreover, nodes in a media session also incur the overhead of sending keep-alive traffic to many relays.

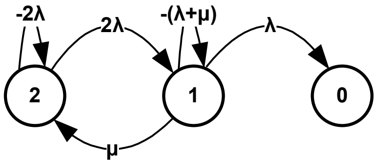


Figure 1: Markov chain for a 2-relay with-replacement scheme.

4.3 With-replacement scheme

This scheme is similar to the no-replacement scheme in that k relays are selected at the beginning of a call, and a call is switched to a backup relay if the primary relay fails. However, when a caller or callee detects that one of the k relays has failed, it launches a search to replace the failed relay. Suppose it takes μ time units to detect that a relay has failed and find a new relay. If node lifetime and search time are exponentially distributed, a Markov chain can be used to evaluate the reliability of this scheme [11]. For a single backup relay, the Markov chain is shown in Figure 1. In the reliability literature, this scheme is referred to as 1-out-of-2 active redundancy with constant failure rate λ and constant repair rate μ [11]. This chain can be solved to obtain MTTF, i.e., the time it spends in states (2) and (1), when two and one relays are operational. The failure rate is the reciprocal of MTTF, i.e.,

$$\frac{1}{\lambda_{WR}} = MTTF = \frac{3\lambda + \mu}{2\lambda^2} \tag{9}$$

The subscript WR denotes with-replacement. For $\lambda << \mu$, this scheme approximately behaves like a one relay scheme with constant failure rate λ_{WR} (Birolini [11, page 190]). Let R_{WR} be a random variable that denotes the reliability of this scheme. Since its failure rate is constant, its CDF is $R_{WR}(t) = e^{-\lambda_{WR}(t)}$. When call duration is exponentially distributed with parameter ν, the probability that a call completes before the two relays fail and a search for the replacement relay also fails is:

$$P(R_{WR} > D) = \frac{\nu}{\nu + \lambda_{WR}} \tag{10}$$

When the node failure rates are not constant, either non-homogeneous poisson processes may be used to model the reliability of this scheme or node lifetime can be split into periods where failure rate is constant. However, the difficulty in using such analysis lies in the fact that for heavy tailed distributions, the shape parameter a is often not accurately known. Therefore, we leave such analysis for future work.

4.4 Reliability of Relayed Calls in Skype

We performed experiments to determine if the Skype application employs a no-replacement or a with-replacement scheme. We blocked direct traffic between two machines running in our lab using NetPeeker [4] and then ran Skype applications on them and established a call. Since the traffic was blocked between the machines, the Skype applications were forced to use a relay to exchange signaling and media traffic. Using NetPeeker [4], we blocked the media traffic between caller machine and the relay, which is similar to emulating a relay failure. Within 2-4 seconds, the Skype applications chose a new media relay. We then immediately blocked traffic between this new relay and the caller Skype application which resulted into the call getting disconnected. The experiment shows that when a call is established that requires a relay, the Skype application chooses a backup relay at the start of the call. When both relays fail simultaneously, the call is disconnected.

To determine if a Skype application searches for a new relay when the primary relay fails and the call is shifted to the backup relay, we gradually increased the time between primary and backup relay failure from 30 s to two minutes. Our experiments indicate, that a Skype application waits for more than a minute before searching for a new relay. Thus, it employs a 'periodic-recovery' scheme for replacing a failed relay instead of a 'reactive-recovery' scheme. We periodically failed the primary relay every 90 s for a call lasting 15 minutes and found that the Skype application was able to find a backup relay and the call did not get disconnected.

All the experiments were performed during the first week of December 2009 and more than 70 calls were established over a period of seven days.

4.5 Evaluation and Discussion

We evaluate the analytical model for the number of relays, and reliability improving techniques using simulations. We wrote an event driven simulator in which nodes form an overlay network using Chord. We use a relay selector which randomly selects a relay from the pool of online relays having sufficient network capacity. The inter-arrival time between requests for relayed calls is exponentially distributed and its mean is adjusted over the course of the simulation so that the cumulative network load of relayed calls does not exceed a target aggregate network utilization of the peers. In the results presented in this section, the aggregate uplink network utilization of all the peers never exceeded 40%. Thus, in our simulations, the relayed calls only fail due to relay failure and not due to the scarcity of relays. We run the simulation for 10 days of simulated time and repeat the experiments until 10^7 call attempts have been made. The warm up period is excluded from the reported results.

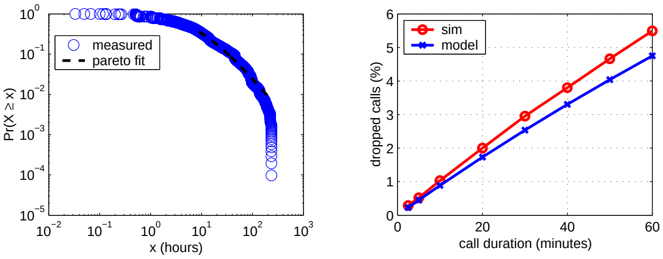


Figure 2: (Left) CCDF of the node lifetimes and the pareto fit for Skype data set (right) percentage of dropped calls through simulations on the Skype data set and using a pareto model when only one relay is used.

We use three node lifetime data sets. The first two data sets contain synthetically generated exponential and pareto node uptime and downtime with a mean of 300 minutes. The pareto parameters a and b were chosen as 2 and 5, respectively. The third data set contains the uptime and downtime of 4,000 Skype super nodes measured for 25 days [17]. The uptime of Skype nodes was measured by sending a specially crafted Skype message to these nodes every 30 minutes. We randomly selected 1,740 nodes from this data set of 4,000 nodes because this is the maximum number of end nodes for which all pair ping latency data is available [18]. We use this data for designing a distributed relay search mechanism that minimizes latency of the relayed calls in Section 5.3.

All 1,740 nodes can potentially provide the relay service. The median and mean uptime of these 1,740 nodes was 256 and 711 minutes, respectively. The pareto parameters, a and b, computed using the method of maximum likelihood and Kolmogorov-Smirnov statistic, are 1.4916 and 8.9833, respectively. Figure 2 (left) shows the CCDF of Skype node lifetimes and the pareto fit indicated by a dashed straight line. Towards the end of the tail, the measured lifetimes exhibit a knee of the curve. This happens because the node lifetimes do not strictly exhibit a pareto behavior and the measurement is stopped after T time units. For the Skype data set, Figure 2 (right) shows the percentage of dropped calls when a call is assigned to one relay through simulations and those predicted by the model $P(R < D)$. The relative error with respect to simulations was less than 15%. Wang [31] suggested that there is an inherent inaccuracy in computing the exact parameters of the node lifetime distribution when they are sampled every T time units. We note that such a bias depends on the ratio of the mean node

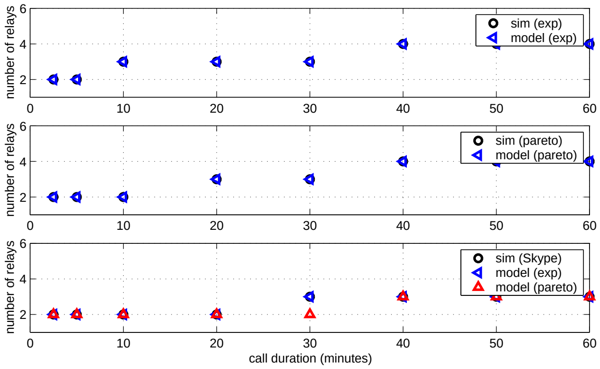


Figure 3: Number of relays for exponential (top) pareto (middle) and Skype (bottom) node lifetime data set to maintain call success rate of 99.9%. The mean node lifetime for exponential and pareto distributions was 300 minutes.

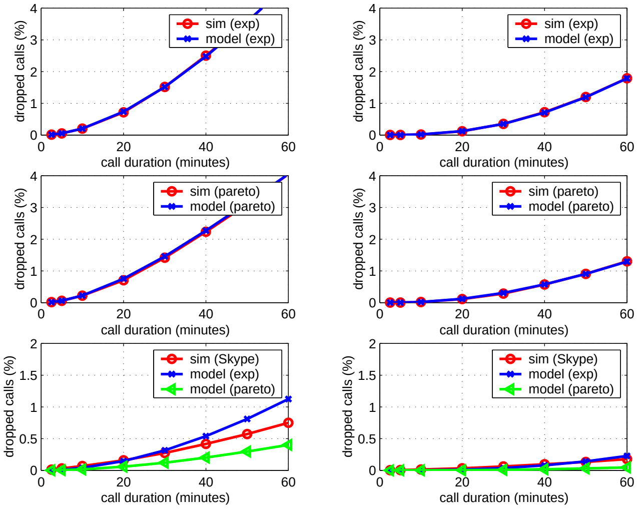


Figure 4: Proportional of failed calls using simulations and model for exponential (top), pareto (middle), and Skype (bottom) node lifetimes. The figures on the left and right are for a 2-relay and 3-relay no-replacement scheme, respectively.

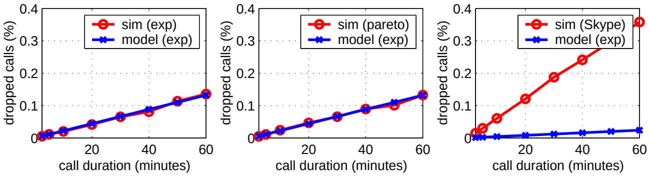


Figure 5: Proportion of failed calls using simulations and Markov model for 2-relay with-replacement scheme for exponential (left), pareto (middle), and Skype (right) node lifetimes.

lifetime and the sampling interval: the higher the ratio, the lesser the inaccuracy and vice versa. Nevertheless, we note that when the real lifetime data is used for churn simulations, such a bias will always be present.

A key consideration is to realistically set the upload and download bandwidth of a relay peer since it cannot relay an arbitrary number of calls. Dischinger *et al.* [14] have measured the upload and download bandwidth for a range of broadband hosts and we set the relay bandwidths according to their reported distribution. We assume that a relay call needs an uplink and downlink bandwidth of 128 kb/s (using the G.711 codec). Modern codecs such as SILK [7] which has a bit-rate between 4-40 kb/s can bring down the required bandwidth at a relay to 8-80. To simulate the effect of network traffic belonging to other applications, we randomly set the uplink network utilization of a node between 10-30% of its uplink capacity at the start of the simulation. Depending on its spare capacity, a relay peer can relay more than one call.

Figure 3 shows the number of relays for exponential, pareto, and Skype node lifetimes for a range of exponentially distributed call holding times. Guha [17] showed that 95% of Skype relayed calls last less than an hour. The approximation from (7) is used to calculate the number of relays when pareto distribution is used to model node lifetimes. For pareto node lifetimes (second row in the figure) and call duration of 60 minutes, the relative error of the approximation was less than 1%. The results from the simulation show that for the Skype data set and for call durations of 60 minutes or less, three relays are sufficient to achieve a call success rate of 99.9%. Observe that modeling the Skype node lifetimes as exponential and pareto resulted in a minimum relay prediction of three relays which matches the simulations. For call duration of 30 minutes, the pareto model under predicts the number of relays. However, this is expected as Skype node lifetimes do not exactly follow the pareto model (Figure 2). Also, for the results shown, note that although only three or four relays or less are needed to achieve call drop rate of 0.1% or less for call duration of 60 minutes, the number can be higher when node lifetimes are smaller. As an example, when the node lifetimes are exponential with a mean of one hour, at least ten relays per call are needed to achieve a success rate of 99.9% for mean call duration of 60 minutes.

Figure 4 shows the reliability of a 2-relay and 3-relay no-replacement scheme for exponential, pareto, and Skype node lifetime data sets computed using (8). As expected, there is a good match between analytically computed (using (8)) and simulated call success rates for exponential and pareto node lifetimes. For the Skype data set, the simulations show that a 2-relay scheme achieves a 99.9% success rate for call durations of 10 minutes or less where as for call duration of 60 minutes, the success rate is 99.25%. For 2-relay no-replacement scheme, using exponential and pareto node lifetimes to model Skype node lifetimes results in over predicting and under predicting the number of dropped calls by approximately a factor of two, respectively.

Figure 5 shows the reliability of a 2-relay with-replacement scheme for exponential, pareto, and Skype node lifetime data sets. The time to detect if a relay has failed and consequently to search a new relay is exponentially distributed with a mean of 60 s. As expected, the Markov model accurately predicts the call drop rate when node lifetimes are exponential. The results also indicate that the Markov model may be a reasonable approximation for pareto node lifetimes. For Skype data set and for call duration of 60 minutes, this scheme achieves a call success rate of 99.65%, an improvement of 0.3% over a 2-relay no-replacement scheme. The improvement is small because node lifetimes have a large mean (711 minutes). When node lifetimes have a small mean, it may be necessary to incorporate a with-replacement scheme to avoid dropped calls. Since Skype employs a 2-relay with replacement scheme with a relay search time of approximately 60 s, the results from our simulations indicate that the drop rate of relayed calls is likely to be small. However, Skype's relay mechanism is not completely random and is biased towards low latency and high bandwidth relays. Such a bias may result in higher drop rates for relayed calls [16]. Nevertheless, an implication of the results is that for Skype node lifetimes, simple schemes for reliability improvement such as two relay no-replacement and with-replacement give reasonable reliability performance thereby obviating the need for a sophisticated reliability improvement scheme.

4.5.1 *Practical implications of these schemes*

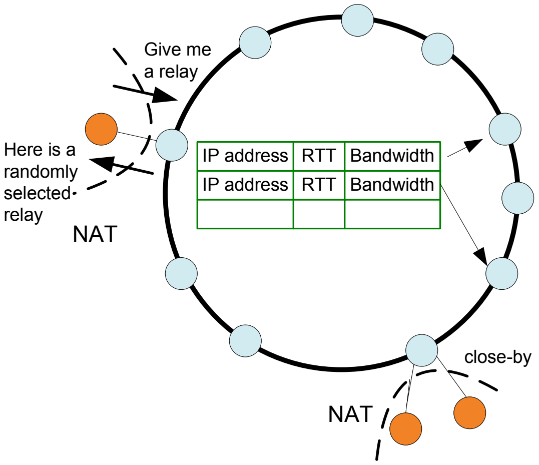
In a k-relay no-replacement scheme, both caller and callee exchange information about k relays at the time of call establishment. After a call has been established, they must periodically check the liveness of all k relays. The liveness period should be adjusted so that when the primary relay fails, there is a high likelihood that the new relay to be incorporated is alive. However, the reliability returns of maintaining a large number of backup relays at the start of the call are diminishing, especially under high churn. For this reason, a with-replacement scheme is attractive. Such a scheme can potentially start with 2 or 3 relays, and find a replacement for a failed relay. However, the caller and callee must exchange information about the new relay in subsequent signaling messages. For a no-replacement scheme, no such exchange is required.

4.5.2 *Other reasons for call failure*

Relay failure is not the only reason why relayed calls may fail. Such calls can also fail during call switching. Also, since nodes may use silence suppression, it may take more time to correctly distinguish between silence periods and a failed relay because the frequency of heart-beat messages is likely to be lower than real-time voice or video packets. If a search for a relay is launched at the time when all relays fail, the caller and callee can perceive a silence gap in the conversation. If the duration of the perceived gap is long, the call participants may simply terminate the call.

5. RELAY SELECTION

In this section, we devise distributed techniques to find a relay that address several practical issues. The first issue is that the distributed relay search must find a relay in a timely manner to minimize the call establishment time and to quickly recover from relay churn. Also, the relaying of media session can interfere with the user applications and impair their performance. It is important to select relays in a way that minimize this interference. Besides minimizing interference, latency and increasing reliability are key objectives for relayed calls. Addressing all these factors is a multi-objective optimization problem which is NP-hard.



IP address	RTT	Bandwidth
IP address	RTT	Bandwidth

Figure 6: Two-tiered overlay implementing a *local-random* scheme for relay selection.

In Section 5.1, we devise a distributed relay selection technique that can find a relay in $O(1)$ hops and compare its performance to a scheme that randomly selects a relay from the global pool of all relays. Section 5.2 introduces the notion of user annoyance. In Section 5.3, we augment the distributed relay selection scheme to devise heuristics for finding a relay that, for a relayed call, minimizes user annoyance or latency or both, and evaluate their performance.

5.1 Distributed Relay Selection

We devise a relay selection scheme where a node requesting a relay can find a relay in $O(1)$ hop. As mentioned earlier, quickly finding a relay is necessary to reduce call establishment time and to recover from relay churn. The key idea to accomplish this goal is to construct a two tier peer-to-peer network. All peers in the top tier provide routing services and can also potentially provide relay services. The peers form the top tier network using any structured or unstructured p2p protocols. Each peer maintains a data structure called a routing table to maintain connectivity with other peers in the overlay. Each entry in this table contains the network address and round-trip time of a reachable peer in the overlay. As part of keep-alive messages to check the liveness of entries in its routing table, a peer also exchanges information with its routing table entries on how many relay calls they can support, their uptime and the time for last user keyboard or mouse activity.

The nodes in the lower tier are connected to peer(s) in the top tier that are close by in terms of network latency and may need a relay peer for establishing a media session. A node in need of a relay sends a request to its connected peer which consults its routing table and returns to the requesting node a set of available relays. If none of the peers in the routing table can fulfill the relay request, the peer forwards the request to a randomly selected peer in its routing table, which in turn consults its routing table for available relay peers. The number of forwarding hops is bounded by a constant such as four. As an example, if on average 30% of the nodes in a peer's routing table are busy routing a call, then the probability of not finding an available relay after traversing four randomly selected hops is less than one

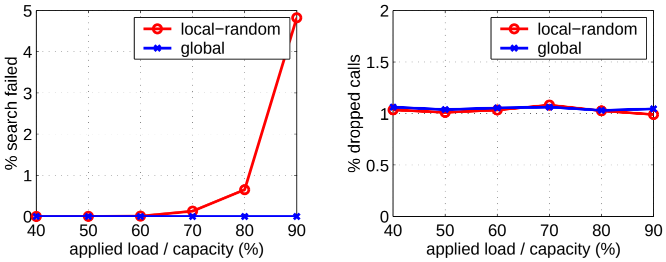


Figure 7: Performance of local-random scheme vs. global scheme as a function of system load (left graph). Percentage of dropped calls when one relays fails (right graph).

percent (0.81%).

If the number of relay requests are low and uniformly distributed across all peers, this scheme is likely to find a relay in $O(1)$ hops. We refer to this scheme as *local-random* scheme because it selects a relay by leveraging the local overlay view of a peer. This scheme is in contrast to a global random scheme, which has knowledge of all relays in the system and randomly picks a relay from this global pool. Figure 6 shows an illustration of this scheme.

We evaluate the performance of this scheme through simulations and use Chord [28] as the overlay protocol. Each Chord peer maintains a randomized routing table [16] instead of a deterministic table, i.e., for the interval $[ID + 2^i, ID + 2^{i+1})$, it picks up any node with an ID in this interval. This is so because Godfrey *et al.* [16] showed that the randomized scheme for populating routing tables has a better performance against churn. The Chord network is run on 1,740 nodes that follow the Skype node lifetime distribution as discussed in Section 4.5. The requests for relaying a media session arrive at any relay and are uniformly distributed across relays. Intuitively, the local-random scheme may have poor performance when relay requests are concentrated on a few peers. However, this issue is easily addressed if a peer unable to fulfill the relay request forwards it to a randomly selected peer in its routing table.

The metric for evaluating the performance of this scheme is its ability to find a relay compared to a scheme with global knowledge of all relays for an increasing number of relay requests. The inability to find a relay impacts the success rate of relayed calls (equation (1)). The relay search is likely to fail when the number of relay requests is close to or exceeds the network capacity of the peers. If the percentage of relay peers that are relaying calls is low, then local-random scheme is likely to find a relay. However, this may not be the case when the number of relay requests is close to the capacity of the system. Figure 7 plots the percentage of calls that fail to find a single relay. For the results shown, the local-random scheme did not forward the relay request to any peers. The x-axis is the ratio of the applied load to the total relay capacity of all relays. The figure shows that the performance of local-random scheme is poor when there are few relays that can relay the calls. However, it gives comparable performance in terms of percentage of dropped calls due to relay failure even under heavy relay request load.

5.2 User Annoyance

A key difference between p2p file-sharing and communication systems is in their approach to free-riders. The tit-for-tat mechanism in BitTorrent-like filesharing mechanisms aims to minimize the impact of free-riders that are not willing to share files or are behind restrictive NAT and firewalls. Such nodes can only download files at a reduced rate [2]. Reducing rate may not be an option in p2p communication networks because it can affect the quality of audio, video, and conference calls. Thus, in contrast to a p2p file-sharing system, a p2p communication system must provide acceptable service to nodes behind restrictive NATs and firewalls. This key requirement means that nodes with unrestricted connectivity must relay calls for nodes with restrictive network connectivity and the relayed calls may interfere with user applications running on these altruistic peers. We refer to such interference as 'user annoyance'.

We focus on characterizing the user annoyance and augmenting the relay selection scheme to minimize user annoyance. User annoyance for relayed calls can also be reduced by providing incentives. However, in a system where proportion of relayed calls is much smaller than the number of available relays, it may be possible to avoid peers where a relay call is likely to cause a high interference with the user applications, and thus bypassing the issue of providing incentives.

The question is how to measure user annoyance. Since relay jobs are network centric and since it is difficult to accurately estimate the perceivable impact of the relay jobs on user applications, we use the spare network capacity to estimate user annoyance. This simplistic measure may not accurately measure user annoyance; however, it is more practical than the other approaches. The higher the spare network capacity, the smaller the likelihood of annoyance of a user whose machine is used as a relay. A peer can periodically perform its uplink and downlink capacity measurements (say every 30 minutes) and by determining the current network usage, gauge its spare network capacity which it can then advertise to peers in its routing table. We use this technique in our PlanetLab implementation (Section 5.4).

5.2.1 *Estimating Spare Network Capacity*

Measuring user annoyance requires estimating of the capacity of the network link. Unlike CPU, memory, and disk, it is non-trivial to estimate the network capacity. To an extent, this depends on the type of network link. On point-to-point dialup connections, the maximum link speed is typically determined by the speed of the modem. As DSL and cable Internet penetrates homes and the use of WiFi routers at home becomes common, a device no longer directly connects to the ISP in a way similar to dialup; rather, a device connects to a WiFi router which connects to the DSL or cable modem, which in turn is connected to the ISP. Using the link speed of the connected WiFi link will highly overestimate the machine-to-ISP link capacity.

We suggest three approaches for determining the machine-to-ISP link capacity in the presence of intermediate devices such as WiFi routers, and cable or DSL modems. The first approach uses the fact that link capacity is agreed upon between ISP and customer when the latter purchases a broadband plan. The idea is to design protocols which allows ISP to pass this link capacity to the cable or DSL modem which in turn passes this information to downstream devices such as WiFi routers or laptops. This idea can be implemented as a DHCP option, for example. The problem with this

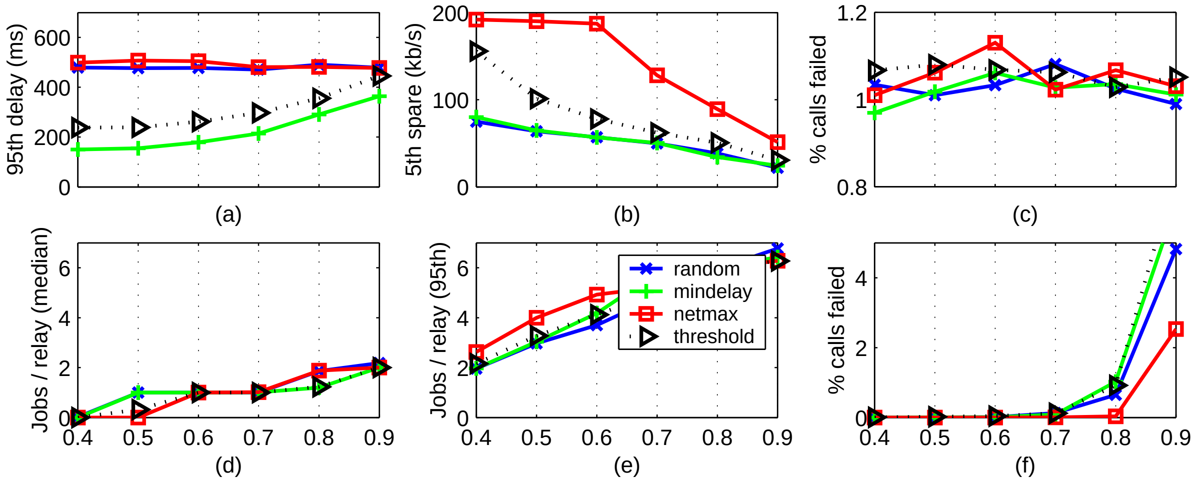


Figure 8: The x-axis represents the ratio of bandwidth consumption of total number of calls in the system to the total network capacity of all nodes. (a) 95th delay (ms) of completed calls (b) 5th percentile of spare network capacity (c) percentage of failed calls due to relay churn (d)(e) median and 95th percentile of number of jobs per relay (f) percentage of calls that fail to find a relay.

approach is that ISPs typically perform statistical multiplexing on multiple flows, and the instantaneous capacity of the link may be less than the purchased capacity. Also, this technique requires changing the already deployed cable/DSL modems and WiFi routers, which is a non-trivial task. Nevertheless, it is a solution that does not require p2p applications to perform any network capacity measurements. In the second approach, a p2p application can perform measurements to estimate the link capacity by sending a train of packets to other peers in the p2p network using tools such as LinkWidth [13] or Pathchar [5]. Third, an operating system or the p2p application can keep track of the maximum data rate seen on the link within a recent time window and use it as an estimate of link capacity. However, this approach heavily depends on the network usage of the machine. We use the second approach in our PlanetLab implementation.

5.3 Heuristics

Besides minimizing user annoyance, it is necessary to minimize the delay of a relayed call and increase its reliability. In essence, this is a multi-objective optimization problem. We devise heuristics to optimize these metrics and evaluate their performance.

In Section 5.1, we constructed a two tier overlay network and peers in the top tier maintain information about the round-trip time, spare network capacity, and uptime of the nodes in their routing table. Peers can periodically exchange this information, perhaps as part of keep-alive messages. A node searching for a relay then sends a request to its connected peer which applies the heuristics and returns a set of candidate nodes.

Below, we discuss the heuristics for selecting a relay peer from a candidate set returned by the local-random scheme.

- *Random:* Select a random node.
- *NetMax:* Select a node with the maximum spare network capacity.
- *MinDelay:* Select a node that has the smallest RTT.
- *Threshold:* Select a node that does not add more than 200 ms of delay on top of the direct network latency between caller and callee, and has maximum spare bandwidth. If no candidate meets the criteria, randomly select any.

Figure 8 shows results for these heuristics. The results were obtained through simulations on a 1,740 Chord network, with node lifetimes taken from the Skype data set as described in Section 4.5. We assume that the network latency between the clients and their connected peers is very small (close to zero). This assumption is reasonable because clients will likely connect to minimum latency peers to use overlay services. The heuristics are evaluated according to several metrics. The first metric is the 95th percentile of the total delay of a relayed call minus the direct latency between session peers. The second metric is the median and 95th percentile of the number of jobs per relay. The third metric is the 5th percentile of absolute spare capacity on relay nodes. The fourth metric is the percentage of calls that fail due to relay failure. The last metric is the percentage of calls that cannot find a relay.

The results shows that *MinDelay* heuristic gives the best delay performance (Figure 8(a)). *NetMax* heuristic ensures that relays with large spare network capacity are preferred over relays with small spare capacity and achieves the best performance for user annoyance. However, this has a consequence that more calls can be assigned to high capacity nodes, making these calls more vulnerable to relay failure (Figure 8(c)). The *Threshold* approach gives the best performance in terms of minimizing latency and user annoyance. The *Threshold* scheme has a slightly high call drop rate due to failed relays but this can be improved by biasing relay selection towards idle nodes, e.g., machines with no keyboard or mouse activity within a time period. All heuristics have similar performance in terms of their ability to find a relay under increasing load.

As mentioned in Section 5.2, spare network capacity is a simplistic measure to estimate user annoyance. In addition to spare network capacity, machine idle time is a useful measure for relay selection. The idea is to select a relay with spare capacity that has been idle for sometime. The use of idle time as a relay selection metric is motivated by SETI@home project [6]. SETI@home runs compute jobs as a screen saver on idle machines that are distributed around the world. Using this approach in a p2p communication network, peers participating in the top-level hierarchy inform peers in their routing table how long they have been idle and whether they are in the screen saver mode. A node in need of a relay then selects a peer that meets the delay constraint, has been idle, and has the maximum spare capacity.

Figure 7 showed that search for relays start to fail when the requests for relay calls are close to or exceed the total network capacity of the system. This is unacceptable for an overlay provider like Skype. The only solution for the overlay provider is to provision the p2p applications with centralized media relay servers. When nodes establishing a media session fail to find a relay peer, they send a request to the media relay server to relay the media session. Such a hybrid solution is necessary for a commercial p2p VoIP provider, if it needs to guarantee call establishment when there are not enough relays in the system.

5.4 PlanetLab Deployment

To examine the feasibility of relay selection schemes, we have implemented the *Random* and *Threshold* scheme in our OpenVoIP [10] system. OpenVoIP is a two-level hierarchical overlay network deployed on PlanetLab that uses the Kademlia DHT [26]. We have successfully scaled the top-level network to 1,000 peers that run on 500 PlanetLab machines. Each peer in the top-level network fully participates in the overlay and can act as a relay peer using TURN protocol [22]. Further, each peer periodically performs uplink and downlink TCP throughput measurements and shares this information with its routing table nodes. Using TCP throughput provides a conservative estimate of the link capacity than tools such as LinkWidth [13] or Pathchar [5]. In addition to sharing its uplink and downlink capacity measurements, a peer also shares its uptime with its routing table nodes. We have integrated p2p functionality with an open source SIP phone. This P2PSIP phone fully participates in the overlay if it is not behind a NAT or a firewall. Otherwise, it participates as a client. When two P2PSIP phones behind a restrictive NAT cannot establish a media session directly, they use a peer in the top-level hierarchy to relay the media session.

We have implemented the *Random* and *Threshold* scheme for relay selection. Our implementation of the *Threshold* scheme uses delay and spare network capacity metric. We do not use a SETI@home like technique for determining whether a machine is idle as the PlanetLab machines are not user desktop machines. The results for the *Threshold* scheme indicate that relay selection is biased towards nodes with maximum spare network capacity and low latency. We note that these relayed calls are real voice calls between two SIP user agents and are not emulated.

6. RELATED WORK

There has been extensive research on constructing proximity aware DHTs [25] and to minimize the impact of churn on DHT routing [16]. Ren *et al.* [23] showed through measurements that many relay peer selections in Skype are sub optimal, waiting time to select a peer can be quite long, and there are a large number of unnecessary probes. They designed an autonomous system aware p2p protocol (ASAP), which considers autonomous systems into peer relay selection. Their approach suffers from three limitations. First, when using DHTs, the network address of all relay peers within the same AS can get stored on a single node, creating a single point of failure. Second, their techniques do not incorporate interference of a relay session with the user applications. This is critical because users will not altruistically run a p2p application if it actively interferes with their applications. Finally, they provide no guidance on how many relay peers are needed to achieve desired reliability. Leonard *et al.* [20] analyze node connectivity in DHTs for exponential and pareto residual lifetimes. However, our focus is on charaterizing the reliability of relayed calls. Godfrey *et al.* [16] analyzed the impact of churn on the DHT routing performance and suggested techniques to minimize such impact. Our relay selection techniques uses their random selection approach. However, it is imperative to explicitly devise schemes to prevent dropped calls. Tan *et al.* [30] present analysis to improve the reliability of DHT-based multicast by improving its delivery ratio. Delivery ratio is not an appropriate metric to for analyzing reliability in peer-to-peer communication systems.

Connectivity issues due to NAT and firewalls also arise in p2p file sharing networks such as Kazaa [21] and BitTorrent [1]. BitTorrent allows nodes behind restrictive NAT and firewalls to download file chunks, albeit at a lower rate. To improve the download rate, BitTorrent FAQ recommends users to configure the ‘port forwarding’ feature of NATs [2]. Lowering rate is not an option in p2p communication networks because it can impact the quality of a call. Further, a user of the p2p communication may find it difficult to configure the NAT device and may abandon the p2p application in favor of a configuration-less communication application.

7. CONCLUSION

We have formalized the notion of reliability in peer-to-peer communication systems and designed a simple analytical model that predicts the reliability of relayed calls as a function of node lifetime and call duration distributions. Our analysis shows that for Skype node lifetimes and for call durations of 60 minutes or less, at least 2-3 relays are needed to achieve a 99.9% call success rate. We have presented two techniques for relay selection, namely, no-replacement and with-replacement, and used reliability theory to analyze them. We have observed that Skype follows a 2-relay with-replacement scheme, and it uses periodic recovery to replace a failed relay, and the search period is more than a minute. Our results indicate that exponential distribution, despite its limitations, is useful in analyzing the reliability of relayed calls.

We also introduced the notion of user annoyance which measures the interference of a p2p communication application relaying a call with other applications running on a machine. We have devised a distributed technique to find a relay in $O(1)$ hop. We augment this technique to find a relay that minimizes the latency of a relayed call and user annoyance. Finally, we have explored the feasibility of our relay selection schemes on a 1,000 node peer-to-peer communication system deployed on PlanetLab. In the future, we will extend our reliability analysis to p2p audio and video conferencing.

8. REFERENCES

[1] BitTorrent [accessed June 2010]. `http://www.bittorrent.com/`.

[2] BitTorrent FAQ [accessed June 2010]. `http://dessent.net/btfaq/\#ports`.

[3] NAT tester [accessed June, 2010]. `http://nattest.net.in.tum.de/`.

[4] NetPeeker [accessed June 2010]. `http://www.net-peeker.com/`.

[5] Pathchar [accessed June 2010]. `http://www.caida.org/tools/utilities/others/pathchar/`.

[6] SETI@home [accessed June 2010]. `http://setiathome.ssl.berkeley.edu/`.

[7] Skype Silk codec [accessed June 2010]. `https://developer.skype.com/silk/`.

[8] Vonage [accessed June 2010]. `http://www.vonage.com/`.

[9] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[10] S. A. Baset and H. Schulzrinne. OpenVoIP: An Open Peer-to-Peer VoIP and IM System. In *Proc. of SIGCOMM (demo)*, Seattle, WA, USA, September 2008.

[11] A. Birolini. *Reliability Engineering: Theory and Practice*. Springer-Verlag, 2004.

[12] D. Bryan, P. Matthews, E. Shim, D. Willis, and S. Dawkings. Concepts and Terminology for Peer-to-Peer SIP. Internet draft (work-in-progress), July 2008.

[13] S. Chakravarty, A. Stavrou, and A. Keromytis. LinkWidth: A Method to Measure Link Capacity and Available Bandwidth using Single-End Probes. Technical Report (cucs-002-08), Department of Computer Science, Columbia University, January 2008.

[14] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proc. of IMC*, San Diego, California, USA, 2007.

[15] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-Peer Communication Across Network Address Translators. In *Proc. of USENIX Tech. Conf.*, Anaheim, CA, USA, 2005.

[16] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing Churn in Distributed Systems. In *Proc. of SIGCOMM*, Pisa, Italy, 2006.

[17] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proc. of IPTPS*, February 2006.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency Between Arbitrary Internet End Hosts. *SIGCOMM Comput. Commun. Rev.*, 32(3):11–11, 2002.

[19] W. Kho, S. A. Baset, and H. Schulzrinne. Skype Relay Calls: Measurements and Experiments. In *Proc. of IEEE Global Internet Symposium*, Phoenix, AZ, USA, April 2008.

[20] D. Leonard, V. Rai, and D. Loguinov. On Lifetime-based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks. In *Proc. of SIGMETRICS*, Banf, Alberta, Canada, June 2005.

[21] J. Liang, R. Kumar, and K. Ross. Understanding Kazaa, 2004.

[22] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN). Internet draft (work-in-progress), April 2010.

[23] S. Ren, L. Guo, and X. Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. In *Proc. of ICDCS*, Lisbon, Portugal, 2006.

[24] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc. of USENIX Tech. Conf.*, Anaheim, CA, USA, 2004.

[25] S. C. Rhea. *OpenDHT: A Public DHT Service*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2005.

[26] J. Risson and T. Moors. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. RFC 4981, September 2007.

[27] J. Rosenberg. Interactive Connectivity Establishment (ICE). RFC 5245, April 2010.

[28] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[29] K. Suh, D. R. Figuieredo, J. Kurose, and D. Towsley. Characterizing and Detecting Relayed Traffic: A Case Study using Skype. In *Proc. of IEEE INFOCOM*, Barcelona, Spain, April 2006.

[30] G. Tan and S. A. Jarvis. Stochastic Analysis and Improvement of the Reliability of DHT-Based Multicast. In *Proc. of IEEE INFOCOM*, Anchorage, Alaska, May 2007.

[31] X. Wang, Z. Yao, and D. Loguinov. Residual-Based Estimation of Peer and Link Lifetimes in P2P Networks. *IEEE/ACM Transactions on Networking*, 17(3):726–739, 2009.

[32] I. V. Zaliapin, Y. Y. Kagan, and F. P. Schoenberg. Approximating the Distribution of Pareto Sums. *Pure and Applied Geophysics*, May 2005.

A Virtual and Distributed Control Layer with Proximity Awareness for Group Conferencing in P2PSIP [*]

Alexander Knauf[1] Gabriel Hege[1] Thomas C. Schmidt[1] Matthias Wählisch[2]
[1]HAW Hamburg, Dept. Informatik, Berliner Tor 7, D–20099 Hamburg, Germany
[2]Freie Universität Berlin, Inst. für Informatik, Takustr. 9, D–14195 Berlin, Germany
alexander.knauf@haw-hamburg.de hege@fhtw-berlin.de {t.schmidt,waehlisch}@ieee.org

ABSTRACT

There is an increasing demand to access voice or video group conferences without the burden of a dedicated infrastructure, but at any place and in an ad hoc fashion. Corresponding solutions require a lightweight, fully distributed cooperation among parties that share and manage the conference in an efficient, self-adaptive way. The technology framework of P2PSIP can be seen as a promising starting point to meet these objectives. In this paper, we make several contributions towards such a distributed, virtualized control layer based on P2PSIP that seamlessly scales and adapts to the user needs. We propose a P2P-signaling protocol scheme for a distributed conference control with SIP, that splits the semantic of Identifier and Locator of a SIP conference URI in a standard-compliant manner. This protocol scheme serves as further basis for a virtualization in RELOAD. We further design and evaluate a self-organizing communication layer that provides load sharing and churn resilience with proximity-awareness. Finally, we address key aspects of security and trust, as well as compatibility for conference unaware clients.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications—*SIP*; C.2.4 [**Distributed Systems**]: Distributed applications—*Conferencing*

General Terms

Scalability, Reliability, Security

Keywords

Overlay virtualization, ID locator split, tightly coupled SIP conferencing, distributed conference control

1. INTRODUCTION

Voice and Video over IP (VVoIP) conference applications follow a trend to become independent tools driven by end users, since the capabilities at end systems (CPU, Memory) and the connectivity to broadband Internet are increasing continuously. They not only offer an alternative to traditional telephony, but liberate users from provider-bound infrastructure at common service charges. These changes are even more visible in the mobile domain with its spread of intelligent smartphones, and a foreseeable decline of operator control of end systems. In addition to traditional telecommunication services, VVoIP deployments open the realm to richer and more flexible use cases such as ad-hoc multi-party conversations of variable sizes.

Popular lightweight group communicators such as Skype [1] are built from proprietary models and protocols, while multiuser multimedia conferencing systems based on the Session Initiation Protocol (SIP) standard [2] are mainly deployed on dedicated server systems. Recent IETF activities, though, emerge to a new, infrastructureless session management using P2PSIP overlays and a control layer that converges to the form of REsource LOcation And Discovery (RELOAD) [3]. Conferencing solutions built by SIP and non-SIP means are still almost exclusively constructed with the help of one central conference controller per session, which — in a P2P setup — severely limits scalability and reliability of the application. Distributed conference session management has not yet been taken up by the P2PSIP community.

In this paper, we propose a virtual and distributed conference management architecture and a protocol that operate in a P2P ad-hoc mode independent of infrastructure. By separating the locator from the identifier of the conference controller, the *focus* [4] or conference Unified Resource Identifier (URI), we show how the multi-party session management can be distributed among multiple peers. We introduce a simple routing scheme that transparently guides conference signaling through the focus cloud, but still requires a globally routable physical focus instance for an initial conference contact. To overcome the dependence on individual peers, we virtualize the focus addressing within RELOAD. The conference URI, which commonly provides global routability to a dedicated focus, is published on the P2PSIP overlay network as a key to several end system devices. Further on, the transparent routing is transfered to operate on a proximity-aware overlay identifier space and gives rise to a self-adaptive tuning of the mutual communication flows.

[*]This work is supported by the German Ministry for Research and Education within the projects *Mindstone* (http://mindstone.hylos.org) and H∀Mcast (http://hamcast.realmv6.org).

The overall result of this work is a decentralized serverless system for distributed conference management with SIP coined DisCo. It solves the open problem of organizing conferences in a spontaneous, scalable and robust way based on the emerging standards of P2PSIP technologies. Evaluations reveal that the use of proximity-aware identifiers in an adaptive routing lead to a seamless self-organization with efficient neighborhood selection in our solution. These mechanisms are also designed as base implementations for a distributed media mixing which scales up to a large number of participants, and remains reliable against node departure or failures.

The remainder of this paper is organized as follows. Section 2 presents an overview of background technologies and related work on the subject, followed by a discussion of the distributed conferencing problem and its requirements. Our core distribution mechanisms for a SIP conference focus are outlined and evaluated in Section 3. Section 4 is dedicated to the conference virtualization in P2PSIP and its adoption of the core distribution scheme; RELOAD usages and kinds are defined here along with the self-organizing procedures, authentication and trust aspects and a protocol evaluation. Finally, Section 5 is dedicated to conclusions and an outlook.

2. DISTRIBUTED CONFERENCING: PROBLEM STATEMENT & RELATED WORK

2.1 Traditional Conferencing

Three models of multi-party communication have been defined in the discussion process at the IETF. The *loosely coupled* model does not provide a signaling relationship between conference participants. Membership is achieved by joining multicast groups and control information are learned out of band or from the application transport protocol (e.g., RTCP [5]). In a *fully distributed* model, each participant somehow manages a signaling dialog to all other remote participants. Finally, in the *tightly coupled* model signaling relationships are established between participants and one central point of control, that negotiates media parameters to establish media sessions. In SIP, this central point of control is called the *focus* of a conference [6]. It is identified and located by a conference-specific SIP URI that must be globally unique and routable. The first two models are not further defined, leaving details and complexity to further specifications. In the tightly coupled approach, a conference-specific URI will be obtained by querying a dedicated conferencing server. This allocates and publishes a conference URI (e.g., *sip:meeting@muppets.com*) and instantiates its corresponding focus . The focus then serves as interface towards SIP user agents that are interested in joining the multimedia session. In addition to media negotiations, a conference focus may comprise presence and conference state [7] notification services. The focus also enforces a predefined conference policy (e.g., permitted participants) and controls the media mixing components.

2.2 Peer-to-Peer SIP Overview

The P2PSIP working group is dedicated to provide a virtualized communication infrastructure for IP-based session services. It decided to rely on a structured peer-to-peer approach. Structured P2P systems are based on Distributed Hash Table (DHT) algorithms that can provide resource location and storage in an application layer overlay network. The overlay routing and data storage efforts are equally distributed among the participating peers and scales up to a very high number of joining nodes. The benefits of DHTs originate form its performance properties of typically $O(log(N))$ routing hops on average, and for a requirement of $O(log(N))$ routing table entries per node, where N is the number of overlay members. DHTs such as Chord, Pastry, CAN, and Kademlia [8, 9, 10, 11] have proved for their distributed, robust and scalable characteristics and now experience a wider deployment in various file sharing applications (e.g., BitTorrent [12]).

Proximity Aware Overlays.

Overlay network identifier are typically generated from hash-functions (e.g., SHA1 in Chord) for maintaining a uniform flat address space. These IDs normally do not have any relation to relative network positions of a nodes in the underlay. Numerical neighbors in the overlay can be physically far apart. Improved structuring of P2P overlays [13, 14] therefore may account for proximity information. One class of approaches is built from landmarks. To determine the relative network position p of a node, the round-trip times (RTT) are measured against a fixed set of well known landmarks $l_0, l_1, .., l_n$. These measurement results will be ordered according to the landmark index with the result of a *landmark vector* $< l_1, l_2, .., l_3 >$. Thereafter, the entire address space will be divide into equally sized *regions*. The definition of a region depends on the DHT and its address structure in use. The ring-type address space like in Chord can be cut into equal slices; subtrees in Pastry can define a region or an n-dimensional space in CAN. Each landmark vector permutation produces exactly one related region. A node then joins the overlay at a 'random' point in the region, that belongs to its landmark vector permutation. A node may then be assigned to a relative position according to its overlay ID, since every node has constructed its ID using the same calculations. A disadvantage of this ID construction is caused by an uneven population of the address space. This may cause peers to become responsible for much larger address ranges than others. Load-balancing algorithms can handle this problem by relocating responsibilities for overlay spaces to less loaded peers.

Other approaches construct mappings between overlay IDs and position information that are stored in the overlay. Assuming the relative position p for node-ID ID_n, then $p' = hash(ID_n)$ is an overlay identifier, for position p of node n. Node n will then be mapped with p' into the overlay region, stored on the node that is responsible for this address range. The nodes n_1 and n_2 are close to each other if the difference of $|p_1 - p_2|$ is low, with p_1 and p_2 were retrieved by a lookup operation on $p_1{}'$ and $p_2{}'$.

P2PSIP approaches.

Traditional SIP-oriented service architectures depend on proxy servers that assist in call routing, user location, NAT and Firewall traversal, as well as additional functionalities. This orientation on static infrastructure limits deployment and motivates approaches to relocate the proxy roles into a P2P overlay for SIP sessions. Peers query the overlay by using a P2P signaling protocol, and may contact an address of the desired user agent without further hindrance. Core procedures for call establishment should still be achieved by using standard SIP mechanisms. K. Singh *et al.* [15] and D.

Bryan *et al* [16] presented two different approaches, using a Chord overlay network for replacing the SIP client-server infrastructure. Both are using SIP messages within their P2P signaling protocol which are routed throughout the overlay. For example, sending REGISTER requests is mapped to the meaning of a DHT *join* message. Note that the semantics of these SIP messages are either changed or extended by new extension-header fields. Fessi *et al.* [17] presented a hybrid model, connecting a user agent to a dedicated SIP server, and likewise to a P2P SIP overlay. In this way, the authors gain the benefits of the traditional SIP of low signaling latencies and a trustworthy instance for security considerations. In the case of a SIP server failure, a user agent may regain connectivity by the P2P SIP overlay. To provide backwards compatibility, a *CoSIP* proxy server is proposed as gateway from SIP to the P2P protocol.

The necessity for an P2PSIP storage and lookup service overlay was adopted by the IETF. The P2PSIP working group is now standardizing a signaling protocol for REsource LOcation and Discovery (RELOAD) [3]. The intention is to establish a P2P overlay network based on a improved Chord DHT, providing a storage and resource location platform for different kinds of data. It is firstly designed for a usage for SIP [18], but can be extended for new kinds with similar requirements. We use this flexibility to define a new Usage for RELOAD for a virtual and distributed conference, mapping contact data and positioning informations into the overlay.

2.3 Related Decentralized Approaches

Several approaches have already dealt with the problem space of distributed conferences. S. Romano *et al.* [19] presented a framework that allows to receive information about conferences from various distributed conference servers. Therefore, they foresee signaling relations between multiple instances of dedicated centralized conferencing servers. A user agent can query its local conference server about multi-party sessions running at remote XCON Servers. The local conference server then requests the remote server for the required parameters to participate and passes them to the requesting client. An approach for a P2P SIP conference construction were developed by K. Tirasoontorn *et al.* [20]. The conference URI, created by a *Conference Factory* placed on a dedicated Server, will be announced in an P2P overlay network including the required media and contact information to join the conference. The user agent that stored the conference information in the overlay, is responsible to perform conference operations. It remains as a single contact point to manage the multi-party conversation. Y. Cho *et al* [21] presented a distributed architecture for signaling and media mixing. In this hierarchical approach, a dedicated *primary focus* server schedules conference participation requests among a set of *regional focus* servers. The latter are responsible to include the new participants and grant their access to the provided media data. The encoding effort thereby will be distributed onto several devices providing large-scale multimedia conferences.

2.4 Problems and Requirements for Virtualized Distributed Conferences

The traditional way to manage a multi-party conference is to create a central point of control. Thereby SIP correctly identifies a conference as a single logical entity and maps its identity to single point of control. The centrality of the latter is limited by scalability and stability, and conceptually forms the main problem for any approach of distributing a conference architecture [22]. This conference controller in SIP is called *focus* and plays the role of an interface to participants, serves as negotiator for media parameters, and often provides conference state notification services. The focus is identified and located by a Globally unique Routable User agent URI (GRUU). Each request of a callee will be routed to the physical device behind this address. This results in a *single point of failure* problem, as the conference breaks down with failures in this device or its connections. In a P2P scenario, the reliability of the conference controlling node cannot be guaranteed and may cause a complete failure of the conference on regular departure. Apart from signaling, the chain of decoding, mixing and again encoding of media data demands high computational effort. Therefore, common solutions for multiuser voice and video conferencing are placed at dedicated server systems. They are capable to reliably serve a fixed amount of media streams at limited numbers (video solutions are typically designed for about 20 participants). Common end user systems are only able to handle a fraction of this amount due to computing efforts. Deployed P2P-streaming (e.g., Zattoo [23]) solutions challenge the possibilities of using the end-user systems for distributed media streaming or mixing in pure audio. Currently, many approaches apparently remain at a borderline quality, but provisioning of reliable media streams will be soon enabled by the continuous dissemination of high speed Internet connections in home networks and the rising computational power of consumer computer.

From this perspective, we follow the need to design a distributed conferencing scheme in a P2P fashion as a future standard-based solution for fully distributed voice and video conferences. Therefore we define a set of requirements to be met by our distributed conferencing protocol DisCo:

- **Ad-hoc conference creation** Any user agent implementing the conferencing scheme, must be able to create a multi-party session at any time. The creation must be independent from a server infrastructure.

- **Splitting the central conference control** The conference focus must be divisible into several independent end systems. The split of the focus must thereby be transparently achieved with respect to standard-compliant SIP implementations and should appear as one single entity. The focus distribution should be activated prior to a focus peer management resource exhaustion. Any party should be enabled to discover other potential focus peers within among active members.

- **Robustness against focus failure** It must be possible to *re-arrange* (not to re-create) a conference, as one or more controlling peers fail, and thus to increase the reliability as compared with centralized solutions.

- **Availability of a conference** To provide accessibility to a distributed conference, it must be announced on a stable platform. For this purpose, a well-defined conference data structure must be stored redundant in a P2P network, that allows to resolve a conference URI, that points to several independent conference managers as entry points.

- **Proximity aware participation** The proposed conference signaling topology should serve road map for the transfered media data. The media processing peers should be arranged. New participants should be able to select the physically closest focus peers, to minimize signaling and data transfer delays.
- **Security and Privacy** A distributed conference must ensure that only authorized participants can attend the conference. Also needs to be ensured that only determined user can change and manage a conference state.
- **Backwards compatibility** A virtualized and distributed conference must be accessible by client implementations that do not support our DisCo Usage.

3. DISTRIBUTING A FOCUS WITH SIP

3.1 Protocol Scheme

The first step for designing a distributed conference is to separate the central control of the focus at the SIP layer. A conference URI refers per se to a dedicated focus. Our Scalable Distributed CONference (SDCON) [24] approach splits the meaning of the conference URI into identifier and locater. This is achieved by introducing a source routing approach, which transparently forwards data among conference controllers that share a common conference URI. The *focus service* of a conference is distributed among several participating user agents supporting the SDCON scheme. This leads to two classes of focus. First, the *primary focus*, which initially arranged and managed a multi-party conference. Second, the *secondary focus*, which is a participating user agent requested by the primary focus to become part of the distributed conference controllers. There is no functional difference between primary and secondary focus. Participants can have a signaling relation to either a primary or secondary focus. Both provide the same conferencing operations and notification services based on the same predefined policies. However, the conference URI is bound to the primary focus. We propose the virtualization of the conference identifier in section 4. This allows to completely decouple the conference URI from dedicated peers.

Conference initiation, control, and management is performed by the participating user agents adapting to the size of a dynamically growing conference. Therefore, SDCON defines a focus discovery procedure, call delegation, and state synchronization mechanisms. As the primary focus delegates a call to a secondary focus, it also transfers the used SIP Call-ID and session identifier. Using this information, a secondary focus is able to seamlessly send a re-invitation to the transfered user agent and negotiate new media parameters. To implement the source routing, the secondary focus inserts a *Record-Route* header field carrying its Globally unique Routable User agent URI (GRUU). Further signaling is thus routed to the secondary focus. An example of the SIP re-invite request is shown below:

```
INVITE sip:elmo@sesamestreet.com SIP/2.0
Call-ID: 0818@141.22.26.55
CSeq: 1 INVITE
From: <sip:puppets.meet@conf.muppets.com>;tag=134652
To: <sip:elmo@sesamestreet.com>;tag=643684
...
Contact: <sip:puppets.meet@conf.muppets.com>;isfocus
Record-Route: <sip:kermit@sesamestreet.com>
...
```

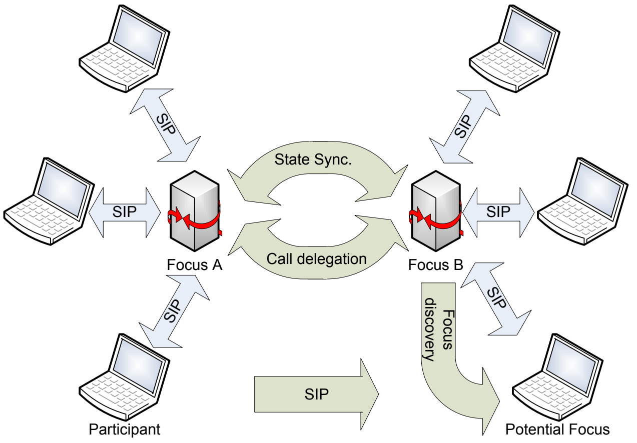


Figure 1: A distribute conference control scenario

The Record-Route header is usually added by SIP proxies to force further requests in a SIP dialog to be routed via these entities [2]. In the example above, the secondary focus *kermit* adds its own SIP URI into the Record-Route header and forces the re-invited user *elmo* to send subsequent SIP requests via him. Those source-routed requests to secondary focus peers are intercepted by them and processed. Only the focus peers are aware of the distributed fashion of conference control. Participants do not recognize the ID/Locator split, thus, the compatibility to SIP standard compliant implementations is achieved.

Figure 1 shows the main functionalities supported by SDCON user agents. The focus peers maintain signaling relations mainly by two message flows: the *State synchronization* messages and the *Call delegation* request messages. Call delegations occur when a focus is fully booked and needs to refer additional calls to less loaded focus peers. This is realized by sending standard SIP compliant REFER requests. Plain calls that address the conference URI are routed to the primary focus. Call delegation, thus, will mainly be performed for secondary focus peers. Synchronization messages are sent on change of state in any single focus entity, e.g., announcing the arrival of a new participant. These messages have to reach every controller to keep a consistent view on the conference. Synchronizations are sent within SIP NOTIFY messages carrying an XML document defined by the Event Package for Conference State [7], which is extended for multi-focus demands. The additional elements include information about each focus capacities, list of the participants that are connected to it, and shows the signaling relation to other focus peers. The capacity information is used to prevent a call delegation to an already busy focus. In the case that the synchronization process has not been completed while a call delegation is performed, each focus peer can use SIP 4xx response messages types [2] to advertise its status as busy. Another function consists in the ability to discover focuses capabilities among participating peers [24]. The *focus discovery* procedure is initiated before a focus reaches its threshold for serving new clients.

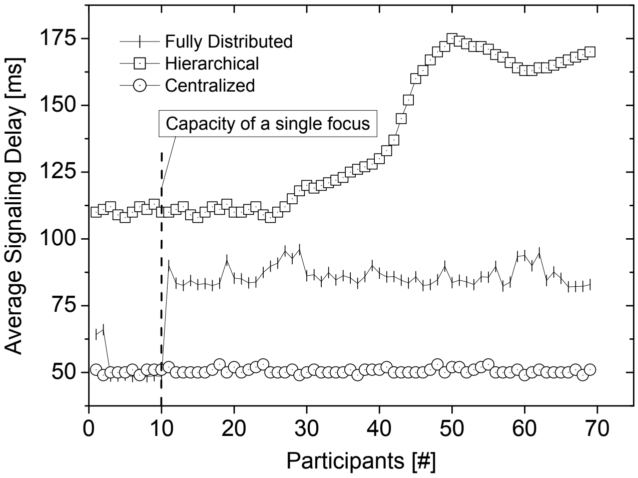


Figure 2: Time to completion of an INVITE request for a newly arriving peer

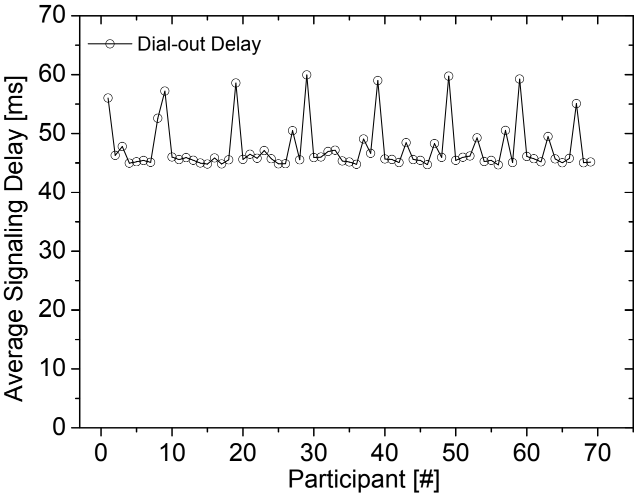


Figure 3: Third-party participation via REFER requests

3.2 Evaluation

To validate the operation and test the scalability of SDCON signaling, we implemented a prototype application and performed experimental measurements. The prototype is based on the NIST Jain SIP stack [25], which represents the reference implementation for Java. All measurements were performed in emulation mode. A minimal SIP proxy implementation was executed on a Pentium D 2*2.80 GHz 2 with 2 GB RAM. The emulated participants and conference focus peers have been executed on an Intel(R) Xeon(R) CPU 16*2.33 GHz with 16 GB RAM. The capacity of a single focus was fixed to 10 conference members. Each measurement result presents the average signaling delay of 50 independent runs.

Figure 2 presents the average signaling delay to participate a conference, i.e., sending SIP INVITE requests towards the conference URI. It compares our fully distributed conference management with a centralized [4] and hierarchical [21] approach. The later implements a recursive call delegation starting from the primary focus along the focus servers. For small conferences, where all parties can be served from a single focus, our results agree with delays of a centralized approach. The redistribution of the focus attachment in our scheme causes one additional REFER message and thus slightly doubles the signaling times. Apart from this delay enhancement, the distributed conferencing admits almost constant delays, in contrast to the hierarchical scheme. The latter experiences increasing delays of approximately linear scale with growing conference size.

The signaling delay for a third-party invitation is presented in Figure 3. In this scenario, each recently joined conference member initiates a third-party request to its related conference focus peer by sending corresponding SIP REFER messages. The measurements follow our previous observations. Most third-party participations are handled in a constant signaling delay around 45 ms. Delay peaks reflect overloaded focus peers with a maximum of 10 conference members. On reaching more than 10 members, a focus initiates the focus discovery procedure and delegates further participation requests to the new capable focus peer.

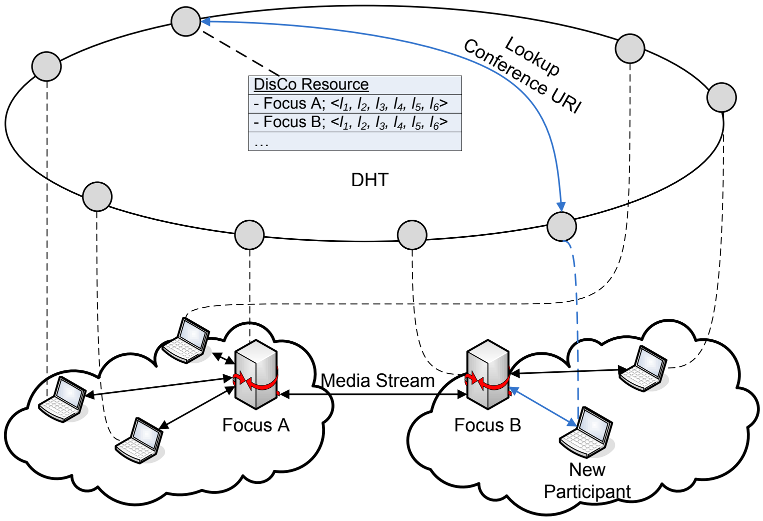
4. VIRTUALIZED CONFERENCE CONTROL WITH P2PSIP

The aim of virtualizing the conference is to separate its logical ID from any physical instance. The P2PSIP overlay RELOAD [3] facilitates a corresponding mapping and lookup of currently available conference management peers. By using P2PSIP with RELOAD we gain the benefits of an open and extensible signaling protocol that provides solutions for common problems in traditional SIP and P2P systems.

RELOAD serves as a P2P service platform providing a message transport protocol, data storage and lookup functionalities, as well as connection establishment for different types of applications. Since connectivity of many peers in an overlay may be limited by NATs or firewalls, Interactive Connectivity Establishment (ICE) [26] is supported for NAT and firewall traversal. RELOAD also provides a security framework based on public/private-key certificates to establish trust relations and message authentication.

Overlay messages are designed with a simple and lightweight forwarding header reducing forwarding effort and increasing the routing performance. A noteworthy feature of RELOAD is that the overlay algorithm to be used is not fixed, but left to the implementation. However, the current version of the RELOAD draft foresees a deployment on an improved Chord distributed hash table (DHT). To support different applications, RELOAD allows for the specification of new *Usages.* A Usage defines the data structures (*kinds*) to be stored, the corresponding data identifier (*kind-ID*), *access control* rules to those resources and how the resources' overlay IDs are to be formed.

Our concept of a virtual and distributed conference control uses these RELOAD benefits to provide a reliable, flexible and scalable conferencing service in a P2P fashion. We define a RELOAD Usage for separating the conference URI from any specific focus entity and map it to the set of participants that act as a focus instance. The proposed RELOAD data structure provides network positioning information to enable a proximity based focus selection. Based on this kind definition, our Distributed Conference Usage (*DisCo*)



Lookup Conference URI
DisCo Resource
- Focus A; <l_1, l_2, l_3, l_4, l_5, l_6>
- Focus B; <l_1, l_2, l_3, l_4, l_5, l_6>
…
DHT
Media Stream
Focus A
Focus B
New Participant

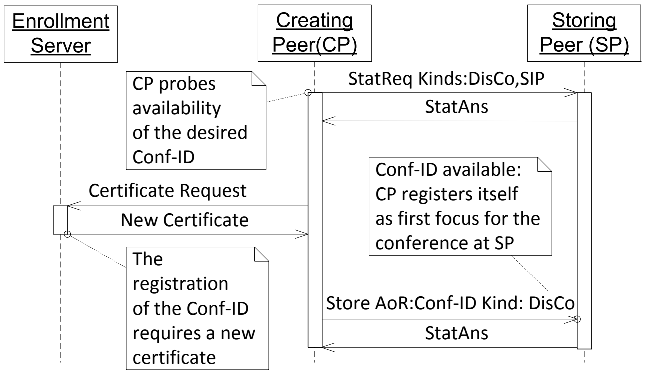
Figure 4: Discovery of a secondary focus using proximity information

[27] allows for the ad hoc creation of multimedia conferences without a dedicated server infrastructure. Conference signaling is performed using the call delegation and synchronization mechanisms as described in the previous section.

4.1 Distributing a Focus in RELOAD

DisCo defines a distributed SIP conferencing Usage that publishes all available entry points to a conference in a P2P fashion. The inter-focus SIP signaling is performed using the SDCON protocol scheme presented in the previous section. This keeps the conference state in sync and performs load balancing whenever focus peers are reaching their service threshold for hosting clients. The DisCo Usage allows a SIP user agent to create a tightly coupled conference in P2P fashion, without assistance of a dedicated conference server. Figure 5 displays the procedure of how to register a distributed conference in a RELOAD instance. The creating peer (CP) of a conference generates the desired conference URI (Conf-ID) and first probes whether this address is available. This is performed by using the RELOAD StatReq message which is routed to the storing peer (SP) responsible for the overlay ID. Overlay storage is organized according to keys obtained by hashing the conference URIs. The corresponding StatAns messages contains all meta data about the RELOAD *resources* already stored at this *resource-id*. If no other DisCo or SIP registrations for the selected Conf-ID exist, CP can proceed by querying the enrollment server of this RELOAD instance to obtain a new certificate created for the conference URI. Using this security certificate, CP then creates a DisCo *kind* data structure that comprises tuples of two types of information. At first the address where a joining peer can contact the CP to join the conference, at second a coordinate vector that encodes the relative position of the CP within the underlying network. Using the RELOAD Store operation, CP registers the conference in the overlay.

The distributed conference registration will be treated as a RELOAD *resource* of *Kind* DisCo maintained by the storing peer. The RELOAD overlay itself acts as a registrar and establishes direct transport connections traversing NATs and firewalls.



Enrollment Server
Creating Peer(CP)
Storing Peer (SP)
StatReq Kinds:DisCo,SIP
StatAns
CP probes availability of the desired Conf-ID
Conf-ID available: CP registers itself as first focus for the conference at SP
Certificate Request
New Certificate
The registration of the Conf-ID requires a new certificate
Store AoR:Conf-ID Kind: DisCo
StatAns

Figure 5: Creation of a distributed conference

DisCo-enabled peers intending to participate in the conference need to look up the hash of the conference URI as displayed in figure 4. They retrieve the DisCo conference resources, i.e., a RELOAD *dictionary* data structure in which each single dictionary entry points to a distributed conference focus. In a RELOAD *dictionary* data model, each value stored is indexed by a *key*. Using this index scheme, a focus peer can explicitly update its own contact and coordinates information maintaining its own overlay ID as dictionary key. The contact information of the conference focus can be of two different types, an Address-of-Record or a RELOAD overlay ID. In the first case, if the retrieved Address-of-Record (AOR) is a GRUU, the participating peer simply establishes a regular SIP session by sending a SIP INVITE request towards the announced contact. Otherwise the received AOR is registered with the standard SIP Usage for RELOAD and must be resolved following the SIP Usage protocol. If the retrieved contact is a RELOAD overlay ID, a participating peer needs to perform a RELOAD *appattach* request to establish a direct connection to the remote overlay peer. This request will be routed along the overlay with ICE parameters and defines the desired application protocol

```
enum {sip_focus_uri (1), sip_focus_node_id (2)
} SipDistConfRegistrationtType;
struct {
  opaque  coordinate<0..2^16-1>
  select (SipDistConfRegistration.type) {
      case sip_focus_uri:
        opaque  uri<0..2^16-1>
      case sip_focus_node_id:
        Destination destination_list<0..2^16-1>
  }
} SipDistConfRegistrationData
struct {
    SipDistConfRegistrationType     type;
    uint16                          length;
    SipDistConfRegistrationData     data;
} SipDistConfRegistration
```

Figure 6: Proposed RELOAD data structure for a distributed conferencing kind

as SIP. After the *appattach* request has succeeded, an ordinary SIP session will be build upon the newly created transport connection. A new conference member can advertise its focus ability by adding an *allow event* to the multi-focus conference state event package in the `INVITE` request.

Each contact in the data structure is complemented by coordinate values that indicate the relative position of the peer within the underlying network. Based on this information, a joining peer may choose the focus from the dictionary entries that is closest according to the proximity selection mechanism explained in the following section.

After a *DisCo*-enabled peer has established a SIP session by sending an INVITE, it is free to decide on advertising its own capacities. To do so, it registers as a *potential focus* to the conference storing its contact and network positioning information within the same *DisCo* resource. Focus functions will be activated either by a new joining peer that chooses this potential focus as (nearest) entry point, or by the focus *discovery procedure* explained in section 3. As a potential focus is requested by a user agent to participate via SIP signaling, it first accepts the call and establishes the requested media sessions to this client. Afterwards, the active focus will advertise its new status to all other active peers managing the conference. It subscribes its related focus to the extendedconference event package while transmitting its focus capacities and contact and media information of the new participants. The request focus will interpret this message as indication for a new user agent acting in the role of a focus and notifies all remote conference controller about this change of state. It further responds with a SIP `SUBSCRIBE` request to the new focus, transmitting the conference state XML document. This finishes the focus acceptance and the potential focus is a known active focus. All focus peers take the same authorities and responsibilities to manage the distributed conference as the initial focus.

The definition of the distributed conference registration kind is shown in figure 6. Every focus peer is allowed to store or update mapping bindings using its node-id as the dictionary key. The mappings stored can be of two varieties corresponding to the types allowed in the SIP-REGISTRATION: The first type *sip_focus_uri* contains the Address-of-Record of a focus peer and the second *sip_focus_node_id* returns

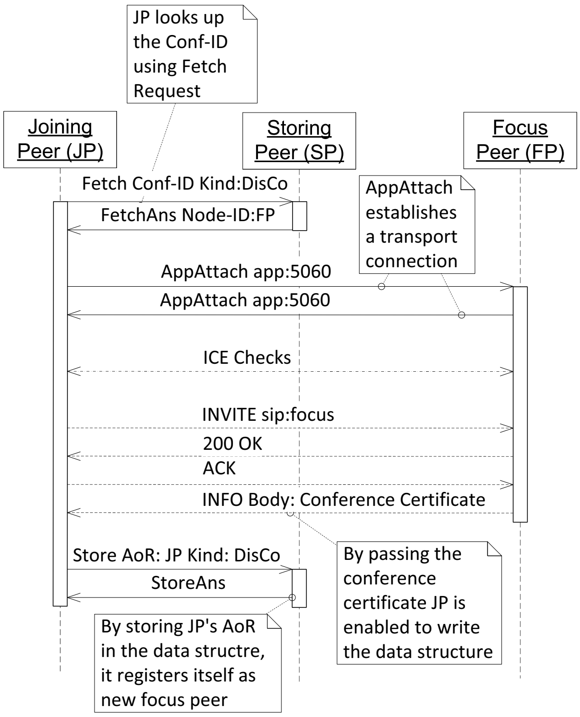

JP looks up the Conf-ID using Fetch Request
Joining Peer (JP)
Storing Peer (SP)
Focus Peer (FP)
Fetch Conf-ID Kind:DisCo
FetchAns Node-ID:FP
AppAttach establishes a transport connection
AppAttach app:5060
AppAttach app:5060
ICE Checks
INVITE sip:focus
200 OK
ACK
INFO Body: Conference Certificate
Store AoR: JP Kind: DisCo
StoreAns
By passing the conference certificate JP is enabled to write the data structure
By storing JP's AoR in the data structre, it registers itself as new focus peer

Figure 7: Joining a distributed conference and advertising focus abilities

the a RELOAD *destination list* containing overlay node-IDs. The destination list feature in RELOAD is used, to enable a requesting peer to perform a recursive overlay source routing. We define for the DisCo Usage, that the accompanying coordinates value belongs to the final target of the destination list. If storing an AoR, the related coordinates value must define the relative position of the AoR location. The coordinates value is stored as an opaque string containing the relative network defining a *landmark vector*. A *landmark vector* represents a set of Round-Trip-Times (RTT) measurements against well-known landmarks. A more detailed explanation follows in the next section. We use is explicit coordinate value, because it can not be assumed that used overlay algorithm in an RELOAD P2PSIP instance supports proximity awareness. The proposed Chord overlay in the RELOAD base definition for example, does not support proximity information.

4.2 Self Organization with Proximity-aware Load Sharing

The DisCo conference construction is performed using relative network position information. Each joining participant chooses its closest focus, and every new peer managing parts of the conference establishes an SDCON relation to its nearest active focus node. A benefit of this proximity peer selection arises from an optimized mesh build-up causing short signaling paths by default. The single steps to joining a virtual and distributed conference are the following as displayed in figure 7:

1. **Determining coordinates:** Before a peer joins the multiparty conversation, it determines RTTs against a set of

stable Internet hosts $l_1, l_2, .., l_n$ serving as landmarks. The measurement results are ordered along a landmark index that is equal for all parties and focus peers. Ordered in this manner, the measurement results in milliseconds comma-separated define our *landmark vector* representing a peers relation network position in an n-dimensional Cartesian space with n is the number of landmarks (e.g. $< 311, 87, 42, 137, 228, 75\ , .., 55 >$). We thereby follow the landmarking approach from Ratnasamy *et al.* for proximity-aware server selection [13] without the explicit binning of peers whose landmark vectors equal each other. It just serves as am abstract descriptor for a peer's relative position in the network and is not used to identify a peer.

2. **DisCo data structure retrieval:** To obtain alle available focus peers for a conference the joining peer (JP) achieves a RELOAD *fetch* request that is routed to the storing peer thats own the resource-id for the hashed Conference URI. It thereby in the sets the *kind* value in the request to the DisCo kind-id querying for the complete conference dictionary.

3. **Calculating the closest entry point:** On successful received the conference information, a peer compares each retrieved coordinates value representing the focus landmark vector with its own. Our approach subtracts the each focus landmark vector with that of the joining peer and builds the scalar product over the result of a substation. The joining peer then chooses that focus with the smallest scalar product result as entry point to the conference.

4. **Connecting to a Focus:** Using contact information deposited dictionary entry, JP establishes a transport connection to the selected focus peer (FP) using the RELOAD's AppAttach operation. It is routed throughout the overlay to FP and indicates a desired SIP signaling connection by setting the *application* field to *5060*. After FP finalized the AppAttach progress JP and FP perform ICE checks [26] to detect whether any of them if located behind a NAT and additional TURN server are needed for application session establishment.

5. **SIP Session establishment:** The established transport connection is then used to enter the ordinary SIP signaling progress thus JP can successfully join the multiparty conversation. Additionally, JP can pass JP writing permission to the DisCo registration by transmitting the shared certificate within a SIP INFO message.

6. **Advertising focus abilities:** JP can optionally advertise itself as available focus peer for the distributed conference, by mapping its contact to the existing DisCo data structure at the storing peer.

The joining peer hereby adds its own landmark vector coordinates as an URI parameter *coord* base64 encoded to its URI in the SIP *contact* header. The *coord*-parameter is used by the requested focus in case of overloading. It then performs the *call delegation* mechanism and selects a focus candidate according to the new participants network positioning. If the selected focus is capable to serve new clients it accepts the SIP call. Further, it published the new membership by achieving the SDCON synchronization mechanism explained in section 3, to keep the conference state consistent.

Because every new member chooses its closest focus, the conference will be constructed unified distributed among all controlling peers, like shown in figure 4. Following this regular construction, participants and focus peers will arrange themselves to an unbalanced distribution tree. To reduce the diameter of this tree, hence minimizing the delay times between the nodes, it is possible to establish cross connections. This kind of mesh optimization are highly dependent on the types of used media streams, and is therefore out of scope of this paper.

4.3 Resilience to Focus Failures

A problem in traditional tightly coupled conferences, originates from the focus that acts as single point of failure. If it breaks down, all signaling and media sessions are disconnected. In our scenario, the distributed structure of the conference prevents the breakdown of the entire multimedia session as one focus peer fails. As a focus fails, it can be substituted by potential or active focus peers re-collecting lost conference participants.

We use this redundancy to build a recovery mechanism. As a DisCo-enabled participant notices that its related focus does not any more deliver signaling or media packets, it will connect to one of the remaining managers of the conference. It therefore achieves the same DisCo protocol steps explained previews, however, without redetermining its landmark vector.

Conference participants not supporting the DisCo Usage will get a different treatment in case of focus error. A conference focus selects one or more active focus peers, that will serve as *backup* focus. The selection is done according to the relative network *coordinates* by choosing the closest peers. The backup selection will be announced to all other conference controller within the conference state XML document. In the case of node appearance, the detecting focus firstly notifies the conference managing peers about failure to share knowledge. It then immediately refers all disconnected participants to the *backup* focus peers. In this way, participants related to the malfunctioning conference controller just notice a temporally connection loss and recover via a re-invitation mechanism. Whenever the malicious focus returns, it re-joins the conference normally. Otherwise, the dictionary entry of this peer will be deleted by the resource owner, after the *lifetime* value expires in RELOAD.

New participating peers who try to connect to a disappeared focus will receive a 404 Not Found response message, according to the RELOAD protocol. These peers then try to connect to the focus, whose landmark coordinates are the second closest to their own. The stored DisCo data is protected against failure of the resource owner, by the provided replication algorithms in the used DHT running the RELOAD P2P SIP instance.

4.4 Security & Trust Aspects

The DisCo Usage defines a set of security and trust aspects in a P2P environment. A common problem in distributed P2P systems arises from the fact, that connections will be established, even though the corresponding partners do not necessarily trust each other. In our conference scenario, we assume that participating peers can authenticate each other in person based on the received voice and video transmissions. Built on this, we introduce a graduated trust delegation system for distributed conferences.

RELOAD provides a set of access control policies, defining whether a peer is allowed to perform a certain store request or not. For our distributed conferencing resource, we use the defined *user-match* access policy. Each stored data can be written if the request is signed with a key associated with a certificate whose hashed user name equals the resource's overlay ID. Since our DisCo resource needs to be updated by multiple peers, using the *user-node-match* policy used by the SIP Usage for RELOAD is not an option. We use our trust delegation mechanism, allowing peers to obtain write access to the shared resource. To receive write permissions for the distributed conference overlay resource, the private key for the certificate of the stored resource will be transmitted within a SIP `INFO` message to allowed focus peers. Using this key, a user is able to authenticate itself against the owner of the conference data structure, and can register as potential focus.

On conference creation, the initiating peer can setup the distributed conference policy for a layered authentication. In an *open access* model, every peer interested to join the conference can do so by just inviting one of the multi-party focus peers. No authentication is required for participating. An open access model may be suitable for a group conversation of public interest, for example a political debate. Because in this open model, an attacker could easily become a focus peer and send malicious packages, we define an *open access focus authenticate* model. The conference initiator can specify that peers wanting to become a focus need to authenticate themselves using any of the standard authentication mechanisms allowed in SIP. The corresponding credentials need to be transmitted to those peers by a non-SIP, non-overlay mechanism. As a new participant invites the conference, it uses ordinary SIP authorization. After validation of the presented credentials the called focus is then allowed to pass the conference's certificate key to the recently joined conference member. To create a closed multimedia conference, it is also possible to set an authentication scheme required for participation in a *closed access* model. Thus, only users who present valid authorization credentials are allowed to join. By combining the *closed access* and the *focus authenticate* model, our layered access model defines different permissions for clients joining the conference only and peers that are allowed to become focus, dependent on their credentials. Focus peers obtain the information needed to validate participants' credentials within the conference XML document (e.g. a conference password or certificate), to be able to authorize new members. The used access model will be stored in the conference state XML extension, thus every controlling peer is aware of the used access model.

Providing these access layers, a user initiating a conference is able to setup its desired privacy policies for the multi-party conversation. It can be suggested, that in closed conferences an unknown conference member will be detected by the participating users, for example by not recognizing its voice or outwards appearance in video. Those unsuspected users can be excluded by a conference focus peer by disconnecting signaling and media sessions.

4.5 Supporting Conference-unaware Parties

Participation a virtual and distributed conference is not be exclusive for those peers that implemented our RELOAD Usage definition. Standard compliant participation is transparently provided to peers unaware of the distributed conference construction. This section describes the backward compatibility to user applications implementing the SIP Usage [18] for RELOAD, and describes how connectivity to SIP-only user agents is achieved.

The SIP Usage for RELOAD defines a *kind* data structure for storing an AoR for a SIP user agent. It likewise uses the *destination list* feature in RELOAD and provides the storage of GRUUs as contact addresses for SIP session establishment. To provide backward compatibility to RELOAD peers only implementing the SIP Usage, a conference initiator can decide to register the conference URI as SIP-Registration *kind* parallel to the DisCo kind. The SIP Usage registration is then performed using the *destination list* feature, registering the amount of active and potential focus peers as entries in the destination list. Peers attending to join requesting resolving conference URI using SIP-Registration kind-ID, retrieve the destination list containing the conference entry points. The connection to a conference focus then will be achieved in accordance with the SIP Usage. Those peers will not be aware of the distributed structure of the multi-party conversation.

Because the SIP usage for RELOAD access model is *user-node-match*, other focus peers will not be able to update the stored data. The conference initiator must update the SIP-Registration kind continuously, on appearance or disappearance of focus peers. Hence, it depends to the conference initiator to keep the destination list up to date and valid. To achieve maximal accessibility in the case that the latter permanently leaves the multi-party conversation, it has to set the *lifetime* value on a high level. By just using the SIP-registration kind, conference joins can not be performed under proximity selection. However, a conference created with DisCo can provide access to the multi-party, although a client does not implement our usage.

The participation for ordinary SIP user agents is performed by another mechanism. Since a virtualized conference URI is stored in a RELOAD overlay, a standard SIP user agent can not resolve it with traditional mechanisms and a direct participation is not possible. Instead, participation will be achieved through third-party initiated from within the conference. An established multi-party member requests its related focus to invite the new attendees sending SIP REFER requests. By using the protocol mechanisms for transparent focus distribution explained in section 3, the requested conference manager invites the new attendee sending a SIP INVITE request.

4.6 Evaluation

To verify our concept of the proximity-aware focus selection, we conducted experimental measurements based on the PlanetLab platform [28]. PlanetLab nodes are globally distributed and thus allows for geographically placement of conference peers. Although this real-world experimental facility is biased in the sense that significant nodes are located at well-connected university networks, it gives a good approximation of delay characteristics for this part of the Internet.

CAIDA Monitor	Location
mnl-ph.ark.caida.org nrt-jp.ark.caida.org she-cn.ark.caida.org	ASIA
dub-ie.ark.caida.org lej-de.ark.caida.org her-gr.ark.caida.org pna-es.ark.caida.org	Europe
sea-us.ark.caida.org mty-mx.ark.caida.org amw-us.ark.caida.org yto-ca.ark.caida.org wbu-us.ark.caida.org	North America
hlz-nz.ark.caida.org	Oceania
gig-br.ark.caida.org scl-cl.ark.caida.org	South America

Table 1: Selected landmark nodes chosen from CAIDA measurement monitors

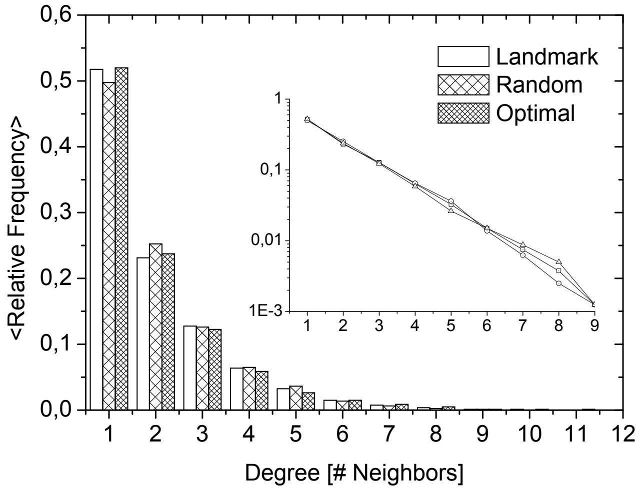


Figure 8: Degree distribution

Experimental Setup

The experiment considers small and medium size conferences. The principle setup is the following: We deployed DisCo on a varying number of peers chosen from a predefined subset of all PlanetLab nodes. These peers create focus instances and corresponding relationships based on the landmarking approach described in section 4.2. Relative network positions are determined using 15 landmark nodes outside of the PlanetLab system. The experiment is conducted until measurements are converged.

100 nodes are selected from the overall PlanetLab nodes to create the list of potential DisCo peers. To mitigate local system disturbances, we included only hosts that exhibit an appropriate system load. The selected nodes are located in Asia, Europe, South and North America to emulate a globally distributed conference and observe long range delay effects.

In general, the quality of landmark approaches depends on an appropriate number of landmark nodes and their placement. However, there is no common sense on the number of dimensions to create a coordinate system [29]. They may range typically of 7 to 9 [30], but also depend on the dataset. In order to evaluate proximity-awareness for an almost generic scenario with respect to the selected DisCo peer, i.e., without any dedicated landmark optimization, 15 landmarks are chosen from the set of CAIDA [31] monitor points (cf. Table 1). This has two advantages: First, CAIDA monitors are globally reachable and not located behind NATs or firewalls, which is important and a realistic deployment assumption for landmark nodes. Second, they are globally distributed covering different geographic locations. The landmark selection process omits suspicious nodes that reply unusually on ICMP echos.

Performance Metrics

We analyze the quality of our proximity-aware self organization of (focus) peers based on the following metrics:

Degree corresponds to the number of neighbors. Nodes that have a degree of 1 only participate in the conference without replicate data. Nodes with a larger degree operate as focus. This metric, thus, reflects implicitly the load on a peer.

Delay Stretch measures the ratio of the average delay caused by the overlay and the average delay using native distribution. It follows the idea of the relative average delay (RAD) defined by Castro et al. [32]. This metric represents the relative delay penalty.

Foci Ratio describes the ratio of overlay peers that attain the role of a focus. This metric quantifies the distribution of conference management load among peers.

The results are compared with a complete random selection of focus nodes, and an optimal solution of the focus and peer topology.

Results

The degree distribution of inter-peer relations was measured and displayed in Figure 8. For all schemes, the majority of nodes are single-attached and thus pure leafe nodes. Focus nodes that exhibit a degree ≥ 2 dominantly admit low degrees and thus suffer little load of packet replication and forwarding. More significantly and clearly visible from the insert, the probability of higher degrees exponentially decreases leaving negligible weight to the occurrence of overloaded peers or unsuitable conferencing demands.

A more sensitive measure on detailed conference performance is given by the delays imposed mutually related peer neighborhoods on the overlay. Figure 9 compares the average delay stretch of our landmarking scheme with a random neighbor selection and the optimal set-up. While a routing via arbitrary conference neighbors in our global conference evaluation may lead to alienating delay enhancements of 15 to 30 times, our landmarking scheme remains within favorable bounds around 2 to 3, which is very close to the optimal solution. Most importantly, the delay stretch remains constant with respect to the numbers of conferencing parties and thus promotes arbitrary scalability of our proposed adaptive self-organization scheme. It should be noted that randomized neighbor selection leads to a linearly increasing stretch.

Finally, we examine the relative portion of peers that attain the role of a conference controller assuming the absence

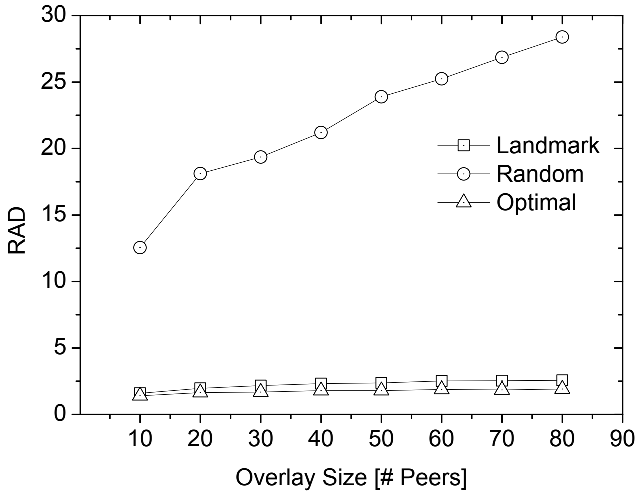


Figure 9: Delay stretch based on the Average Delay Ratio (RAD)

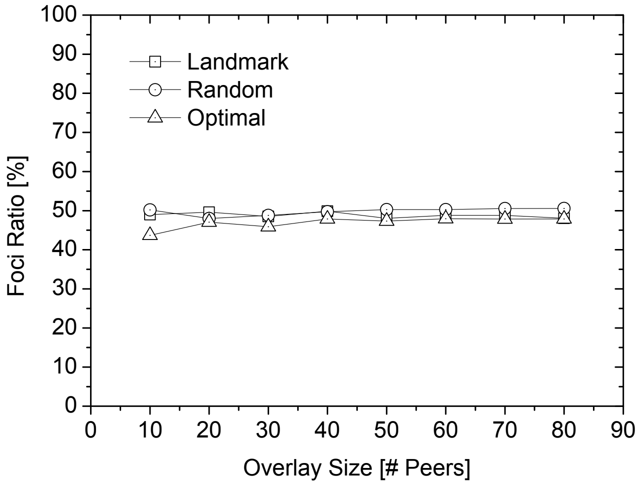


Figure 10: Ratio of overlay peers attaining the role of a focus

of NATs and firewalls. As displayed in Figure 10, the relative portions of focus peers is bound to about 50 %, independent of the conference size, as well as the adaptation scheme in use. Peers thus encounter a probability of 0.5 to be utilized as conference supporters at a scale the remains fully independent of conversational parties.

5. CONCLUSION AND OUTLOOK

In this paper, we presented a virtual and distributed conference control solution, self-organizing and adapting to the demands of a scalable, infrastructure-resilient multi-party conversation. Presenting a protocol scheme that transparently splits a SIP conference focus onto multiple peers, an address virtualization of the conference URI separates the logical ID from any physical instance. We demonstrate how this concept is implemented in a RELOAD DHT of P2PSIP, providing independence from any server infrastructure. To meet the requirements of a transient P2P environment, the presented protocol schemes maintain operations for call delegation, load balancing and state synchronization. To reduce signaling delays, we proposed a method for routing with respect to relative network position of peers and to enable a proximity-aware focus selection.

The conducted experimental measurements revealed close to optimal results for our presented concepts. We showed that the signaling delay remains constant during an increasing conference. Furthermore, our measurements on the PlanetLab platform displayed, that our proximity-aware focus selection achieves a low delay stretch. Reducing the edge degree per node and diameter of the arising tree-like mesh topology, we expect to apply further optimizing algorithms for future work. We propose to bring the concept of a virtualized and distributed conference Usage into the IETF standardization process.

6. REFERENCES

[1] "The Skype homepage," http://www.skype.com, 2009.

[2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," IETF, RFC 3261, June 2002.

[3] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol," IETF, Internet-Draft – work in progress 08, March 2010.

[4] J. Rosenberg, "A Framework for Conferencing with the Session Initiation Protocol (SIP)," IETF, RFC 4353, February 2006.

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF, RFC 3550, July 2003.

[6] O. Levin and R. Even, "High-Level Requirements for Tightly Coupled SIP Conferencing," IETF, RFC 4245, November 2005.

[7] J. Rosenberg, H. Schulzrinne, and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State," IETF, RFC 4575, August 2006.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 149–160.

[9] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Application-Level Multicast Using Content-Addressable Networks," in *Networked Group Communication, Third International COST264 Workshop, NGC 2001, London, UK, November 7-9, 2001, Proceedings*, ser. LNCS, J. Crowcroft and M. Hofmann, Eds., vol. 2233. London, UK: Springer–Verlag, 2001, pp. 14–29.

[10] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, ser. LNCS, vol. 2218. Berlin Heidelberg: Springer–Verlag, Nov. 2001, pp. 329–350.

[11] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proc. of the 1st Int. Workshop on Peer-to*

Peer Systems (IPTPS '02), Cambridge, MA, USA, 2002, pp. 53–65.

[12] "The BitTorrent Homepage," http://www.bittorrent.com/, 2010.

[13] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proc. of 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '02)*, Washington, DC, USA, 2002, pp. 1190–1199.

[14] Z. Xu, C. Tang, and Z. Zhang, "Building topology-aware overlays using global soft-state," in *Proc. of the 23rd Int. Conf. on Distributed Computing Systems (ICDCS '03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 500.

[15] K. Singh and H. Schulzrinne, "Peer-to-peer internet telephony using sip," in *Proc. of the int. workshop on Network and operating systems support for digital audio and video (NOSSDAV '05)*. New York, NY, USA: ACM, 2005, pp. 63–68.

[16] D. A. Bryan, B. B. Lowekamp, and C. Jennings, "Sosimple: A serverless, standards-based, p2p sip communication system," in *Proc. of the 1st Int. Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications(AAA-IDEA '05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 42–49.

[17] A. Fessi, H. Niedermayer, H. Kinkelin, and G. Carle, "A cooperative sip infrastructure for highly reliable telecommunication services," in *Proc. of the 1st int. conf. on Principles, systems and applications of IP telecommunications (IPTComm '07)*. New York, NY, USA: ACM, 2007, pp. 29–38.

[18] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "A SIP Usage for RELOAD," IETF, Internet-Draft – work in progress 04, March 2010.

[19] S. Romano, A. Amirante, T. Castaldi, L. Miniero, and A. Buono, "A Framework for Distributed Conferencing," IETF, Internet-Draft – work in progress 06, January 2010.

[20] K. Tirasoontorn, S. Kamolphiwong, and S. Sae-Wong, "Distributed p2p-sip conference construction," in *Int. Conf. on Mobile Technology, Applications, and Systems (Mobility '08)*. New York, NY, USA: ACM, 2008, pp. 1–5.

[21] Y.-H. Cho, M.-S. Jeong, J.-W. Nah, W.-H. Lee, and J.-T. Park, "Policy-Based Distributed Management Architecture for Large-Scale Enterprise Conferencing Service Using SIP," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 10, pp. 1934–1949, Oct. 2005.

[22] T. C. Schmidt and M. Wählisch, "Group Conference Management with SIP," in *SIP Handbook: Services, Technologies, and Security*, S. Ahson and M. Ilyas, Eds. Boca Raton, FL, USA: CRC Press, December 2008, pp. 123–158, on invitation. [Online]. Available: http://www.crcpress.com/product/isbn/9781420066036

[23] "The Zattoo Homepage," http://www.zattoo.com/, 2010.

[24] A. Knauf, T. C. Schmidt, and M. Wählisch, "Scalable Distributed Conference Control in Heterogeneous Peer-to-Peer Scenarios with SIP," in *Mobimedia '09: Proc. of the 5th International ICST Mobile Multimedia Communications Conference*. Brussels, Belgium: ICST, Sep. 2009, pp. 1–5. [Online]. Available: http://dx.doi.org/10.4108/ICST.MOBIMEDIA2009.7436

[25] "The NIST JAIN-SIP homepage," http://jain-sip.dev.java.net/, 2009.

[26] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," IETF, RFC 5245, April 2010.

[27] A. Knauf, G. Hege, T. C. Schmidt, and M. Wählisch, "A RELOAD Usage for Distributed Conference Control (DisCo)," individual, IETF Internet Draft – work in progress 00, June 2010. [Online]. Available: http://tools.ietf.org/html/draft-knauf-p2psip-disco

[28] "The PlanetLab homepage," http://planet-lab.org/, 2010.

[29] B. Abrahao and R. Kleinberg, "On the Internet Delay Space Dimensionality," in *Proc. of the 8th ACM SIGCOMM Conf. on Internet Measurement (IMC'08)*. New York, NY, USA: ACM, 2008, pp. 157–168.

[30] L. Tang and M. Crovella, "Virtual Landmarks for the Internet," in *Proc. of the 3rd ACM SIGCOMM Conf. on Internet Measurement (IMC'03)*. New York, NY, USA: ACM, 2003, pp. 143–152.

[31] "The Cooperative Association for Internet Data Analysis homepage," http://www.caida.org/home/, 2010.

[32] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays," in *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom 2003)*, vol. 2. Washington, DC, USA: IEEE Computer Society, 2003, pp. 1510–1520.

Pr2-P2PSIP: Privacy Preserving P2P Signaling for VoIP and IM

Ali Fessi, Nathan Evans, Heiko Niedermayer, Ralph Holz
Technische Universität München
Boltzmannstrasse 3
Munich, Germany
{fessi|evans|niedermayer|holz}@net.in.tum.de

ABSTRACT

In the last few years, there has been a good deal of effort put into the research and standardization of P2P-based VoIP signaling, commonly called P2PSIP. However, there has been one important issue which has not been dealt with adequately, privacy. Specifically *i)* location privacy, and *ii)* privacy of social interaction in terms of who is communicating with whom. In this paper, we present Pr^2-$P2PSIP$, a Privacy-Preserving P2PSIP signaling protocol for VoIP and IM. Our contribution is primarily a feasibility study tackling the privacy issues inherent in P2PSIP. We leverage standard security protocols as well as concepts and experiences learned from other anonymization networks such as Tor and I2P where applicable. We present the design and on-going implementation of Pr^2-P2PSIP and provide a threat analysis as well as an analysis of the overhead of adding privacy to P2PSIP networks. Particularly we analyze cryptographic overhead, signaling latency and reliability costs.

Categories and Subject Descriptors

C.2 [**Network Architecture and Design**]: Miscellaneous; K.4.1 [**Public Policy Issues**]: Privacy

General Terms

Privacy, anonymization, Peer-to-Peer(P2P), Session Initiation Protocol (SIP)

Keywords

P2P signaling, P2PSIP, location privacy, social interaction privacy, onion routing, reliability costs

1. INTRODUCTION

The Session Initiation Protocol (SIP) [30] is a protocol standardized by the IETF for setting up multimedia sessions, in particular Voice over IP (VoIP) sessions. It can also be used for Instant Messaging (IM) [29]. There has been a lot of effort in research and standardization in the last few years related to P2PSIP [6]. The concept behind P2PSIP is that the location of a SIP User Agent (UA) (IP address and port number) is published not to a SIP Registrar, but in a Distributed Hash Table (DHT). This data is stored at other peers with peer identifiers (IDs) uncorrelated to the SIP UA. These peers, called replica nodes, reply to queries from any other peer looking for the UA. This makes the UA available for incoming VoIP phone calls and chat messages. However, the SIP UA has no control over knowing which peers have asked for its current location. Curious and malicious peers can perform a lookup for the SIP URI of the UA regularly. The IP addresses of the UA could then be mapped to geographic locations [1]. Using this information, attackers could build location profiles of a user. Even worse, attackers could "crawl" the P2PSIP network and harvest location profiles of all participants. This issue has been left out-of-scope in the IETF P2PSIP working group (WG) [2]. On the other hand, location privacy had been thought of early in the GSM standardization process. Thus, it seems to be necessary to consider this privacy issue in P2PSIP networks as well.

Another privacy threat in P2PSIP is that replica peers can observe that communication is established between two SIP UAs and deduce knowledge about the social interaction of the two users.

In this paper, we tackle the two privacy issues illustrated above; the former, *location privacy* and the latter, *social interaction privacy*, by developing a new protocol which we call *Privacy-Preserving P2PSIP* (*Pr^2-P2PSIP*). The rest of this paper is organized as follows. In Section 2, we present our on-going work on the design and implementation of Pr^2-P2PSIP. Section 3 provides an evaluation of Pr^2-P2PSIP in terms of threat analyses as well as an analysis of the overhead of adding privacy to P2PSIP networks in terms of cryptographic overhead, signaling latency and reliability costs. Section 4 provides an overview of related work and Section 5 concludes our findings in this paper.

2. DESIGN OF PR^2-P2PSIP

In this section, we introduce Pr^2-P2PSIP.

2.1 Model and Notation

First, we introduce the model and notation used in the rest of the paper.

2.1.1 SIP UAs and Public Identities

The SIP UAs provide the means for users to perform their

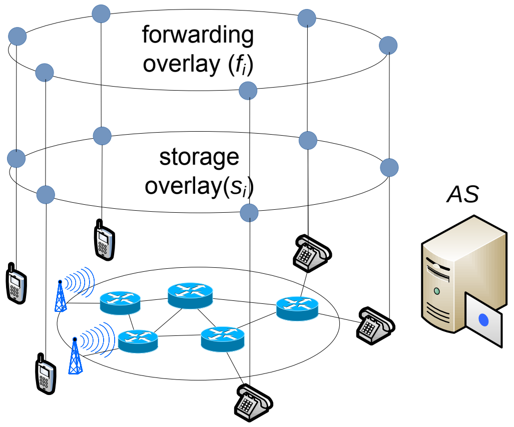


Figure 1: Architecture of Pr^2-P2PSIP

social interactions. They send chat messages and initiate phone conversations on behalf of the users. Let $\mathcal{N}$ be the set of UAs in a P2PSIP network and $n = |\mathcal{N}|$ the number of UAs. In this paper, we use capital letters, e.g., A, B or $A_i, i \in \{1, 2, ...n\}$ to denote interchangeably (unless otherwise explicitly mentioned) a user name, her SIP UA, or her SIP URI.

Note that we use the term "UA" and "peer" interchangeably.

2.1.2 Authentication Server

Pr^2-P2PSIP functions with a central authority, which is an *authentication server AS*. The AS authenticates a user A using a long-term preshared key, e.g., user password, or a high entropy key stored in the (U)SIM card of the user's smart phone. After successful authentication, the AS provides the UA with a certificate that binds the user's public key $+K_A$ to her public identity A. The AS is indispensable for Pr^2-P2PSIP as it provides verifiable identities at the application layer. This enables UAs to mutually authenticate each other and establish secure channels for encryption and integrity-protection at the application layer (SIP signaling and multimedia streams). The AS provides verifiable identities at the overlay layers as well (Pr^2-P2PSIP includes two different overlays, explained in Section 2.1.3) in order to prevent attacks on the overlays, e.g., Sybil and eclipse attacks. Another attack that would be possible without a central authority would be the so-called *chosen-location attack* where malicious peers choose a convenient peer ID where they could, eclipse (hide) other peers, or eclipse the content they would be responsible for. In the context of privacy, chosen-location attacks would allow malicious peers to choose a strategically "good" position where they could monitor the activities of certain other peers.

2.1.3 Storage and Forwarding Overlays

In addition to its public identity, a UA A_i has two *pseudonyms* f_i and s_i which it uses for participating in two different overlays as sketched in Figure 1. $s_i, i = 1, \ldots, n$ is the *storage* overlay. $f_i, i = 1, \ldots, n$ is the *forwarding* overlay.

Storage.

Storage is the common service that DHT's provide. The DHT stores information required to contact other UAs for sending them application layer signaling messages. However, the information stored in the Pr^2-P2PSIP DHT differs from P2PSIP. Specifically, it does not reveal the actual location of UAs. The content of this information is explained in Sections 2.2.2 and 2.3.2.

Table 1: Notation

$+K_e$	Public key of an entity e
$-K_e$	Private key of an entity e
$K_{a,b}$	Shared secret key between entities a and b
$\{m\}_{K_{a,b}}$	Message m encrypted and integrity-protected with the symmetric key $K_{a,b}$ (See Section 2.4).
$\{m\}_{+K_e}$	Message m encrypted with the public key of entity e.
$l(e,t)$	Location (IP address and port number) of the entity e at a certain point of time t
$L(A,t)$	Data stored in P2P network required to reach UA A at a certain point of time t

Forwarding.

Forwarding is an additional function that peers need to perform in Pr^2-P2PSIP. It differs from typical forwarding in DHT algorithms with recursive routing, e.g., Chord or Pastry, given that these DHT algorithms were not designed with privacy in mind. Message forwarding in Pr^2-P2PSIP is explained in 2.2.1.

Overlay Algorithm.

We currently use Kademlia [20] as our DHT overlay algorithm. However, Pr^2-P2PSIP could be used with other DHTs. We do not claim that the choice of the overlay algorithm is orthogonal to the impact of Pr^2-P2PSIP on user privacy. Thus, this design decision requires further investigation in future work. For this paper, we use the Kademlia RPCs FIND_NODE, FIND_VALUE, PING and STORE in the storage overlay. Since the forwarding overlay is used only for finding other peers (i.e., no data stored in the DHT, see Section 2.2.1 for details), the forwarding overlay makes use only of the FIND_NODE and PING RPCs.

Pseudonyms in the Storage and Forwarding Overlays.

The pseudonyms f_i and s_i are temporal identities which are unlinkable to the UA's public identity A_i (we use non-capital letters to denote pseudonyms). Pseudonyms f_i and s_i belong to an identifier space $\mathcal{K}$, e.g. $\mathcal{K} = \{0, \ldots, 2^{160} - 1\}$. Each pseudonym is linked to a public key as well: $(f_i, +K_{f_i})$, $(s_i, +K_{s_i})$. As such, a UA uses different public/private key pairs for different purposes.

By "UA A_i", we mean the UA with public identity A_i while "UA f_i" or "UA s_i" is the UA with pseudonym f_i or s_i respectively. Table 1 provides additional notations used throughout this paper.

2.1.4 Threat Model

Given a UA $A \in \mathcal{N}$, we assume that an attacker M wants to collect as much information as possible about A, in particular:

1. its current locator $l(A,t)$

2. its location profile: a history of $l(A,t)$

3. a social interaction profile: a history of social interactions $A \rightarrow B$ or $B \rightarrow A$ for any $B \in \mathcal{N}$.

Note that man-in-the-middle, eavesdropping and message forgery attacks on the application data (chat messages and phone conversations) can be successfully countered (unless the AS turns malicious) using the UA's certificates provided by the AS. Note also that the AS guarantees that each UA receives a single pseudonym f_i and a single pseudonym s_i, so Sybil attacks can be excluded and eclipse attacks are difficult (since the overlay routing algorithm provides multiple disjoint paths between two arbitrary peers).

We consider the following attackers in Pr^2-P2PSIP:

1. a single malicious UA participating in the Pr^2-P2PSIP network: $M \in N$. In this case, we assume every UA operates on its own. Different malicious UAs do not exchange information for the sake of breaking other users' privacy. Thus, each UA can observe only the messages it sends and it receives. Additionally, if it forwards a message from one peer to another, it can decrypt only the messages (or message parts) for which it has the appropriate key.

2. a *partial* observer in the network underlay observing that communication is taking place between different IP addresses. The attacker may be able to observe some traffic and deduce some conclusions about the location or social interaction of some UAs.

2.2 Protocol Overview

In this section we describe how Pr^2-P2PSIP handles data storage and message forwarding. Storage and forwarding in the Pr^2-P2PSIP network differ from a "regular" P2PSIP network, because UAs seek to keep their location and social interaction private.

2.2.1 *Message Forwarding*

An application layer message (e.g., SIP MESSAGE for IM or SIP INVITE for establishing a phone call) from a UA A to a UA B is sent via intermediate forwarding peers using so-called *onion routing* [15]. In onion routing, the sender of a message m chooses intermediate forwarding peers which route the message to B on behalf of A. A orders these peers in series and encrypts m several times recursively. One layer of encryption is removed at each of the forwarding peers, so that the final peer in the tunnel has the original unencrypted message.

In Pr^2-P2PSIP, peers establish *inbound tunnels* and *outbound tunnels* (see Figure 2). The choice of tunnel length has some effects on privacy which are discussed in detail in Section 3. For illustration purposes, we consider a tunnel length of three hops throughout Section 2.

A UA A uses its pseudonym ($f_{O_0} = f_{I_0}$ in Figure 2) to communicate with the first hop of each tunnel. For outbound tunnels, A (sending application layer messages) generates symmetric keys for protected communication (i.e. encrypted and integrity protected) with each of the outbound forwarding peers (f_{O_1}, f_{O_2} and f_{O_3}). For inbound tunnels, A (receiving application layer messages) generates symmetric keys for protected communication with each of the inbound forwarding peers f_{I_1}, f_{I_2} and f_{I_3}. In both cases, A uses the public keys of the forwarding peers to distribute the required symmetric keys which will be used during the tunnel lifetime. Additionally, the forwarding peers establish TLS sessions for hop-by-hop security. Figure 3 sketches the resulting encryption and integrity-protection layers. The layered encryption ensures that the message looks different for each hop.

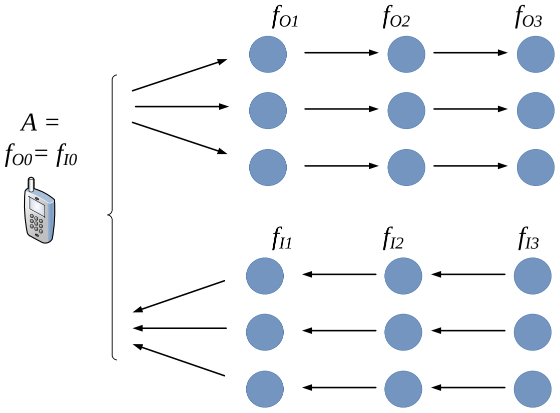


Figure 2: Inbound and outbound tunnels of sender/receiver A

While the end-to-middle symmetric keys are valid only for the tunnel lifetime, a hop-by-hop TLS session may be multiplexed for several inbound and outbound tunnels serving several sender/receiver peers and can be long-lasting. This design decision is borrowed from Tor and should make traffic analysis more difficult. Unlike Tor where all peers are connected in a full mesh and establish TLS tunnels to each other, Pr^2-P2PSIP TLS tunnels are established on demand, since otherwise Pr^2-P2PSIP could not scale to more than few thousand peers.

Forwarding Pool.

To discover forwarding peers, peers query the forwarding overlay. Additionally, each peer keeps a local pool of the forwarding peers it has learned about, and which it can ask to be a part of its tunnels. This pool should be kept up-to-date, so a peer can refresh its inbound or outbound tunnels.

The peer will occasionally learn about other forwarding peers as a side effect of overlay maintenance. However, it is crucial for the privacy goals of Pr^2-P2PSIP to not rely solely on overlay maintenance for re-filling its forwarding pool and not to simply choose peers from its overlay routing table. Instead, a UA A should perform node lookups (a FIND_NODE RPC in Kademlia) for *random* identifiers in the forwarding overlay when it needs to update its forwarding pool, in order to prohibit an attacker M from being able to force A to select her (M) as a forwarding peer in her tunnels (i.e., path selection attack; see Section 3.1).

2.2.2 *Contact Data Storage*

The contact data of all UAs are stored in a DHT. For each UA A there exists a value stored in the DHT with the contact data of A under the key $h(A)$. The contact data is a tuple $(+K_A, L(A,t))$. $L(A,t)$ does not reveal any information about A's real location $l(A,t)$. Instead, $L(A,t)$ includes information about the entry points of A's inbound tunnels (i.e., the forwarding peers furthest from A in her

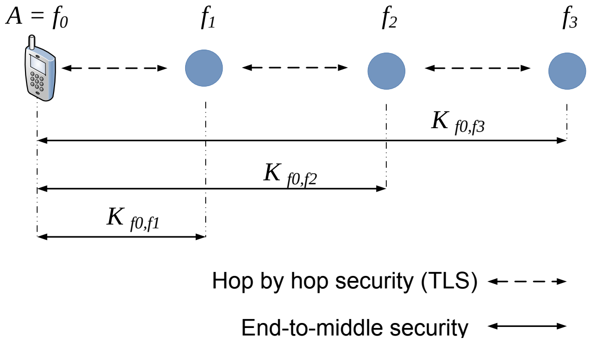


Figure 3: End-to-middle and hop-by-hop encryption and data-integrity layers in Pr^2-P2PSIP

inbound tunnels), which will forward incoming messages towards A. Details on the structure of $L(A, t)$ are provided in Section 2.3.2.

2.3 Protocol Operations

In this section we provide more low level details on the protocol operations of Pr^2-P2PSIP.

2.3.1 Tunnel Setup

Up to this point we have differentiated between inbound and outbound tunnels. However, the procedure for setting up both kinds of tunnels and the per-hop state required for them is the same. A forwarding peer can be unaware of the type of tunnel it is participating in. This reduces the complexity of Pr^2-P2PSIP.

In fact, in both cases communication takes place in both directions, for instance to acknowledge tunnel setup and to tunnel RPC responses backwards to the initiator of a RPC (this is the case for publishing data in the DHT; see Section 2.3.2; and retrieving data from the DHT; see Section 2.3.3).

Forwarding peers need to store state information that is required to process incoming and outgoing messages for each tunnel. Let A be the UA which initiates the tunnel setup for sending or receiving application layer messages. Let f_1, f_2 and f_3 be the forwarding peers chosen by A to build the tunnel (as in Figure 3). A uses its pseudonym f_0 to communicate with the first hop in the tunnel, f_1. The state stored at each forwarding peer $f_i, i = 1, 2, 3$, called the *tunnel binding* in Pr^2-P2PSIP, is a tuple which consists of the following data:

- *tunnel ID*: a tunnel ID α used for multiplexing between different tunnels,
- *successor* and *predecessor*: the pseudonyms, public keys and locations of the successor and the predecessor peers in the tunnel: $(f_{i+1}, +K_{f_{i+1}}, l(f_{i+1}, t))$ and $(f_{i-1}, +K_{f_{i-1}}, l(f_{i-1}, t))$,
- *end-to-middle symmetric key*: K_{f_0,f_i}.

This data is distributed by A during the tunnel setup. Furthermore, f_{i+1} and f_{i-1} are used at each forwarding peer locally to determine whether it has already established TLS sessions with the successor and predecessor peers.

The data for the tunnel binding is sent by A onion-encrypted along the tunnel. For each node $f_i, i = 1, 2, 3$, A sends (indirectly) a message:

$$\begin{aligned} m_i = \quad & (\alpha, \\ & f_{i+1}, +K_{f_{i+1}}, l(f_{i+1}, t), \\ & f_{i-1}, +K_{f_{i-1}}, l(f_{i-1}, t), \\ & K_{f_0,f_i}) \end{aligned} \tag{1}$$

For f_3, the information about the successor is marked with *null* values:

$$\begin{aligned} m_3 = \quad & (\alpha, \\ & null, null, null, \\ & f_2, +K_{f_2}, l(f_2, t), \\ & K_{f_0,f_3}) \end{aligned} \tag{2}$$

Of course, this has the consequence that f_3 can deduce that it is the last hop in the tunnel. The impact of this information available to f_3 will be discussed in Section 3.1. The message flow for setting up the tunnel initiated by A looks as follows.

$$\begin{aligned} f_0 \leftrightarrow f_1 \quad &: \quad TLS \quad handshake \\ f_0 \rightarrow f_1 \quad &: \quad \{m_1, \{m_2, \{m_3\}_{+K_{f_3}}\}_{+K_{f_2}}\}_{+K_{f_1}} \\ f_1 \leftrightarrow f_2 \quad &: \quad TLS \quad handshake \\ f_1 \rightarrow f_2 \quad &: \quad \{m_2, \{m_3\}_{+K_{f_3}}\}_{+K_{f_2}} \\ f_2 \leftrightarrow f_3 \quad &: \quad TLS \quad handshake \\ f_2 \rightarrow f_3 \quad &: \quad \{m_3\}_{+K_{f_3}} \end{aligned} \tag{3}$$

The TLS handshakes take place only if two successive forwarding peers have not yet established a TLS session. After tunnel setup, A (i.e., f_0) can exchange messages with f_3 without revealing her location $l(A, t)$ or her identity (neither the public identity A, nor her pseudonym f_0). f_3 knows only the information about f_2.

A message m from A to f_3 is forwarded as follows:

$$\begin{aligned} f_0 \rightarrow f_1 \quad &: \quad \{\alpha, \{\alpha, \{\alpha, m\}_{K_{f_0,f_3}}\}_{K_{f_0,f_2}}\}_{K_{f_0,f_1}} \\ f_1 \rightarrow f_2 \quad &: \quad \{\alpha, \{\alpha, m\}_{K_{f_0,f_3}}\}_{K_{f_0,f_2}} \\ f_2 \rightarrow f_3 \quad &: \quad \{\alpha, m\}_{K_{f_0,f_3}} \end{aligned} \tag{4}$$

while a message from f_3 to A is forwarded as follows:

$$\begin{aligned} f_3 \rightarrow f_2 \quad &: \quad \alpha, \{m\}_{K_{f_0,f_3}} \\ f_2 \rightarrow f_1 \quad &: \quad \alpha, \{\{m\}_{K_{f_0,f_3}}\}_{K_{f_0,f_2}} \\ f_1 \rightarrow f_2 \quad &: \quad \alpha, \{\{\{m\}_{K_{f_0,f_3}}\}_{K_{f_0,f_2}}\}_{K_{f_0,f_1}} \end{aligned} \tag{5}$$

The tunnel setup (message flow (3)) is acknowledged by the last forwarding peer f_3. Thus, the acknowledgement message is the first message sent from f_3 to A via f_2 and f_1. Note that the acknowledgement of the tunnel setup by f_3 is crucial for the reliability of Pr^2-P2PSIP. This will be discussed in detail in Section 3.2.

2.3.2 Publishing UA Contact Data

Publishing the contact data of a UA in the DHT makes use of outbound tunnels and the Kademlia STORE RPC. A UA A publishes its application layer public key ($+K_A$) as well as the pseudonyms, the public keys and the locations of the entry points of its inbound tunnels. For example, assume A has three parallel inbound tunnels. Then, the value stored

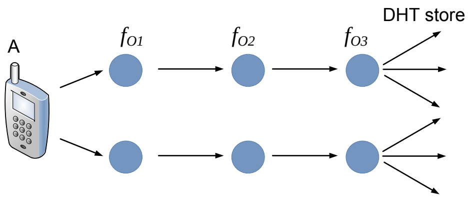


Figure 4: Publishing UA contact data in the DHT

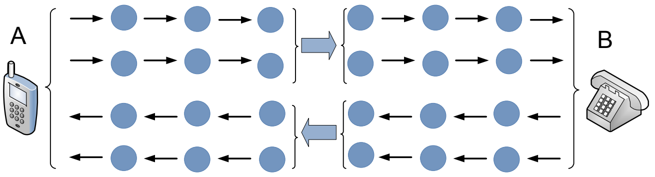


Figure 5: Bidirectional signaling in Pr^2-P2PSIP

in the DHT under the key $h(A)$ is a tuple $(+K_A, L(A,t))$ where

$$\begin{aligned} L(A,t) \;\; = \;\; & (f_{I_3}, +K_{f_{I_3}}, l(f_{I_3},t), \alpha), \\ & (f'_{I_3}, +K_{f'_{I_3}}, l(f'_{I_3},t), \beta), \\ & (f''_{I_3}, +K_{f''_{I_3}}, l(f''_{I_3},t), \gamma) \qquad (6) \end{aligned}$$

where f_{I_3}, f'_{I_3} and f''_{I_3} are the entry points of the different inbound tunnels; and α, β and γ the respective tunnel IDs. The STORE RPC request is sent from A to f_{O_3} using message flow (4). This is depicted in Figure 4. It is crucial that the STORE RPC responses received by f_{O_3} are forwarded back to A (using message flow (5)). The reason for this is that A can not be sure that all peers in the outbound tunnel (f_{O_1}, f_{O_2} and f_{O_3}) are still online since the tunnel has been established or refreshed. If A does not receive a response to her STORE request from f_{O_3}, she needs to re-initiate the RPC using another outbound tunnel. The time interval between two successive RPC requests is a trade off between latency and signaling overhead. In the extreme case, A could send STORE RPCs simultaneously along several outbound tunnels. However, this parallelism may produce a large unnecessary signaling overhead depending on the stability of the network (and thus, the stability of the outbound tunnels). As a trade off, we use an aggressive timeout of $1s$ before the next outbound tunnel is invoked.

2.3.3 *Retrieving Contact Data*

Looking up data in the DHT is quite similar to publishing data in the DHT except the Kademlia RPC used is FIND_VALUE. A uses one of her outbound tunnels and asks the last peer in the tunnel to lookup the data on behalf of her. The same procedure with timeouts is performed if no response is received from an outbound tunnel.

Using the same procedure for publishing and retrieving data in/from the DHT reduces the complexity of the protocol.

2.3.4 *Bidirectional Signaling*

Once A has found the entry points of the inbound tunnels of B, she can use her outbound tunnels to send application layer messages to B. A may include her real location $l(A,t)$ (encrypted with $+K_B$) in the first signaling message to B or $L(A,t)$ if she does not want to reveal her location to B. The same holds for the response of B to A. Every SIP message is acknowledged end-to-end, i.e., if B receives a message from A through one of his inbound tunnels, he sends an acknowledgement through one of his outbound tunnels.

The same procedure with timeouts applies here as well: if A sends a SIP message to B and the acknowledgement does not reach A within $1s$ another end-to-end path, i.e., another combination of an outbound tunnel of A and an inbound tunnel of B is used. At this point, it is worth it mentioning that Pr^2-P2PSIP is designed with signaling in mind and is not optimized for real-time communication. The main problem with real-time communication is the accumulated one-way-delay in both directions between A and B, given that there are four to six hops between A and B (depending on the tunnel length).

2.4 Cryptographic Primitives

In this section, we provide implementation details on the cryptographic primitives used in Pr^2-P2PSIP.

Symmetric Cryptography.

As mentioned in Table 1, $\{m\}_{K_{a,b}}$ is a message m encrypted and integrity-protected with the shared key $K_{a,b}$. This is used to provide end-to-middle security in the inbound and outbound tunnels (see Figure 3). However, it is well known that different keys should be used for different purposes and for each direction [13]. Thus, four symmetric keys are derived from $K_{a,b}$ on both sides using a cryptographic key expansion function. These keys are derived at the tunnel setup and used during the tunnel lifetime.

Public Key Cryptography.

Given that Pr^2-P2PSIP makes extensive use of public key encryption, in particular for inbound and outbound tunnel setup, it is crucial to optimize the use of the public key cryptographic primitives. We use two solutions for this purpose:

- a message m from a to b encrypted with the public key $+K_b$ is actually encrypted with a temporary symmetric key $K_{a,b}$ generated by a. Then, $\{m\}_{K_{a,b}}$ is sent together with the temporary key $K_{a,b}$ encrypted with $+K_b$. Thus, $\{m\}_{+K_b}$ is actually implemented as $(\{K_{a,b}\}_{+K_b}, \{m\}_{K_{a,b}})$.
- an important design decision in Pr^2-P2PSIP is to use Elliptic Curve Cryptography (ECC) [31] instead of RSA for public key encryption. The reason is the convenient key length without necessarily sacrificing performance. An ECC key length of 194 bits provides comparable entropy to a 2054 bit RSA key[1].

The impact of the design decisions on the cryptographic primitives are further discussed in Section 3.3.

2.4.1 *Pitfalls*

In this section, we explain a few details that need to be taken into account when implementing Pr^2-P2PSIP. These details were skipped in the previous sections for the sake of simplicity.

[1] The choice of the private key for RSA is limited by the choice of prime numbers, while any random number can be used as a private key for ECC.

Outbound tunnels used by A for publishing $L(A, t)$ should not be used for other purposes, e.g., retrieving contact data of another UA B. The last hop in the outbound tunnel of A, f_{O_3} sees only the hash value of A when the data is stored in the DHT. However, if f_{O_3} has a list of user names, it can determine whether A is one of them. If the same outbound tunnel is used for retrieving the contact data of B, f_{O_3} can deduce that A is about to send a SIP message to B. Thus, the social interaction privacy of A would be broken.

In the description of the tunnel setup in Section 2.3.1, the tunnel ID α remains constant along the tunnel. However, this raises a privacy threat especially for inbound tunnels. Intermediate hops (f_{I_1}, f_{I_2} and f_{I_3}) are all aware of the tunnel ID α published in the contact data of A in the DHT: $L(A, t)$. Thus, by crawling the DHT, f_{I_1} can discover which UA A has published its contact data $L(A, t)$ with α as tunnel ID, and can deduce the public identity of A. Since f_{I_1} has direct IP communication with A, the location privacy of A is broken. In order to defeat this attack, the tunnel ID has to be changed at each hop. Thus, each forwarding peer has two different tunnel IDs, one shared with the predecessor and another one shared with the successor. Since A needs to know the final tunnel ID at f_{I_3} in order to publish its contact data $L(A, t)$ in the DHT, f_{I_3} informs A about the tunnel ID to be published when it confirms the tunnel setup to A. Since f_{I_3} and A use end-to-middle encryption to secure their communication, f_{I_1} and f_{I_2} can not deduce which tunnel ID is published in the DHT.

3. EVALUATING PR²-P2PSIP

3.1 Threat Analysis

In this section, we evaluate whether Pr^2-P2PSIP fulfills its goals, i.e., whether it can thwart attacks on location privacy and social interaction privacy. Additionally, based on an extensive threat analysis, we deduce appropriate recommendations for the tunnel length.

The threat analysis of Pr^2-P2PSIP benefits from attacks on anonymization networks that have been described in the literature. Therefore, we provide an overview of those attacks that are relevant to Pr^2-P2PSIP first. We then evaluate whether these attacks can be applied to Pr^2-P2PSIP and if Pr^2-P2PSIP introduces new attack vectors.

3.1.1 Attacks on Anonymization Networks

Attacks on anonymization networks can be classified into *passive* and *active* attacks. Passive attacks are attacks where the attacker monitors communication between other peers. For this purpose, the attacker may try to become part of one of the victims tunnels. However, in passive attacks, attackers do not alter the data they observe or forward. In contrast to passive attacks, active attacks involve a participant actively altering or injecting data in the network. Nevertheless, an attacker may combine passive and active attacks in order to reach his malicious goals. As with all privacy preserving networks, a trade off exists between usability and security.

Traffic Analysis.

Traffic analysis is a general term referring to monitoring data as it passes through a network to glean useful information. In an onion routing network over the Internet this typically means monitoring underlying network communications or data handled by a participant in the network overlay. A subset of traffic analysis called *timing analysis* measures *when* data enters or exits the network or nodes in the network. All of the attacks described herein utilize some form of traffic analysis. As discussed in [3, 33] an attacker that is able to observe both ends of a tunnel may be able to correlate that two peers (identified by IP addresses) are communicating by analyzing inbound and outbound packet counts between every two peers. This attack is depicted in Figure 6. However, the attacker can not be sure that the two peers are communicating, since they could simply be forwarding data for other peers.

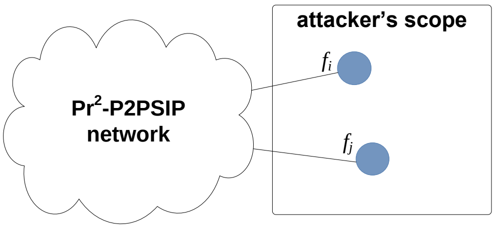


Figure 6: Passive attacks on Pr^2-P2PSIP

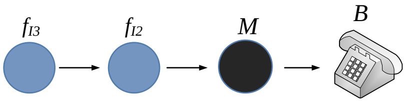


Figure 7: Path selection attacks on Pr^2-P2PSIP

Path Selection Attacks.

Another type of passive attack is the path selection attack [5]. The attacker forces particular peers to be chosen for a tunnel, preferably controlled by the attacker. Since we assume peers do not collude in Pr^2-P2PSIP, this attack is useful only if the attacker is on an end of the tunnel directly connected to the victim as in Figure 7. Given that peers choose forwarding peers using *random* identifiers in the forwarding overlay, the probability of a successful path selection attack when a peer builds its inbound tunnels is inversely proportional to the size of the network. However, given that a peer occasionally has to change the peers in its inbound tunnel, the probability of a successful path selection attack grows over time.

Most other passive attacks [3, 9] require a *global* passive adversary, outside of the threat model for our work.

Congestion Attacks.

The congestion [23] or circuit clogging [21] attack combines typical traffic and timing analysis with an active denial or reduction of service attack. The basic layout of this attack is depicted in Figure 8. In this type of attack, a malicious peer initiates a “legitimate” communication with the victim. Using this communication, she alternates between periods of sending data and being silent on the tunnel. She concurrently builds tunnels between all (or some subset of) possible other peers in the network and sends probe traffic down each. If she can correlate the sending periods on the legitimate tunnel with traffic on the probe tunnels she has discovered that some peers on the probe tunnel are also part

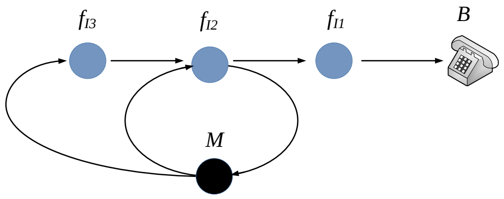


Figure 8: Congestion attacks on Pr^2-P2PSIP

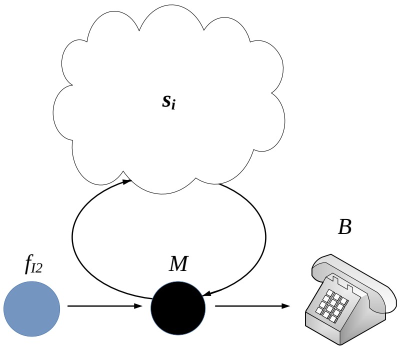


Figure 9: Attacks on two-hop inbound tunnels

of the legitimate one. This method works if forwarding peers have to split resources equally between their tunnels; utilizing one tunnel therefore alters the latency properties of the other tunnels. By building repeated probe tunnels through different sets of possible peers she can eventually determine exactly which peers are being used. Provided that the peers on the legitimate tunnel are rotated over time (as is the case in Pr^2-P2PSIP) and the victim will be the only peer which will be always part of the tunnels, the attacker could discover the actual IP address of the victim.

3.1.2 Attacks on Pr^2-P2PSIP

In this section, we provide a security threat analysis of Pr^2-P2PSIP on inbound and outbound tunnels for different tunnel lengths.

Attacks on One-hop Inbound Tunnels.

Using one-hop inbound tunnels, the only inbound forwarding peer and potentially malicious peer $M = f_{I_1}$ is directly connected to the victim B. The contact data of B published in the DHT points to f_{I_1}. Thus, by crawling the DHT (i.e. the storage overlay), f_{I_1} can find out which UAs have published their contact information with f_{I1} as a tunnel entry point. f_{I1} might be the tunnel entry point for several peers, let' say B, B' and B''. After collecting the data $\{L(B,t), L(B',t), L(B'',t)\}$ from the DHT, f_{I1} can correlate the tunnel IDs in $L(B,t)$, $L(B',t)$ and $L(B'',t)$ with the tunnel bindings it has previously setup and can unambiguously deduce the location of B, B' and B''.

Attacks on Two-hop Inbound Tunnels.

Using two-hop inbound tunnels, as shown in Figure 9, a similar attack remains possible. A malicious peer M can trivially recognize from communication with the successor and the predecessor in the tunnel that she is not the entry point of the tunnel. Thus, M can deduce its position in the tunnel and that its predecessor is the initiator of the tunnel (B) and its successor is the entry point of the tunnel (f_{I_2} in Figure 9). By crawling the content of the DHT, M can find out which UAs have published their contact information with f_{I_2} as a tunnel entry point, again let' say B, B' and B''. The difference to the one-hop case is that M can not necessarily identify which one of these peers is the initiator of the tunnel she is part of. This is because the tunnel ID is not constant along the tunnel. Nevertheless, M could significantly reduce the number of possible public identity of the tunnel initiator, potentially to one. This would lead to an unambiguous link between the public identity of B and his current location $l(B,t)$.

Depending on the size of the network, B may have changed its inbound tunnels while M is still crawling the DHT, and the data M is looking for in the DHT may become unavailable. However, we can not rely on this assumption, if M has sufficient resources.

One possible approach to reduce the probability of this attack could be the concept of *entry guards* [25], which were suggested for thwarting attacks on discovering the origin of *hidden services* in Tor. These attacks are based on path selection attacks. The concept of entry guards is as follows; instead of choosing uniformly at random from the set of all peers for the crucial hop (the nearest to the hidden server in Tor, the nearest to the UA in the inbound tunnel in Pr^2-P2PSIP, i.e., f_{I_1})), a small set of peers are chosen initially and one of these is always utilized in that position. Choosing forwarding peers uniformly at random gives a patient attacker the chance to be chosen as the crucial hop with a high probability if B rotates his tunnels regularly, whereas the probability of choosing the attacker with "final guardians" is only g/n where g is the total number of guardian nodes used (and n the overall number of peers as mentioned in Section 2.1).

Nonetheless, since malicious peers have the chance here to discover the public identity of B and its location with an effort estimated by $O(n)$ (crawling the DHT), we consider the attack on one-hop and two-hop inbound tunnels as a real threat to Pr^2-P2PSIP.

Attacks on Three-hop Inbound Tunnels.

Using three-hop inbound tunnels, a possible attack scenario is a variant of the circuit clogging attack, where the participants of a tunnel can be deduced. In this scenario the attacker M initiates a communication with the victim B (Figure 8). M wants to discover the IP address of B. To do so, she actively builds tunnels through many peers which she uses to send a steady stream of data to herself. She then sends a certain pattern to B (for example, via chat), which can be detected on the tunnels that she is monitoring because of interference [21, 23, 28]. Since M may not necessarily obey to the agreed inbound tunnel length in the network, she could conceivably connect to every peer with a one hop tunnel back to herself and send the pattern to B (via his legitimate inbound tunnel). If the pattern is detected, this reveals either B or a part of his tunnel. By repeating the same procedure for each of B's multiple inbound tunnels, M can eliminate B's tunneling peers, because B will be the only peer present on *each* of the inbound tunnels used.

This attack becomes more difficult as the number of peers in the network increases, because the attacker needs to monitor them all for the pattern she is sending. False positives or false negatives may occur due to other traffic in the network at the same time as the attacker's probe or pattern traffic. The attack may also take a prohibitively long amount of time to mount; if the attacker cannot monitor all nodes in the network at once, she will need to perform this attack by monitoring only some subset of the network at a time.

General Attacks on Outbound Tunnels.

No matter how long the outbound tunnel is, the last hop in the tunnel (furthest from A) which is used for publishing the contact data of A in the DHT should not be used for other purposes as mentioned in Section 2.4.1. Otherwise, the social interaction privacy of A would be broken.

Attacks on One-hop Outbound Tunnels.

If the outbound tunnel of a UA A consists of one hop only, when A publishes her contact data in the DHT, the outbound forwarding peer f_{O_1} receives the STORE RPC from A directly, and thus, can trivially discover the public identity of A and correlate it with her IP address. This would break the location privacy of A.

Attacks on Two-hop Outbound Tunnels.

Attacks on two-hop outbound tunnels become more difficult. The last peer in the outbound tunnel f_{O_2} may misuse the property of Pr^2-P2PSIP that communication in both inbound and outbound tunnels takes place in both directions, and send certain traffic patterns to f_{O_1} which are forwarded to A and thus may be the basis for a congestion attack.

Conclusions.

Given the threat analysis above, we conclude that:

- Passive attacks are of limited use because while they may reveal that two peers are participating in the network and connected, this does not indicate whether the peers are forwarding data for other peers or actually communicating.

- Path selection attacks require that the attacker be chosen as the victim nodes final inbound hop. The probability of the success of such an attack is inversely proportional to the size of the network. Though it increases over the time by changing the tunnel. Unless entry guards are chosen as crucial hop.

- Congestion attacks may be feasible, but at high cost, take a long time and are susceptible to false positives and false negatives.

- A tunnel length of *three hops* for *inbound tunnels* and *two hops* for *outbound tunnels* provide location and social interaction privacy at a high and satisfactory degree.

3.2 Reliability Cost Analysis

In this section, we provide a model of Pr^2-P2PSIP based on reliability theory [26]. This model will then be used for estimating the overhead generated by adding privacy to P2PSIP. First, we start with some basic knowledge in reliability theory from [26] which is required to understand the model.

3.2.1 Reliability Theory

Reliability theory provides tools for estimating the reliability of a whole system by estimating the reliability of the single units/components of the system. Let T be the *time to failure* of a unit, i.e., the time elapsed between when the unit is put into operation until it fails for the first time. T can be assumed to be continuously distributed with a density function $f(t)$ and distribution function:

$$F(t) = Pr(T \leq t) = \int_0^t f(u)\,du \tag{7}$$

The reliability $R(t)$ is the probability that the unit will be still operating at time t:

$$R(t) = 1 - F(t) = Pr(T \geq t) \tag{8}$$

A structure of units is *series* if the operation of the structure depends on the operation of all units in this structure. A *parallel* structure is a structure which operation requires at least one of the units operating.

Let a structure consisting of k units with independent failures[2] and equal reliabilities $R_i(t) = R(t)$ for all units $i = 1, \dots, k$. If the structure is series, the reliability of the structure is

$$R_\wedge(t) = R_1(t)R_2(t)\dots R_k(t) = R^k(t) \tag{9}$$

If the structure is parallel, the reliability of the structure is

$$\begin{aligned} R_\vee(t) &= 1-(1-R_1(t))(1-R_2(t))\dots(1-R_k(t)) \\ &= 1-(1-R(t))^k \end{aligned} \tag{10}$$

3.2.2 Modeling Pr^2-P2PSIP Networks with Reliability Theory

A Pr^2-P2PSIP (or P2PSIP) network is a system which consists of multiple units, which are the peers. The time to failure of a peer is the time interval between the time when the peer goes online until it leaves the network, i.e., T is the peer lifetime. Different studies of P2P networks for file sharing, in particular KAD [35] and for VoIP, in particular Skype [16] have shown that the peer lifetime is heavy-tailed distributed. Since it is difficult to estimate appropriate parameters for a P2PSIP network, we focus on a generic analytical model first. Note that Skype is not necessarily a good representative since Skype clients are mainly installed on PCs/laptops. Skype shows a high number of peers during working days and middays, while peers in a P2PSIP network could be running, e.g., on some fixed hardphones which are permanently online, or on mobile smart phones, which may change their IP addresses more frequently than laptops. Nevertheless, Skype is the most similar application to P2PSIP and Pr^2-P2PSIP and the study in [16] will help us to interpret the results of our reliability costs analysis as shown below.

Reliability Model of Pr^2-P2PSIP.

A UA B refreshes its contact data in the DHT as well as its inbound tunnels periodically with a refreshing period e.g., $\tau = 20mn$, in order to make sure it remains reachable in the Pr^2-P2PSIP network with high probability. This high probability is a target reliability, e.g., $\bar{R} = 1 - 10^{-5}$.

When B performs a refresh operation at $t = k\tau, k \in \mathbb{N}$, it receives acknowledgement messages for both the storage

[2] which is a dominant assumption in reliability theory

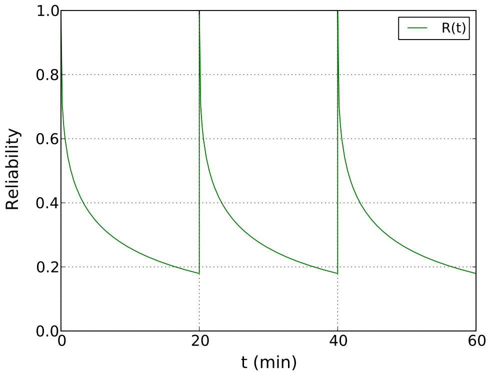


Figure 10: Example reliability of a single storage unit s_i or inbound forwarding unit f_i with periodic refreshes. $\tau = 20mn$.

and the tunnel refresh/setup (as described in Sections 2.3.1 and 2.3.2). Thus, we assume the probability that a peer/unit, either involved in the storage of the contacts of B or involved in one of the inbound tunnels for B, is online at $t = k\tau$ is 1. Then, this probability decreases over the time to the minimum value. An example of this behavior is shown in Figure 10. We denote by μ the minimum reliability of a peer at the end of each refreshing period.

$$\mu = \liminf_{t \to (k+1)\tau} R(t) \quad k \in \mathbb{N} \tag{11}$$

μ could be estimated autonomously by B through measurements. It is the probability that if another UA is observed online at t, the UA will remain online until $(t + \tau)$. μ can be considered as a metric for the churn in the network. If the measured value for μ is too low, then the UA may have to decrease τ, and thus increasing μ.

Furthermore, the following assumptions are required for our reliability analysis:

- We assume that all peers are cooperative, i.e., as long as a peer is online, it will perform requests from other peers to create tunnels, forward messages and store data.
- We assume that peers/UAs leave and join the network independently. A UA which leaves the network deletes all contact data and tunnel bindings of other peers.
- We assume a DHT model like in KAD [35] where peers which publish data are responsible for refreshing this data themselves, i.e., replica nodes do not re-publish data among each other, in particular when some of them leave the network, or new nodes close to the key of the data enter the network.
- We assume that routing in the DHT always succeeds. In particular if A is looking for the contact data of B and there is at least one replica node s_i storing this data, then A will be able to reach s_i and find the contact data of B.

Figure 11 shows the resulting reliability model under these assumptions. A UA A calling B needs to reach at least one of the storage peers s_i which have stored the contact data of B. Then, A needs to find at least one inbound tunnel to B where all peers which build the tunnel are still online. As shown in Figure 11, let m be the number of storage peers, p the length of B's inbound tunnels and q the number of parallel inbound tunnel.

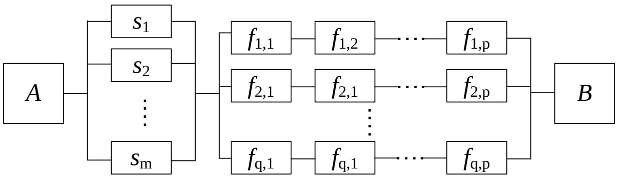


Figure 11: Reliability model of Pr^2-P2PSIP

Estimating the Overhead of Privacy.

If $p = 0$, then we have a regular P2PSIP network. Let m_0 the number of required parallel storage peers, then it follows from equation (10):

$$1 - (1 - \mu)^{m_0} \geq \bar{R} \tag{12}$$

Thus, the number of required storage peers for an inbound tunnel length $p = 0$ can be estimated by:

$$m_0 \geq \frac{ln(1 - \bar{R})}{ln(1 - \mu)} \tag{13}$$

If $p \geq 1$, then the reliability of the storage part at the end of each refreshing period can be estimated as:

$$(1 - (1 - \mu)^m) \tag{14}$$

and the reliability of the inbound forwarding part:

$$(1 - (1 - \mu^p)^q) \tag{15}$$

Let $\bar{R}_s$ the target reliablity of the storage part and $\bar{R}_f$ the target reliability of the inbound forwarding part. Thus, m and q can be estimated as follows:

$$m \geq \frac{ln(1 - \bar{R}_s)}{ln(1 - \mu)} \tag{16}$$

$$q \geq \frac{ln(1 - \bar{R}_f)}{ln(1 - \mu^p)} \tag{17}$$

and the reliability of the whole system:

$$(1 - (1 - \mu)^m).(1 - (1 - \mu^p)^q) \geq \bar{R}_s\bar{R}_f = \bar{R} \tag{18}$$

As it can be seen in Figure 11, the overall number of peers required for each UA in order to be reachable is $(m + pq)$.

By varying the ratio $\bar{R}_s/\bar{R}_f$ for a constant system target reliability $\bar{R} = 1 - 10^{-5}$ we obtain different values for $(m + pq)$ which are slightly better than equal target reliabilities for both parts, i.e., $\bar{R}_s/\bar{R}_f = 1$. Thus, we determine numerically the optimum value of $(m+pq)$ by varying $\bar{R}_s/\bar{R}_f$ for different values $p \in \{0, 1, 2, 3\}$ and $\mu \in (0, 1]$ and $\bar{R} = 1-10^{-5}$ (values of μ are chosen stepwise with steps of 0.01). Figure 12 shows the result. The number of peers required for a UA to be reachable for incoming SIP message increases to infinity if $\mu \to 0$ (i.e., average peer lifetime is $\varepsilon \to 0$) and converges to $(p + 1)$ for $\mu \to 1$ (i.e. a static network with peers never leaving).

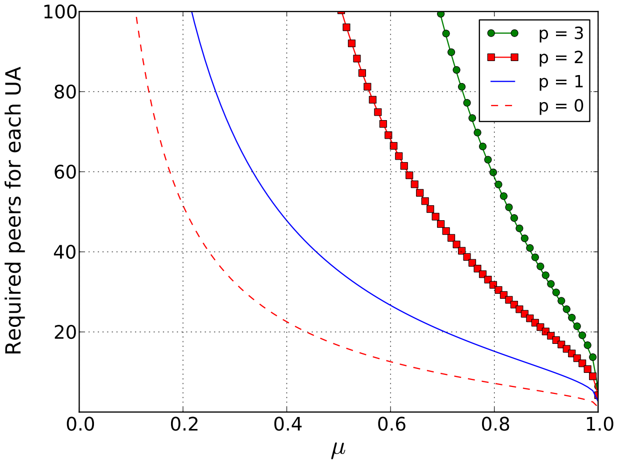


Figure 12: Number of peers required to keep a UA reachable in a Pr^2-P2PSIP network with target reliability $\bar{R} = 1 - 10^{-5}$

Interpretation based on Skype Traces.
Using the Skype network as an example, according to [16], around 87% of the Skype super-peers have a peer lifetime more than $30mn$ and 78% more than $1h$. We interpolated these values to estimate the privacy overhead for $p = 3$ with different refreshing periods. The result is shown in Table 2. E.g., assuming a refreshing period of $20mn$ in Pr^2-P2PSIP, then around 33 peers would be required to keep a UA reachable for incoming calls. However, taking only Skype *super-peers* into consideration means that in Pr^2-P2PSIP only stable peers should be used for storage and inbound tunnels.

Table 2: Estimation of the privacy overhead based on Skype traces

Refreshing period (τ)	μ	Number of storage peers (m)	Number of inbound tunnels (q)	Total number of peers ($m+pq$)
10.0 mn	0.95	5	7	26
20.0 mn	0.91	6	9	33
30.0 mn	0.87	6	12	42
40.0 mn	0.84	7	14	49
50.0 mn	0.81	8	17	59
60.0 mn	0.78	9	19	66

Note that if a UA needs around 33 peers for storage and inbound tunnels, this means also that each UA will receive on average 33 requests within $20mn$ from other peers to store data or be a part of an inbound tunnel. Additional signaling is required for the outbound tunnels, overlay maintenance and DHT lookups.

Conclusions.
The reliability analysis above provides an estimation of the impact of adding privacy to P2PSIP. The signaling overhead generated by Pr^2-P2PSIP to keep a target reliability of $(1-10^{-5})$ should not be underestimated. Further, the overhead is sensitive to the stability of the storage and forwarding peers. This may have different consequences depending on the types of devices used for the UAs. Processing a few requests per minute for storage, tunnels, DHT lookups and overlay maintenance may not be a problem for fixed hardphones, but would mean a large resource consumption for mobile devices, in particular if they are constantly awoken from standby mode (at least, this is a problem today). Given that the signaling overhead is sensitive to the stability of the storage and forwarding overlay networks, it is crucial for Pr^2-P2PSIP to exclude peers with a short lifetime from these overlays.

3.3 Cryptographic Overhead

Given the design decisions described in Section 2.4, the overhead of the public key encryption of a message m sent from a to b using a 194 bit ECC key $+K_b$ and a 128 bit temporary symmetric key $K_{a,b}$ for AES encryption in CBC mode consists of:

- the length of $\{K_{a,b}\}_{+K_b}$, which results in an ECC block size of 194 bits,
- the length of the initialization vector used for the symmetric encryption in CBC mode: 128 bits,
- and a maximum padding of 128 bits for the symmetric encryption,

which results in an overall overhead between 322 and 450 bits, i.e,. approximately between 40 and 56 bytes. Thus, even if a message is onion-encrypted with three layers the overhead in terms of message length remains acceptable.

However, the cryptographic overhead of Pr^2-P2PSIP in terms of the number of public key operations increases linearly with the number of tunnels per UA and the number of peers per tunnel. Thus, the same conclusions hold here as in Section 3.2.

3.4 End-to-end Signaling Latency

The signaling latency from UA A to UA B is affected by:

1. the processing overhead at each forwarding peer,
2. the tunnel length, or the number of forwarding peers used for inbound and outbound tunnels,
3. the accumulated one-way-delay along the full path between A and B,
4. the probability that all forwarding peers in a path are online since they were last.

As mentioned in Section 2.4, once a tunnel is setup, only symmetric cryptography is used. Thus, the cryptographic processing is certainly not a bottleneck. As for the tunnel length and the accumulated delay, we believe that Pr^2-P2PSIP deployed with the recommended tunnels lengths in Section 3.1 does not necessarily involve more signaling hops than server-based SIP networks used in practice today, in particular, where quite a few components are involved in the signaling for different purposes, e.g., lawful interception, billing, etc.

As for the probability that all forwarding peers in a path are online, as mentioned in Section 2.3.4, A tries another end-to-end path, i.e., another combination of outbound tunnel of A and inbound tunnel of B if it does not receive an acknowledgement to a SIP message within $1s$.

Thus, the maximum overall signaling latency is expected to be within a few seconds. If peers in the forwarding overlay are stable, it becomes more likely that the tunnels are available and the signaling succeeds at the first attempt, thus reducing the latency by an order of magnitude. If Pr^2-P2PSIP is used for chat, the same tunnels should be used for subsequent chat messages, since once tunnels have been successfully used, they are likely to remain available for the next chat messages, assuming a heavy-tailed distribution of the peer lifetime.

4. RELATED WORK

Location privacy was not a main concern when the Internet was conceived, because hosts were fixed. However, it was considered early on in GSM standardization. In GSM and UMTS networks, each mobile devices has a unique identifier called the International Mobile Subscriber Identity (IMSI). However, temporary pseudonyms called Temporary Mobile Subscriber Identities (TMSI) are usually used for communication with base stations. Nevertheless, both GMS and UMTS authentication protocols allow an attacker to impersonate a base station and request the User Equipment (UE) to send its IMSI for authentication.

P2PSIP was suggested initially by [7] and [34] and raised much interest and follow up work. Seedorf [32] discusses the security issues inherent in P2PSIP and mentions privacy briefly. In [4], the authors investigate a game theoretical approach for the security threats of P2PSIP such as SPIT and attacks on overlay routing. However, privacy is not addressed.

RELOAD [18], the base protocol for P2PSIP allows for different overlay algorithms to be plugged in. The IETF P2PSIP WG charter [2] does not preclude the deployment of anonymization networks. However, it can not be assumed that any general purpose anonymization network could be used. The Internet draft [17] describes SIP usage for RELOAD and mentions explicitly that “all RELOAD SIP registration data is public. Methods of providing location and identity privacy are still being studied”. Thus, Pr^2-P2PSIP is right on target to address this issue.

Reliability theory has been used in [35] for modeling P2P networks in the context of the KAD file sharing network. In [19], the authors investigate self-tuning behavior of DHTs in order to optimize the reliability costs in the context of Pastry. However, they consider only the reliability of overlay routing. In [38], the authors investigate the costs of maintenance and lookup in DHTs with different ratios of super peers. Their work considers regular DHT functionality without privacy. Nonetheless, our work can be enhanced in the future with a similar analysis in order to provide better insight on the signaling overhead of Pr^2-P2PSIP with different ratios of fixed and mobile devices with different resources. In [8,36] the authors demonstrate how the end points of P2P VoIP streams, e.g. Skype streams, can be identified. Thus, they demonstrate how one could break location and social interaction privacy. However, Skype peers do not consider each other as potentially malicious.

There are many anonymization networks which utilize onion routing [15] or a derivative, notably Tor [10], JAP [12], MorphMix [28] and I2P [11]. They all share characteristics and sometimes differ only in subtle ways. Our intention is not to invent a new anonymization network or new anonymization techniques, but to leverage existing techniques, particularly onion routing and inbound and outbound tunnels to address the privacy issues of P2PSIP. Nevertheless, Pr^2-P2PSIP can still be clearly differentiated from existing anonymization networks in several aspects. Approaches for anonymization networks can be classified into centralized and P2P approaches. Pr^2-P2PSIP is a P2P approach. Centralized approaches, e.g., Tor [10], Crowds [27] and MorphMix [28] rely on centralized databases (although eventually redundant as in the Tor case) to get a list of relay nodes. Pr^2-P2PSIP relies on a forwarding overlay. Likewise, Tor hidden services, which can be compared to Pr^2-P2PSIP inbound tunnels, are accessed via service descriptors stored in a central database. In Pr^2-P2PSIP, peers get the contact data from the DHT before they contact the inbound tunnel entry points.

In P2P anonymization networks, such as I2P [11], Salsa [24], Cashmere [37], Tarzan [14] and AP3 [22], there is no central authority as in Pr^2-P2PSIP, which makes them vulnerable to Sybil attacks. Further, peers select forwarding peers from their P2P routing tables. This makes them vulnerable to attacks where malicious peers attempt to dominate the routing tables of other peers. Pr^2-P2PSIP uses a separate overlay for forwarding and chooses forwarding peers randomly.

Pr^2-P2PSIP allows anonymous routing only within the network. Other anonymity networks such as JAP [12], Cashmere [37], Tarzan [14], MorphMix [28] and Crowds [27] are designed to allow communication with normal servers in the Internet. Thus, they need to support outbound connections. On the other hand, the clients do not have to be reachable for incoming communication as in Pr^2-P2PSIP.

In summary, Pr^2-P2PSIP benefits from the design of Tor and other anonymization networks and experience learned from them, while it has been designed exclusively to provide the P2P-based SIP user registration and session establishment, while preserving the privacy of the network participants. To the best of our knowledge, there has been no work which provides a dedicated solution to the privacy needs of P2PSIP with such an extensive analysis of the implications.

5. CONCLUSIONS

Our conclusions are as follows: Pr^2-P2PSIP provides location and social interaction privacy with a tunnel length of three for inbound tunnels and two for outbound tunnels. Cryptographic overhead is not a hindrance for Pr^2-P2PSIP, in particular if ECC is deployed. Signaling latency improves as the forwarding overlay becomes more stable. The signaling overhead to keep a target reliability of $(1 - 10^{-5})$ should not be underestimated. Further, the signaling overhead is sensitive to the stability of the forwarding overlay. Thus, it is crucial for a successful deployment of Pr^2-P2PSIP that stable peers, i.e., those with a long lifetime, are preferentially chosen for building tunnels.

6. ACKNOWLEDGEMENT

The authors would like to thank Christian Grothoff, Georg Carle and the anonymous reviewers for their valuable feedback and support for this paper. This work is partially funded by the EU project ResumeNet (FP7-224619) and by Deutsche Forschungs Gemeinschaft (DFG) under ENP GR 3688/1-1.

7. REFERENCES

[1] Geo ip tool - view my ip information. `http://www.geoiptool.com/`. Last checked on Feb. 10th 2010.

[2] IETF P2PSIP working group charter. `http://www.ietf.org/dyn/wg/charter/p2psip-charter.html`. Last checked on Feb. 10th 2010.

[3] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Proceedings of Information Hiding Workshop (IH 2001)*, pages 245–257. Springer-Verlag, LNCS 2137, April 2001.

[4] S. Becker, R. State, and T. Engel. Using game theory to configure P2P SIP. In *Proceedings of IPTComm '09, Atlanta, Georgia*, pages 1–9. ACM, 2009.

[5] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of service or denial of security? How attacks on reliability can compromise anonymity. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 92–102, New York, NY, USA, October 2007. ACM.

[6] D. A. Bryan and T. Broadband. P2P SIP. `http://www.p2psip.org`. Last checked on Feb. 10th 2010; last updated Jul. 2009.

[7] D. A. Bryan, B. B. Lowekamp, and C. Jennings. Sosimple: A serverless, standards-based, p2p sip communication system. In *AAA-IDEA '05: Proceedings of the First International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, pages 42–49, Washington, DC, USA, 2005. IEEE Computer Society.

[8] S. Chen, X. Wang, and S. Jajodia. On the anonymity and traceability of peer-to-peer voip calls. *IEEE Network*, 20(5):32–37, 2006.

[9] G. Danezis. Statistical disclosure attacks: Traffic confirmation in open environments. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421–426, Athens, May 2003. IFIP TC11, Kluwer.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[11] I. P. M. et. al. I2p tech intro.

[12] H. Federrath. Jap: Anonymity and privacy. http://anon.inf.tu-dresden.de, 2000-2006.

[13] N. Ferguson and B. Schneier. *Practical Cryptography*. John Wiley and Sons (1st edition), 2003.

[14] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing tarzan, a peer-to-peer anonymizing network layer. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 121–129, London, UK, 2002. Springer-Verlag.

[15] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[16] S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system. In *IPTPS'06: The 5th International Workshop on Peer-to-Peer Systems*, 2006.

[17] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. A SIP Usage for RELOAD. draft-ietf-p2psip-sip-04, Internet Draft, Work in Progress, 2010.

[18] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD). draft-ietf-p2psip-base-08, Internet Draft, Work in Progress, 2010.

[19] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *In IPTPS'03*, 2003.

[20] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-To-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002*, pages 53–65, 2002.

[21] J. McLachlan and N. Hopper. Don't clog the queue! circuit clogging and mitigation in p2p anonymity schemes. In *Financial Cryptography*, pages 31–46, 2008.

[22] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach. Ap3: cooperative, decentralized anonymous communication. In *EW 11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 30, New York, NY, USA, 2004. ACM.

[23] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 183–195, Washington, DC, USA, May 2005. IEEE Computer Society.

[24] A. Nambiar and M. Wright. Salsa: A structured approach to large-scale anonymity. In *Proceedings of CCS 2006*, October 2006.

[25] L. Øverlier and P. Syverson. Locating hidden servers. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 100–114, Washington, DC, USA, May 2006. IEEE Computer Society.

[26] M. Rausand and A. Hoyland. *System Reliability Theory; Models, Statistical Methods, and Applications.* Addison-Wesley Publishing Company (2nd Edition), Reading, Massachusetts, 2004.

[27] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1), June 1998.

[28] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 91–102, New York, NY, USA, November 2002. ACM.

[29] J. Rosenberg. A Presence Event Package for the Session Initiation Protocol (SIP). RFC 3856 (Proposed Standard), Aug. 2004.

[30] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630.

[31] M. Rosing. *Implementing elliptic curve cryptography.* Manning Publications Co., Greenwich, CT, USA, 1999.

[32] J. Seedorf. Security challenges for peer-to-peer sip. *IEEE Network*, 20(5):38–45, 2006.

[33] A. Serjantov and P. Sewell. Passive attack analysis for connection-based anonymity systems. In *Proceedings of ESORICS 2003*, October 2003.

[34] K. Singh and H. Schulzrinne. Peer-to-peer internet telephony using sip. Technical report, Columbia University CUCS-044-04, 2004.

[35] M. Steiner, T. En Najjary, and E. W. Biersack. A global view of KAD. In *IMC 2007, ACM SIGCOMM Internet Measurement Conference, October 23-26, 2007, San Diego, USA*, 10 2007.

[36] X. Wang, S. Chen, and S. Jajodia. Tracking anonymous peer-to-peer voip calls on the internet. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 81–91, New York, NY, USA, 2005. ACM.

[37] L. Zhuang, F. Zhou, U. C. Berkeley, B. Y. Zhao, and A. Rowstron. Cashmere: Resilient anonymous routing. In *In Proc. of NSDI*. ACM/USENIX, 2005.

[38] S. Zoels, Z. Despotovic, and W. Kellerer. Cost-based analysis of hierarchical dht design. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 233–239, Washington, DC, USA, 2006. IEEE Computer Society.

Online Non-Intrusive Diagnosis of One-Way RTP Faults in VoIP Networks Using Cooperation

A. Amirante, S. P. Romano
Computer Science Department
University of Napoli Federico II, Napoli, Italy
{alessandro.amirante, spromano}@unina.it

K. H. Kim, H. Schulzrinne
Department of Computer Science
Columbia University, New York (NY), USA
{khkim, hgs}@cs.columbia.edu

ABSTRACT

We address the well-known issue of one-way RTP flows in VoIP communications. We investigate the main causes that usually lead to this type of fault, and we propose a methodology allowing for their automated online detection and diagnosis. The envisaged approach exploits node cooperation and is based on a more general framework for network faults diagnosis called *DYSWIS* (Do You See What I See). As most of the problems associated with one-way RTP can be ascribed to the presence of NAT elements along the communication path, one of the key features of the proposed methodology resides in the capability to detect such type of devices. Besides, another important aspect of this work is that the diagnosis is non-intrusive, meaning that the whole process is based on the passive observation of flowing packets, and on silent active probing that is transparent to the users. In this way, we also avoid the possibility of being classified as SPIT (SPam over Internet Telephony). We provide a thorough description of the various steps the diagnosing process goes through, together with some implementation details as well as the results of the validation process.

1. INTRODUCTION

We tackle the challenge of automatically detecting faults occurring in SIP-based Voice over IP (VoIP) networks. We first illustrate the most common fault scenarios that characterize a complex communication infrastructure comprising entities which handle end-to-end data, both in the control plane (proxies, back-to-back user agents, etc.) and in the data plane (NATs, Application Level Gateways, relays, etc.). We then focus on one of the most critical faults that can happen when trying to setup a multimedia communication in a SIP [1] network, namely the impossibility of creating a real-time bi-directional communication channel between a caller and a callee. Such fault, which is known in the literature as the "one-way RTP issue", can be due to a number of different yet often interdependent causes and represents one of the most cumbersome problems VoIP architects have to face when deploying and maintaining their networks. We deal with the above mentioned issue by leveraging a novel peer-to-peer architecture for network diagnosis, called DYSWIS (Do You See What I See) [2], which has been conceived at the outset as an extensible infrastructure for non-intrusive, cooperation-based detection of network faults. We will describe how we extended DYSWIS in order to let it support both the SIP and the RTP [3] protocol state machines. The paper embraces an engineering approach. It delves into some of the details of the most notable implementation choices characterizing our contribution. It also illustrates how the most common real-world scenarios which suffer from the one-way RTP issue can be addressed with the approach we propose. At the best of our knowledge, no other approaches addressing the one-way RTP problem have been proposed as yet. The paper is structured as follows. In Section 2 we report the main causes of the problem. In Section 3, we first introduce the DYSWIS architecture as a framework for automated network faults diagnosis; then we show how we added to it support for the SIP, SDP [4] and RTP protocols. The section explains how we devised an approach based on passive tests and silent active probing. Section 4 contains some implementation details, while in Section 5 we show the results of our validation process. Finally, Section 6 concludes the paper by summarizing the main achievements while also presenting the main directions of future work.

2. ONE-WAY MEDIA FLOWS: A WELL KNOWN ISSUE

The problem of one-way RTP flows is very common in VoIP communications. In this section, we provide a classification of the causes that lead to such kind of fault, by splitting them into four main categories.

2.1 Configuration problems

Into this category fall all the problems that can be ascribed to some error in the configuration of the machine hosting a User Agent (UA). First of all, there are possible oversights in the configuration of the UA itself (e.g., wrong audio capture device selected). Then, we have network interface configuration errors, that are quite common especially in multi-homed systems. In fact, it can happen to see RTP packets being received and sent on two different network interfaces, for example on machines having both a wired and wireless connection up (this is not unlikely on Unix-based systems, and is usually due to the configuration stored in the `/etc/hosts` file). The presence of software firewalls not properly configured can also cause one-way media flows: for example, if we want both audio and video to be involved in the call, it would not be sufficient to open a couple of ports, since each call leg consumes two ports (one for RTP and the other for RTCP). Finally, we also classify IP address conflicts in the network as a local configuration problem.

As we will see in Section 3.3, it is easy to diagnose problems falling into this category.

2.2 NAT-related problems

Most of the factors that can cause one-way media flows fall into this category and are related to the presence of NAT elements along the communication path. Several NAT traversal solutions have been proposed by the Internet Engineering Task Force (IETF), namely the STUN *(Session Traversal Utilities for NAT)* [5], TURN *(Traversal Using Relay NAT)* [6] and ICE *(Interactive Connectivity Establishment)* [7] protocols and the *Application Level Gateway* (ALG) and *RTP proxy* elements. If no such solution is employed, the User Agent is unable to receive RTP packets. Even worse, even if a NAT traversal technique is employed, it can happen that the "natted" party is anyhow unable to see incoming packets. This is the case of the most widespread NAT traversal solution: the STUN protocol. STUN is actually helpful in a number of cases; though, it is useless when a User Agent is behind a *symmetric NAT*[1], in which case it experiences one-way media flows. Furthermore, one more scenario where the STUN usage does not avoid one-way RTP flows is when both the caller and the callee happen to be in the same subnet, since a lot of NAT elements discard packets received from the private network and destined to their own public IP address. The last situation can happen also if the STUN protocol is not employed, but the NAT box has built-in SIP Application Level Gateway (ALG) functionality. This is becoming very common, as many of today's commercial routers implement such feature. Unfortunately, poorly implemented ALGs are quite common, too, and in some cases they can be the cause of the problem rather than the solution[2]. Finally, very often the same device handles both NAT and firewall functions; in these cases, port blocking issues have to be taken into account.

2.3 Node crash problems

The sudden crash of a network node also causes the inability to receive RTP packets. We remark that the crashed node could be neither the caller nor the called party, but a possible RTP proxy that belongs to the media path.

2.4 Codec mismatch

A lot of SIP clients offer the possibility to select only a subset of media codecs, among the ones supported. Unfortunately, sometimes this choice is not reflected in the capabilities offered in the SDP, so it can happen that the result of the media negotiation is a codec that has been disabled. As a consequence of this, one of the parties involved in the call would not hear the voice or see the video of the other, even if it is actually receiving the corresponding RTP packets. We report this kind of problem just for the sake of completeness, as in this case we are not experiencing one-way media flows since RTP packets flow in both directions. Consequently, our work does not address this issue.

3. DIAGNOSIS: THE DYSWIS APPROACH

As previously introduced, this work is based upon a network diagnosis architecture that is currently under development at Columbia University, called DYSWIS[3], which leverages distributed resources in the network, called *DYSWIS nodes*, as multiple vantage points from which to obtain a global view of the state of the network itself. Each DYSWIS node is capable to detect fault occurrences and perform or request diagnostic tests, and has analytical capabilities to make inferences about the corresponding causes.

3.1 Architecture overview

From a very high-level perspective, a DYSWIS node tries to isolate the cause of a failure by asking questions to peer nodes and performing active tests. The architecture is depicted in Fig. 1; in the following, we do not dwell on architectural details, since these are beyond the scope of this work. We just remark that a modular approach is adopted, in order to allow support for new protocols in an easy fashion. Specifically, each time a new protocol has to be added, protocol-specific *Detect* and *Session* modules have to be implemented, together with a representation of the fault. Furthermore, new tests and probes have to be implemented, too, when required. Finally, the rules that drive the diagnosis process have to be written. In fact, each DYSWIS node relies on a rule engine that triggers the invocation of the probes on the basis of the type of fault and of the result of previous tests.

As probing functions need to be executed on remote nodes that have specific characteristics, a criterion to identify such nodes is needed, as well as a communication protocol. For example, we could be interested in selecting a peer that has a public IP address, rather than a node that belongs to a given subnet. At the time of writing, remote peers are discovered by means of a centralized repository where each node registers all its useful information as soon as it becomes available. However, an alternative approach, exploiting a *Distributed Hash Table* (DHT), has been implemented in order to better fulfill scalability requirements.

In order to communicate among each other, as well as to convey information about detected failures and request a probe to be run, the DYSWIS nodes exploit a request-response protocol. For further details about how this functionality is provided, refer to Section 4, which discusses implementation aspects.

Finally, when the probing phase is completed, the *Analysis* module produces the final response and presents it to the user.

3.2 Adding SIP/RTP diagnosing features to the framework

For the purpose of this work, we added support for both SIP and RTP to the DYSWIS architecture. The detection part is simply performed by "sniffing" packets on the SIP standard ports 5060 and 5061, as well as on the media ports indicated by the SDP's *m-lines*. In Fig. 2, instead, we show the SIP Finite State Machine (FSM) we devised for the session module. We note that the detection process is based on the observation of packets flowing through a host's network interface, so it is a bit different from the classical SIP state machine.

[1] For a thorough description of the different types of NAT, the reader can refer to [5].

[2] See `www.voip-info.org/wiki/view/Routers+SIP+ALG`.

[3] See `http://www.cs.columbia.edu/irt/project/dyswis/`

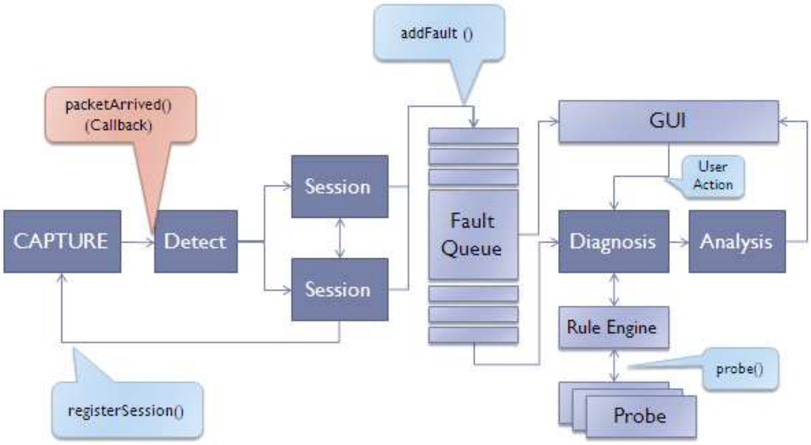


Figure 1: DYSWIS architecture

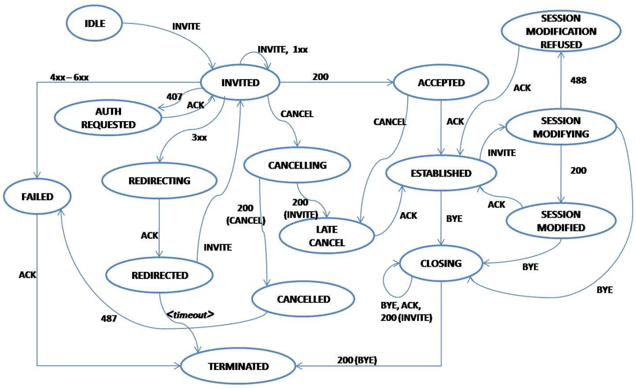


Figure 2: SIP finite state machine

The creation of a new SIP session is triggered by a new `INVITE` message and, within a SIP session, one or more RTP sessions could be created, each one representing a single medium. Specifically, the creation of an RTP session starts with the first SIP message that carries an SDP body (that could be either an `INVITE` or a `200`) and is completed as soon as the second SDP-carrying message is seen (a `200` or an `ACK`, respectively). An RTP session could also be created or modified by re-`INVITE` messages; we took into account such possibility since it is of key importance when both parties of the call make use of the ICE protocol. When the ICE negotiation ends, in fact, the caller sends a re-`INVITE` to update the media-specific IP address and port.

3.3 Proposed diagnosis flow

As already stated, the goal of this work is to diagnose one-way RTP faults by identifying the source of the problem among the ones presented in Section 2. We represent the whole process by means of a flow chart (see Fig. 3) that applies to both UAC and UAS scenarios. It takes into account all the scenarios that can lead to one-way media flows and, even if we will not thoroughly analyze all the possible branches, we provide, in Section 5, some reference scenarios that will help the reader understanding our work. In the diagram, the "local" adjective is used to identify elements or functionality that belong to the same subnet of the DYSWIS node which experienced the fault, while "remote" elements or functionality belong to the same subnet of the other party. We also make a distinction between *tests* and *probes*: the former class only exploits local information, while the latter plays an active role by introducing packets into the network. Finally, we explicitly mark the probes that need the help of a cooperating node in order to be performed.

We observe that it is not always possible to exactly identify the cause of the problem. The capability of making an accurate diagnosis, in fact, strictly depends on the complexity of the network topology under consideration and on actual availability of "remote" DYSWIS nodes, too. The ability to identify such nodes is of key importance and is far from trivial. In fact, when a remote node belongs to a private network environment (i.e., the remote party of the call is natted), its IP address is not helpful for our purpose. Even the node's *reflexive address*[4] can be not helpful in cases where hierarchies of NATs are involved, like the one depicted in Fig. 4. We will explain in the following subsection how we coped with this issue.

It is worth remarking that one of our goals was to carry out diagnosis in a non-intrusive way. In other words, we did not want to allocate new "real" SIP call towards the caller or the callee, because they would be annoying and could be easily classified as SPIT. Instead, a DYSWIS node tries to collect as much information as possible: (i) from the observation of flowing packets, and (ii) with silent active probes (e.g., a STUN transaction to determine its own reflexive address). When an actual SIP session needs to be set up for diagnosing purposes, it is established between two DYSWIS nodes without using the default SIP ports, so that possible softphones running on those machines would not be alerted.

3.4 Description of tests and probes

In this subsection we provide a thorough description of the probing functions we designed and implemented. These

[4] From RFC 5389: the *reflexive transport address* is the public IP address and port created by the NAT closest to the server (i.e., the most external NAT)

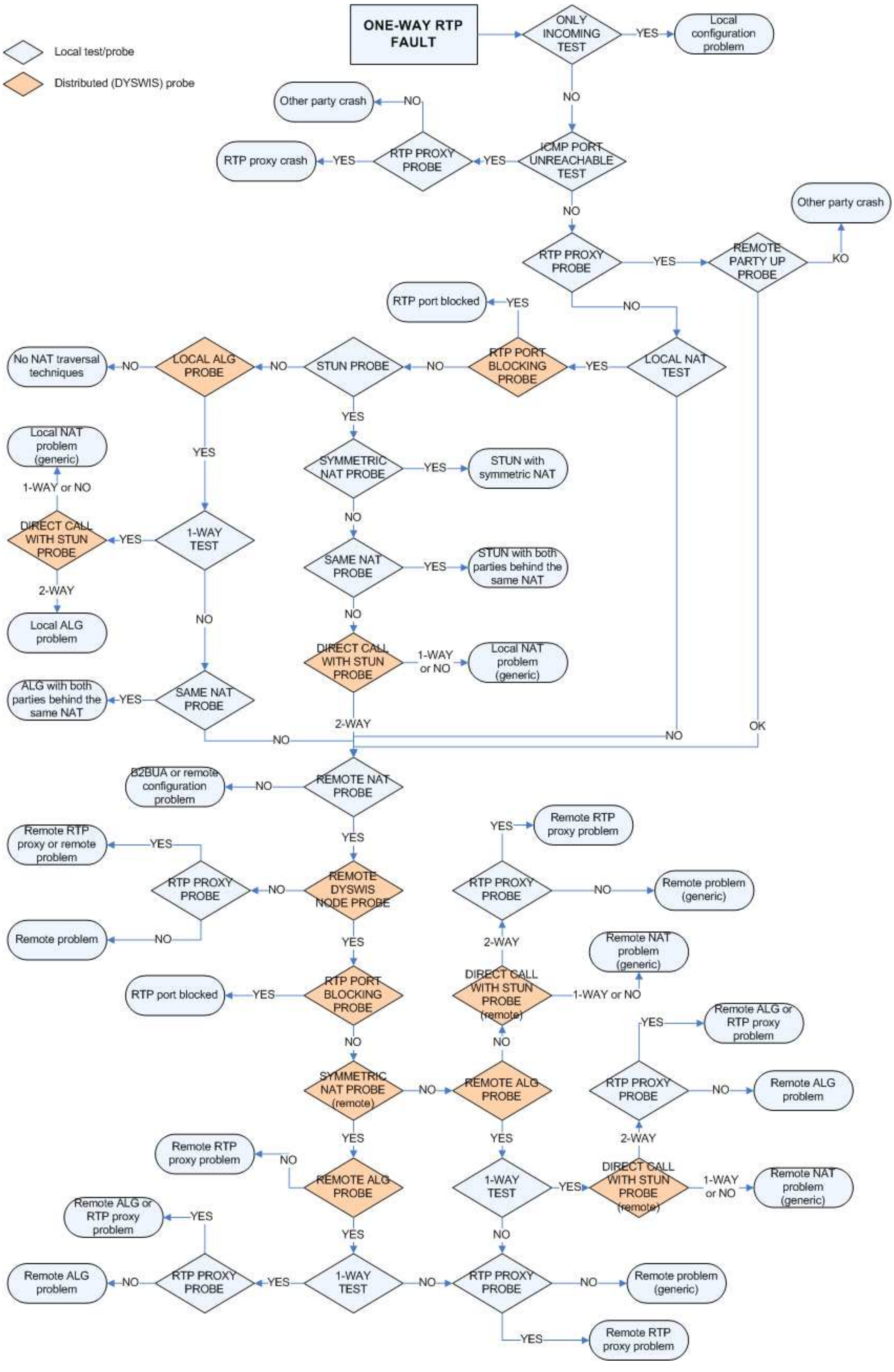


Figure 3: Flow diagram representing the whole diagnosis process

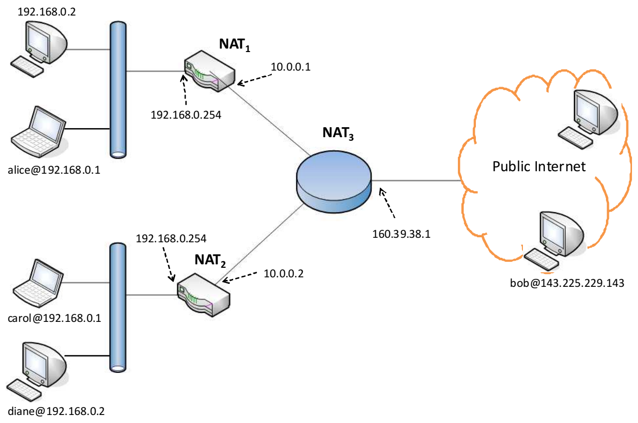


Figure 4: An example of NAT hierarchy that complicates the identification of "remote" peers

probes allow us to test the network environments close to either the caller or the callee (e.g., NATs, ALGs), as well as possible external nodes, like RTP proxies.

3.4.1 *Only incoming test*

This is an easy test that checks whether the detected one-way RTP flow is only incoming or only outgoing.

3.4.2 *ICMP port unreachable test*

Here, we check if there are incoming *ICMP port unreachable* packets, which would be a clear symptom that the process that was supposed to receive data is not active. Herein, we refer to this situation as a node crash.

3.4.3 *RTP proxy probe*

This probe determines if there is an RTP proxy along the media path. An RTP proxy could be manually configured in the SIP client (e.g., a TURN server) or its usage might have been forced by a SIP proxy by modifying the SDP payload of the messages it forwards. We take into consideration both cases. For the former, we compare the IP address contained in the `Contact` header of an **incoming** message with the SDP's *c-line* **of the same message**: if they are different, we can presume that there is an RTP proxy. As to the latter case, instead, we inspect **outgoing** SIP packets, checking if the IP address contained in the SDP's *c-line* is different from both the local interface address and the reflexive IP address that is retrieved by means of a STUN transaction.

3.4.4 *Remote party up probe*

Whenever an RTP proxy is employed, we are not capable to detect a possible crash of the remote node, since we would not receive any ICMP packet. In these cases, we check the availability of the remote party by sending a SIP `OPTIONS` message to it. Such message is sent through all the SIP proxies included in the signaling path, if any, in order to cross a possible remote NAT, making use of the `Record-route` and `Route` SIP headers.

3.4.5 *Local NAT test*

This test determines if the local node (i.e., the node which experienced the fault) is behind a NAT by checking if the local interface has a private IP address.

3.4.6 *RTP port blocking*

This probe verifies that the port number used for the RTP flow is not being blocked by a possible firewall running on the NAT box.

3.4.7 *STUN probe*

Here we determine if the local node is making use of the STUN protocol. This probe consists in a STUN transaction to learn the local reflexive IP address. The result is then checked against the address contained in the SDP's *c-line* of an **outgoing** SIP message.

3.4.8 *Local/Remote ALG*

This probe consists of a direct call attempt to a public DYSWIS node (i.e., a DYSWIS node that has a public IP address). As long as this call attempt is performed without exploiting any NAT-traversal technique, as well as without the SIP extension for Symmetric Response Routing [8], it lets us detect if the local or remote NAT has built-in Application Level Gateway functionality. In fact, the call attempt would succeed only if the private IP address, inserted by the client in the SIP message, is being modified by the NAT element before forwarding it. As previously said, we do not make use of the standard SIP ports for this call.

3.4.9 *Direct call with STUN*

This probe differs from the previously described one only because the call attempt employs the STUN protocol.

3.4.10 *Same NAT probe*

The public (reflexive) IP of the remote party is compared with the local reflexive address: if they match, the two parties are assumed to be behind the same NAT.

3.4.11 *Symmetric NAT probe*

One functionality offered by the STUN protocol is the possibility to discover which type of NAT (Full Cone, Restricted Cone, Port Restricted Cone or Symmetric) is deployed. We use such feature to determine if there is a symmetric NAT, that, as already introduced, might be the cause of the fault we are trying to diagnose.

3.4.12 *Remote NAT probe*

One of the main issues we had to face is the detection of remote NAT elements. In other words, we wanted to learn if the remote party is in a private network environment. Sometimes this is easy because, parsing a received SIP message, we find a private IP address (e.g., it could be in the SIP `Contact`, `From` or `To` headers, or in the SDP's *c-line* or *o-line*). Unfortunately, this depends on the specific implementation of the SIP element: for instance, some clients, when using STUN, put their public address in the SDP's *o-line*, while others do not. Similarly, some ALGs just parse outgoing messages and substitute every occurrence of a private IP, while others perform better thought-out replacements. When we cannot find any occurrence of private IP, we exploit a modified version of the IP traceroute we developed on our own, that sends a SIP `OPTIONS` message gradually increasing the *IP Time-To-Live* value. We send such request towards the public IP address of the remote node and, if we get an ICMP *TTL exceeded* packet whose source address is the original target of our request, it is a clear indication of the presence of a remote NAT element. Otherwise, we could either receive a SIP response (e.g., a 200) or do not receive any response at all. In the latter case, after having retried to send the message, with the same TTL value, for a couple of times (to take care of possible packet losses), we infer that there is a remote NAT box that is not a *Full Cone*. Consequently, our SIP message is being filtered. Finally, if we receive a response to the OPTIONS query, we cannot state there is no NAT along the path, yet. In fact, in the standard specification [9], there is no constraint for a NAT element to decrease the TTL value while forwarding packets. This topic has been discussed a lot on the BEHAVE[5] mailing list of the IETF, where both personal opinions and implementation reports were provided. It turned out that a NAT does not always decrease the TTL of packets received on the public interface, while, for diagnostic reasons, it always decreases it for packets generated in the private environment and forwarded outside. Then, in order to take into account this possibility, when we receive a response to the aforementioned SIP OPTIONS query, we check the TTL value of the IP packet and try to infer whether it comes from a end-host or it has been modified by a NAT. This check is performed by considering that host operating systems have distinctive values for the initial TTL. Then, if the packet did not go through a NAT, the received TTL value would be equal to one of such initial TTL values, decreased by the number of "hops" returned by the traceroute. Otherwise, we infer the presence of a NAT. Further details of these OS-specific TTL values can be found in [10].

For the sake of completeness, we report a draft proposal [12] that has been recently submitted to the IETF and that might prove helpful for the NAT detection problem. It introduces a new SIP header field called Debug whose purpose is to convey extra debugging information.

3.4.13 Remote DYSWIS node probe

We conclude the description of the probing functions by showing how we realized the selection of a DYSWIS node that belongs to the same subnet of the remote party of the call. As we already said, a selection merely based on the public IP address would not be sufficient whenever there is a hierarchy of NATs. Then, after having selected all the DYSWIS nodes characterized by the same public IP address as the remote party, by means of the criterion described at the beginning of Section 3, we need to verify if one (or more) of them can be exploited for our purposes. We achieve this goal by sending a SIP INFO message in broadcast over the LAN. Such INFO message has to be sent within the dialog existing between caller and callee, so that, according to the INFO's RFC [11], *"A 481 Call Leg/Transaction Does Not Exist message MUST be sent by a UAS if the INFO request does not match any existing call leg"*. This is achieved by making the node aware of the To and From tags and of the Call-ID, so that it could be able to generate a request within a specific dialog. Therefore, each selected node would receive a non-481 response only if the remote party belongs to its same subnet.

Among all the methods envisaged by the SIP protocol, the only two that MUST[6] send an error response whenever they do not find any existing call leg are INFO and UPDATE. We chose to exploit the first one because, even if it is not mandatory, it is widely implemented in almost all the clients currently available.

4. IMPLEMENTATION DETAILS

In this section we provide some brief information about the implementation choices. Besides Java, that has been chosen at the outset as the programming language for the whole framework for its well known platform-independence characteristic, the framework exploits the Jess rule engine [13] to control the diagnosis process. Jess uses an enhanced version of the *Rete* algorithm [14] to process rules, making Java software capable to "reason" using knowledge supplied in the form of declarative rules. Consequently, we implemented the whole flow chart presented in Fig. 3 as a set of rules in the Jess scripting language. The example below shows the rules allowing for the detection of a node's crash, when incoming ICMP packets are detected:

```
(defrule MAIN::RTP_ONEWAY
 (declare (auto-focus TRUE)) => (rtp_oneway (fetch FAULT))
)

(deffunction rtp_oneway (?args)
 "one-way RTP diagnosis"

(bind ?result (LocalProbe "RtpOnlyIncomingTest" ?args))(
  if (eq ?result "ok") then
    (bind ?finalresponse "Local configuration problem")
  else then
    (bind ?result (LocalProbe "IcmpDestUnreachTest" ?args))(
    if (eq ?result "ok") then
      (bind ?result (LocalProbe "RtpProxyTest" ?args))(
      if (eq ?result "ok") then
        (bind ?finalresponse "RTP proxy crash")
      else then
        (bind ?finalresponse "Other party crash")
    )
    else then
      ...
```

As to the SIP/SDP functionality, we adopted the JAIN APIs [15] developed by the National Institute of Standards and Technology (NIST).

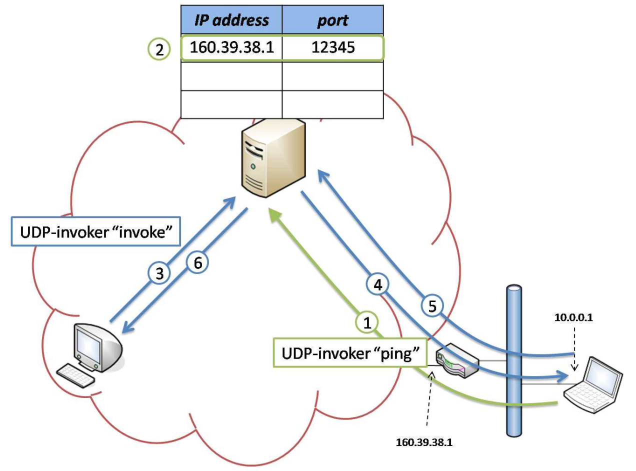
For the invocation of remote probes on nodes that happen to be in natted environments, we chose to make use of the *udp-invoker* library [16], slightly modifying it in order to fit our needs. More precisely, a remote natted node is contacted by means of a relay agent, as shown in Fig. 5: as soon as a DYSWIS node belonging to a private environment becomes available, it sends a *udp-invoker ping* message to the relay agent, which in turn stores the related public IP address and port. Such message is sent periodically, in order to properly refresh the binding in the NAT table. Then, if the probing functionality provided by a private node needs to be exploited, the *invoke* message is sent through the relay agent. We remark that, in such way, we managed to cross any type of NATs. On the other hand, when the peer has a public IP address, the XML-RPC protocol [17] is exploited. Since it uses HTTP as the transport mechanism, it is more reliable than udp-invoker and, in some cases, it helps crossing restrictive local NATs.

Finally, the Jpcap library [18] allowed us to "sniff" packets from the network interfaces and send ad-hoc formatted packets, as well.

5. VALIDATION

[5] BEHAVE (Behavior Engineering for Hindrance Avoidance) is the working group of the IETF which deals with the behavior of NATs

[6] In the IETF jargon, the capitalized word "MUST" represents an absolute requirement of the specification.



IP address	port
160.39.38.1	12345

Figure 5: Remote probing functionality of natted nodes leveraging a relay agent

In this section we provide the results of our validation. We tested our work with several different SIP clients. Specifically, we exploited the following softphones: *X-Lite* [19] (Windows), *SJPhone* [20] (Windows and Linux), *Ekiga* [21] (Linux) and *PJSIP-UA* [22] (Linux). As SIP and RTP proxies, we used *OpenSIPS* [23] and its *RTPproxy* [24] component, respectively. Finally, we developed our own implementation of a basic SIP ALG, since we could not find any suitable open-source library. With all these components, we set up a distributed testbed between the IRT lab at Columbia University and the COMICS lab at the University of Napoli. For the sake of conciseness, we do not present all the possible diagnosis paths that result from the flow chart in Fig. 3, which nonetheless have all been tested. Instead, we just provide a couple of representative scenarios, which show how the diagnosis process takes place.

5.1 Scenario 1: problem with the local ALG

The first scenario we examine is characterized by the use of an ALG in the local network. We deliberately modified our ALG library in order to induce the one-way RTP fault. Specifically, we let our ALG function modify the *c-line* in the session-level section of the SDP message, without changing the same parameter in the media description section. So, since the session-level parameter is overridden by an analogous one in the media description, if present, the remote party will send its RTP packets to a private, non-routable, IP address.

In Fig. 6 we show a snippet of the whole flow diagram that applies to this situation, whose understanding is quite straightforward. We just clarify the last steps. The call attempted by the *Local ALG probe* can take place, thus revealing the presence of an ALG. Though, the resulting RTP flow is still one-way and this definitely represents a clue that the source of the problem might be the ALG itself. Such conjecture is confirmed by the *Direct call with STUN probe*. In fact, as long as we employ the STUN protocol before placing the call, the ALG does not come into play, since there would be no private IP addresses to replace.

5.2 Scenario 2: remote RTP proxy crash

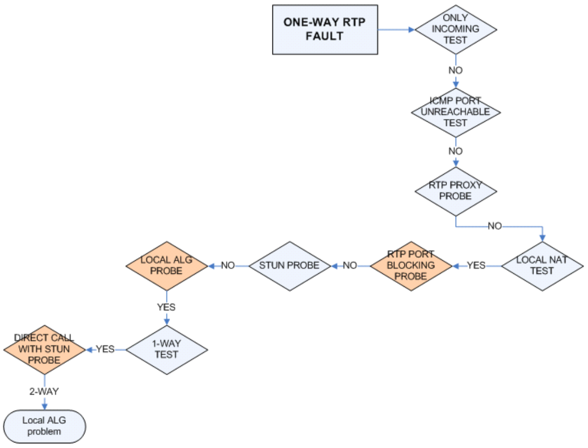


Figure 6: Local ALG problem

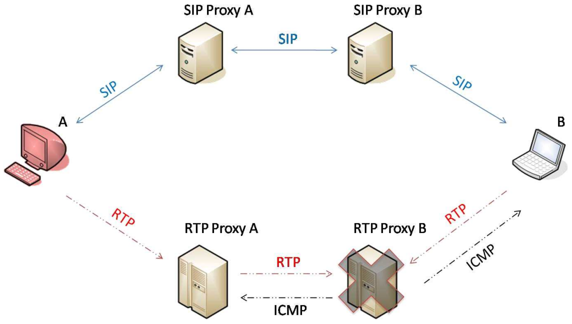


Figure 7: Remote RTP proxy crash: network topology

In this scenario, we suppose that both caller and callee use an RTP proxy. If the proxy used by the remote party crashes, the local DYSWIS node will experience a one-way RTP fault. Furthermore, it will not see any incoming ICMP packet (see Fig. 7).

In Fig. 8 we show the diagnosis steps in this scenario. We are supposing that the remote node is behind a non-symmetric NAT that has no built-in ALG functionality. However, even changing such hypotheses, we are still able to identify the cause of the fault. In general, when the diagnosis process involves the remote subnet, the results of the various probing functions allow us to narrow down the set of possible sources of the problem. In this case, we first get ensured that the problem cannot be ascribed to a remote ALG; then, we exclude that it could be somehow related to the remote NAT's behavior, since the SIP+STUN call involves two-way media flows. This brings us to the final verdict. We observe that, in this lucky case, we are able to detect the exact cause of the fault, while in other cases, when the network topology is particularly complex, we are able to narrow down the fault space to two possible choices.

6. CONCLUSIONS AND FUTURE WORK

In this work we dealt with RTP faults in VoIP networks. Specifically, we addressed the well-known problem of one-

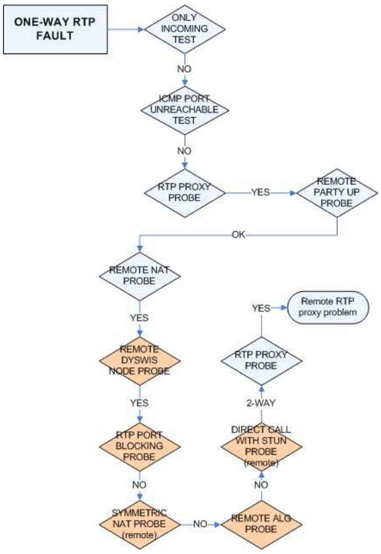


Figure 8: Remote RTP proxy crash: diagnosis flow

way media flows, by first introducing the main causes and, then, by proposing a methodology allowing for its online detection and diagnosis. The proposed approach leverages distributed resources in the network that cooperate in order to isolate the source of the fault, as envisaged by the wider framework for network fault diagnosis, called DYSWIS, it is based upon. The diagnosis process is completely transparent to the users and does not generate any unsolicited calls. We showed that most of the times we are able to exactly identify the source of the problem, while, in the worst cases, we manage to narrow down the fault space to two possible choices. We provided the reader with a thorough description of the diagnosis process, also presenting some reference real-world scenarios, in order to ease its understanding. Finally, implementation details about the prototype we realized have been provided, too, together with the results of the validation we conducted.

The framework described in this paper paves the ground to future research challenges. Besides its enrichment with new protocols and new fault scenarios, we see a big potential in the exploitation of the DYSWIS framework for security purposes. For example, as long as we consider an intrusion as a type of network fault, we might follow the DYSWIS approach in order to build a distributed IDS (Intrusion Detection System). In such context, nodes cooperation is also helpful in the reaction/remediation process. Finally, security issues must be faced in order to avoid that the active probing functionality is exploited for bad purposes by malicious users. Then, it is worth providing the framework with intrinsic mechanisms that guarantee its robustness against possible attacks.

7. ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme INSPIRE (FP7/2007-2013) under grant agreement no. 225553.

This work has been carried out with the financial support of Intel Corporation.

8. REFERENCES

[1] J. Rosenberg, H. Schulzrinne et al., *SIP: Session Initiation Protocol*, RFC 3261, June 2002.

[2] V. K. Singh, H. Schulzrinne and K. Miao, *DYSWIS: An Architecture for Automated Diagnosis of Networks*, Network Operations and Management Symposium 2008, April 2008, 851-854.

[3] H. Schulzrinne et al., *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, July 2003.

[4] M. Handley, V. Jacobson and C. Perkins, *SDP: Session Description Protocol*, RFC 4566, July 2006.

[5] J. Rosenberg, R. Mahy, P. Matthews and D. Wing, *Session Traversal Utilities for NAT (STUN)*, RFC 5389, October 2008.

[6] J. Rosenberg, R. Mahy and P. Matthews, *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*, RFC-to-be 5766, February 2010.

[7] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*, RFC-to-be 5245, February 2010.

[8] J. Rosenberg and H. Schulzrinne, *An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing*, RFC 3581, August 2003.

[9] P. Srisuresh and K. Egevang, *Traditional IP Network Address Translator (Traditional NAT)*, RFC 3022, January 2001.

[10] T. Miller, *Passive OS Fingerprinting: Details and Techniques*, `http://www.ouah.org/incosfingerp.htm`.

[11] S. Donovan, *The SIP INFO Method*, RFC 2976, October 2000.

[12] V. Pascual et al., *A SIP Flight Data Recorder Extension*, work in progress, July 2009.

[13] Jess rule engine's web site: `http://www.jessrules.com/`

[14] C. L. Forgy, *Rete: a fast algorithm for the many pattern/many object pattern match problem.* In Expert Systems: A Software Methodology For Modern Applications, P. G. Raeth, Ed. Ieee Computer Society Reprint Collection. IEEE Computer Society Press, Los Alamitos, CA, 324-341

[15] Jain project's web site: `https://jain-sip.dev.java.net/`

[16] UDP-Invoker project's web site: `http://code.google.com/p/udp-invoker/`

[17] XML-RPC project's web site: `http://www.xmlrpc.com/`

[18] Jpcap's web site: `http://netresearch.ics.uci.edu/kfujii/jpcap/doc/`

[19] X-Lite's web site: `http://www.counterpath.com/x-lite.html`

[20] SJPhone's web site: `http://www.sjphone.org/`

[21] Ekiga's web site: `http://ekiga.org/`

[22] PJSIP's web site: `http://www.pjsip.org/`

[23] OpenSIPS' web site: `http://www.opensips.org/`

[24] Sippy RTPproxy's web site: `http://www.rtpproxy.org/`

Work in Progress:
A Communications-Enabled Collaboration Platform

John Buford, Kishore Dhara, Venky Krishnaswamy, Xiaotao Wu
IP Communications Department
Avaya Labs Research
buford, dhara, venky, xwu@avaya.com

Mario Kolberg
Dept. of Computing Science and
Mathematics, Univ. of Stirling
mko@cs.stir.ac.uk

ABSTRACT

Existing online collaboration tools and platforms provide basic communications integration and the ability to include some real-time information sources. For enterprise use there are requirements for extending these tools with better integration with existing intelligent communication systems, simplifying the collaboration life cycle, enabling the collaboration process, and being able to support long-term collaborations in a variety of ways. We present a new model for such a collaboration environment called *ConnectedSpaces*. Like a number of existing systems, ConnectedSpaces uses a collaboration space as the basic construct. We present important feature sets of ConnectedSpaces, including views, spaces as communication endpoints, space persistence and structuring, and a variety of types of embedded objects. We then describe novel features of the ConnectedSpaces framework, including space history, embedded gadgets and robots, semantic processing, and integration with other collaboration frameworks. Finally we illustrate specific ConnectedSpaces functionality with examples from experimental work.

Categories and Subject Descriptors

H.4.3 [**Communications Applications**]: Computer communication and information browsers

General Terms

Management, Design

Keywords

Collaboration Tools, Enterprise Communication, Feature sets

1. INTRODUCTION

Today's enterprise collaboration platforms include well known web conferencing systems, online document editing, shared document repositories, and voice and video conferencing. Rudimentary communications integration is starting to appear, with softphone and instant message components being integrated into web browsers and collaboration tools through Web 2.0 programming.

More than twenty years of research in groupware, computer-supported cooperative work, and mixed reality systems has demonstrated a rich set of potential features for collaboration environments. However the convergence of internet-scale telephony, messaging, RIA, web, online media, social networking, and real-time information feeds has rapidly enlarged the design choices. It has also made it possible to launch mass market collaboration applications which are distinguished not by major feature differences but by stylistic associations such as tweeting, yammering, skyping, IM-ing, and blogging.

We envision the following areas of evolution to these tools and platforms for increasing their utility for information workers in enterprises, and are particularly interested in seamless integration of intelligent communication capability:

- Highly composable collaboration spaces including space addressing and nesting
- Collaboration spaces as communication endpoints
- Space history and temporal control which includes semantic time markers and layered time relationships
- Group management and information security

In this paper we present the following results:

1. We describe features for ConnectedSpaces, an enterprise-oriented collaboration platform, and compare these features with existing collaboration platforms.
2. We present the ConnectedSpaces framework for building scalable collaboration platform.
3. We present implemented components of ConnectedSpaces-like functionality as illustration of key ideas.

Section 2 summarizes related work in contemporary collaboration systems. Section 3 describes the ConnectedSpaces collaboration framework. Section 4 presents the categories of ConnectedSpaces features, and describes example use cases. Section 4 describes the ConnectedSpaces collaboration framework. Section 5 describes implementation work on ConnectedSpaces components, and Section 6 concludes this paper.

2. RELATED WORK

Collaboration platforms vary from wikis, blogs, and shared documents to web-based collaboration systems to 3D collaboration spaces offered by virtual worlds. The focus in this paper is on the features of collaboration systems and is independent of the underlying collaboration tools used. A brief discussion of the existing classes of collaboration tools helps in understanding the new features of collaboration for enterprises that are discussed in the rest of this paper. For a survey of collaboration platforms, see for example [1].

Web based collaborations such as wikis, blogs, conferencing systems such as WebEx or Meeting Exchange are used for collaboration in enterprises. While wikis and blogs are used as collaborative authoring tools for a large number of users, other web-based conferencing systems are used to create a space that combines users' communication links with desk-top application sharing. Typically, these include audio and video conferencing and features such as side-bar, remote-party mute, etc. These systems are based on the notion that there is a common space that is accessed through a browser and users can collaborate in that space.

Microsoft Groove [3] and Sharepoint offer an alternate approach for collaboration on a set of files or documents. The collaboration client is a thick application and not a generic browser based client. Besides the client, the major variation of this approach is the individual view of data until it is synchronized. That is, each user in the collaboration session can have their view of the data that they work on remotely and synchronize through various means to a common repository. This synchronization is enabled in the client by providing tools for communication between users and by displaying the presence status of various users that belong to the collaboration session.

Other new collaboration platforms such as Google's Wave [1] and Thinkature [4] offer real-time collaboration tools that allows users to create and manage their own collaboration spaces. The ability to create a collaboration space allows users to tailor collaboration space to the needs of a project or for a particular collaborative effort. Persisting these spaces allows users to continue a collaboration in a given space, and continue to use all the contacts, content, and other tools previously added to the space. Further, Google Wave allows threading of a collaborative effort as a Wave and allows user-defined applications (gadgets) and automated participants (robots) to act on such waves. In this paper, we argue that while these are important steps in enhancing collaboration spaces, for enterprise collaboration additional features are needed in those collaboration spaces.

There is another set of collaboration platforms that are based on virtual worlds, such as Second Life [5], Kaneva [6], and There.com [7]. These virtual words offer features such as immediacy (real-time interaction), interaction (ability to view, create, modify, and act) on a shared space that is closer to replicating reality. While these platforms offer rich user-experiences, often the creation of collaboration spaces and the navigation in those spaces is not easy. The taxonomy of networked virtual environments [8] discusses the need for designing a network architecture within virtual environments.

Broll et al. [9] defined an approach for inter-world communication. Bouras et al. [10] defined an approach for inter-world communication. Bouras et al. [11] proposed a distributed virtual reality networking platform for multi-user interaction. Sallnas [12] provides a comparative study of different modes of communications. All of these efforts improve communication and interaction among users of virtual worlds, but are limited to instant messaging or in-world voice. In this paper we propose novel concepts for integrating enterprise communications in collaboration platforms that is mixing in-world (virtual) communication with real world enterprise communication systems. Voice over IP (VoIP) based services provided by companies like Vivox [13] offer communication within virtual worlds, but are not enterprise grade with respect to their scope and their features.

3. CONNECTEDSPACES

3.1 Motivation and Goals

Today's collaboration tools are powerful and widely used. Nevertheless we observe:

- the need for better integration of intelligent communication capability with collaboration environments.
- the value of simplifying the creation and initialization of new collaborations.
- the importance of being able to structure collaborations and treat them as persistent and externally reference-able, since enterprise collaborations are often long-term, deal with complex information, and are important to document.

To achieve these goals, our ConnectedSpaces framework uses increased automation, meta (view) mechanisms, integration with external information and communication resources, and semantic processing where feasible.

The following table summarizes key concepts in our collaboration model.

Table 1 Concepts in ConnectedSpaces

Concept	Definition
space (collaboration space)	A collaboration space provides a shared persistent container in which users perform collaboration activities. It requires resources, such as computation, communication, and storage devices, to support those activities. For

	example, Google Wave, Microsoft Sharepoint, and many virtual worlds, such as the Second Life, are all collaboration spaces.
view	A view of a shared space is a user, a group, or a project specific meta perspective of the collaboration space that itself can be shared, annotated, analyzed, and stored for further retrieval.
entity	An agent that can view and modify the space and its attributes. Entities are also referred to as members of a space. Each entity has a unique identifier.
contact	Any entity which a given user may share a space with
user	A human entity
robot	A system owned entity that can automatically perform some actions in the space.
avatar	The representation of an entity in a space
object	A component embedded in a space that users and robots can operate on. It can be system created or created by users. Objects include content, gadgets, real-time information sources, other spaces, and gateways to components of other collaboration platforms.
gadget	An object that contains application logic that may affect other entities or communicate with applications outside of the collaboration space.
application	A collaboration application is used to provide certain functions to manipulate entities in a collaboration space.
event	An event driven collaboration space uses events to notify one entity about the system and other entities' states and activities.
session	A collection of collaboration activities among users, robots, and objects. It spans a certain period of time, contains some specific semantic information, and requires resources, such as communication channels, storage, and network bandwidth, to support the collaboration activities. A space may contain multiple sessions.
template	A pre-initialized set of objects that can be inserted into a space that provide a pattern for some collaboration activity.
policy	A rule specified by the entities managing a space and enforced by the collaboration framework which specifies constraints on sharing and accessing the space and its objects.

Conventional collaboration tool features include creating a new collaboration space, adding collaboration tools and applications, initiating communication to members of the space, and managing access controls to the collaboration space. In the rest of this section we present collaboration features that are important for enterprise users.

3.2 Enterprise Collaboration Model

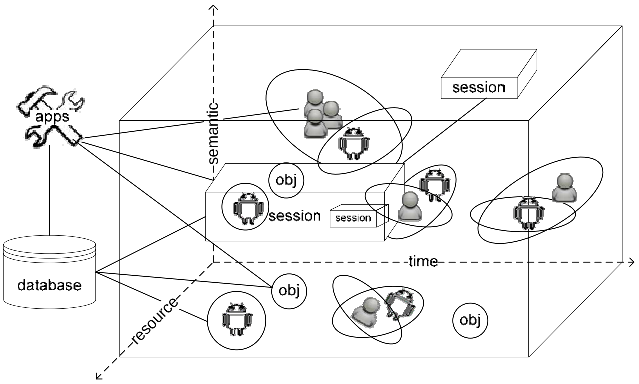


Figure 1 ConnectedSpaces Collaboration Space

As shown in Figure 1, a collaboration space in ConnectedSpaces is represented in three dimensions: *resources*, *time*, and *semantics*. Each object in the collaboration space uses some resources, spans a certain period of time (life cycle of the entity), and has certain semantic properties (either pre-defined or dynamically updated).

Each space has one or more entities which are members of the collaboration. Each entity has a unique identity. Entities can be organized in groups, and groups can be members of a collaboration space. Identities of entities are managed by the collaboration system. We call system owned entities *collaboration robots* or simply robots. In the collaboration space, there can also be sharable objects that member entities space can operate on, such as documents and images. Spaces can be nested, and as in Figure 1, a space can include or refer to another space.

An important concept in the collaboration space is *session*. A session represents a collection of collaboration activities among users, robots, and objects within a space. It spans a certain period of time, contains some specific semantic information, and requires resources, such as communication channels, storage, and network bandwidth, to support the collaboration activities.

A space will include one more sessions. There can be session specific robots and objects. A wavebot becomes active only if a user invites it to a session. A robot may be associated with a specific user. For example, a user may have an assistant robot to help her manage her sessions, such as preparing documents, automatically creating a session and inviting her to join, and recording the session.

Outside of the space, there can be applications that can manipulate objects in the space or provide collaboration channels. For example, call routing functions can be considered as collaboration applications. Embedded communications widgets [14] are examples of such applications. In addition, the manipulation of user preferences and policies about appropriate collaboration behavior in space can also be considered as collaboration applications. Such policies, preferences, and the history of the collaboration activity information can be saved in database for later mining by analytical functions.

3.3 Collaboration Views

While setting up sharing in collaboration spaces is essential for enterprises, valuable meeting time is lost in bringing appropriate content to the shared spaces. The persistence of a collaborative space allows instant access to the previously shared content and a set of commonly used tools. However, it does not address a fundamental issue of *view* in shared collaboration spaces. A view of a shared space is a user, a group, or a project specific meta perspective of the collaboration space that itself can be shared, annotated, analyzed, and stored for further retrieval. In ConnectedSpaces, we add such a notion to instantly bring user specific dynamic context to a collaboration space.

1. User-Specific Views: Based on users' personal data and preferences, ConnectedSpaces allows an overlay of views to collaboration sessions. An example of such a feature is a gadget or an object in a shared space that presents user specific dynamic data such as their interactions across enterprise data that is not shared with all the participants in the session. This overlay presents appropriate information to be presented privately to a user for their active session.

   Figure 2 presents views in their simplest form. The figure depicts a simple collaboration space of an end-user. It depicts the overlay of a user's collaboration space with two views that contain data mined from user's data. The first of these views is a *relevant contacts view* that captures the user's collaboration context, mines data from user's previous sessions, email, calendar, and other data sources to present a list of contacts that the user may need during the collaboration session. The second view is a *relevant documents view* that presents documents that may be useful for the user in the current session. Figure 2 also shows a third personal view that is related to the context of a session. It shows a list of shared colleagues with the remote party of a session.

   While these examples of views are simple, they present two important aspects. One is that views enhance a user's interaction in a collaboration session. A second aspect is the dynamic nature of views that is context-dependent. In contrast, the contacts gadget in Google Wave is a personalized view but is static and does not depend on the collaboration context.

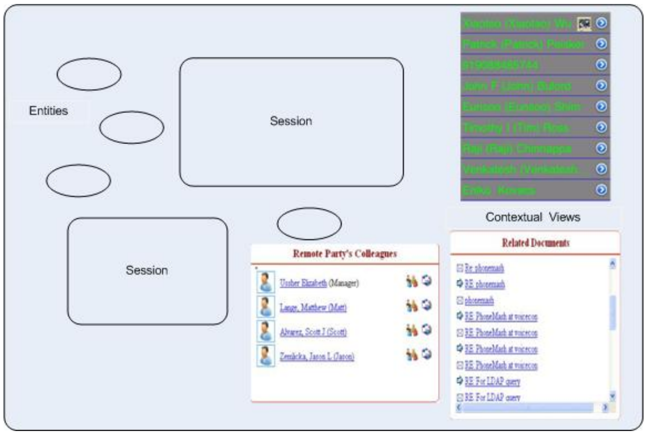


Figure 2 V*iews* in a Collaboration Space

2. Sharing Views: With appropriate access control mechanisms and authentication, users can share personal views with other users or with users who are not participating in the collaboration sessions.

   This feature could be used as a side-bar between a group of users in a collaboration session. Also, in enterprise collaboration, where access to information and resources is often hierarchical, a manager may wish to share views with a delegate to make appropriate decisions during a collaboration session.

3. Managing Views: Views can be attached to a specific space. For views that are dynamic, then robots ensure that they are synchronized appropriately with the content of the space.

3.4 Sharing Space and Navigation

Typically, collaboration tools provide capabilities such as a desktop application sharing, document sharing, audio/video conferencing, and the ability to add new tools to shared collaboration spaces. Despite being part of a shared space, these tools are independent. That is, the navigation controls and context of these tools are not visible to the other tools or gadgets in the collaboration space. Users have to work with each of these tools appropriately and try to connect with the context of their collaboration. Some static context such as participants and existing documents can be shared in some collaboration space gadgets, but this notion is not extended to inter-gadget communication or navigation. ConnectedSpaces offers extensions to provide new features that include dynamic exchange of context and navigation in across gadgets in a collaboration space.

1. Inter-object communication: ConnectedSpaces allows objects that communicate with each other during a collaboration session. As an example, consider a collaboration session with a tool (gadget) that handles shared relevant documents. If a new user joins the collaboration space through a communication session, the shared relevant documents gadget automatically

updates its content to include documents that relate to the new participant.

2. Nested Spaces: As discussed in the previous section, a collaboration space can have nested spaces. These spaces allow users to focus on a particular issue or permit a sub-session that contains confidential data. The participants in a nested space can be a subset of those participants for the parent space. The nested space has a unique that can be externally reference, for example, by a another space.
3. Navigation: ConnectedSpaces allow navigation within a gadget or an object to automatically be reflected in other objects.

3.5 Managing Collaboration Topics and Patterns

Apart from the basic management of starting, ending, and persistenting collaboration spaces, ConnectedSpaces provides additional features that assist user interactions with collaborations sessions.

Automatic Initiation of Collaborations: Based on the information available in existing spaces, ConnectedSpaces robots can automatically create new spaces or initiate communication sessions in existing spaces. Suggested collaborations spaces or sessions can be topic-based, and may be related to content in existing collaboration space or the availability of participants. The robot in some sense predicts the participants, the gadgets or objects required, and the data required for the collaboration session.

Collaboration Template: Collaboration has structure, and the structure of discussion depends on the purpose of the collaboration. For example, parties may collaborate on negotiation, project planning, hiring, investment, and so forth. A template is a predefined set of objects and tools designed to support a collaboration for a specific purpose. When the collaboration is initiated, the template can be selected by the creator of the collaboration, saving the users some time in preparing the space for the intended collaboration.

In addition, a collaboration space can be saved for use as a template for a future collaboration.

3.6 Collaboration Spaces as Communication Endpoints

In ConnectedSpaces, the space itself represents a communications endpoint. The advantages of such representation are as follows.

- Each communications within a space is part of that space's content and history.
- Communications capability to all space members is by default integrated in each space without additional effort by the user.
- Different spaces can be used to organize one's past and future communications.
- Communications to non-members can be provided by embedding specific communications gadgets with those participants.

This means that the space is addressable for communications signaling and that all members of the space are notified for call initiation. Potentially, non-members can also call the space. One way to obtain addressability is to associate a unique identifier in a telephony network with each space instance. For this purpose, we assume the framework includes or integrates a SIP stack or other call stack, and automatically registers each space with the appropriate registrar.

Each space has a default communications device representation, such as a softphone interface in a 2D space or a 3D representation in a virtual world. This representation is in turn bound to one or more personal communication devices. A member uses their local device representation as the interface. When initiating a call, it can be set up as conference call to all the members of the space, a subset, or other endpoints. Robots which are members of the space can be on calls or initiate calls through the space provided the media type of the call is supported by the given robot.

Example 1: Alice defines two spaces, one for work and the other for recreation, and Bob is a member of each space. Alice selects the communications device for the space to initiate a call to Bob. Bob gets a call initiation indication on his device representation(s) for the given space.

Example 2: Alice, Bob, and Charlie are members of a space. When one of them initiates a call, both members receive a call initiation indication on their device representation(s). This is a type of follow-me conferencing. If Jim (a non-member) initiates a call to the addressed assigned to the space, then the associated endpoints of Alice, Bob, and Charlie receive a call initiation indication.

Example 3: Alice uses the communications device in the recreation space to call Bob. The call events are included in the recreation space timeline. Later Alice calls Bob using the communication device in the work space. The call events are included in the work space timeline.

3.7 Context Aware Collaboration

Enterprise collaborations have two factors that distinguish them from other forms of collaborations. One is the context that surrounds the collaboration and the other is the need for a sequence of related collaboration sessions over a period of time. Note that though the participants are important, often it is the case that the context and temporal aspects are important. For example, collaborations that involve a project continue even if members of the team leave. Discussion of context is beyond scope of this paper. However, we use *context* as a general term to capture key aspects of collaboration sessions such as the intent of the collaboration, temporal nature of data, content associated with the collaboration, information about participants, etc.

One feature of such context aware collaborations is to allow applications, such as relevant contacts, to use the context to mine relevant data to generate a user specific view for the session. One can see that the intent of one participant, a customer, could be the context of a collaboration. This collaboration would involve an appropriate customer agent with one or more experts trying to resolve the customer issue.

3.8 Groups in Collaboration Spaces

ConnectedSpaces provides a notion of a group, where a set of users can be identified as a group. Their capabilities and access controls can be managed as a group. This group could have a separate group view that contains data mined from the group's information and shared among members of the group. The ability to have groups allows collaborations to include a large set of people without requiring all of them to be part of the space and without managing their individual identities.

Table 2 Comparison of Features

Feature	Wave	Virtual world	Web-Based Collaboration Tools	Groove (Sharepoint)	Connected-Spaces
User-View	Yes (Static only)	Partially	No	Yes (static)	Yes (Static and Dynamic)
Sharing Views	No	No	No	No	Yes
Managing Views	Partially	Partially	No	No	Yes
Initiation of Sessions	Yes	Partially	No	Partially	Yes
Session Template	Yes	Yes	Yes (limited)	Yes	Yes
Inter-Session Communication	No	No	Yes (in a limited way)		Yes
Nested Session	No	No	No		Yes
Navigation	Limited dependency with contacts, presence and participation	Yes	Independent	Limited dependency with contacts, presence and participation	Yes
Collaboration Spaces as end-points	No	No	No	No	Yes
Groups and Groups Views in Collaboration Sessions	No	No	No	No	Yes

Table 1 compares the proposed features in ConnectedSpaces with other collaboration platforms. The term 'static only' indicates that a collaboration platform allows the feature to be included in its space but only as a static feature. That is, while Google Wave allows personalized contact view, the view is static and does not change based on activity across the collaboration space. Further, the term 'independent' in the Navigation row suggests that within a collaboration session, gadgets have to be independently navigated and do not automatically react to changes in other gadgets.

4. FRAMEWORK

Figure 3 shows the ConnectedSpaces framework. Based on the collaboration space model in Figure 1, the framework consists of three layers. The bottom layer manages the three dimensions of the collaboration space; the mid-layer manages the entities in the collaboration space; and the upper layer consists of collaboration applications. All layers can access data entries through the data access API.

In the bottom layer, the semantic store manages information mined from persistent collaboration data, such as keywords extracted from users' emails and conversation tags. The timer manages timestamps and can generate timeout events. The resource manager handles devices and media servers. The space manager contains helper functions that can manage multiple collaboration spaces. For example, our ConnectedSpaces framework may embed collaboration spaces of Google Wave and Avaya web.alive. In this case, different models of collaboration spaces need to be translated to a sharable view.

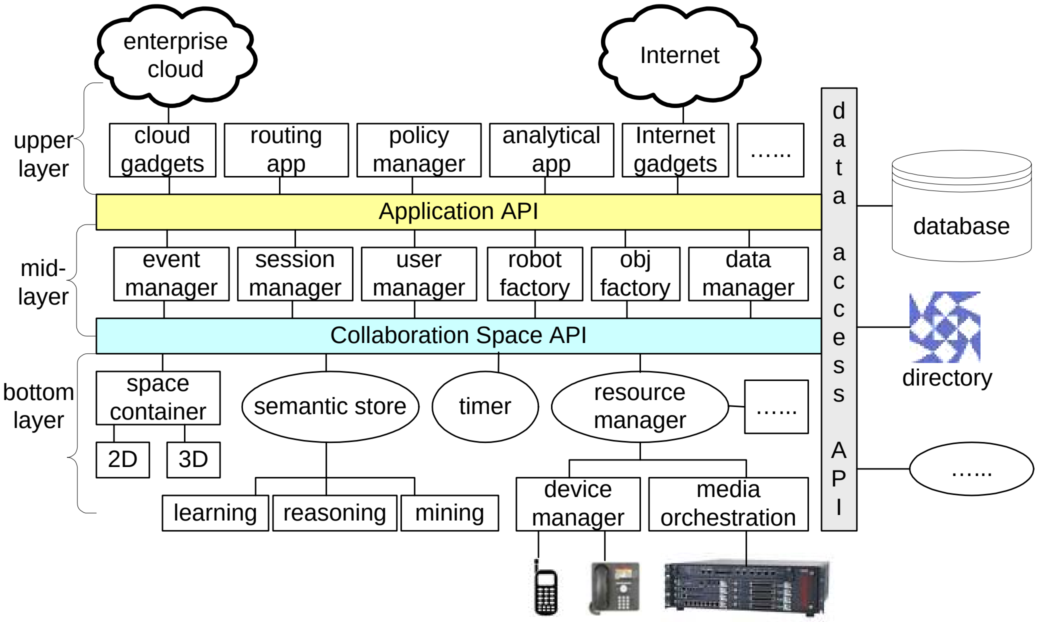


Figure 3. Enterprise Collaboration Framework

In the mid-layer, the robot factory handles system created robots, and the object factory manages the objects in the collaboration space. The user manager handles user registration and manages user profiles. The session manager can create, update, and delete sessions and maintains session information. The event manager and data manager contain helper functions that manage events and session data in the mid-layer.

The upper layer contains different applications that can manipulate sessions, users, robots, and objects. The applications can subscribe to the event manager to get event notification. They can also interact with other applications in enterprise cloud or over the Internet.

Figure 4 shows an example of nesting two sub-spaces in a ConnectedSpaces space and sharing views across spaces. In this figure, Alice's view of Bob is a personalized version of Bob's social profile that is specific to Alice. This personalized social profile can be generated by mining into Alice's wave conversations. Alice's avatar in the ConnectedSpaces space can then access and bring this view to the collaboration space in the Second Life, a virtual world environment. When Alice meets Bob in the Second Life, this view can be shown along side of Bob. Alice can also share this view with the third user, Tom, for a specific duration of time in the ConnectedSpaces space. During the sharing period, when Tom meets Bob in the Second Life space, Tom may also see the view. To achieve this feature, we need the data manager in the mid-layer to collect data, the analytical application in the upper layer to mine the data and generate the view, and the semantic store in the bottom layer to store the view. The space container in the lower layer can manage the relationship of the ConnectedSpaces space, Google Wave space, and the collaboration space in the Second Life. The policy manager in the upper layer and the user manager in the mid-layer can handle access control. When two users meet in Second Life, the event manager gets the event and the session manager creates a session with two users. During the session, the object factory creates a view object from the ConnectedSpaces space and presents it in the Second Life.

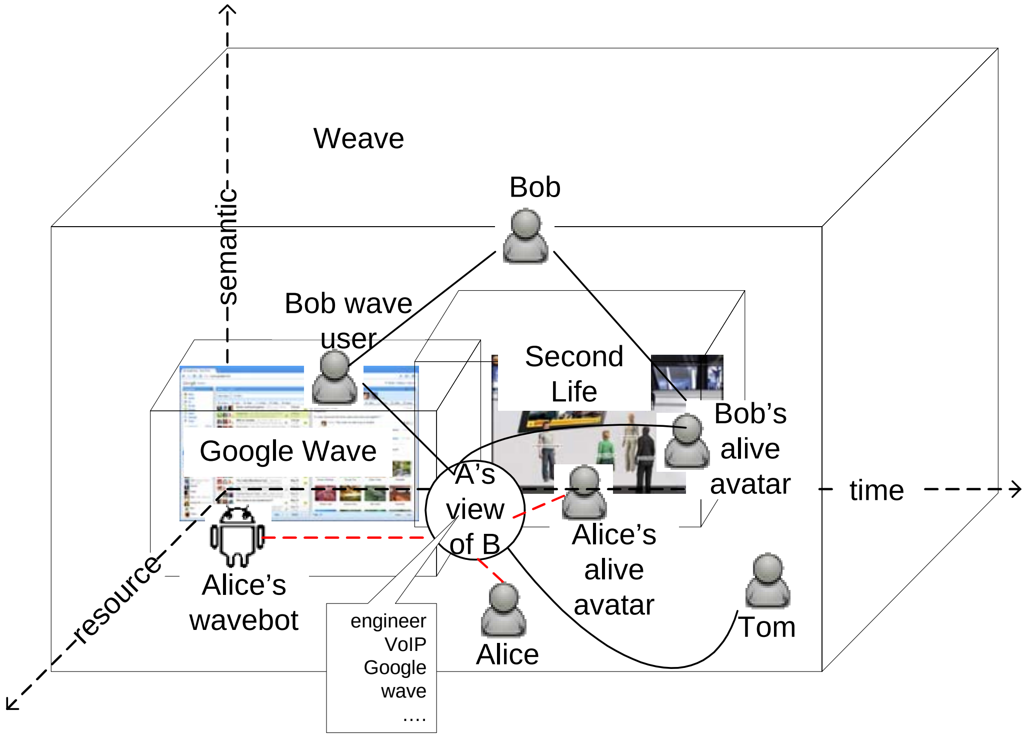


Figure 4. Cross space view sharing

The semantic meaning of entities can enable many new collaboration features. For example, in Figure 4, Alice may group people in her contact list based on the views of those contacts. She can then perform certain activities based on those semantic groups, such as "*sending Google Wave invitation to all the engineers in my contact list*". Note that the "view mining" and "view sharing" features in Figure 4 enables this "semantic grouping" feature. If the mined semantic information is inaccurate, the "semantic grouping" features may misbehave.

5. IMPLEMENTATION WORK

Based on the model we introduced in Section 0, we defined our ConnectedSpaces collaboration framework as shown in Figure 3. We are in the process of implementing the framework presented in Figure 3. In this section, we present several components from our current prototype implementations and relate them to the features and the framework discussed in this paper.

In the bottom layer of the framework, we are building a semantic store by mining users' emails, call histories, and other documents and generate different views of users' collaboration space information. We have also implemented functions that can import views from our collaboration space into Google Wave and the Second Life, as shown in Figure 5.

5.1.1 Session Extension to Google Wave

We extended [19] Google Wave to bring session context information, such as related documents and recent shared contacts, from our collaboration space into Google Wave. In addition, we also allow Google Wave users to control their enterprise voice communication session. The most difficult part of the integration is to allow enterprise information to cross enterprise boundary and enters Google Wave space. We use a border gateway for data access and use a Google wavebot to retrieve the information and a wave gadget to present the information. Figure 5 shows the architecture of the integration [19].

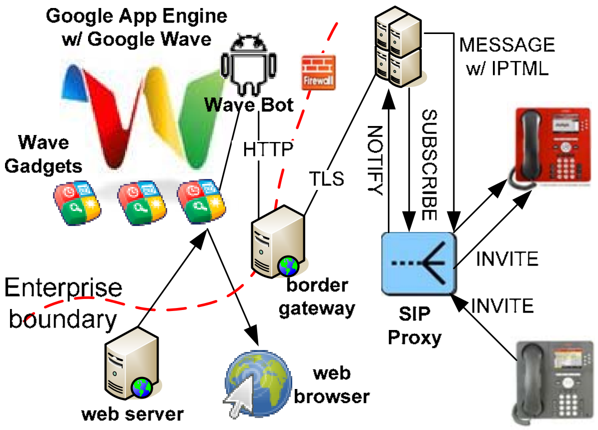


Figure 5 Session Context Integration [19].

5.1.2 Integration with the Second Life

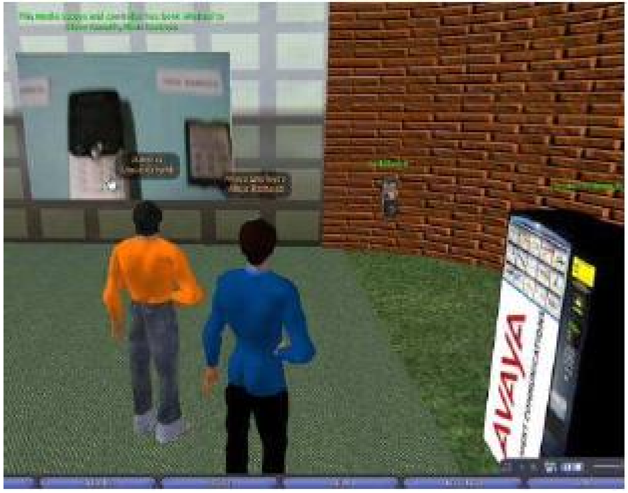


Figure 6 Collaboration Space in Virtual Worlds [20]

Figure 6 captures two avatars interacting in Second Life in the collaboration space, a customer care center, we created. This customer care center contains various interactive 3D objects, communication objects, and access control mechanisms that are tied back to enterprise servers. Some components of our architecture and a use case scenario of our implementation is described in [20]. We limit the discussion of our implementation to its relation to the concepts discussed in this paper, which are as follows.

1. Personal Views: Avatars can come in and check the status of their requests. Also, agents can come in and check the status of their pending jobs.
2. Sharing Views: Some users can come in and check the status of pending requests and can offer help if they can (like a passerby helping in a real-world scenario).
3. Managing Spaces: Objects in the collaboration space are managed by the enterprise as resources via a resource manager as depicted in Figure 2. Managing resources includes access controls, allocating, and clearing up resources.
4. Context Aware Collaboration: Communication enabled from within this collaboration space captures the context and sends it back to enterprise. In Figure 4, this communication is initiated by the object termed as Avaya. Based on the context, in this case a service request by a customer, the enterprise service can bring in appropriate agent, resources, and/or initiated communication sessions.

6. SUMMARY

Existing online collaboration tools and platforms provide basic communications integration and the ability to include some real-time information sources. For enterprise use there are requirements for extending these tools with better integration with existing intelligent communication systems, simplifying the collaboration life cycle, enabling the collaboration process, and being able to support long-term collaborations in a variety of ways. We presented a new model for such a collaboration environment called *ConnectedSpaces*. Like a number of existing systems, ConnectedSpaces uses a collaboration space as the basic construct. We presented important feature sets of ConnectedSpaces, including views, spaces as communication endpoints, space persistence and structuring, and a variety of types of embedded objects. We then described novel features of the ConnectedSpaces framework, including space history, embedded gadgets and robots, semantic processing, and integration with other collaboration frameworks. Finally we illustrated specific ConnectedSpaces functionality with examples from experimental work. Separately we have discussed new types of feature interactions in ConnectedSpaces and an approach to feature interaction detection [22].

7. REFERENCES

[1] J. Rama, J. Bishop. Survey and Comparison of CSCW Groupware Applications. Proceedings of SAICSIT 2006

[2] Google Wave. http://wave.google.com

[3] Microsoft Groove. http://connect.microsoft.com/groove

[4] Thinkature. http://thinkature.com

[5] SecondLife. www.secondlife.com

[6] Kaneva. www.kaneva.com

[7] There.com. www.there.com

[8] B. Book. Moving Beyond the Game: Social Virtual Worlds, Virtual Economy Research Network, Unpublished manuscript, http://www.virtualworldsreview.com/papers/BBook_SoP2.pdf

[9] T. K. Capin, I. S. Pandzic, N. Magnenat-Thalmann, D. Thalmann. Avatars in Networked Virtual Environments, John Wiley & Sons, Inc., New York, 1999, ISBN: 0471988634.

[10] W. Broll. Bringing People Together-An Infrastructure for Shared Virtual Worlds on the Internet, WET-ICE '97: Proceedings 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises, IEEE Computer Society, pp. 199-204

[11] C. Bouras and T. Tsiatsos. Distributed virtual reality: Building a multi-user layer for the EVE platform, Journal of Network and Computer Applications, April 2004, 27(2), 91-111.

[12] E.-L. Sallnas. Collaboration in multi-modal virtual worlds: comparing touch, text, voice and video, The social life of avatars: presence and interaction in

shared virtual environments, Springer Verlag, 2002, ISBN: 1-85233-461-4, pp: 172-187.

[13] Vivox. http://www.vivox.com

[14] X. Wu and V. Krishnaswamy, Widgetizing communication services, ICC 2010, Capetown, South Africa

[15] M. Roseman , S. Greenberg, TeamRooms: network places for collaboration, Proceedings of the 1996 ACM conference on Computer supported cooperative work, p.325-333, November 16-20, 1996,

[16] T. Rodden. Awareness and Coordination in Shared Workspaces. In Proceedings of the 1996 ACM Conference on Computer-Supported Cooperative Work (CSCW'96). pp. 87-96.

[17] S. Bly, S. Harrison, and S. Irwin. Media Spaces: Bringing People Together in a Video, Audio, and Computing Environment. *Communications of the ACM* 36 (1), pp. 27-47.

[18] S. Benford, C. Greenhalgh, T. Rodden, J. Pycock. Collaborative virtual environments. *Commun. ACM* 44, 7 (Jul. 2001), 79-85.

[19] X. Wu, V. Krishnaswamy, C. Mohit. Integrating Enterprise Communications into Google Wave. IEEE Consumer Communications and Networking Conference (CCNC 2010). Jan. 2010.

[20] S. Vijaykar, M. Kadavasal, K. Dhara, and V. Krishnaswamy. Virtual Worlds as a Tool for Enterprise Services. IEEE Consumer Communication and Networking Conference, Las Vegas, Jan 2009.

[21] C. Gutwin , M. Roseman , S. Greenberg, A usability study of awareness widgets in a shared workspace groupware system, Proceedings of the 1996 ACM conference on Computer supported cooperative work, p.258-267, November 16-20, 1996, Boston, Massachusetts, US.

[22] M. Kolberg, J. Buford, K. Dhara, V. Krishnaswamy, X. Wu. Feature Interaction Analysis for Collaboration Spaces with Communication Endpoints. IEEE Globecom 2010. Nov. 2010.

ISBN 3-937201-15 - 7

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)