Network Architectures
And Services
NET 2009-11-1

Seminar FI
WS 09/10

Proceedings of the Seminar
Future Internet (FI)
Winter Semester 2009/2010

Munich, Germany, 12.10.2009

Editors

Georg Carle, Corinna Schmitt

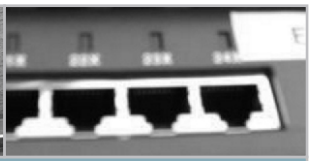
Organisation

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

Technische Universität München







Network Architectures
and Services
NET 2009-11-1

Seminar FI
WS 09/10

Proceedings zum Seminar
Future Internet (FI)
Wintersemester 2009/2010

München, 12.10.2009

Editoren: Georg Carle, Corinna Schmitt

Organisiert durch den Lehrstuhl Netzarchitekturen und Netzdienste (I8),
Fakultät für Informatik, Technische Universität München

Technische Universität München

Seminar FI WS09/10
Seminar: Future Internet

Editors:

Georg Carle
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Technische Universität München
D-85748 Garching b. München, Germany
E-mail: carle@net.in.tum.de
Internet: http://www.net.in.tum.de/~carle/

Corinna Schmitt
Lehrstuhl Netzarchitekturen und Netzdienste (I8)
Technische Universität München
D-85748 Garching b. München, Germany
E-mail: schmitt@net.in.tum.de
Internet: http://www.net.in.tum.de/~schmitt/

Cataloging-in-Publication Data

Seminar FI WS09/10
Proceedings zum Seminar Future Internet
München, Germany, 12.10.2009
Georg Carle, Corinna Schmitt
ISBN: 3-937201-08-4

ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
Lehrstuhl Netzarchitekturen und Netzdienste (I8) NET 2009-11-1
Series Editor: Georg Carle, Technische Universität München, Germany

Vorwort

Wir präsentieren Ihnen hiermit die Proceedings zum Seminar „Future Internet“ (FI), das im Wintersemester 2009/2010 an der Fakultät Informatik der Technischen Universität München stattfand. Im Seminar FI wurden Beiträge zu unterschiedlichen Fragestellungen aus dem Gebiet Future Internet vorgestellt. Die folgenden Themenbereiche wurden abgedeckt:

- Evolution und Revolutionen im bisherigen Internet
- IP Fast Reroute (IPFRR)
- Datagram Congestion Control Protocol (DCCP)
- Netzwerkmanagement mit NETCONF und YANG
- Sammeln von Würmern und Bots mit Honeypots
- Analyse von Würmern und Bots
- Angriffe und deren Nutzen auf BitTorrent und andere P2P File Sharing Netzwerke
- Gruppen-Management in Peer-to-Peer VPNs

Wir hoffen, dass Sie den Beiträgen dieses Seminars wertvolle Anregungen entnehmen können. Falls Sie weiteres Interesse an unseren Arbeiten habe, so finden Sie weitere Informationen auf unserer Homepage http://www.net.in.tum.de.

München, November 2009

Georg Carle

Corinna Schmitt

Preface

We are very pleased to present you the interesting program of our main seminar on “Future Internet” (FI) which took place in the winter semester 2009/2010.

In the seminar FI we deal with issues of Future Internet. The seminar language was German, and the majority of the seminar papers are also in German. The following topics are covered by this seminar:

- Evolution and Revolutions in the current internet
- IP Fast Reroute (IPFRR)
- Datagram Congestion Control Protocol (DCCP)
- Netzwork Management using NETCONF and YANG
- Collecting Worms and Bots using Honeypots
- Analysis of Worms and Bots
- Attacks and exploits targeting Bit Torrent and other P2P file sharing networks
- Group Management in Peer-to-Peer VPNs

We hope that you appreciate the contributions of this seminar. If you are interested in further information about our work, please visit our homepage http://www.net.in.tum.de.

Munich, November 2009

Seminarveranstalter

Lehrstuhlinhaber

Georg Carle, Technische Universität München, Germany

Seminarleitung

Corinna Schmitt, Technische Universität München, Germany

Betreuer

Lothar Braun, *Technische Universität München, Wiss. Mitarbeiter I8*
Benedikt Elser, *Technische Universität München, DFG Emmy Noether Research Group Member*
Nils Kammenhuber, *Technische Universität München, Wiss. Mitarbeiter I8*
Andreas Müller, *Technische Universität München, Wiss. Mitarbeiter I8*
Marc-Oliver Pahl, *Technische Universität München, Wiss. Mitarbeiter I8*

Kontakt:

{carle,schmitt,braun,elser,kammenhuber,mueller,pahl}@net.in.tum.de

Seminarhomepage

http://www.net.in.tum.de/de/lehre/ws0910/seminare/

Inhaltsverzeichnis

Session 1: Future Internet

Session 2: Protokolle

Session 3: Würmer und Bots

Session 4: P2P und DHTs

Evolutionen und Revolutionen im bisherigen Internet

Rolf Siebachmeyer
Betreuer: Nils Kammenhuber
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: rolf@siebachmeyer.de

***Kurzfassung*— Die Geschichte des Internets ist durchsäht von kleineren und größeren Veränderungen. Nicht nur technische Grundlagen, auch Anwendungen, das soziale Verhalten im Netz und die Kommerzialisierung (um nur Einige zu nennen), haben sich seit Beginn es Internetzeitalters stark gewandelt. Diese Arbeit befasst sich in erster Linie mit einigen ausgewählten technischen Änderungen und der Frage, warum diese durchgeführt wurden.**

***Schlüsselworte*— ARPAnet, IP, IPv4, IPv6, Hosts-Datei, DNS, Class-A/B/C-Netze, BGP, CIDR, NCP, TCP, TCP-Congestion-Control.**

I. Einleitung

Rückblickend hat das Internet seit seiner Entstehung im Jahr 1969 einen stetigen Wandel durchlebt. Selten gab es tiefe Einschnitte und Veränderungen in die zugrundeliegenden Konzepte und Technologien. Diese Ausarbeitung befasst sich sowohl mit den Evolutionen des Internets, also dem stetigen Wandel, als auch mit den Revolutionen in der Internettechnologie, also den schlagartigen Veränderungen.

Durch das rasante Wachstum wurde bereits sehr früh klar, dass mit dem Internet ein Schneeball ins rollen kam, der eine ganze Lawine auslösen wird. Exponentielles Wachstum (siehe Abbildung 1) steigerte die Anzahl der Host von gut 200 im Jahr 1981 auf mehr als 1.000 bis Ende 1984. Weitere drei Jahre später waren es bereits mehr als 5.000 und zu Beginn der neunziger Jahre tummelten sich um die 300.000 Hosts im Netz [1].

So wünschenswert dieses Wachstum auch ist, so vielfältig sind die Probleme die dadurch entstehen. Als das Internet noch „klein“ und „ansehnlich“ war, wurden Lösungen eingesetzt, die sehr schnell an ihre Grenzen stießen. Ein anschauliches Beispiel dafür ist die hosts.txt, eine zentral verwaltete Datei, in der IP-Adressen einprägsame Namen zugeordnet werden (dazu mehr in Kapitel IV). In der Anfangszeit des Internets lud sich jeder Host regelmäßig diese hosts.txt von dem Standort, an dem sie gepflegt wurde. Bei ein paar hundert, vielleicht auch bei ein paar tausend Hosts ist das kein Problem. Jedoch ist auch ohne hellseherische Kräfte offensichtlich, dass bei einem solch schnellen Zuwachs an Hosts sehr bald eine Grenze erreicht wird, ab der eine solche Datei nicht mehr mit beherrschbarem Aufwand verwaltet werden kann.

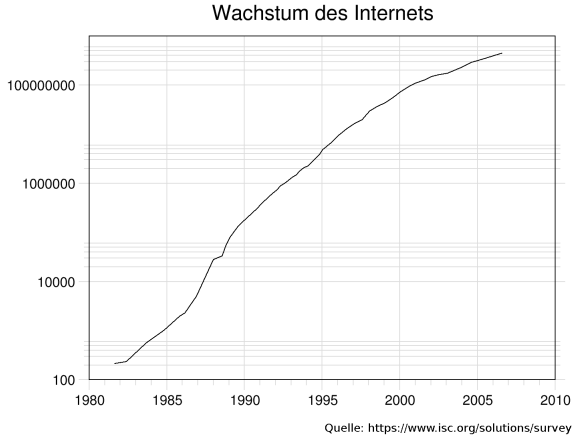


Abbildung 1. Anzahl der Hosts im Netz

Die folgenden Kapitel beschäftigen sich mit dem Wandel des Internets, der durch immer neue Notwendigkeiten stetig vorangetrieben wird. Auch wenn das Internetzeitalter mit Begriffen wie „Kurzlebigkeit“ oder „schneller Wandel“ assoziert wird, so stellt man doch fest, dass es an den Kernkomponenten des Netzes nur mit großer Vorsicht und sehr selten zu Veränderungen kommt.

II. NCP und TCP/IP

Das ARPAnet war das erste groß angelegte Paketvermittlungsnetzwerk und gilt als der Vorläufer des heutigen Internets [2]. Für die Umstellung der zu Grunde liegenden Architektur, damals NCP (=Network Control Program), auf TCP/IP, wurde am 1. Januar 1983, dem sogenannten flag-day, der Grundstein für das heutige Internet gelegt. Diese Umstellung war die vielleicht letzte große Revolution, die das Internet bis heute erlebt hat. Vermutlich war dieser Tag bis heute auch der letzte Tag, an dem ein solcher flag-day möglich war, denn damals war es „nur“ notwendig, etwa 400 Netzknoten auf die neuen Protokolle umzustellen. Bei der seit dem immer größer werdenden Anzahl an Rechnern müssten alle weiteren Änderungen verteilt werden, ohne das bis dahin bestehende Netzwerk zu kompromittieren [3].

A. Das Network Control Program

Sowohl für den Datentransport, als auch für die Vermittlung wurde im ARPAnet bis Ende 1982 das NCP (=Network Control Program) verwendet [4]. Zu diesem Zeitpunkt war die Anpassung oder Umstellung des Netzwerkprotokolls unumgänglich. Ein großes Problem bei NCP bestand darin, dass lediglich acht Bit für die Zieladresse eines jeden Pakets vergeben werden konnten. Damit war es nicht möglich, mehr als 256 Hosts direkt zu adressieren.

Die Anpassung der Länge der Zieladresse von NCP wäre eine Möglichkeit gewesen, um dieses Problem zu behebend. Stattdessen entschied man sich jedoch, die Transport- und die Vermittlungsschicht zu trennen und führte mit dem flag-day die Protokolle TCP (=Transmission Control Protocol)[1] und IP (=Internet Protocol)[2] ein.

Die letzte große Revolution war gelungen. Mittels TCP/IP wurde das Internet flexibler gestaltet und man hatte nun die Möglichkeit, Anpassungen an der Transport- bzw. Vermittlungsschicht vorzunehmen, ohne die jeweils andere Schicht zu beeinflussen.

B. Das Transmission Control Protocol

In seiner ersten Version war das TCP kaum mehr als ein Hilfsmittel zum Versenden von Daten, doch nun stand ein eigenständiges Protokoll auf der Transportschicht zur Verfügung, welches man verändern konnte, ohne die anderen Schichten zu kompromittieren. TCP war und ist sehr erfolgreich. Ein Großteil des Internettraffics wird über dieses Protokoll abgewickelt. Nichts desto trotz hat auch TCP Nachteile. Ein entscheidendes Problem ist der Overhead, welcher beim Versenden von Daten mittels TCP entsteht. Nicht nur der aufgeblähte Header (im Vergleich zu UDP), auch die Bestätigung (engl. Acknowledgement) des Empfängers zu jedem Paket und der Aufbau einer Verbindung zwischen Sender und Empfänger belasteten das Netz sehr stark. Schon Mitte der achtziger Jahre wurde die Überlastung des Netzes (engl. Congestion) zu einem nicht unerheblichen Problem, aufgrund dessen das Internet des öfteren kollabierte. Nach und nach wurde TCP um QoS-Eigenschaften und Congestion Control Algorithmen erweitert, um diesem Problem Herr zu werden (mehr zu Congestion Control mit TCP in Kapitel III).

C. Das Internet Protocol

Auch aus Sicht der Vermittlungsschicht hat sich die Revolution gelohnt. Das IP (damals Version 4, oder kurz IPv4) hat sich nach seiner Erscheinung nicht mehr geändert und es dauerte fast zehn Jahre, bis erste Probleme absehbar wurden. Durch die steigende Zahl an Rechnern im Internet und die viel zu großzügige Adressvergabe (siehe auch Kapitel VI) war es nur eine Frage der Zeit, bis der gesamte verfügbare IPv4-Adressraum aufgebraucht ist. Doch obwohl diese Problematik schon Anfang 1990 erkannt wurde und obwohl dies eine enorme Schwäche von IPv4 offenbarte, ist es bis heute nicht zu einer erneuten Revolution gekommen. Es wurde nachgebessert, bspw. kamen mit CIDR und NAT zwei neue Technologien hinzu, die das IP in seiner damaligen Form bis heute erhalten konnten.

Doch obwohl es bereits seit 1998 mit IPv6 [5] einen geeigneten Nachfolger gibt, ist es bisher nicht zu einer kompletten Umstellung gekommen. Im heutigen Internet fristet IPv6 noch immer ein Schattendasein und kommt nur selten zum Einsatz (dazu mehr im Kapitel VII).

III. Congestion Control

Aufgrund einiger Zusammenbrüche des Internets Mitte der achtziger Jahre, konnte auf die schnelle Einführung einer funktionierenden Congestion Control (zu deutsch: Überlastung- oder Staukontrolle) nicht mehr verzichtet werden [3]. Da die Überlastung des Netzes jedoch in erster Linie ein Problem der Vermittlungsschicht war[3], wäre eigentlich eine strukturelle Änderung an genau dieser Schicht oder zwischen Transport- und Vermittlungsschicht notwenig gewesen. Eine solche Änderung hätte jedoch einen sehr hohen Aufwand bedeutet.

Die einfachste Lösung war hingegen, TCP um einen Mechanismus zur Staukontrolle zu erweitern. Schließlich war TCP durch seine starke Verbreitung der Hauptverursacher von Netzüberlastungen.

A. Congestion Control in TCP

In TCP ist es ohne Weiteres möglich, Staus zu erkennen. Der Absender kann einfach davon ausgehen, dass sein Paket aufgrund eines Staus nicht angekommen ist, falls er nicht innerhalb einer bestimmten Zeit ein Acknowledgement (ACK) erhalten hat. Selbst wenn die Fehlerquelle kein Stau, sondern bspw. eine korrumpierte Routingtabelle eines Routers ist, so ist die Reduzierung der Netzlast immerhin kein schlechtes Verhalten.

Um einen Stau schnell aufzulösen, sollte jeder Sender, der einen Stau erkennt, mit Netzlastreduzierung reagieren. Dazu wurden die vier folgenden Algorithmen definiert [6]:

- Slow Start
- Congestion Avoidance
- Fast Retransmit
- Fast Recovery

Slow Start zusammen mit Congestion Avoidance

Um schon zu Beginn der Datenübermittlung einer möglichen Überlastung vorzubeugen, wird mit dem Slow Start Algorithmus das Congestion Window bestimmt. Da zu Beginn der Übertragung jedoch noch nichts über die Netzlast bekannt ist, wird anfänglich nur eine kleine Datenmenge verschickt. Der Empfänger antwortet auf jedes Datenpaket mit einem ACK, worauf der Sender selbst mit einer Vergrößerung des Congestion Window reagiert. Sobald die Pakete des Senders nicht mehr schnell genug mit ACKs bestätigt werden, ist die Slow Start Phase beendet.

[1] Zuständig für den Datentransport.
[2] Zuständig für die Datenvermittlung.
[3] Schließlich kann jedes beliebige Protokoll der Transportschicht Staus verursachen.

Danach tritt die Congestion Avoidance in Kraft. Dabei wird das Congestion Window nur noch dann vergrößert, falls alle Pakete aus dem Congestion Window mit einem ACK bestätigt wurden.

Kommt es im späteren Verlauf zu einem Timeout, setzt der Sender das Congestion Window wieder auf den Startwert und der Slow Start beginnt von Neuem. Diesmal wird die Slow Start Phase jedoch verkürzt, sodass das Congestion Window bei häufigen Paketverlusten nicht wieder zu schnell wächst.

Fast-Retransmit und Fast-Recovery

Es ist wünschenswert, dass nach Paketverlusten möglichst schnell auf Überlastungen reagiert wird. Zu diesem Zweck werden die Algorithmen Fast Retransmit sowie Fast Recovery verwendet. Hierzu muss der Empfänger dem Sender mitteilen, falls er Pakete in der falschen Reihenfolge erhält. Dies geschieht, indem Duplicate ACKs versendet werden. Eine Duplicate ACK ist eine Bestätigung des letzten korrekt empfangenen Pakets, für genau ein Paket welches außer der Reihe empfangen wurde.

Bemerkt der Sender Duplicate ACKs, beginnt er nach dem dritten Duplicate ACK sofort, das verlorene Paket erneut abzuschicken, ohne noch auf ein ACK dieses Pakets zu warten. Dies wird Fast Retransmit genannt, da nicht auf den Ablauf des Timers gewartet wird. Solche Duplicate ACKs sind ein Indiz dafür, dass es zwar zu einem Datenverlust gekommen ist, aber alle darauf folgenden Pakete den Empfänger erreicht haben. Deshalb wird das Congestion Window nach einem Fast Retransmit nicht auf den Startwert gesetzt, sondern nur halbiert.

Des Weiteren kann das Congestion Window um die Anzahl der Duplicate ACKs vergrößert werden, schließlich steht jedes Duplicate ACK für ein weiteres Paket, welches erfolgreich empfangen wurde. Durch dieses Prinzip kann der Sender nach einem Übertragungsfehler wieder schneller zur maximalen Übertagungsrate zurückkehren, weshalb man hier von Fast Recovery spricht.

Mit diesen vier Algorithmen hat man das Stauproblem vorübergehend in den Griff bekommen. Doch ist diese Evolution eher ein Notbehelf, als eine wahre Verbesserung der bestehenden Infrastruktur. Andere Protokolle, wie bspw. UDP werden in diesen Ansatz der Staukontrolle nicht mit einbezogen und können damit weiterhin das Netz überlasten. Da bis heute jedoch neben TCP kaum ein anderes Protokoll große Mengen an Netzlast verursacht hat, reichte diese Art der Staukontrolle bisher aus. Mit der immer stärkeren Nutzung von Video-Diensten, Onlinespielen und anderen UDP-basierten Anwendungen wird diese Evolution jedoch in absehbarer Zukunft an seine Grenzen stoßen.

IV. Der Domain Name Service

Die Einführung von DNS (=Domain Name Service) kann als eine echte Evolution angesehen werden. Die Nutzung einer eigenen Hosts-Datei blieb auch durch die Einführung des DNS möglich und am bereits bestehenden Teil des Internets mussten keine Änderung durchgeführt werden, die weitere Änderungen in anderen Teilen des Netzes nach sich gezogen hätten.

A. Die Hosts-Datei

Schon sehr früh wurde klar, dass die ursprünglich angedachte Lösung einer Hosts-Datei [7] zur Namensauflösung nicht ausreichen würde. Das exponentielle Wachstum des Internets machte die Idee einer zentral verwalteten, von Hand aktualsierten Liste, schnell obsolet.

Die Hosts-Datei würde schlicht und ergreifend zu groß werden und der Aufwand eine solche Datei auf dem aktuellen Stand zu halten, wäre innerhalb weniger Jahre mit unvorstellbarem Aufwand verbunden gewesen. War eine solche Liste in den Anfangszeiten des ARPAnets mit wenigen hundert Hosts noch praktikabel, so stellte sich bald heraus, dass eine andere Lösung gefunden werden musste.

Eine Möglichkeit wäre gewesen, ganz auf die Namensgebung zu verzichten, doch wer will und kann sich schon die Adresse 209.85.135.106 merken, um auf google.de zu gelangen? Auch hätte jeder Nutzer selbst eine Liste zur Benennung von IP-Adressen anlegen können. Doch wäre der Aufwand schon für wenige ausgewählte Seiten sehr hoch. Mit DNS hat sich jedoch eine bis heute praktizierte Lösung durchgesetzt.

B. Der Domain Name Service

DNS ist nichts Anderes, als ein weltweit verteilter Verzeichnisdienst, dessen ursprüngliche Aufgabe es war, die Hosts-Dateien durch eine effizientere Namensauflösung zu ersetzen. Durch den hierarchischen Aufbau (siehe Abb. 2) von DNS wurde dieses Problem gelöst und die Namensauflösung an unterschiedliche Administratoren deligiert.

In erster Linie wird DNS zur „Übersetzung“ von Domainnamen in IP-Adressen verwendet[4]. Gleiches gilt zwar auch für Hosts-Dateien, jedoch hat DNS den Vorteil, dass es linear skaliert und somit für das Internet deutlich besser geeignet ist. Zusammengefasst ergeben sich noch weitere Vorteile, hier die Wichtigstens im Überblick:

- Hierarchische Struktur (wodurch eine dezentrale Verwaltung umsetzbar ist)
- Eindeutigkeit
- Erweiterbarkeit

Sobald ein Client einen Namen auflösen möchte, stellt er eine Anfrage an einen DNS-Root-Server. Je nach Methodik, liefert dieser entweder den nächsten DNS-Server an den Client zurück (iterativ) oder er erfrägt beim nächsten DNS-Server die Adresse für den Client (rekursiv).

Durch spätere Erweiterungen des DNS konnten auch noch andere Anwendungen umgesetzt werden. So wurde es im Laufe der Zeit bspw. möglich, einem Domainnamen mehrere IP-Adressen zuzuweisen und somit per DNS die Last auf unterschiedliche Server zu verteilen[5] [8]. Diese und andere, später hinzugefügte Anwendungen wurden bei der Evolution der Hosts-Datei zu DNS jedoch noch nicht bedacht, weshalb hier nicht weiter auf sie eingegangen wird.

[4] Wobei der umgekehrte Weg auch möglich ist.
[5] Load Balancing

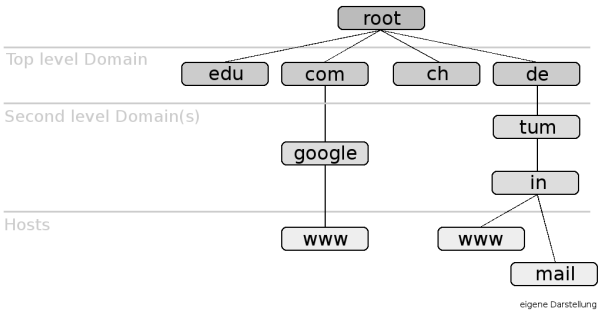


Abbildung 2. Teil des DNS-Namensraumes

V. BGP

Das BGP (=Border Gateway Protocol) wurde vor allem entwickelt, um sogenanntes policy routing zu ermöglichen. Dabei handelt es sich um die Fähigkeit eines Routing-Protokolls, verschiedene Regeln beim Routen von Paketen zu beachten.

Es wäre zum Beispiel denkbar, dass ein Paket von einem fremden System zu einem anderen fremden System geschickt werden soll und dieses Paket dabei das eigene autonome System durchlaufen will. Für zahlende Kunden wäre das durchaus in Ordnung. Nicht zahlende, fremde Parteien möchte man jedoch gerne daran hindern, da diese sonst das eigene System schnell überlasten könnten. Außerdem möchte man vielleicht Daten nur über Netzwerke anderer Parteien verschicken, falls es bei diesen zu einem geringen Preis möglich ist.

In der frühen Phase des Internets, als die Kommerzialisierung noch nicht weit vorangetrieben war, bestand hierfür keine Notwendigkeit. Durch Wachstum und Erweiterung auf immer neue Geschäftsfelder und Zielgruppen, wurde policy routing jedoch zwingend erforderlich. Erst dadurch war es wirtschaftlich sinnvoll, Anderen die eigene Netzinfrastruktur für deren Datenverkehr zur Verfügung zu stellen.

Heute wird BGP bereits in seiner vierten Version betrieben (kurz BGP4), welche ausschließlich zum Einsatz kommt. Der Vorteil von BGP4 gegenüber seinen Vorgängern ist die Möglichkeit, CIDR-Blöcken verarbeiten zu können (siehe Kaptiel VI).

VI. CIDR

Viele neue Technologien und Spezifikationen verdanken ihre Entwicklung und Nutzung vor allem dem rapiden Wachstum des Internets. Auch das Verfahren CIDR (=Classless Inter-Domain Routing) stellt hier keine Ausnahme dar. Anders als bei Classfull routing (mehr dazu in Abschnitt VI-A), werden bei CIDR die IPv4 Adressen weitaus weniger großzügig verteilt.

Durch eine frühzeitige Entwicklung und Einführung von IPv6 wäre die Entwicklung von CIDR für IPv4 nicht zwingend notwendig gewesen. Kurzfristig war es jedoch weniger Aufwendig, den bestehenden Adressraum besser aufzuteilen, als das alte Protokoll auf der Vermittlungsebene durch ein Neues zu ersetzen.

A. Class-A/B/C-Netze

Bis in das Jahr 1993 waren Class-A/B/C-Netze die verwendete Unterteilung des IPv4-Adressraums in kleinere Teilnetze. Da mit dieser Methodik der IPv4-Adressraum jedoch schon bald ausgeschöpft war, musste etwas passieren. Die Unterteilung in nur drei verschiedene Teilnetzgrößen war schlichtweg zu verschwenderisch, um dem rasanten Wachstum gerecht zu werden. Tabelle I zeigt, wie wenig Teilnetze nur mit Class-A -B und -C Netzen zur Verfügung stehen:

Tabelle I
ÜBERSICHT DER NETZKLASSEN

Netzklasse	Anzahl der Netze	Anzahl der Hosts pro Netz
Klasse A	128	16.777.214
Klasse B	16.384	65.534
Klasse C	2.097.152	254

Wie bereits erwähnt, wäre die wohl effektivste Lösung dieses Problems, der Umstieg auf einen größern Adressraum gewesen (siehe dazu Kapitel VII), denn selbst mit einer sparsameren Aufteilung der verfügbaren Adressen war abzusehen, dass ein Umstieg in nicht all zu ferner Zukunft notwendig wird. Eine solche Umstellung auf IPv6 ist jedoch nicht nur mit hohem Aufwand, sondern auch mit hohen Kosten verbunden, weshalb man sich für eine andere Lösung entschied.

B. Classless Inter-Domain Routing

CIDR teilt den Adressraum nicht in nur einige „wenige“ Teilnetze auf, sondern ermöglicht eine deutlich effizientere Nutzung des Adressraums. War es mit Classfull Routing notwendig, einer Organisation mit 400 Hosts ein Class-B-Netz zuzuteilen (und damit gut 65.000 IP-Adressen zu blockieren), so besteht mit CIDR die Möglichkeit, ein Teilnetz mit 512 IP-Adressen zu erzeugen. Damit wurde das Problem des zu kleinen Adressraums zwar nicht gelöst, aber zumindest in die Zukunft verschoben.

Neben diesem durchaus wichtigen Effekt, bot CIDR noch einen weiteren Vorteil. Für einen Router war es nun machbar, Routingtabellen deutlich zu verkleinert. Schließlich ist es mit CIDR häufig möglich, mehrere Adressen die über den gleichen Next Hop[6] erreicht werden, in einem Adressblock zusammenzufassen.

VII. INTERNET PROTOCOL VERSION 6

Die Umstellung von IPv4 nach IPv6 kommt nur schleppend voran. Hier handelt es sich um eine echte Evolution. Immer mehr Rechner und immer mehr Router werden für die neue Version des Internet Protocols „fit“ gemacht, weiterhin bleibt der Betrieb mit IPv4 möglich. Es gibt bereits einige Netzwerke, welche IPv6 für die Vermittlung verwenden und diese Netzwerke können ohne weiteres über 6to4-tunnel mit oder durch IPv4-Netzwerke kommunizieren. Dieser Parallelbetrieb ist jedoch aufwendig, was eigentlich zu einer raschen Umstellung führen sollte. Offensichtlich ist die Notwendigkeit jedoch

[6]Nächster bevorzugter Router, an dein ein IP-Paket weitergeschickt wird, falls es für den entsprechenden Adressbereich bestimmt ist.

noch nicht groß genug, um den Aufwand einer kompletten Umstellung zu rechtfertig.

Neben der wohl entscheidenden Änderung, endlich genug IP-Adressen zu Verfügung zu stellen[7], wurden bei der Entwicklung von IPv6 noch weitere wesentliche Ziele verfolgt:

- Verkleinerung von Routingtabllen
- Vereinfachung der Paket-Header, um Pakete von Routern schneller verarbeiten zu können
- Starke Unterstützung von Multicasting
- Die Möglichkeit für Host, ohne Adressänderung den Standort zu wechseln
- Sicherheitsfragen wurden berücksichtigt
- Abwärtskompatibilität

Obwohl, oder vielleicht gerade weil, IPv6 mit IPv4 über kleinere Umwege kompatibel ist, dauert die Umstellung schon sehr lange an und es ist heute noch nicht abzusehen, wann diese abgeschlossen sein wird. Da der Bedarf für IP-Adressen aber weiterhin stark steigt und große Teile der Erde bei der bisherigen Vergabe der Adressen nahezu leer ausgegangen sind, ist eine Umstellung auf lange Sicht unausweichlich.

VIII. Ausblick

Eins ist sicher, das Internet wird weiter wachsen und mit diesem Wachstum entstehen immer neue Anforderungen an Verfügbarkeit, Stabilität, Performanz, Kosten, Sicherheit,... weshalb immer wieder Änderungen an der bisherigen Architektur notwenig werden.

Doch in seiner jetzigen Form ist das Internet bereits viel zu groß, um eine Technologie komplett durch eine Andere zu ersetzen. Die Kosten wären nicht abschätzbar, die Kompatiblilität mit allen Teilnehmern im Netz könnte nicht gewährleistet werden und es ist nicht absehbar, ob eine neue Technologie in einem solchen Maßstab fehlerfrei funktioniert.

Für weitere Revolutionen, wie es sie das letzte Mal am 1. Januar 1983 gab, ist deshalb wohl kein Platz mehr. Mit Evolutionen kann man sich jedoch behelfen. Diese können die bereits funktionierende Infrastruktur nutzen, erweitern und verbessern, sowie alte Technologien - die ausgedient haben - nach und nach verdrängen.

Literatur

[1] "Internet systems consortium – www.isc.org/solutions/survey/history."
[2] James F. Kurose, Keith W. Ross, *Computer Networking – A Top-Down Approach*, 4th ed.
[3] M. Handley, "Why the internet only just works," *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, Juli 2006.
[4] S. Crocker, "Protocol notes," RFC 36, March 1970.
[5] Andrew S. Tanenbaum, *Computernetzwerke*, 4th ed., 2003.
[6] M. Allman, "Tcp congestion control," RFC 2581, April 1999.
[7] M. Kudlick, "Host names on-line," RFC 608, Januar 1974.
[8] T. Brisco, "DNS support for load balancing," RFC 1794, April 1995.

[7]Würde man die Erde mit mit IPv6-Adressen bedecken, wäre auf jedem Quadratzentimeter der Erdoberfläche Platz für etwa 667 Billiarden Adressen.

IP Fast Reroute

Deniz Ugurlu
Betreuer: Nils Kammenhuber
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: ugurlu@in.tum.de

***Kurzfassung*— Diese Arbeit beschreibt, wie lokaler Schutz für Datenverkehr in rein IP-basierten Netzwerken bei Auftreten einzelner Verbindungs- oder Routerausfälle erreicht werden kann. Das Ziel dabei ist, Ausfallzeiten und damit zusammenhängende Datenverluste im Falle solcher Fehler zu minimieren, während durch die Router neue optimale Pfade für die geänderte Netzwerktopologie berechnet werden. IP Fast Reroute beschreibt Techniken, welche ein schnelles Reagieren auf Ausfälle im Netzwerk durch Verwendung zuvor berechneter alternativer Next-Hops oder über Not-Via Adressen ermöglichen. Das Ausmaß des Schutzes (Node- und/oder Linkprotection) hängt dabei von der Topologie des jeweiligen Netzwerkes ab.**

***Schlüsselworte*— IPFRR, Fast Reroute, Link-Protection, Node-Protection, Micro Loop, Downstream Pfad**

I. Einleitung

Nach einer Verbindungsunterbrechung in rein IP-basierten Netzwerken benötigen Router mit aktuellen Methoden noch Zeiträume in der Größenordnung mehrerer hundert Millisekunden, um Reparaturpfade zu initialisieren und Daten über diese umzuleiten. Für z.B. multimediale Anwendungen wie VoIP oder Business-Lösungen reicht dies oft nicht aus, da diese empfindlich auf Datenverluste reagieren können, welche über zehntel Millisekunden hinausgehen.

Als IP Fast Reroute, folgend abgekürzt als IPFRR, werden Techniken bezeichnet, die eine möglichst schnelle Erholung von solchen Ausfällen ermöglichen. Dazu werden Alternativen für Verbindungen zwischen Routern berechnet, um Datenverkehr im Fehlerfall über diese abgespeicherten Reparaturpfade zu leiten. Andere Methoden verwenden Not-Via Adressen, um dadurch anzuzeigen, über welchen Punkt des Netzwerks ein Router, bei Störung einer gegebenen Verbindung, erreichbar ist. IPFRR dient dabei als übergangsweise Notlösung, während für die geänderte Netzwerktopologie neue optimale Pfade berechnet und eingesetzt werden können. Ziel beim Einsatz von IPFRR sind Datenverluste im Bereich von wenigen Zehntel Millisekunden.

Durch Verwendung alternativer Next-Hops soll ein Maximum an Fehlerfällen, etwa Link- oder Nodeausfälle, abgedeckt werden, wobei die Wahl der Absicherung von der Netzwerktopologie und dem aufgetretenen Fehler abhängt. Meist kann dadurch kein vollkommener Schutz eines Netzwerks gewährleistet werden.

Mittels der Not-Via Adressen hingegen ist eine Abdeckung von 100% des Netzwerkes möglich, allerdings wird hierfür ein deutlich höherer Rechenaufwand benötigt.

Im Folgenden wird in Abschnitt II die Terminologie erklärt, gefolgt von den unterschiedlichen Fehlerszenarios einschließlich der Zusammensetzung der Ausfallzeit in Abschnitt III. Die Abschnitte IV und V stellen IPFRR Verfahren im Detail vor, und veranschaulichen diese an ausgewählten Beispielen. Abschließend werden Vor- und Nachteile der Techniken gegenübergestellt und ein Ausblick auf die weitere Entwicklung gegeben.

II. Terminologie

Die in dieser Arbeit verwendeten Begriffe und Abkürzungen im Bezug auf IP Fast Reroute entsprechen den im IPFRR Framework von M. Shand und S. Bryant Eingeführten. [1]

Für den aktuell berechnenden Router, dem Ausgangspunkt also, wird die Bezeichnung S(ource), für den Router am Zielpunkt entsprechend D(estination) verwendet. Der primäre Next-Hop, also die Übertragung zum erste Knotenpunkt bzw. Router auf dem kürzesten Pfad von S zu D, wird als E notiert. Existieren mehrere kürzeste Pfade nennt man diese ECMP (equal cost multi-path) und deren primäre Next-Hops E1, E2, E3 etc. Der i-te Nachbar-Router zu S heißt Ni. Darüber hinaus werden die Abkürzungen LFA (loop-free alternate), SPF (shortest path first) und SPT (shortest path tree) verwendet. LFA steht dabei für einen kreisfreien Ersatz, dessen Berechnung und Funktionsweise in Abschnitt III erklärt wird. Mittels SPF werden von jedem Router individuell optimale Pfade aufgrund von Wegekosten bestimmt. Häufig wird dazu der Dijkstra-Algorithmus verwendet. Der resultierende Baum mit dem jeweils berechnenden Router als Wurzel ist der SPT. Um die optimale und kürzeste Distanz zwischen zwei Punkten A und B zu bestimmen, wird die Notation D_opt(A, B) eingeführt.

Abbildung 1 zeigt eine Beispieltopologie. Der kürzeste Pfad von S nach D läuft in diesem Fall über E.

III. Fehlerszenarios

IP Fast Reroute kann in einem Netzwerk Schutz gegen einzelne Ausfälle verschiedener Komponenten bieten. Dabei treten jedoch, abhängig von der Topologie des Netzwerks, unterschiedliche Problematiken auf, wenn maximaler Schutz gewährleistet werden soll. Um den Grad an Absicherung differenzieren zu können, werden verschiedene Arten von Fehlern unterschieden.

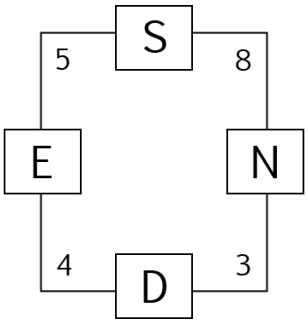


Abbildung 1. Einfache Topologie

A. Linkausfälle

Kommt es zum Ausfall einer Verbindung zwischen zwei Routern, also einem Linkausfall, gehen alle Daten verloren, welche weiterhin über diese Verbindung weitergeleitet werden.

B. Nodeausfälle

Von einem Nodeausfall spricht man, wenn ein Router nicht mehr erreichbar ist. Das führt dazu, dass alle Verbindungen zwischen diesem und anderen Routern oder Präfixen unterbrochen werden.

C. Fehlerdauer

Tritt einer dieser Fehler im Netzwerk auf, kommt es unwiederbringlich zu Verlust von Daten, da angrenzende Router den Fehler erst erkennen und dann den entsprechenden Datenverkehr umleiten müssen. Die Ausfallzeit setzt sich dabei aus den Zeiten zusammen, die vergehen, bis benachbarte Router den Fehler erkennen, den Ursprung identifizieren, die Informationen über den Fehler im Netzwerk verbreiten, neue SPF-Berechnungen durchführen und die Ergebnisse in den Routingtabellen installieren. Dabei entfallen weniger als 20ms auf das Feststellen des Fehlers, das Orten dauert unter 10ms. Bis die Information über die neue Topologie an alle Router im Netzwerk weitergeleitet, entsprechende SPF-Berechnungen neu durchgeführt und Routingtabellen aktualisiert wurden, verstreichen mit bisherigen Methoden mehrere 100ms, je nach Anzahl an Knoten und Präfixen. [2]

Mittels der IPFRR-Techniken soll an dieser Stelle erreicht werden, dass die Zeit nach dem Erkennen eines Fehlers bis zur Übernahme der SPF-Ergebnisse durch zuvor berechnete alternative Reparaturpfade ohne Datenverlust überbrückt werden kann.

IV. Loop Free Alternates

Tritt in einem Netzwerk ein Fehler auf, wird dieser zuerst von den direkt benachbarten Routern bemerkt. Unterstützen diese Router IPFRR, können deren gespeicherte Reparaturpfade verwendet werden, ohne dass andere Router über die Änderung informiert werden müssen.

Als Beispiel dient eine Topologie wie in Abbildung 1 zu sehen. Bei Berechnung des kürzesten Pfades von Router S zu D ergibt sich E als primärer Next-Hop. Da Router S IPFRR verwendet, versucht dieser, einen zusätzlichen alternativen Pfad zu D zu finden, bei dem die Verbindung zwischen S und E nicht durchlaufen wird. Ermöglicht wird das durch Router N. Kommt es zum Ausfall der Verbindung zwischen S und E und hat S dies bereits erkannt, wird der abgespeicherte alternative Next-Hop zu N als neuer primärer Next-Hop für Traffic mit Ziel D installiert. Diese Berechnung von alternativen Next-Hops wird analog von allen Routern und für alle Zieladressen durchgeführt.

A. Definition + Absicherung

Ein kreisfreier Pfad über einen alternativen Next-Hop (LFA) kann genau dann gefunden werden, wenn ein Nachbar des primären Next-Hop die Bedingung aus Ungleichung (1) erfüllt. [3]

$$D\_opt(N, D) < D\_opt(N, S) + D\_opt(S, D) \qquad (1)$$

Im o.g. Beispiel trifft dies auf Router N bereits zu, da $3 < 8 + 9$. Geht man aber von geänderten Wegekosten 30 zwischen N und D aus, ändert sich die Ungleichung zu $30 \not< 8+9$ und Router N kann keine Kreisfreiheit mehr garantieren. Alternative Next-Hops können einen unterschiedlichen Grad an Absicherung gegen Ausfälle bieten. Unterschieden wird dabei zwischen Absicherung gegen Verbindungsausfälle, was als Link-Protection bezeichnet wird, und Absicherung gegen Routerausfälle, auch Node-Protection genannt. Probleme bei Ausfällen können dann auftreten, wenn beispielsweise ein Routerausfall eintritt, der alternative Next-Hop aber nur Link-Protection bietet. In solchen Fällen können sich sogenannte Micro Loops bilden, bei denen Daten zwischen Routern im Kreis hin- und hergeleitet werden, bis diese verworfen werden.

B. Micro Loops

Anhand von Abbildung 2 lässt sich dies verdeutlichen.

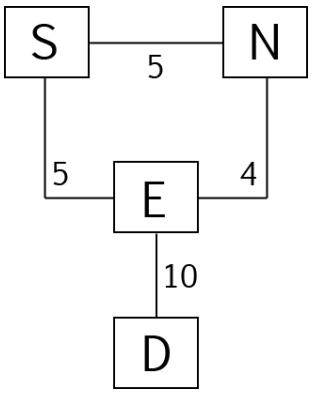


Abbildung 2. Entstehung von Micro Loops

Die Berechnung von Router S hat E als primären Next-Hop und N als alternativen Next-Hop zum Ergebnis. Diese werden abgespeichert und in seiner Routingtabelle eingetragen. Router N kann den Traffic dabei im Fall eines Verbindungsfehlers zwischen S und E absichern. Fällt aber Router E vollständig

aus, kann auch über den alternativen Next-Hop N keine Reparatur erfolgen. Tritt der Fall ein, dass E ausfällt, bemerken sowohl S als auch N dies und leiten ihren Traffic über die jeweils abgespeicherten Alternativen um. Das hat zur Konsequenz, dass S versucht, über N weiterzuleiten, während N versucht, Daten über S zu senden. Dieses Verhalten bezeichnet man als Micro Loop. [4]

Wie in [3] beschrieben, können nicht nur Fehler, die größere Ausmaße als geplant annehmen, sondern auch das gleichzeitige Auftreten mehrerer Fehler dazu führen, dass sich Micro Loops bilden, in denen Daten mit hoher Wahrscheinlichkeit verloren gehen.

1) Downstream Pfade: Eine Möglichkeit, die Bildung von Micro Loops zu verhindern, besteht darin, die Auswahl alternativer Next-Hops auf Router zu beschränken, welche sich auf Downstream Pfaden zwischen S und D befinden. Ein alternativer Next-Hop verläuft auf einem Downstream Pfad, falls Ungleichung (2) von dem Router erfüllt wird.

$$D\_opt(N, D) < D\_opt(N, S) + D\_opt(S, D) \qquad (2)$$

Alle Router entlang eines Downstream Pfades müssen also näher am Zielpunkt liegen bzw. haben geringere Wegekosten zum Ziel als ihr jeweiliger Vorgänger. Bezogen auf das Beispiel in Abbildung 2 bedeutet dies, dass von S zwar Router N als Downstream Alternative gewählt werden kann, jedoch umgekehrt N nicht S als Downstream Alternative verwenden darf. N hat deshalb keine Möglichkeit einen kreisfreien alternativen Next-Hop zu finden, weshalb der Traffic im Fehlerfall von N verworfen und die Entstehung eines Micro Loops verhindert wird. Es zeigt sich, dass in einem Netzwerk, in dem ausschließlich Downstream Pfade verwendet werden sollen, die Auswahl an alternativen Next-Hops drastisch eingeschränkt sein kann. Dadurch vermindert sich auch gleichzeitig die Anzahl an Reparaturwegen. Abgesehen von Downstream Pfaden bieten alternative Next-Hops mit Node-Protection eine weitere Möglichkeit, solche Micro Loops zu vermeiden.

C. Berechnung von Alternativen

Um für ein bestimmtes Ziel D eine LFA zu berechnen, benötigt ein Router diverse Informationen über kürzeste Pfade und deren Wegekosten. Die kürzeste Distanz zwischen S und D, D_opt(S, D), ist in der Regel nach der SPF-Berechnung durch das Routing-Protokoll direkt verfügbar. Entfernungen aus Sicht eines Nachbarrouters N, also D_opt(N, D) und D_opt(N, S), erhält man durch zusätzliche SPF-Berechnungen, welche aus Sicht von N durchgeführt werden. Das bedeutet, dass N als Wurzel des SPT angenommen wird und somit von N aus alle kürzesten Pfade im Netzwerk errechnet werden.

Um kreisfreie alternative Next-Hops nach [3] zu garantieren, muss mindestens Ungleichung (1) durch einen Nachbarrouter erfüllt sein. Dadurch wird sichergestellt, dass weitergeleiteter Traffic keinen Kreis bildet, falls eine Verbindung zwischen Routern ausfällt. Um gegen Ausfälle eines ganzen Routers E Schutz bieten zu können, ist es erforderlich, dass ein Nachbar N kreisfrei bezüglich E und D ist. Der Pfad von N zu D führt also nicht über E. Bietet N eine kreisfreie Alternative und wird zusätzlich Gleichung (3) erfüllt, nennt man N eine Node-Protecting LFA.

$$D\_opt(N, D) < D\_opt(N, E) + D\_opt(E, D) \qquad (3)$$

Wird hingegen Gleichung (4) erfüllt, ist es möglich, dass mehrere Pfade mit gleich niedrigen Wegekosten existieren. Es besteht somit auch die Möglichkeit, dass einer dieser Pfade Schutz bei Ausfall von E bietet. Gleich wahrscheinlich ist aber auch, dass ein anderer Pfad dies nicht gewährleistet. Da der berechnende Router keinerlei Einfluss auf die Wahl eines konkreten Pfades hat, muss davon ausgegangen werden, dass ein alternativer Next-Hop keine Node-Protection Eigenschaft besitzt, wenn Gleichung (3) erfüllt wird.

$$D\_opt(N, D) = D\_opt(N, E) + D\_opt(E, D) \qquad (4)$$

D. Equal-Cost Multipath

Existiert ein Equal-Cost Multipath (ECMP), folgt daraus, dass auch mehrere primäre Next-Hops verwendet werden, um Traffic zu einem bestimmten Ziel weiterzuleiten. Fällt einer dieser primären Next-Hops aus, sollte dessen alternativer Next-Hop verwendet werden. Die Alternative wiederum kann selbst einer der anderen primären Next-Hops sein, muss es aber nicht. Der Schutz anderer primärer Next-Hops reicht gegebenenfalls nicht aus, um den aktuell aufgetretenen Ausfall abzufangen.

In Abbildung 3 ist ein Aufbau mit drei primären Next-Hops E1, E2 und E3 zu sehen, bei denen sich die primären Next-Hops untereinander unterschiedlich starken Schutz bieten können.

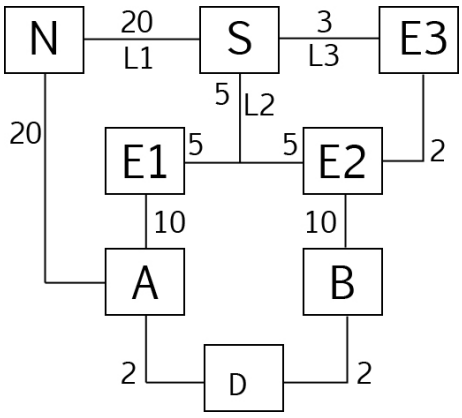


Abbildung 3. ECMP

Die Pfade über E1, E2 und E3 zu D bieten alle die selben Wegekosten. Die möglichen Absicherungen gegen Fehler verhalten sich dabei wie folgt:

- Der primäre Next-Hop L2 zu E1 kann von L3 zu E3, einem anderen primären Next-Hop, gegen Link- und Routerausfälle abgesichert werden.

- Verbindung L2 zu E2 wird von L2 zu E1 lediglich gegen Routerausfälle und durch L3 zu E3 gegen Linkausfälle abgesichert.
- L3 zu E3 kann von L2 zu E1 und auch L2 zu E2 gegen beide Fehlerarten geschützt werden.
- Mit L1 als alternativem Next-Hop werden sowohl für L2 zu E1 als auch für L2 zu E2 Link- und Routerausfälle abgedeckt.

E. Broadcast Links

Bei Punkt-zu-Punkt Interfaces gilt für kreisfreie alternative Next-Hops, dass ein Router mit Node-Protection Eigenschaft immer auch Link-Protection liefert. Ein Problem tritt allerdings bei sog. Broadcast Links auf, d.h. eine Verbindung von einem zu mehreren anderen Routern. Abbildung 4 zeigt eine Topologie, bei der von S ausgehend ein Broadcast Link zu N und E besteht. Dieser Link wird als Pseudonode(PN) dargestellt, dessen Wegekosten zu angrenzenden Routern 0 sind, um eine Berechnung von kreisfreien Alternativen zu ermöglichen.

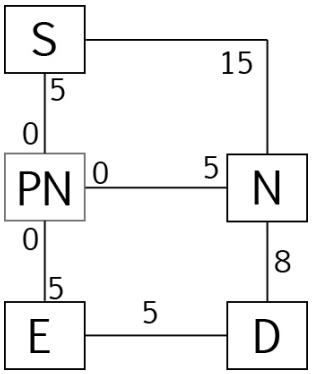


Abbildung 4. Broadcast Link

In Topologien mit Broadcast Links muss zur Untersuchung und zum Nachweis von möglichen Absicherungen eine genauere Betrachtung als bei simplen Punk-zu-Punkt Netzwerken erfolgen. Im o.g. Beispiel zeigt sich, dass Router N eine kreisfreie Link-Protecting Alternative bezüglich S und der Pseudonode darstellt. Allgemein lässt sich für einen Router N, der einen Broadcast Link kreisfrei absichern soll, folgende Bedingung aufstellen:

$$D\_opt(N, D) = D\_opt(N, PN) + D\_opt(PN, D) \qquad (5)$$

Der kürzeste Pfad der Pseudonode zu D verläuft über Router E, daher ist ein Nachbarrouter, der den Ausfall von E absichert, gleichzeitig auch kreisfreie Link-Protecting Alternative. Eine Ausnahme davon besteht nur, wenn der einzig mögliche Pfad von Router S zu Router N über die selbe Pseudonode führt. Dies ist der einzige Fall, in dem Node-Protection nicht aber Link-Protection durch einen Router vorliegt. Um in solch einer Topologie Absicherung gegen Verbindungsausfälle zu ermöglichen, ist es notwendig, dass sowohl der Pfad des ausgewählten alternativen Next-Hop als auch die Verbindung zwischen S und dieser Alternative nicht über den Broadcast Link verlaufen. Da bei Netzwerken mit Punkt-zu-Punkt Verbindungen Letzteres nicht möglich ist, muss für primäre Next-Hops, welche über Broadcast Links erreicht werden, bei der Auswahl der Alternativen beachtet werden, ob diese gegen Verbindungsausfälle absichern können. Anhand von Abbildung 4 kann dies beispielhaft erläutert werden.

Um über N gegen Verbindungsausfall des Broadcast Links abzusichern, darf der kürzeste Pfad von N nach D nicht über die Pseudonode verlaufen. Des Weiteren darf auch der von S gewählte alternative Next-Hop die Pseudonode nicht durchlaufen. Der kürzeste Pfad von S zu N verläuft im Beispiel allerdings über die Pseudonode, das heißt, S muss einen Next-Hop zu N finden, der die Pseudonode vermeidet, dafür aber höhere Wegekosten in Anspruch nehmen.

F. Auswahlverfahren

Bei der Auswahl der alternativen Next-Hops durch Router, welche die Spezifikation [3] unterstützen, soll für jeden primären Next-Hop versucht werden, mindestens eine LFA zu ermitteln. Gleichzeitig soll eine maximal mögliche Abdeckung von Fehlerfällen durch diese LFA erzielt werden. Dies wird dadurch erreicht, dass Router S bei der Berechnung diejenige LFA bevorzugt wählt, welche Node-Protection bietet. Ist solch eine LFA nicht verfügbar, kann eine kreisfreie Alternative gewählt werden, die Verbindungsausfälle absichert. Für den Fall, dass eine kreisfreie Alternative sowohl Link- als auch Node-Protection bietet, und eine zweite Alternative nur mit Node-Protection nicht aber Link-Protection zur Auswahl steht, sollte die Link- und Node-Protecting LFA von S gewählt werden.

Existieren mehrere primäre Next-Hops, sollte entweder einer der anderen primären Next-Hops oder eine kreisfreie Alternative mit Node-Protection als LFA gewählt werden. Ist solch eine Alternative nicht verfügbar und kann keiner der anderen primären Next-Hops die Verbindung schützen, sollte eine kreisfreie Alternative Link-Protection verwendet werden. Durch Anwendung dieser Priorisierung wird ein größtmöglicher Ausfallschutz ermöglicht.

G. Einsatzdauer der Reparaturpfade

Die Dauer, über die ein Router einen alternativen Next-Hop verwenden sollte, sobald sein primärer Next-Hop ausgefallen ist, muss begrenzt werden. Dadurch wird sichergestellt, dass die optimalen Pfade durch die SPF-Berechnung der geänderten Netzwerktopologie installiert und verwendet werden.

Bei der Umstellung der alternativen Next-Hops auf die neu berechneten primären Next-Hops muss jedoch beachtet werden, dass sich Micro Loops bilden können, wenn Router S den neuen primären Next-Hop sofort verwendet, während bei anderen Routern noch SPF-Berechnungen andauern. Geht man z.B. von einer Topologie wie in Abbildung 5 aus und die Verbindung von S zu E wird unterbrochen, kommt N1 als alternativer Next-Hop zum Einsatz.

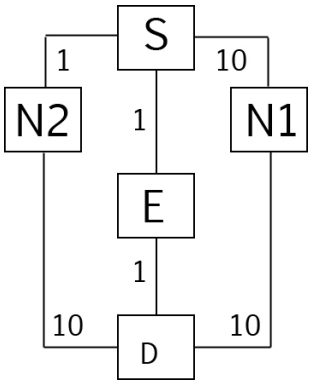


Abbildung 5. Einsatzdauer von LFAs

Bei der darauf folgenden SPF-Berechnung ergibt sich N2 als neuer primärer Next-Hop für das Ziel D. Wird in Router S sofort nach Berechnung N2 als neuer primärer Next-Hop installiert, ist es wahrscheinlich, dass bei N2 noch die primären Next-Hops der alten, fehlerfreien Topologie im Einsatz sind. Dies resultiert darin, dass S seinen Traffic für D über N2 weiterleitet, während bei N2 der optimale Pfad zu diesem Ziel noch über S installiert ist und somit ein Micro Loop entsteht. Dieser Kreis besteht solange, bis in N2 die aktualisierten primären Next-Hops übernommen wurden. Unter der Annahme nur einzeln auftretender Fehler kann solch ein Verhalten unterbunden werden, indem die Verwendung der neuen primären Next-Hops von Router S verzögert, und der Traffic solange weiter über die alternativen Next-Hops geleitet wird. Die alternativen Next-Hops können durch die neu Berechneten ersetzt werden, falls der neue primäre Next-Hop kreisfrei bezüglich der Topologie vor Auftreten des Fehlers ist oder eine vorher definierte Zeitspanne abgelaufen ist. Diese Zeitspanne sollte eine Obergrenze für die worst-case Laufzeit des Übergangs während der Netzwerkkonvergenz darstellen. Abgesehen davon kann der neue primäre Next-Hop sofort verwendet werden, falls der betroffene Router über einen unabhängigen neuen Fehler im Netzwerk informiert wurde.

H. Präfixe in Multihomed Netzwerken

Bei Multihomed Präfixen, d.h. Präfixe, die über mehrere verschiedene Router erreichbar sind, kann es bei der Berechnung der Alternativen zu einer nicht optimalen Lösung kommen.

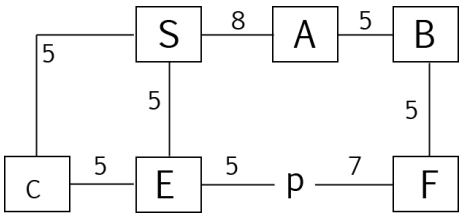


Abbildung 6. Multihomed Präfixe

Beispielsweise verläuft in Abbildung 6 der kürzeste Pfad zwischen S und p über Router E. Wird hierfür eine Alternative nur über den Router mit dem kürzesten Pfad zu p gesucht, erhält man C als LFA. Dieser ermöglicht Link-Protection, im Gegensatz zu A, welcher zwar einen längeren Pfad zu p hat, aber gegen kompletten Ausfall von E absichern kann. Daher ist es bei Multihomed Präfixen wichtig, alternative Pfade auch über alle anderen Router zu suchen, an welche das Präfix angeschlossen ist. Durch dieses Verhalten kann wieder maximal möglicher Schutz erreicht werden.

I. Sicherheit

Bei Verwendung der LFA Mechanismen für IP Fast Reroute werden keinerlei Änderungen an den Nachrichten des Routing-Protokolls vorgenommen. Es entstehen dadurch also keine zusätzlichen Sicherheitslücken bezüglich z.B. Paket-Änderungen oder Replay-Attacken. [3]

Auch die übergangsweise Verwendung von Next-Hop Routern, welche bei einer Änderung im Netzwerk ohne IPFRR nicht eingesetzt würden, kann nicht als zusätzliche Gefährdung betrachtet werden, da diese bei entsprechenden Ausfällen ohnehin verwendet werden. Mittels der beschriebenen Technik zur Verwendung von LFAs kann laut [1] in 80% der Fehlerfälle ein Reparaturpfad gefunden werden.

V. Not-Via Adressen

IPFRR mittels LFAs bietet keine Möglichkeit, den Ursprung eines Fehlers aufzuzeigen, an andere Router weiterzuleiten und Reparaturpfade explizit um diesen Punkt herum zu leiten. Dies beeinflusst auch die Abdeckung an Fehlerfällen in Abhängigkeit der Topologie eines Netzwerks.

Die im Folgenden vorgestellte Technik der Not-Via Adressen ermöglicht es, Pakete im Fehlerfall zu kapseln und an eine Netzwerkkomponente zu senden, welche vom betroffenen Router über seine Not-Via Adresse angegeben ist. Wird das Netzwerk durch den Ausfall nicht partitioniert, ist mit Hilfe von Not-Via Adressen in jedem Fall eine Reparatur möglich.

Not-Via Adressen werden von jedem Router in einem Netzwerk für jede Verbindung einzeln angegeben. So erhält ein Router B für die Verbindung zwischen A und B zusätzlich zu seiner regulären IP Adresse eine sog. Not-Via Adresse. Diese Adresse zeigt an, über welchen Knoten im Netzwerk Router B erreichbar ist, ohne dass die Verbindung von A zu B verwendet wird.

A. Berechnung mittels Not-Via Adressen

Bei diesem Mechanismus ist es notwendig, dass jeder Router für einen Ausfall jedes anderen Routers einen Pfad über dessen Not-Via Adresse berechnet, den er dann im Fehlerfall sofort verwenden kann. Dazu simuliert ein Router X nacheinander Ausfälle der anderen Router im Netzwerk und berechnet einen neuen SPF mittels der Not-Via Adresse. Für ein Netzwerk mit n Routern bedeutet dies, dass jeder Router n-1 zusätzliche SPF-Berechnungen durchführen muss, weshalb diese Methode vergleichsweise rechenintensiv ist.

In [2], [5] werden jedoch Optimierungen zur Reduzierung des Rechenaufwandes bei der alternativen Wegfindung vorgestellt. Dazu wird in einem ersten Schritt überprüft, ob die

zugehörige Verbindung der fraglichen Not-Via Adresse in der aktuellen Topologie verwendet wird. Ist dies nicht der Fall, muss natürlich keine neue Berechnung von kürzesten Pfaden hierfür erfolgen. Eine weitere deutliche Reduktion der Komplexität bietet der Einsatz des ISPF Algorithmus (Incremental SPF, [6]) in Verbindung mit vorzeitiger Terminierung statt einer normalen SPF-Berechnung.

Der ISPF Algorithmus berechnet lediglich diejenigen Pfade neu, welche von der vorliegenden Topologieänderung betroffen sind. Vorzeitige Terminierung bedeutet in diesem Zusammenhang, dass die Berechnung abgebrochen wird, sobald der Pfad zur Not-Via Adresse gefunden wurde.

Möchte man Not-Via Adressen und LFAs zusammen verwenden, existiert noch eine weitere Möglichkeit zur Optimierung. Für einen Link, welcher durch die Not-Via Adresse gesichert werden soll, kann überprüft werden, ob bereits ein LFA berechnet wurde, welcher statt der Not-Via Adresse zu Reparaturzwecken verwendet werden kann. [5]

B. Vor- und Nachteile

Es zeigt sich also, dass sowohl der Einsatz von LFAs als auch Not-Via Adressen Vor- und Nachteile hinsichtlich Ressourcenaufwand und Größenordnung der Absicherung besitzen. Soll ein Netzwerk ausschließlich mittels Not-Via Adressen abgesichert werden, beträgt der erforderliche Rechenaufwand in einem realen Umfeld (Backbone mit 600 Nodes, alle Links abgesichert) etwa das 15-fache eines vollständigen SPF. [5] Andererseits ist es dadurch möglich, 100% aller Verbindungen in allen Topologien abzusichern (Ausnahme: Partitionierung des Netzwerks durch den Ausfall).

LFAs bieten dagegen eine Absicherung von etwa 70-80%, der benötigte Aufwand bei deren Berechnung ist allerdings signifikant niedriger.

VI. Zusammenfassung und Ausblick

Aufgrund der Vor- und Nachteile von kreisfreien Alternativen sowie Not-Via Adressen gibt es Ansätze, beide Methoden zu kombinieren. Dabei werden kreisfreie Alternativen berechnet, um den Großteil eines Netzwerkes abzusichern. Für die restlichen Verbindungen können danach Not-Via Adressen eingesetzt werden, um die vollständige Abdeckung an Fehlerfällen zu ermöglichen. Somit wird mit annehmbarem Aufwand die größtmögliche Sicherheit erreicht.

Da IPFRR eine vergleichsweise junge aufkommende Technologie darstellt, gibt es in realen Umgebungen meist Probleme, da nicht alle Router-Hersteller ISPF Implementierungen unterstützen und nicht alle Router über ausreichende Rechen- sowie Speicherkapazitäten verfügen. Davon abgesehen müssen für den Einsatz von Not-Via Adressen alle Router untereinander kompatibel sein.

Das Ziel, die Ausfallzeiten in Netzwerken zu verringern, kann IPFRR bei großer Fehlerabdeckung aus technischer Sicht bereits erreichen. Die Entwicklung von IPFRR wird z.B. von Cisco und der IETF vorangetrieben. Cisco unterstützt IPFRR LFA im seinem Routermodell CRS-1 [7], und viele IPFRR bezogene Drafts bzw. RFCs der IETF werden von Cisco Mitarbeitern mitgestaltet. [1], [5]

Weitere Verbesserungen an der Reaktionszeit zwischen dem Auftreten eines Fehlers und der Reparatur können durch Verringerung im Bereich der Fehlererkennung erreicht werden, welche die meiste Zeit beim IP Fast Rerouting beansprucht. [2]

Literatur

[1] M. Shand and S. Bryant, “IP Fast Reroute Framework,” IETF, draft-ietf-rtgwg-ipfrr-framework-12, June 2009.

[2] S. Previdi, “IP Fast reroute technologies,” Cisco-Talk, 2006.

[3] A. Atlas, “Basic specification for IP fast reroute: Loop-free alternates,” RFC 5286, September 2008.

[4] *IP Fast Reroute: Overview and Things We Are Struggling to Solve*. NANOG, January 2005.

[5] M. Shand, S. Bryant, and S. Previdi, “IP Fast reroute using not-via addresses,” IETF, draft-ietf-rtgwg-ipfrr-notvia-addresses-04, July 2009.

[6] J. McQuillan, I. Richer, and E. Rosen, “ARPANET routing algorithm improvements,” BBN Technical Report 3803, 1987.

[7] *IOS XR Routing Configuration Guide, Release 3.8*, Cisco.

DCCP - Datagram Congestion Control Protocol

Benjamin Peherstorfer
Betreuer: Andreas Müller
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: pehersto@in.tum.de

***Kurzfassung*— Für zeitkritische Anwendungen wie VoIP oder Online-Games wird immer öfter UDP eingesetzt. Diese Anwendungen sind darauf angewiesen, dass Pakete innerhalb eines gewissen Zeitfensters beim Empfänger eingehen, und damit ist eine erneute Übertragung von verlorenen Paketen meist unerwünscht. Durch den vermehrten Einsatz von UDP werden immer größere Datenmengen mit UDP übertragen, was zu Stauproblemen im Internet führen kann. Mit DCCP soll dies verhindert werden. Dieses neue Protokoll verbindet die Vorteile von UDP mit Staukontrolle. DCCP bietet eine Art Framework, auf welches man unterschiedliche Staukontroll-Mechanismen aufbauen kann. Die Möglichkeit aus verschiedenen Staukontroll-Mechanismen auszuwählen, soll Entwickler von Anwendungen den Umstieg von UDP auf DCCP erleichtern.**

***Schlüsselworte*— UDP, DCCP, Staukontrolle, TFRC, CCID, timeliness**

I. Einleitung

Der Datenverkehr im Internet wird durch zwei Protokolle auf der Transportschicht dominiert: TCP und UDP. Die früheren Protokolle benutzen UDP wegen des geringen Overheads. Bei DNS (oder auch SNMP) werden nur geringe Datenmengen verschickt und auf jede Anfrage folgt genau eine Antwort. Den teuren und langwierigen Verbindungsaufbau (*startup delay*) von TCP kann man mit UDP umgehen. Außerdem schützt die Zustandslosigkeit (*statelessness*) vor Angriffen (z.B. SYN-Flooding). Dafür handelt man sich die fehlende Zuverlässigkeit ein.

In letzter Zeit haben sich jedoch Anwendungen (z.B. VoIP, Multimedia Streaming, Online-Games) vermehrt aus anderen Gründen für UDP entschieden: Diese Anwendungen verschicken Daten, die extrem zeitkritisch sind. Kommt ein Paket innerhalb einer vorgegebenen Zeitspanne nicht beim Empfänger an, ist es nutzlos (*timeliness over reliability*). Daher braucht die Anwendung absolute Kontrolle über die versendeten Pakete, welche erst durch die Unzuverlässigkeit von UDP ermöglicht wird. Erst beim Versenden des Pakets entscheidet die Anwendung, was als Payload mitgegeben wird (z.B. die letzten Ortsinformation der Spielfiguren bei einem Online-Game). Des Weiteren spielt ein Datenverlust bei solchen Anwendungen keine große Rolle. So bevorzugen die meisten Benutzer von Videostreaming einen kurzen Qualitätsabfall gegenüber dem Anhalten des Bildes. Dies sind alles Gründe, warum TCP für solche Anwendungen nicht geeignet ist und man sich daher für UDP entschieden hat.

Von Anwendungen wie DNS und SNMP wurde UDP nur für geringe Datenmengen benutzt. Auch das hat sich geändert; zwischen einem Video in HD-Qualtität und einer DNS Abfrage ist ein großer Unterschied. Damit wird UDP auch zur Übertragung großer Datenmengen benutzt. Und das kann zu einem Stauproblem im Internet führen. Eine Staukontrolle ist daher wünschenswert. Man spricht von Stau (*congestion*), falls soviele Pakete bei einem Router ankommen, dass dieser Pakete verwerfen muss [1]. Aber auch wenn dieser Extremfall nicht eintritt, kann eine Staukontrolle hilfreich sein. Teilen sich z.B. mehrere Verbindungen einen Kanal mit fester Bandbreite und wollen alle diese Verbindungen eine konstante Übertragungsrate erreichen, so müssen sich diese selbst beschränken. Vor allem bei VoIP und Videostreaming ist eine konstante Übertragungsrate wichtig.

II. Abgrenzung und Problemformulierung

A. Anforderungen und Probleme von UDP mit Staukontrolle

Die Staukontrolle soll für Anwendungen mit einem langlebigen Datenfluss und viel Datenverkehr konzipiert werden. (Anwendungen, welche diese Annahmen nicht erfüllen (DNS, SNMP), verursachen kaum Datenverkehr und damit keinen Stau [2].) Grob kann man hier drei Typen von Anwendungen unterscheiden [3].

- *Internet-Telefonie*: Hier wird pro Zeiteinheit eine feste Anzahl an Bits übertragen. Der Benutzer merkt bereits kleinste Verzögerungen bei der Ankunft neuer Pakete. Eine Pufferung ist nicht möglich, da der Inhalt der Pakete im Voraus nicht bekannt ist. Oft werden Audio-Codecs verwendet, die eine extreme Komprimierung der Payload erlauben. Dies führt zu sehr vielen kleinen Paketen. Je kleiner die Pakete, desto kleiner sollte auch der Header sein.
  Eine schnelle Anpassung an mehr Bandbreite ist nicht nötig. Genauso soll aber auch die Begrenzung der Bandbreite während einer Stausituation nicht zu schnell geschehen. Da meist während einer Sprechpause gar nichts übertragen wird, muss die gesamte Bandbreite bei der Wiederaufnahme des Gesprächs sofort zur Verfügung stehen (also keine langen Startup-Delays).

- *Audio- und Videostreaming*: Ist ein Zeitstück des Videos beim Empfänger abgespielt worden, muss bereits das Paket mit dem nächsten Zeitstück des Videos vom Sender vorliegen. Kommt es zu einem späteren Zeitpunkt, ist es nicht mehr nützlich. Durch Pufferung, verschiedene Video-Layer (z.B. *key* und *incremental frames* bei MPEG [3]) und ähnliches, können Paketverluste ausgeglichen werden.
- *Online-Games*: Meist wird hier die Ortsinformation von den Spielfiguren in der virtuellen Welt übertragen. Dabei muss das letzte angekommene Paket den letzten Ort der Spielfigur beinhalten. Erneutes Senden (auch in sehr kurzen Zeitabständen) macht keinen Sinn. Außerdem kann erst beim Absenden des Pakets die Anwendung entscheiden, was es enthalten soll. Da es hier durchaus von Vorteil ist, wenn viel Bandbreite vorhanden ist, sollte sich die Staukontrolle schnell an neue Verhältnisse anpassen und neue Bandbreite auch sofort zur Verfügung stellen.

Es gilt, die Vorteile der unzuverlässigen Datenübertragung von UDP zu erhalten (*timeliness over reliabilty*), aber trotzdem eine effektive Staukontrolle darauf aufzubauen. Dies sind völlig widersprüchliche Ziele: zum einen will die Anwendung die absolute Kontrolle des Datenverkehrs haben, zum anderen will die Staukontrolle den Datenverkehr begrenzen, falls es zu einer Stausituation kommt.
Man hat sich in [2] daher für einen Kompromiss entschieden: Damit die Staukontrolle überhaupt funktionieren kann, entscheidet sie, wann ein Paket verschickt wird (sie kann also den Datenverkehr begrenzen), die Anwendung entscheidet aber nach wie vor, welche Daten in diesem Paket stehen. (Das ist bei TCP nicht möglich![1])

Verschiedene Anwendungen haben verschiedene Anforderungen an die Staukontrolle, d.h. wie sich der Staukontroll-Mechanismus bei Stau verhält. Die Anwendung soll daher die Wahl zwischen verschiedenen Staukontroll-Mechanismen für jeden Datenfluss haben. Des Weiteren soll die Hinzunahme von neuen Algorithmen wenig Änderungen an der Anwendung oder dem Protokoll erfordern, so dass zukünftige Algorithmen schnell einfließen könnnen.
Bereits um die Auswahl des Algorithmus zu ermöglichen, muss eine Möglichkeit geboten werden, um Features des derzeitigen Datenflusses auszuhandeln. Dafür führt man *Feature Negotiation* ein. Diese ermöglicht es zuverlässig verschiedenste Parameter auszuhandeln, wie z.B. Senderate der Acknowledgements.

Explicit Congestion Notification (ECN) soll unterstützt werden. Mit ECN ist es möglich, dass Router Stau signalisieren, ohne Pakete zu verwerfen. Beim Senden eines Pakets, setzt der Sender zufällig eines von zwei Bits im IP Header. Erreicht das Paket einen Router in einer Stausituation oder am Beginn einer Stausituation, so setzt dieser beide Bits auf Eins. Der Empfänger reicht schließlich diese Information an den Sender weiter (z.B. beim Acknowledgement hinzugefügt). [5] Der Sender kann damit die Stausituation im Netz abschätzen. Das zufällige Setzen von einem der zwei Bits, verhindert das Fälschen der Information durch den Empfänger (ECN Nonces). [6]

Weitere typische Anforderungen sind z.B. den Header-Overhead möglichst gering halten, da Anwendungen dieser Art oft sehr viele kleine Pakete senden und daher ein großer Header besonders schwer wiegt (z.B. Verwenden von verkürzten Sequenznummern). Aber auch das Zusammenspiel mit Firewalls und NAPTs soll besser sein als bei UDP, wo dies auf die Anwendungsschicht verlagert wurde, da UDP verbindungslos ist.

B. Staukontrolle für ein unzuverlässiges Transportprotokoll

Um Staukontrolle und UDP zu kombinieren, gibt es drei Möglichkeiten: (1) man integriert die Staukontrolle in die Anwendung (also über UDP), (2) die Staukontrolle arbeitet unter UDP, am besten auf einer neuen Schicht zwischen IP und UDP, oder (3) man führt auf der Transportschicht ein neues Protokoll ein, welches UDP ersetzt und die oben angeführten Forderungen erfüllt.

1) Staukontrolle in der Anwendung: Die Staukontrolle ist direkt in der Anwendung und kann daher genau auf den Aufgabenbereich angepasst werden. In [2] wird jedoch davon ausgegangen, dass die meisten Entwickler auch mit einer Standardlösung zufrieden sind, falls diese die Möglichkeit bietet, aus verschiedenen Staukontroll-Mechanismen zu wählen. Der große Nachteil der Staukontrolle in der Anwendung ist, dass für jede Anwendung die Staukontrolle neu implementiert werden muss. Die Staukontrolle ist aber ein sehr komplexes und fehleranfälliges System, wie man an den zahlreichen fehlerhaften Implementierungen von TCP gesehen hat [2], [7]. Weiter wird in [2] argumentiert, dass man eher dazu geneigt ist, die Staukontrolle zu umgehen, wenn sie in der Anwendung integriert ist, als wenn sie im Netzwerk-Stack/Betriebssystem liegt.
Ein technisches Problem ergibt sich außerdem noch mit ECN. Um die ECN Fähigkeit zu nutzen, müsste der Anwendung direkter Zugriff auf den IP Header gegeben werden. Selbst wenn ECN auf einem Netzwerkpfad nicht verfügbar ist, bieten die ECN Nonces immer noch eine Möglichkeit zu überprüfen, ob der Empfänger auch alle Paketverluste korrekt mitteilt. Dies müsste man ebenfalls aufgeben. Weitehin ist damit das Firewall-Problem von UDP nicht gelöst; wieder müssten Firewalls ein Protokoll der Anwendungsschicht ansehen, um Verbindungen erkennen zu können.
Von derzeitigen Anwendungen wird meist eine sehr grobe Staukontrolle direkt in der Anwendung benutzt. So wechselt z.B. Skype die Codecs, falls die Bandbreite kleiner wird [8].

[1]TCP_NODELAY muss sich der Staukontrolle unterordnen, siehe z.B. Abschnitt 4.2.3.4 in [4].

2) Staukontrolle unterhalb von UDP: Eine Grundvoraussetzung für jede Staukontrolle ist ein Feedback-Mechanismus (z.B. Acknowledgements). Hierfür werden Sequenznummern benötigt. Will man die Staukontrolle unter UDP einordnen, so muss der Feedback-Mechanismus entweder in der Anwendung stattfinden und die Informationen an einen *Congestion Manager* weitergeben, oder man verlegt auch den Feedback-Mechanismus selbst unter UDP. Ein Feedback-Mechanismus in der Anwendung hat die gleichen Probleme als würde man die Staukontrolle in die Anwendung integrieren. Verlegt man den Feedback-Mechanismus unter UDP, so muss man irgendwo die Sequenznummern unterbringen. Dazu braucht man einen zusätzlichen Header. Man müsste dann auch eine Protokollnummer im IP Header dafür reservieren und hätte damit schon fast ein neues Protokoll eingeführt. Lösungen um den Header im Optionen-Feld von IP zu verstecken, führen vermutlich zu Problemen bei verschiedenen IP Implementierungen, siehe [2].

3) Staukontrolle auf der Transportschicht: Insgesamt spricht daher alles für eine Implementierung der Staukontrolle auf der Transportschicht. Anwendungen müssten wenig an ihrem Code ändern, um das neue Protokoll zu verwenden. Auch an der Struktur unter der Transportschicht müsste nichts geändert werden, d.h. einer schnellen Ausbreitung des Protokolls stehen keine langwierigen Änderungen an Routern usw. bevor.

C. Staukontrolle auf der Transportschicht

Auch wenn nun entschieden ist, dass die Staukontrolle auf der Transportschicht stattfinden soll, so bleibt immer noch die Frage offen, ob man ein neues Protokoll entwickelt, oder ein bekanntes modifiziert. Zwei Protokolle bieten sich hierfür an: TCP und SCTP.

Bei TCP sind sämtliche Mechanismen (Flusskontrolle, Staukontrolle, Zuverlässigkeit usw.) ineinander verwoben. Lässt man etwas davon weg, funktionieren alle anderen nicht mehr. Außerdem benutzt TCP Sequenznummern, welche Bytes im Stream zählen, hier will man aber Datagramme. Man will vielleicht sogar erlauben, dass Datagramme in der falschen Reihenfolge beim Empfänger ankommen (*out-of-order delivery*). Zusätzlich setzen sich Änderungen von TCP nur sehr langsam durch. In [2] wird daher empfohlen, TCP so zu lassen wie es ist.

Auch gegen SCTP finden sich genügend Argumente. Da SCTP das Bündeln von mehreren Streams erlaubt, beinhaltet der Header viel Information (viel Overhead), die hier nicht gebraucht wird. Auch die Wahl von verschiedenen Staukontroll-Mechanismen wird von SCTP nicht unterstützt. Größere Änderungen an SCTP könnten außerdem vorhandene Implementierungen durcheinander bringen.

Nach diesen Überlegungen kommt man zu dem Schluss, dass die Entwicklung eines neuen Protokolls die beste Lösung ist: Das Datagram Congestion Control Protocol (DCCP), welches unzuverlässige Datenübertragung mit Staukontrolle (und verschiedenen Staukontroll-Mechanismen) bietet, sowie ECN benutzt und darüber hinaus noch geringen Overhead hat.

III. Datagram Congestion Control Protocol (DCCP)

A. Überblick und Funktionsweise

DCCP wird in RFC 4340 [6] vorgestellt und definiert. Man unterscheidet die DCCP Kernfunktionalitäten, welche in RFC 4340 beschrieben sind und den Staukontroll-Mechanismus oder -Algorithmus. Der Staukontroll-Mechanismus kann je nach Anwendung beliebig ausgetauscht werden und baut auf den Kernfunktionalitäten wie z.B. Feedback-Mechanismus, Verbindungsaufbau (Handshake) oder Sequenznummern auf. So können neue Staukontroll-Algorithmen entwickelt werden, ohne die Kernfunktionalitäten von DCCP zu ändern.

Sollen Daten mit DCCP übertragen werden, wird eine Verbindung aufgebaut. Ein Acknowledgement Framework informiert den Sender, wieviele und warum Daten/Datagramme verloren gingen. Es werden also Pakete nicht einfach nur bestätigt, sondern es wird zusätzliche Information an den Sender geleitet, so dass beim Sender der Staukontroll-Mechanismus genug Information hat, um seine Staukontrolle korrekt durchzusetzen. DCCP unterscheidet zwischen verschiedenen Arten von Paketverlusten: Daten korrupt, Protokoll Spezifikation verletzt, Puffer voll usw. Eine erneute Übertragung eines verloren gegangen Pakets muss jedoch die Anwendung selbstständig durchführen.

B. Pakettypen und Header

Bei DCCP wird zwischen zehn verschiedenen Pakettypen unterschieden, deren Header sich jeweils unterscheiden. So werden z.B. verschiedene Pakettypen für den Verbindungsaufbau und für die Datenübertragung verwendet. Der Vorteil hiervon ist, dass die nötigen Felder im Header für den Verbindungsaufbau nicht auch bei den Paketen während der Datenübertragung mitgeschleppt werden müssen (Overhead minimieren).
Es gibt die Pakettypen DCCP-Request und DCCP-Response um eine Verbindung aufzubauen, für die Datenübertragung die Pakettypen DCCP-Data, DCCP-Ack und DCCP-DataAck und bei der Terminierung die Pakettypen DCCP-CloseReq, DCCP-Close und DCCP-Reset. Um eine Verbindung zu synchronisieren, wurden noch die Pakettypen DCCP-Sync und DCCP-SyncAck eingeführt. (Warum es nötig sein kann eine Verbindung zu synchronisieren, wird in Abschnitt III-C beschrieben.)

Wie eine Datenübertragung mit Verbindungsauf- und abbau aussieht, ist in Abb. 1 dargestellt. In Teil (1) wird die Verbindung aufgebaut. Dabei initiiert der Client die Verbindung mit einem DCCP-Request Paket. Der Server bestätigt die Verbindung mit DCCP-Response, welches der

Client mit DCCP-Ack bestätigt. Damit ist die Verbindung aufgebaut, d.h. alle Parameter sind ausgehandelt, sowie die Sequenznummern initialisiert (siehe unten). Nach der Datenübertragung in (2) will der Server in (3) die Verbindung mit DCCP-CloseReq trennen. Der Client bestätigt dies mit DCCP-Close. Der Server signalisiert dann mit seinem DCCP-Reset Paket, dass er keine weiteren Daten mehr von dieser Verbindung empfangen wird.

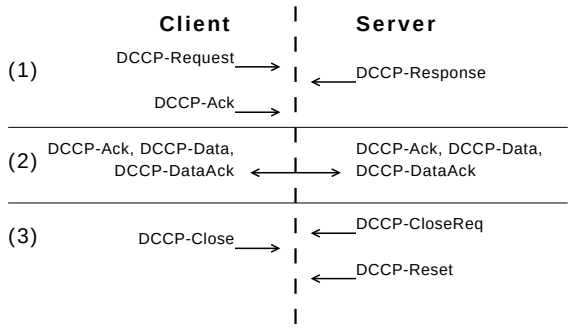


Abbildung 1. Ablauf einer Datenübertragung mit DCCP.

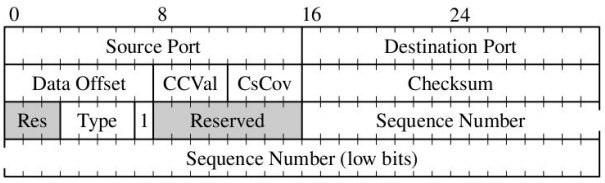


Abbildung 2. Generic-Header eines DCCP Pakets. Quelle: [3].

Ein DCCP-Header besteht aus zwei Teilen. Den *Generic-Header* trägt jedes Paket, der *Packet-Header* ist für jeden Pakettyp spezifisch. In Abb. 2 ist der Generic-Header dargestellt:

- *Source und Destination Ports* (jeweils 16 Bits): Sind wie bei TCP und UDP zu verstehen.
- *Data Offset* (8 Bits): Gibt den Offset vom Anfang des DCCP Headers bis zum Start der Payload (in 32 Bit Worten) an.
- *CCVal* (4 Bits): Wird vom Staukontroll-Mechanismus verwendet.
- *CsCov* (4 Bits): Hier kann angegeben werden, was alles von der Checksumme geprüft wird (nur Header oder auch Payload). Eventuell Performancegewinn möglich.
- *Checksum* (16 Bits): Checksumme über Header bzw. auch Daten.
- *Type* (4 Bits): Gibt den Pakettyp an, d.h. welcher Packet-Header diesem Generic-Header folgt.
- *1* (1 Bit): Dieses Bit gibt an, ob lange oder kurze Sequenznummern verwendet werden. Wie man an Abb. 2 sofort sieht, nimmt die Sequenznummer den Großteil des Platzes ein. Daher kann man zwischen kurzen und langen Sequenznummern wählen (siehe Abschnitt III-C).
- *Sequence Number* (48 oder 24 Bits): Die Sequenznummer dieses Packets.

C. Sequenznummern

Eine der wichtigsten Kernfunktionalitäten von DCCP sind Sequenznummern. Nahezu alle anderen Funktionalitäten (Verbindungsaufbau, Feature Negotiation usw.) werden erst durch Sequenznummern möglich. Zusätzlich werden Sequenznummern z.B. benötigt um Duplikate zu erkennen, verlorene Pakete zu identifizieren, oder Angriffen (*packet injection*) entgegen zu wirken.

Sequenznummern von DCCP zählen Datagramme und keine Bytes wie bei TCP. Jedes Datagramm, auch Acknowledgements, erhöhen die Sequenznummer. DCCP will Staukontrolle auch auf Acknowledgements anwenden und Feature Negotiation (z.B. von Staukontroll-Mechanismen) während der Verbindung erlauben. Dies geschieht aber durch Pakete, welche keine Payload tragen und daher müssen auch diese eine Sequenznummer haben, um sie bestätigen zu können [3].

Wie schon oben bei Abb. 2 angesprochen, bietet DCCP die Möglichkeit, kurze Sequenznummern (*short sequence numbers*) zu verwenden. Eine (lange) Sequenznummer von DCCP besteht aus 48 Bits. Um Overhead zu sparen, können z.B. DCCP-Data Pakete während der Datenübertragung die oberen 24 Bits weglassen und nur noch die unteren 24 Bits übertragen. Die Implementierung auf der anderen Seite erweitert dann diese 24 Bits wieder auf 48 Bits (Pseudocode in [6]). Es wird jedoch empfohlen, dieses Feature mit Sorgfalt zu verwenden, da hierdurch Angriffe leichter möglich sind, siehe [6].

Bei TCP werden *cumulative acknowledgements* verwendet, d.h. die ACK-Nr. bestätigt den Empfang der Pakete/Bytes bis zu dieser ACK-Nr., wobei die nächste erwartete Sequenznummer genau die ACK-Nr. ist. Damit ist der Status der aktuellen Verbindung mit dieser Acknowledgement Nummer beschrieben. Die Teilnehmer wissen wieviel bereits angekommen ist und damit wieviel verloren ging. Für DCCP ist diese Art der Bestätigung nicht sinnvoll, da DCCP niemals erneute Übertragung von Paketen durchführt. Daher hat man sich bei DCCP dazu entschlossen, dass eine ACK-Nr. der Sequenznummer des letzten angekommenen Pakets/Datagramms entspricht. Das hat aber zur Folge, dass aus der ACK-Nr. nicht mehr geschlossen werden kann, wieviele Pakete verloren gingen (*connection history*), was für die Staukontrolle entscheidend ist. Daher gibt es bei DCCP weitere Mechanismen, die genau spezifizieren, was mit den einzelnen Paketen geschehen ist, siehe dazu Abschnitt III-D.

Um Angriffe (wie *packet injection*) zu erkennen, benutzt DCCP (sowie TCP) *Sequence* und *Acknowledgement Number Windows*, welche festlegen, in welchem Bereich die nächsten

Sequenznummern und ACK-Nr. liegen sollen. Kommt ein Paket beim Empfänger an, prüft DCCP die Sequenznummer anhand von Regeln, welche in [6] beschrieben sind. Dabei gilt im Groben, dass eine Sequenznummer gültig ist, falls sie im gültigen Bereich des Fensters liegt.
Kommt es während einer Datenübertragung zu einem Ausfall und gehen dadurch viele aufeinanderfolgende Pakete verloren, so kann es vorkommen, dass die Pakete, welche schließlich wieder bis zum Empfänger durchdringen, bereits eine Sequenznummer außerhalb des gültigen Bereichs haben. (Bei TCP würden nun einfach die verlorenen Pakete durch *cumulative acknowledgements* erkannt und neu übertragen oder überhaupt die Verbindung terminiert.) DCCP weiß in so einem Fall jedoch nicht, ob jemand die Pakete eingeschleust hat, ob sie von einer alten Verbindung stammen, oder ob sie wirklich zur aktuellen gültigen Verbindung gehören. Daher versucht DCCP, die Verbindung wieder zu synchronisieren. Dabei schickt der Empfänger an den Sender ein DCCP-Sync Paket, welches die eingegangene Sequenznummer enthält. Der Partner der Verbindung empfängt das DCCP-Sync und antwortet mit einem DCCP-SyncAck falls er die Sequenznummer als gültig betrachtet. Dann aktualisiert der Empfänger sein Fenster und die Verbindung ist wieder synchron, siehe Abb. 3. Obwohl dieser Ansatz recht einfach und übersichtlich klingt, gibt es doch einige Probleme und Sonderfälle. Wie mit diesen umzugehen ist, wird in [6] besprochen.

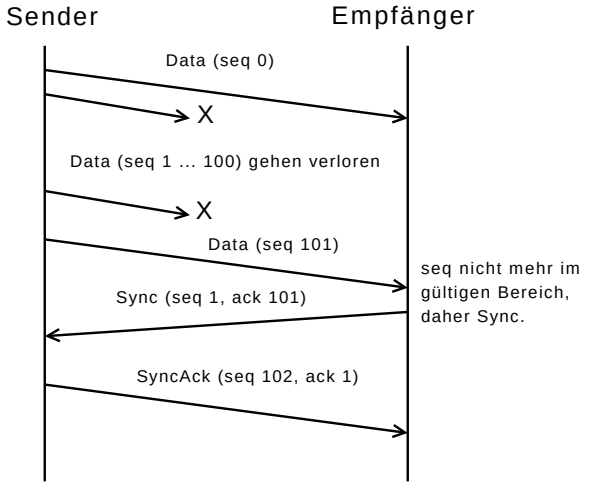


Abbildung 3. Synchronisation einer Verbindung. Nach [6].

D. Acknowledgements

Acknowledgements haben bei DCCP eine andere Bedeutung als bei TCP. Wird bei DCCP ein Datagramm durch ein ACK bestätigt, so heißt dies nur, dass der Header des Pakets erfolgreich bearbeitet wurde (Checksumme stimmt, Optionen wurden bearbeitet usw.), aber nicht, dass das Paket auch bei der Anwendung ankommt. Warten z.B. mehrere Pakete in einem Empfangspuffer auf eine Anwendung, so ist vielleicht nur das letzte empfangene Paket von Bedeutung (z.B. Ortsinformation bei Online-Games), der Rest wird verworfen. Acknowledgements sind bei DCCP hauptsächlich für die Staukontrolle eingeführt worden und die Staukontrolle interessiert nur was an Verlusten wegen der Situation im Netz stattgefunden hat. Daher ist dieser Ansatz vernünftig.

(a)	0010011?	Länge	SSLLLLLL	SSLLLLLL	...
(b)	0010011?	Länge ...	00000000 00000011	11000000 01000000	... 00000101

Tabelle I
ACK VECTOR OPTION

DCCP bietet über Optionen/Features einige Mechanismen an, die zusätzliche Information über die Pakete der aktuellen Verbindung beinhalten. Der eingesetzte Staukontroll-Mechanismus entscheidet dabei, welche Optionen aktiv sind und benutzt werden. So legt z.B. das *Ack Ratio Feature* die Rate fest, mit welcher der Empfänger die DCCP-Acks versenden muss. Damit lässt sich bei Stau eine optimale Rate für den Acknowledgement-Verkehr einstellen. Wieder wird über den Staukontroll-Mechanismus festgelegt, welcher Wert hier genau eingestellt werden soll.

Die wichtigsten Informationen für die Staukontrolle werden jedoch über die *Ack Vector Option* geliefert. Dies ist eine Option im DCCP-Header und beschreibt den Status der letzten Pakete. Der Aufbau eines solchen Ack Vektors ist in Tabelle I angegeben.

Das erste Byte des Vektors spezifiert die Option-Nr., das zweite Byte die Länge des Vektors. Danach folgen Felder (je ein Byte) mit dem Aufbau “SSLLLLLL”. Dabei stehen die ersten beiden Bits “SS” für einen Status (Received (00), Received ECN Marked (01), Not Yet Received (11)) und die folgenden 6 Bits “LLLLLL” geben die Anzahl der aufeinanderfolgenden Pakete mit diesem Status an. Dabei ist das erste Byte in einem Ack Vektor auf das Paket mit der ACK-Nr. bezogen, welche ebenfalls in diesem Paket geschickt wird (und damit ist ein Paket mit Ack Vektor aber ohne ACK-Nr. ungültig). Betrachtet man das Beispiel von Tabelle I in Zeile (b) und geht davon aus, dass die ACK-Nr. des Pakets 100 ist, so bedeuten die fünf SSLLLLLL-Komponenten des Vektors [6]

1) Paket 100 ist angekommen,
2) Paket 99 ist nicht angekommen,
3) Pakete 98 bis 95 sind angekommen,
4) Paket 94 ist mit Status “Received ECN marked” angekommen, und
5) Pakete 93 bis 88 sind angekommen.

Die Information über ein Paket A wird solange in den Ack Vektoren mitgeschickt, bis ein Paket mit Ack Vektor, welches Paket A behandelt, vom Gegenüber bestätigt wird. Dadurch kann es vorkommen, dass eine Sequenznummer/Datagramm in zwei Ack Vektoren behandelt wird, und dass sich diese Informationen widersprechen! Der Empfänger kann z.B. direkt nach dem Abschicken eines Pakets mit Ack Vektor ein noch fehlendes Paket erhalten und die Ack Vektor Information

im nächsten Datagramm dann aktualisieren. Der Empfänger erhält so zwei Pakete mit Ack Vektoren, welche sich bei einem Datagramm unterscheiden. (Zuerst “Not Yet Received” und dann “Received”.) Ein andere Möglichkeit ist, dass im Netz die Acknowledgements einfach umgedreht werden. Was bei solchen Inkonsistenzen zu tun ist, wird in [6] genau festgelegt (die Ack Vektoren werden miteinander kombiniert und das Ergebnis anhand einer Tabelle des RFCs abgelesen).

Wie oben erwähnt, werden Pakete in einem Ack Vektor solange behandelt, bis dies bestätigt wurde. Stellt man sich nun die Verbindung eines Videostreams vor, so sendet ausschließlich der Server Daten an den Client. Der Client bestätigt auch die Daten des Servers mit DCCP-Acks und schickt darin immer den Ack Vektor mit. Wird jedoch ein DCCP-Ack des Clients nicht vom Server bestätigt, so schickt der Client sämtliche Informationen aller Pakete der bisherigen Verbindung in allen DCCP-Ack Paketen mit. Daher muss der Sender einer solchen einseitigen Verbindung Bestätigungen von Acknowledgements verschicken (*acks of acks*). (Dabei sind *acks-of-acks-of-acks* hierbei nicht nötig!) Bei Verbindungen, bei denen Daten in beiden Richtungen fließen, werden diese Acknowledgements in DCCP-DataAck Paketen verschickt. (Wieder kontrolliert der Staukontroll-Mechanismus, wann Ack Vektoren bestätigt werden.)

Zwei weitere Optionen sollen hier kurz beschrieben werden: *NDP Count Option* und *Data Dropped Option*. Da Sequenznummern auch bei DCCP-Acks erhöht werden, kann ohne zusätzliche Information nicht entschieden werden, ob wirkliche Daten der Anwendung verloren gingen. Hier hilft die *NDP Count Option*, welche die Länge von aufeinanderfolgenden Nicht-Daten-Paketen angibt (*non-data-packet*). Ist in einem Paket die NDP Count Option auf n gesetzt, so heißt dies, dass die letzten n Pakete Nicht-Daten-Pakete waren. Gehen z.B. während einer Verbindung 10 Pakete verloren und die Option ist bei dem nächsten eintreffenden Paket gesetzt und hat einen Wert ≥ 10, dann sind keine Anwendungsdaten verloren gegangen.
Erreichen Pakete/Daten nicht die Anwendung, so bietet die *Data Dropped Option* die Möglichkeit, zusätzliche Information an den Sender zu schicken, warum die Daten nicht weitergereicht wurden. Der Aufbau der Data Dropped Option ist ähnlich zum Ack Vektor. Mögliche Gründe warum Daten nicht weitergereicht wurden, sind z.B. “Protocol Constraints” (000), “Corrupt” (011) oder “Application Not Listening” (010).

E. Feature Negotiation

Während und am Anfang einer Verbindung mit DCCP müssen einige Parameter ausgehandelt werden (Staukontroll-Algorithmus, Ack Vector verwenden, NDP Count Option verwenden usw.). Bei DCCP hat man sich für eine sehr klare Lösung entschieden. Es gibt einen allgemeinen Mechanismus, die *Feature Negotiation*, welche es erlaubt, solche Parameter auszuhandeln. Die einzelnen Parameter werden dabei über Codes (*feature number*) identifiziert.

Es wurden vier Optionen eingeführt: *Change L*, *Confirm L*, *Change R* und *Confirm R*. Eine Feature Negotiation wird durch das Setzen einer Change Option eingeleitet und durch eine Confirm Option abgeschlossen. Dabei unterscheiden “L” und “R” ob das Feature bei sich selbst oder beim Partner geändert werden soll. Will z.B. Teilnehmer A ein Feature F bei Teilnehmer B ändern (z.B. Teilnehmer B soll keinen Ack Vektor mehr schicken), dann schickt Teilnehmer A ein Change R an B. Will jedoch Teilnehmer A ein Feature F von sich selbst ändern, so schickt er ein Change L. Entsprechend wird mit Confirm L und Confirm R geantwortet. Wieder können hier einige Sonderfälle auftreten (Change geht verloren, Unbekannte Option usw.), die in [6] besprochen werden.

Damit zwei Partner sich einigen können, gibt es Regeln (*Reconciliation Rules*) die Prioritäten festlegen. Bis jetzt sind zwei Regeln spezifiziert: *Server-Priority* und *Non-Negotiable*. Bei Server-Priority besteht das Feature aus einer Liste von Möglichkeiten. Sendet A die Liste (3,2,1) (von z.B. verschiedenen Staukontroll-Mechanismen) an B, so wählt dieser etwas davon aus und schickt die Antwort zurück. Bei Non-Negotiable handelt es sich nur um einen Wert (z.B. Ack Ratio). Dieser Wert muss akzeptiert werden, falls er gültig ist.

Insgesamt erlaubt diese allgemeine Methode das Hinzufügen von neuen Features ohne große Änderungen am Protokoll.

IV. Staukontroll-Mechanismen

Wie schon oben erwähnt, ist DCCP lediglich ein Framework für die Staukontrolle [3]. Aufbauend auf dieses Framework können verschiedene Staukontroll-Mechanismen und -Algorithmen eingesetzt werden. Bei DCCP wird der Staukontroll-Mechanismus am Anfang einer Verbindung ausgewählt. Jedem Mechanismus ist eine Nummer zugeordnet, der *congestion control identifier* (CCID). Die Funktionsweise eines solchen Staukontroll-Mechanismus wird durch das Staukontroll-Profil (*congestion control profile*) festgelegt. Im Folgenden sollen drei Staukontroll-Mechanismen etwas genauer besprochen werden.

A. TCP-like Congestion Control

Der *TCP-like congestion control* Mechanismus versucht die TCP Staukontrolle basierend auf SACK [9] möglichst genau nachzubilden. Als CCID hat man 2 gewählt. Das zugehörige Staukontroll-Profil findet man in RFC 4341 [10].

CCID 2 passt sich sehr schnell an die verfügbare Bandbreite an und versucht damit auf lange Sicht soviel Bandbreite wie möglich zu erlangen. Dieses schnelle Anpassen an neue Situationen führt jedoch zu großen

und abrupten Schwankungen in der Bandbreite (*additive increase/multiplicative decrease* (AIMD)). Daher wird CCID 2 eingesetzt, falls die Anwendung mit solchen Schwankungen umgehen kann. Grob kann man sagen, dass Anwendungen bei denen eine Pufferung möglich ist, für CCID 2 geeignet sind (z.B. Multimedia Streaming). Anwendung bei denen dies nicht möglich ist (z.B. VoIP), sollten auf andere Mechanismen ausweichen.

SACK TCP Staukontrolle basiert auf *Slow Start* und *Congestion Avoidance*. Zusätzlich werden *Fast Retransmit* und *Fast Recovery* verwendet [9]. Am Anfang einer Verbindung wird während der Slow-Start-Phase für jedes eintreffende ACK das Staufenster (*congestion window*) verdoppelt. Dauert das Eintreffen eines ACKs zu lange, so wird in die Congestion-Avoidance Phase gewechselt. Dort wird nun das Staufenster halbiert und nur noch linear erhöht [11], [12].
Um die Performance dieses Algorithmus zu verbessern, wurden noch weitere Algorithmen entwickelt [12]:

- *Fast Retransmit*: Treffen beim Sender dreimal ACKs mit der gleichen Acknowledgement Nummer ein, wird das Paket bereits als verloren angenommen. (Man wartet also nicht bis ein Timer abläuft.)
- *Fast Recovery*: Beim Verlust eines Pakets wird nicht wieder in die Slow-Start-Phase gewechselt, sondern in die Congestion-Avoidance-Phase.

Mit der Ack Vector Option bietet DCCP einen Mechanismus wie SACK von TCP an. Damit kann der Algorithmus in CCID 2 dann das AIMD Verhalten der SACK TCP Staukontrolle simulieren [3].

B. TCP-friendly Rate Control (TFRC)

TFRC wurde bereits 2003 unabhängig von DCCP vorgestellt und wird in RFC 5348 [13] beschrieben. Bei DCCP hat TFRC CCID 3 und das Profil wird in RFC 4342 [14] festgelegt. Ziel ist es, die abrupten Schwankungen von CCID 2 zu vermeiden, aber trotzdem eine faire Aufteilung der Bandbreite zu ermöglichen.

Der Ablauf einer Anpassung der Bandbreite durch TFRC lässt sich in vier Schritte einteilen [13]:

1) Der Empfänger misst die *Loss-Event-Rate* und schickt diese mit Zeitstempel an den Sender zurück. Die Loss-Event-Rate entspricht in etwa der Anzahl der verlorenen Pakete durch die Anzahl der übertragenen Pakete in einem gewissen Zeitintervall.
2) Der Sender kann mit der Loss-Event-Rate und dem Zeitstempel die RTT berechnen.
3) Die Loss-Event-Rate, RTT sowie Paketgröße und andere Parameter werden in die TCP-Durchsatzgleichung eingesetzt und man erhält die zu benutzende Bandbreite.
4) Der Sender aktualisiert seine Senderate entsprechend.

Schickt der Empfänger wirklich die Loss-Event-Rate an den Empfänger, so kann dieser nicht kontrollieren ob sie richtig ist. D.h. der Empfänger kann sich so mehr Bandbreite erschleichen, als ihm eigentlich zusteht. Darum fordert CCID 3, dass nicht die Loss-Event-Rate an den Sender geschickt wird, sondern *Loss Intervals*. Dabei handelt es sich um eine Option welche angibt, was mit den letzten Paketen passiert ist. Abgesichert wird diese Information über die ECN Nonces. Daraus kann dann der Sender die Loss-Event-Rate berechnen.

C. TCP-Friendly Rate Control for Small Packets (TFRC-SP)

Der *TCP-Friendly Rate Control for Small Packets* Mechanismus ist in RFC 4828 [15] spezifiziert und wird noch als experimentell angesehen. Die CCID ist 4 und das Profil wird in RFC 5622 [16] beschrieben.

Bei Anwendungen wie VoIP werden viele kleine Pakete verschickt. Dies hat zwei Gründe. Die Anwendung kann nicht lange auf neue Daten warten, sondern muss dafür sorgen, dass der Empfänger möglichst aktuelle Daten hat (zeitkritisch) und neue Codecs sind so gut, dass für kleine Audioabschnitte weniger als 20 Bytes Daten anfallen [3]. Man hat nun festgestellt, dass sich TFRC für solche Datenströme nicht eignet.
Während TCP solche Daten einfach in großen Segmenten verschickt, hat DCCP keinen Einfluss darauf, welche Pakete von der Anwendung kommen. Da bei TFRC die Paketgröße in die Durchsatzgleichung einfließt, verhält sich TFRC wie ein TCP-Fluss, der sehr kleine Segmente verschickt. Solch ein TCP-Fluss erzielt aber eine kleinere Durchsatzrate (in Bytes pro Sekunde) als ein TCP-Fluss mit großen Segmenten. Darum treten solche TCP-Flüsse praktisch nie auf [3]. Benutzen ein TCP-Fluss mit großen Segmenten und ein TFRC-Fluss mit kleinen Paketen einen Durchsatzknoten gemeinsam, reduziert (bei gleicher Paketverlust-Rate) TFRC die Senderate (in Pakete pro Sekunden) immer weiter, obwohl der Durchsatz (in Bytes pro Sekunde) des TFRC-Flusses eigentlich viel kleiner ist, als von dem TCP-Fluss. In Abb. 4 ist dies gut zu erkennen. Der TFRC-Fluss reduziert die Senderate immer weiter, obwohl er viel weniger Durchsatz erzeugt als der TCP-Fluss.
Bei TFRC-SP wird in der TCP-Durchsatzgleichung einfach eine andere Paketgröße eingesetzt und man erhält damit einen höheren Durchsatz und trotzdem verhält sich der Algorithmus noch fair gegenüber einem TCP-Fluss [16], siehe Abb. 4.

Diese Änderung zieht jedoch einige Probleme nach, weswegen TFRC-SP auch noch als experimentell angesehen wird. Verwirft man z.B. keine Pakete sondern nur Bytes (und sieht ein Paket als verloren an, falls ein Byte verloren wurde), so verhält sich TFRC-SP unfair gegenüber TCP, wie in [3] gezeigt wird.

V. Ausblick

Derzeit ist DCCP noch nicht weit verbreitet. Es gibt erste Implementierungen von DCCP unter Linux, FreeBSD und

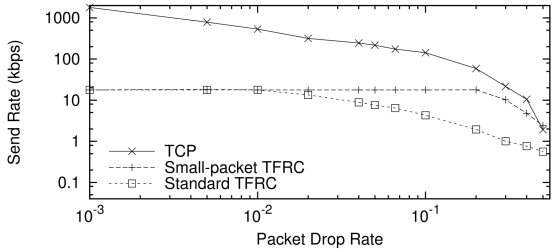


Abbildung 4. Vergleich von TCP (1460-Byte Segmente), TFRC (14-Byte Segmente, Anwendung sendet maximal 12 kbps (inkl. Header)) und TFRC-SP bei vorgegebene Paketverlust-Rate. Quelle [3].

NetBSD, aber noch keine Anwendung die DCCP als (Haupt-) Protokoll verwenden. Dies hat mehrere Gründe. Firewalls und NAPTs verstehen DCCP meist noch nicht. Anwendungen setzen oft auf eigene Mechanismen um Schwankungen in der Bandbreite auszugleichen, wie am Beispiel von Skype in [8] gezeigt wird. Weiter hoffen die Entwickler von DCCP auf mehr Feedback von den Anwendungsentwicklern, da die Staukontroll-Mechanismen noch einige Probleme haben.
Es gibt bereits zahlreiche Untersuchungen, wie sich DCCP mit verschiedenen Staukontroll-Mechanismen bei unterschiedlichen Anwendungen verhält. In [17] wird die Qualität von Videostreams über DCCP untersucht. Dabei kommt TFRC (CCID 3) zum Einsatz. Verschiedene Szenen in einem Video brauchen verschiedene Bitraten, um die Videoqualität gleich zu halten. Da TFRC diese Unterschiede aber nicht beachtet, kommt es zu teilweise starken Schwankungen der Qualität. Eine konstante Qualität während der gesamten Übertragung wäre wünschenswert. In [18] wird die Übertragung von VoIP mit TFRC (CCID 3) in einem echten Netzwerk (und keinem Simulator) untersucht. Dabei zeigt sich, dass TFRC bei langen Pausen (*idle period*) die Senderate zu stark begrenzt, was zu einem Qualitätsverlust führt. Weiter ist z.B. noch nicht klar, wie mit Datenflüssen umgegangen werden soll, welche sehr schnell zwischen einer (hohen) Datenübertragung und einer Pause wechseln.

VI. Zusammenfassung

Mit DCCP wurde ein Protokoll entwickelt, welches Staukontrolle auch für unzuverlässigen Datenfluss ermöglicht. Besonders der modulare Aufbau von DCCP und der Staukontrolle ist essentiell. Wie das Beispiel von VoIP und TFRC-SP gezeigt hat, führen kleine Unterschiede bei Anwendungen bereits zu völlig anderen Anforderungen an die Staukontrolle. Bei einem Protokoll können solche Änderungen an der Staukontrolle nicht so schnell einfließen, daher ist die Entkoppelung von Protokoll und Staukontrolle ein vernünftiger Schritt. Da die Staukontrolle aber auf Feedback-Mechanismen angewiesen ist, mussten diese bei DCCP recht allgemein gehalten werden.

Insgesamt bietet DCCP mit dem modularen Aufbau den richtigen Ansatz, um verschiedenste Staukontroll-Mechanismen zu unterstützen. Auch der Bedarf an einem unzuverlässigen Protokoll mit Staukontrolle ist vorhanden. Jedoch lässt sich nicht sagen, ob und wann sich DCCP wirklich durchsetzt.

Literatur

[1] L. Mamatas, T. Harks, and V. Tsaoussidis, "Approaches to Congestion Control in Packet Networks," *Journal of Internet Engineering*, January 2007.

[2] S. Floyd, M. Handley, and E. Kohler, "Problem Statement for the Datagram Congestion Control Protocol (DCCP)," RFC 4336 (Informational), Internet Engineering Task Force, Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4336.txt

[3] E. Kohler, M. Handley, and S. Floyd, "Designing dccp: congestion control without reliability," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2006, pp. 27–38.

[4] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (Standard), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 4379. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[5] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), Internet Engineering Task Force, Sep. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[6] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," RFC 4340 (Proposed Standard), Internet Engineering Task Force, Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4340.txt

[7] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz, "Known TCP Implementation Problems," RFC 2525 (Informational), Internet Engineering Task Force, Mar. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2525.txt

[8] L. De Cicco, S. Mascolo, and V. Palmisano, "Skype video responsiveness to bandwidth variations," in *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. New York, NY, USA: ACM, 2008, pp. 81–86.

[9] E. Blanton, M. Allman, K. Fall, and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP," RFC 3517 (Proposed Standard), Internet Engineering Task Force, Apr. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3517.txt

[10] S. Floyd and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control," RFC 4341 (Proposed Standard), Internet Engineering Task Force, Mar. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4341.txt

[11] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, August 1988.

[12] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001 (Proposed Standard), Internet Engineering Task Force, Jan. 1997, obsoleted by RFC 2581. [Online]. Available: http://www.ietf.org/rfc/rfc2001.txt

[13] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348 (Proposed Standard), Internet Engineering Task Force, Sep. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5348.txt

[14] S. Floyd, E. Kohler, and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)," RFC 4342 (Proposed Standard), Internet Engineering Task Force, Mar. 2006, updated by RFC 5348. [Online]. Available: http://www.ietf.org/rfc/rfc4342.txt

[15] S. Floyd and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant," RFC 4828 (Experimental), Internet Engineering Task Force, Apr. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4828.txt

[16] E. Kohler and S. Floyd, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)," RFC 5622 (Experimental), Internet Engineering Task Force, Aug. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5622.txt

[17] J. V. Velthoven, K. Spaey, and C. Blondia, "Performance of constant quality video applications using the dccp transport protocol," in *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06)*, Nov. 2006, pp. 511–512.

[18] H. Hwang, X. Yin, Z. Wang, and H. Wang, "The internet measurement of voip on different transport layer rotocols," in *Information Networking, 2009. ICOIN 2009. International Conference on*, Jan. 2009, pp. 1–3.

Netzwerkmanagement mit NETCONF und YANG

Julian Sommerfeldt
Betreuer: Marc-Oliver Pahl
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: sommerfe@in.tum.de

***Kurzfassung*— NETCONF hat mit dem Jahr 2009 einen Status erreicht, den man als weitestgehend vollständig bezeichnen kann. Die Entwicklung durch die IETF Working Group wurde motiviert durch die Tatsache, dass SNMP nicht wie ursprünglich gedacht, zur Erfüllung von Konfigurationsaufgaben für Netzwerkgeräte benutzt wurde, sondern lediglich für Monitoringzwecke. Das XML-basierte Protokoll reagiert darauf mit der Zielsetzung, die bisher üblichen Command-Line-Interfaces zu ersetzen und den Administratoren somit eine flexible, einfach und schnell zu erlernende Möglichkeit zur Verfügung zu stellen.**
Unterstützt wird dieser Reifegrad durch die Datenmodellsprache YANG, die für das Protokoll entworfen wurde und den Zugang noch weiter erleichtert.

***Schlüsselworte*— Netzwerk, Management, NETCONF, YANG**

I. Einleitung

Um die immer komplexeren Netzwerkstrukturen und die immer größer werdende Vielfalt an Geräten die darin interagieren, in den Griff zu bekommen, war eine Standardisierung notwendig, die die Netzwerkadministratoren unterstützt. Mit diesem Ziel wurde 2003 die IETF Working Group gegründet, deren Arbeit nun bereits durch einige Implementierungen, beispielsweise von Cisco, und einer stark wachsenden Anwendergruppe bestätigt wurde. Da vorher im Allgemeinen Kommandozeilenprogramme beziehungsweise sehr spezielle, auf bestimmte Geräte zugeschnittene, Lösungen verwendet wurden, kann das NETCONF Protokoll im Vergleich zu dem, was bisher eingesetzt wurde, als komplette Neuheit bezeichnet werden.

II. Anforderungen

Die Anforderungen und deren Motivation, die zu Beginn festgesetzt wurden, werden hier grob erläutert. [1], [2]

A. Differenzierung verschiedener Datenarten

Innerhalb der Netzwerkgeräte kann grob unterschieden werden zwischen Konfigurationsdaten und Statusdaten. Erste werden primär dazu verwendet, um Einstellungen zu manipulieren und das Gerät somit zu steuern. Zweitere geben beispielsweise Auskunft über Temperatur oder Laufzeit seit dem letzten Start. Da oftmals nur die eine oder die andere Art der Daten benötigt wird, ist es sinnvoll, dass man zwischen ihnen unterscheiden und somit gezielter arbeiten kann und eine kleinere Datentransfermenge hat.

B. Erweiterbarkeit

Die Fülle an unterschiedlichen Geräten bringt auch eine große Menge unterschiedlichster Daten mit sich, die alle ansteuerbar sein müssen. Hierzu ist es notwendig, dass ein hohes Maß an Flexibilität bereitgestellt wird, wodurch die Hersteller mit einem einzigen Protokoll Zugang zu allen Daten gewährleisten.

C. Programmierbare Schnittstelle

Dieser Punkt stellt den Schritt weg von der Kommandozeile hin zu einfacher steuerbaren Programmen dar. Die Administratorentätigkeit wird auf eine wesentlich abstraktere Ebene gehoben, wodurch die Konfiguration leichter fällt, wenn man nicht auf tiefster Ebene mit der entsprechenden Arbeitsweise vertraut ist. Des Weiteren wird durch unterschiedliche Implementierungsmöglichkeiten an Flexibilität gewonnen.

D. Textuelle Repräsentation

Die Konfigurationsdaten sollen rein durch „einfachen“ Text festgelegt werden, um zu verhindern, dass ein spezialisiertes Tool verwendet werden muss. Hierzu eignet sich XML, da dieses sowohl einfach zu erlernen und zu lesen sowie bereits weit verbreitet und anerkannt ist. Jeder kann somit ohne aufwendige Lernnotwendigkeiten relativ einfach Konfigurationen durchführen.

E. Authentifizierung

Da es sich bei der Konfiguration um höchst sensible Daten handelt, ist es notwendig, dass zu Beginn einer Session eine Authentifizierung stattfindet, wobei diese Aufgabe von bereits bestehenden Methoden, wie dem Transportprotokoll, unterstützt wird.

F. Integration existierender Konfigurationsdatenbanksysteme

Um zu verhindern, dass ein völlig neuer Aufbau stattfinden muss, war eine Anforderung, die Fähigkeit der Zusammenarbeit mit bereits bestehenden Systemen, so dass mit diesen weitergearbeitet werden kann.

G. Transaktionen

Damit die Möglichkeit einer Konfiguration durch mehrere Stellen gegeben ist, müssen transaktionsbezogene Funktionen bereitgestellt werden. Hier sind Stichwörter wie *locking* und *rollback* zu nennen, die Komplikationen des Mehrbenutzerbetriebs verhindern sollen.

H. Transportunabhängigkeit

Ein weiterer Punkt, der unter Flexibilität einzuordnen ist, ist die Unabhängigkeit des Transportprotokolls. Das Ziel, alle Geräte, mit unterschiedlichen Protokollen zu unterstützen, erreicht NETCONF dadurch, dass es mit allen gängigen Transportprotokollen implementiert werden kann. Neben SSH im Vordergrund existieren bereits Implementierung die mit BEEP und SOAP arbeiten.

III. Architektur

Die Architektur des Protokolls ist in vier Ebenen aufgeteilt, wie sie in Tabelle I dargestellt ist. [2]

Tabelle I
ARCHITEKTUR DES NETCONF PROTOKOLLS [2, S. 8]

Ebene	Beispiel
Inhalt	Konfigurationsdaten
Operationen	<get-config>, <edit-config>
RPC	<rpc>, <rpc-reply>
Transportprotokoll	SSH, BEEP, SSL, Console

Im Folgenden wird der Aufbau näher erklärt, Beispiele werden erläutert und es wird aufgezeigt, wie auf die Anforderungen aus II eingegangen wird. Die Listings, die dabei verwendet werden, lehnen sich an [2] an.

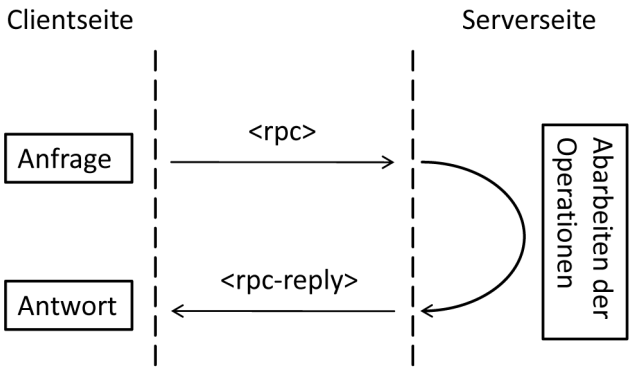
A. Transportprotokoll

Als unterliegendes Transportprotokoll kann jedes gewählt werden, das gewisse Anforderungen erfüllt, wobei eine Implementierung mindestens SSH unterstützen muss, wodurch II-H erfüllt ist. Die Verbindungsorientierung ist wichtig, damit eine persistente Kommunikation zwischen Client und Server stattfinden kann. Wichtig ist dies bei Authentifizierungsdaten, die während einer Session erhalten bleiben.
Das Aufgabenfeld der in II-E erwähnten Authentifizierungsanforderung wird auf das Transportprotokoll ausgelagert und muss somit davon, ebenso wie die Integrität und Vertraulichkeit der Daten, erfüllt beziehungsweise gewährleistet werden.

B. RPC

NETCONF benutzt Remote Procedure Call (RPC) um vom Client aus Anfragen an den Server zu stellen. Wie in II-D gefordert, wird hierzu XML verwendet, wobei alle NETCONF Nachrichten sich im Listing 1 angegebenen Namespace befinden müssen. Ebenso stellt es eine RPC-Anfrage in XML dar, welche wohlgeformt sein muss. Dieser Vorgang und das Verhalten der Akteure werden in Abbildung 1 dargestellt. Durch die Verwendung von XML besteht die Möglichkeit, beliebige Interfaces zu erstellen (II-C) und durch Umformung älterer Konfigurationen neue XML-basierte zu erhalten (II-F).

Abbildung 1. Ablauf des RPC-Requests



Listing 1. RPC

```
<rpc message-id="1337"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

        <!-- Inhalt -->

</rpc>
```

Man sieht den Starttag <rpc>, der zwei Parameter aufweist: Der erste ist eine Id, mit deren Hilfe der Aufruf identifiziert werden kann, was unter anderem für den <rpc-reply>, also die Antwort, notwendig ist, um eine klare Zuordnung zu ermöglichen.
Im folgenden Beispiel (Listing 2) wird die Methode <get>, eine der Basisoperationen von NETCONF, ausgeführt und der gesamte Datensatz zurückgegeben, wobei der „Header" unverändert bleibt.

Listing 2. RPC-Reply

```
<rpc message-id="1337"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <get\>
</rpc>

<rpc-reply message-id="1337"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
        <data>
              <!-- Kompletter Datensatz -->
        </data>
</rpc-reply>
```

Darüber hinaus gibt es noch den Tag <rpc-error>, der bei Fehlern zum Einsatz kommt und den Tag <ok>, der verwendet

wird, wenn kein Fehler aufgetreten ist, es aber auch keine Daten gibt, die zurückgegeben werden.

C. Operationen

Das NETCONF Protokoll stellt Basisoperationen zur Verfügung, mit denen einige Fähigkeiten realisiert werden.

1) Inhaltsbezogene Operationen:

- <get>
  Fordert alle, also Konfigurations- und Statusdaten, an.
- <get−config>
  In II-A wird eine Differenzierung zwischen den Daten gefordert, was unter anderem dadurch realisiert wird, dass hier nur die Konfigurationsdaten abgefragt werden.
- <edit−config>
  Hierbei wird die Konfiguration direkt geändert, wobei folgende unterschiedliche Operationen verfügbar sind: merge, replace, create, delete.
- <copy−config>
  Kopiert eine komplette Konfiguration, je nach Situation, über eine bereits Existierende oder erstellt diese.
- <delete−config>
  Löscht eine Konfiguration.

2) Transaktionsbezogene Operationen (II-G):

- <lock>
  Sperrt eine Konfiguration, um ein zwischenzeitliches Eingreifen zu verhindern.
- <unlock>
  Gibt eine Sperre wieder frei.

3) Sessionbezogene Operationen:

- <close−session>
  Schließt eine Session, wobei alle gehaltenen Locks entsperrt und alle Verbindung geschlossen werden.
- <kill−session>
  Der Operation <close−session> ähnlich, allerdings werden alle laufenden Operationen abgebrochen und es wird ein sofortiges Schließen erzwungen.

D. Inhalt

Der Inhalt ist nicht direkt Teil der Spezifikation des NETCONF Protokolls, da sie kein Datenmodell angibt, sondern der der jeweiligen Implementierung. Die dadurch gewonnene Flexibilität ermöglicht ein Erfüllen der Anforderung II-B. Mit YANG wurde für diesen Teil ein Datenmodell geschaffen, welches aufgrund seiner Eigenschaften für diese Ebene verwendet werden kann. Hierauf wird in V näher eingegangen.

IV. Weitere Fähigkeiten

Neben den bisher dargestellten Grundlagen bietet das Protokoll noch andere Möglichkeiten, welche genaueres Steuern (IV-A) beziehungsweise neue Fähigkeiten (IV-B) bereitstellen. Beide Gebiete werden in [2] näher erläutert. Die folgenden Codebeispiele wurden wieder grob daraus entnommen.

A. Subtree Filtering

Um die Daten besser zu kategorisieren, werden sie in Bäumen verwaltet, die die Datenstruktur bilden. Kinder eines Baumes können entweder Knoten, also Enden, in denen Daten gespeichert sind, sein, oder wiederum Bäume, die sogenannten Subtrees. Subtree Filtering wird darauf angewandt, um Daten expliziter abzufragen, da wir bisher nur die Möglichkeit kennen gelernt haben, beispielsweise durch <get−config> die gesamte Konfiguration abzufragen. Dies führt häufig zu Unübersichtlichkeit und es findet ein unnötig hoher Datentransfer statt, der nur selten notwendig ist. Es gibt einige Typen von Filtern, hier wird allerdings die Filterung, die direkt auf den Inhalt bezogen ist exemplarisch vorgestellt. Wie in III-B bereits erklärt, läuft die Kommunikation über RPC ab. So sieht man im folgenden Listing 3 den umschließenden <rpc> Tag und die darin befindliche Filterung.

Listing 3. Subtree Filtering

```
<rpc message-id="1337"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <get-config>
  <source>
   <running/>
  </source>
  <filter type="subtree">
   <top
    xmlns="http://example.com/schema/1.2/config">
    <users>
     <user>
      <name/>
     </user>
    </users>
   </top>
  </filter
 </get-config>
</rpc>
```

Zuerst wird festgelegt, dass die laufende Konfiguration angefordert wird. Im Filter wird dann durch das *top* Element ein Namespace definiert, innerhalb dessen sich die Daten befinden müssen. Hieraus wiederum werden erst alle User „selektiert“ und dann von jedem Einzelnen jeweils nur das Attribut *name*. Angenommen es gäbe drei User im entsprechenden Namespace, so gibt der Server folgende Daten zurück (Listing 4).

Listing 4. Subtree Filtering Reply

```
<rpc-reply message-id="1337"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <data>
  <top
   xmlns="http://example.com/schema/1.2/config">
   <users>
    <user>
     <name>root</name>
    </user>
    <user>
     <name>Alfred</name>
    </user>
    <user>
```

```
      <name>Heidi</name>
     </user>
    </users>
   </top>
  </data>
</rpc-reply>
```

Wie man sieht, lassen sich mit der Anwendung von Subtree Filtern so exakte Datenabfragen bzw. -manipulationen durchführen und effizient arbeiten.

B. Capabilities

Capabilities sind Implementierungsmöglichkeiten für bestimmte Einsatzzwecke, wobei man sie als eine Art Erweiterung der Basisoperationen verstehen kann. Für viele Capabilities ist eine entsprechende Kommunikation zwischen Client und Server notwendig. Deshalb sendet jedes der beiden Enden am Beginn einer Session eine *hello*-Message, in der neben einer Id alle unterstützten Capabilities eingetragen sind. In Listing 5 sieht man eine solche Nachricht.

Listing 5. Hello Message

```
<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <capabilities>
  <capability>
   urn:ietf:params:netconf:base:1.0
  </capability>
  <capability>
   urn:ietf:params:netconf:capability:startup:1.0
  </capability>
  <capability>
   http://example.net/router/2.3/myfeature
  </capability>
 </capabilities>
 <session-id>4</session-id>
</hello>
```

Als ein Beispiel sei hier die *Rollback on Error Capability* beschrieben, welche die Fähigkeit, beim Auftreten von Fehlern ein Rollback durchzuführen, zur Verfügung stellt. Sollte beispielsweise im Listing 6 beim Setzen des *mtu* Parameters ein Fehler auftreten, so werden alle Änderungen auf den Stand zurückgesetzt, der vor der Ausführung dieser Anfrage aktuell war, also wird der *name* wieder auf den Ursprünglichen gesetzt.

Listing 6. Rollback on Error

```
<rpc message-id="1337"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
   <running/>
  </target>
  <error-option>rollback-on-error</error-option>
  <config>
   <top
   xmlns="http://example.com/schema/1.2/config">
    <interface>
     <name>Ethernet0/0</name>
     <mtu>100000</mtu>
    </interface>
   </top>
  </config>
 </edit-config>
</rpc>
```

Darüber hinaus kann man noch die *With-defaults Capability* nennen, welche die Mitübergabe von Standardwerten der Konfigurationsdaten steuert. Dies ist bei der Eröffnung einer Session wichtig, da eine zu große Datenmenge und die damit verbundenen notwendigen Operationen vermieden werden. Nähere Informationen sind in [3] zu finden.

C. Monitoring Schema

NETCONF definiert ein Schema, mit dem der Client sich über die Funktionalitäten des Servers informieren kann. Es beinhaltet alle Subtrees, also nicht nur die Capabilities, die Informationen liefern können und stellt diese mit Hilfe eines XSD Schemas dar. Mehr wird durch den Internet-Draft [4] spezifiziert.

V. YANG

Als ein Datenmodell abstrahiert YANG (Yet Another Next Generation) die eigentlichen Daten und bildet diese allgemeingültiger ab. Hier beschreibt es die Inhaltsebene der Architektur, also die Konfigurations- bzw. Statusdaten an sich, aber auch die RPCs und Notifications. Es füllt die Lücke, die vom NETCONF Protokoll offen gelassen wird und ermöglicht so eine standardisierte Sprache. Im Folgenden werden nur Ausschnitte aus allen Möglichkeiten, die Yang bietet präsentiert, da dies sonst zu weit führen würde. Weitere Informationen können in [5] nachgelesen werden.

A. Warum YANG?

Da bereits unzählige Datenmodellierungssprachen existieren, stellt sich die Frage, warum eine neue benötigt wird. Es gab einige Überlegungen XSD (XML Schema Definition) oder andere Sprachen zu erweitern, was aber nie wirklich zufrieden stellen konnte. Eines der größten Hindernisse für XSD war, dass es zu schwer zu erlernen und zu lesen ist. So wird es wie RelaxNG (Regular Language Description for XML New Generation) zu komplex und würde damit die Akzeptanz und somit den gesamten Erfolg gefährden. Zusammengefasst ist zu sagen, dass es bisher keine Sprache gibt, die alle Bedürfnisse, die durch die NETCONF Spezifikation entstehen, abdeckt. Weitere Punkte werden in [6] diskutiert.

B. Struktur

YANG arbeitet mit einer Aufteilung in Module und Submodule, welche hierarchisch gesehen die obersten Elemente bilden. Solche Module enthalten viele Informationen, vor allem aber die Moduldefinitionen, die den eigentlichen Inhalt bereitstellen und von denen einige hier vorgestellt werden.

1) Typen: YANG enthält einige Standarddatentypen, wie beispielsweise int32, wobei deren Wertebereich flexibel definiert werden kann (Listing 7).

Listing 7. type int32

```
type int32{
        range "1|1328..1337|max";
}
```

Mit Hilfe des *typedef* Befehls lassen sich aus den Stammtypen weitere Typen ableiten. In Listing 8 ist zum Beispiel der Code dargestellt, mit dem man Prozente realisieren kann.

Listing 8. typedef Prozent

```
typedef percent {
        type uint8 {
                range "0..100";
        }
        description "Percentage";
}
```

2) Leaf: Leafs stellen im Datenbaum die unterste Einheit dar und repräsentieren die Daten an sich. Man definiert also den Hintergrund der XML-tags, die in der tiefsten Ebene stehen. Im folgenden Listing 9 wird ein Element Port beschrieben.

Listing 9. leaf port

```
leaf port {
 type inet:port-number;
 default 22;
 description
 "The port which the SSH server listens to"
}
```

Im NETCONF Protokoll, könnte dieser Datenknoten dann wie folgt verwendet werden (Listing 10):

Listing 10. leaf port Einsatz

```
<rpc message-id="1337"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc=
 "urn:ietf:params:xml:ns:netconf:base:1.0">
 <edit-config>
  <target>
   <running/>
  </target>
  <config>
   <system
    xmlns="http://example.com/schema/config">
    <services>
     <ssh>
      <port>2022</port>
     </ssh>
    </services>
   </system>
  </config>
 </edit-config>
</rpc>
```

Darüber hinaus existiert noch die *leaf-list*, welche eine Liste gleichartiger Elemente beschreibt und mit einem Array von Werten verglichen werden kann.

3) Container: Container können leafs oder andere Container enthalten und bilden somit zusammenfassende Elemente. Die Arbeitsweise ist wieder an einem Beispiel verdeutlicht (Listing 11):

Listing 11. Container

```
container system {
 container services {
  container "ssh" {
   leaf port {
    type inet:port-number;
    default 22;
    description
    "The port which the SSH server listens to"
   }
  }
 }
}
```

Umgesetzt in XML ergibt sich Listing 12.

Listing 12. Container XML

```
<system>
 <services>
  <ssh>
   <port>2022</port>
   </ssh
 </services>
</system>
```

VI. Zusammenfassung und Ausblick

NETCONF versucht, eine bisher existierende Lücke zu füllen und arbeitet dabei mit flexiblen und dem Nativen zugewandten Methoden. Angestrebt wird dadurch eine breite Unterstützung von Netzwerkgeräten, wie sie in größeren Netzarchitekturen zu finden sind. YANG vervollständigt als Datenmodell das Protokoll und bietet so die notwendig Reife für eine Verbreitung.
Da NETCONF relativ einfach zu erlernen ist, besteht auf der Anwenderseite bereits ein großer Zuspruch, von dem noch mehr zu erwarten ist. Notwendig ist jetzt eine Vielzahl von Implementierungen auf Herstellerseite, damit der Gebrauch vermehrt in die Praxis übergehen kann.

Literatur

[1] http://www.ietf.org/dyn/wg/charter/netconf-charter.html
Stand 10.09.2009.

[2] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "NETCONF configuration protocol," Internet-Draft, draft-ietf-netconf-4741bis-01, July 2009.

[3] A. Bierman and B. Lengyel, "With-defaults capability for NETCONF," Internet-Draft, draft-ietf-netconf-with-defaults-03, August 2009.

[4] M. Scott, M. Bjorklund, and S. Chisholm, “NETCONF monitoring schema,” Internet-Draft, draft-ietf-netconf-monitoring-07, July 2009.
[5] M. Bjorklund, “YANG - a data modeling language for NETCONF,” Internet-Draft, draft-bjorklund-netconf-yang-02, February 2008.
[6] B. Lengyel, “Why we need a NETCONF-specific modeling language - YANG,” Internet-Draft, draft-lengyel-why-yang-00, November 2007.

Sammeln von Würmern und Bots mit Honeypots

Sebastian Eckl
Betreuer: Lothar Braun
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: sebastian.eckl@mytum.de

***Kurzfassung*—Eine stetig steigende Zahl von Internetangriffen und die Verbreitung immer ausgeklügelterer Schadsoftware bedingt die Entwicklung und den Einsatz neuer Verfahren und Möglichkeiten bezüglich Schutz und Abwehr. Honeypots sind reale oder virtuelle Rechnerumgebungen mit Netzwerkanbindung, deren Zweck einzig und allein darin besteht, gezielt Opfer manueller oder automatischer Angriffe und Übernahmen zu werden, um somit unerkannt Informationen über den Angreifer zu erlangen. Da nicht jeder Honeypot-Ansatz in jedem Einsatzszenario Erfolg verspricht, soll die folgende Arbeit einen Überblick über die verschiedenen Honeypot Arten und ihren jeweiligen Einsatzzweck geben.**

***Schlüsselworte*—Honeypot, Low-Interaction Honeypot, High-Interaction Honeypot, Client Honeypot**

I. Einleitung

Seit Februar 2000 [4], [7], als die ersten Distributed Denial of Service (DDoS) Attacken auf große E-Commerce Unternehmen und News Seiten gesichtet wurden, hat sich das Angriffspotential im World Wide Web zu einem gewichtigen Problem entwickelt. Spielte zu Beginn eher die blinde Kraft der Zerstörung eine Rolle, so wandelte sich die Intention im Laufe der Zeit in Richtung realer Internetkriminalität – das Erpressen von Unternehmen und das Ausspähen von Daten im Rahmen des Kreditkartenbetrugs seien hier exemplarisch genannt. Die bislang zugrundeliegende Infrastruktur bilden dabei Botnetze, also Netzwerke - oftmals sogar tausender - kompromittierter Rechner, die von einem Angreifer ferngesteuert werden [6]. Um die dabei eingesetzte Schadsoftware aufzuspüren und zu analysieren, wurde das Konzept der Honeypots entwickelt und zum Einsatz gebracht. Im Folgenden sollen nun die drei Honeypot Varianten Low-Interaction, High-Interaction und Client Honeypot vorgestellt, Einsatzbereiche, Vor- und Nachteile herausgearbeitet sowie eine abschließende Bewertung vorgenommen werden.

II. Definition von Honeypots

Ein Honeypot bezeichnet eine reale oder virtuelle Rechnerumgebung, deren Zweck darin besteht, gezielt Angreifer sowie automatisch verbreitende Schadsoftware (sog. Malware) anzulocken, um die dabei verwendeten Strategien und Techniken anschließend genauer analysieren zu können. Dem potentiellen Angreifer wird dabei eine scheinbar ungesicherte bzw. schlecht gesicherte Rechnerkonfiguration vorgetäuscht. Dieser Rechner erfüllt die Voraussetzung, keiner weiteren Benutzung zugeordnet zu sein, womit sich theoretisch gesehen keine ein- bzw. ausgehenden Verbindungen beobachten lassen dürften. Somit lässt sich jeder Verbindungsversuch entweder als Unfall oder, viel wahrscheinlicher, als Angriffsversuch werten. „False Positives“, also die irrtümliche Einstufung an sich harmlosen Datenverkehrs als Gefahr, sind hiermit minimiert bzw. faktisch ausgeschlossen - ein Hauptvorteil der Honeypots.

Der Wert eines Honeypots liegt in Umfang und Art der Informationen, die er generiert. So lassen sich hierbei auch verschlüsselte Daten (z.B. Tastatureingaben des Angreifers) mitschneiden, auf die gängige „Network Intrusion Detection“ Systeme (NDIS) nicht ansprechen. Der signifikanteste Vorteil von Honeypots liegt in der Tatsache, dass bestimmte Konzepte auch mit noch unbekannten Angriffssignaturen zurechtkommen. Letztere ermöglichen die Entdeckung bis dato unbekannter Schwachstellen [8] durch die Analyse ein- und ausgehender Netzwerkverbindungen sowie der Rechneraktivitäten.

Die Funktionsweise des Honeypots erklärt Aufgabe und Einsatzbereich. Ziel ist nicht die direkte Abwehr, sondern das Sammeln von Informationen über Angreifer und Schadsoftware zur Erforschung und Entwicklung langfristig angelegter Abwehrstrategien auf Basis bestehender Schutzsoftware.

Unterteilen lassen sich Honeypots allgemein in die beiden Oberkategorien Server- und Client-Honeypots. Zu den Server-Honeypots zählen sowohl Low-Interaction als auch High-Interaction Honeypots. Sie zeichnen sich im Wesentlichen dadurch aus, dass sie passiv auf Angreifer, in Form von Clients, warten. Client Honeypots versuchen dagegen aktiv, indem sie z.B. einschlägige Webseiten abgrasen, Schadsoftware zu sammeln.

Einzelne Honeypots lassen sich auch zu einem gesamten Netz zusammenschließen und bilden dann ein sogenanntes Honeynet bzw. eine Honeyfarm [10].

Eingesetzt werden Honeypots u.a. von Herstellern von Sicherheitssoftware (z.B. von Antiviren-Software) zur

stetigen Verbesserung der eigenen Produkte bzw. von Universitäten zu Forschungszwecken und Überwachung des eigenen Netzwerkes.

III. LOW-INTERACTION HONEYPOTS

A. Konzept

Low-Interaction Honeypots stellen die einfachste Honeypot Variante dar. Installation und Betrieb erfordern wenig Aufwand. Low-Interaction Honeypots offerieren einem Angreifer nur eine eingeschränkte Bandbreite an Möglichkeiten, indem sie entweder ein Betriebssystem oder nur ausgewählte Systemdienste, Schwachstellen, etc. emulieren. Sie versuchen damit, einem Angreifer in gewisser Weise ein reales System vorzutäuschen, bieten jedoch in Wirklichkeit kein vollwertiges System an. Oft enthält ein Low-Interaction Honeypot beispielsweise nur eine eingeschränkte Anzahl an Netzwerkdiensten und bietet gerade eben so viele Internetprotokolle (im Wesentlichen TCP und IP), wie zur Interaktion mit dem Angreifer notwendig sind, um ihm Glauben zu machen, er verbinde sich mit einem realen System. Der Grad der Interaktionsmöglichkeiten sollte dabei jedoch hoch genug sein, einen potentiellen Angreifer bzw. automatische Malware auszutricksen und nicht von vornherein zu verlieren.

Low-Interaction Honeypots dienen primär dazu, statistische Daten (z.B. Anzahl der Gesamtangriffe) und grundlegende Informationen über bestimmte Angriffsmuster zu sammeln. Aufgrund des einfach gehaltenen Betriebssystems fehlen komplexe Rechneraktivitäten, weshalb tiefergehende Analysen, wie die Erforschung gespeicherter Schadsoftware, an andere Systeme ausgelagert werden müssen.

Bekannte Low-Interaction Honeypot Distributionen sind honeyd und die Nepenthes Plattform. Per honeyd [11], [12] lassen sich komplexe Netzwerke und der TCP/IP Stack beliebiger Betriebssysteme simulieren. Kleine Scripte emulieren dabei reale Systemdienste und bieten dem Angreifer eingeschränkte Interaktionsmöglichkeiten.

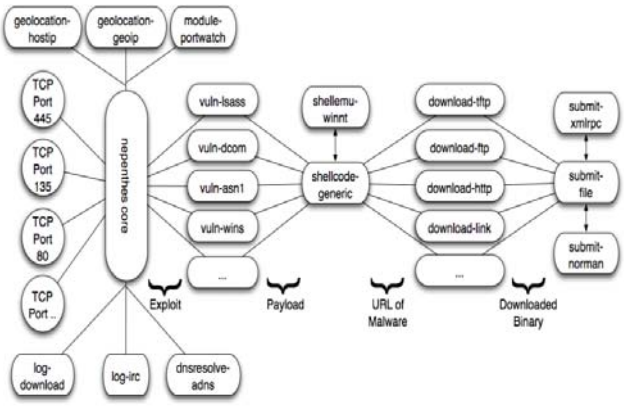


Abbildung. 1. Konzept der Nepenthes Plattform [14]

Die Nepenthes Plattform [14] (siehe Abbildung 1) bietet ein Framework um sich selbst verbreitende Malware im großen Stil zu sammeln. Nepenthes versteht sich dabei nicht als Honeypot per se, sondern vielmehr als Plattform, bestehend aus flexibel austausch- und erweiterbaren Honeypot Modulen. Vorgesehen sind dabei Module zur Emulation von Sicherheitslücken, zum Parsen auszuführender Shellcode Befehle, zum Download der per Shellcode angeforderten Malware und schlussendlich zum Mitschneiden und Speichern sämtlicher Aktivitäten auf dem Honeypot. Eine Erweiterung dieses Konzepts bieten Leita und Dacier mit SGNET [3] (siehe Abbildung 2). Sie verknüpfen Nepenthes mit dem Konzept automatisch dazu lernender Sensoren, die auf verschiedene IP-Adressbereiche weltweit verteilt sind. Sobald ein Angreifer einen Sensor attackiert, werden sämtliche Interaktionen über ein speziell entwickeltes Protokoll an ein zentrales Gateway gesandt und von dortigen Modulen zur Emulation von Systemdiensten und Ausführung von Shellcode abgearbeitet.

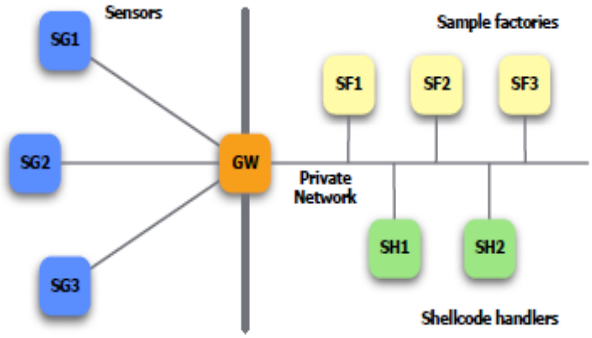


Abbildung. 2. Architektur der SGNET Plattform [3]

B. Abgrenzung

Low-Interaction Honeypots basieren im Gegensatz zu High-Interaction Honeypots auf keinem vollständig ausgestatteten Betriebssystem, sondern emulieren lediglich gewisse Komponenten einer realen Rechnerumgebung. Einem Angreifer ist es somit nicht möglich, ein komplettes System zu kompromittieren und zu übernehmen.

Im Vergleich zu Client Honeypots verhalten sich Low-Interaction Honeypots passiv, d.h. sie warten darauf, angegriffen zu werden. Bestimmte Arten moderner Schadsoftware können somit nicht erfasst werden.

C. Bewertung

Der klare Vorteil von Low-Interaction Honeypots liegt in ihrer Einfachheit. Installation und Inbetriebnahme gestalten sich problemlos. Es werden keine großen Anforderungen an zugrundeliegende Hardware Ressourcen gestellt, weshalb auch betagtere Rechner als Low-Interaction Honeypot in Frage kommen können.

Durch die Simulation bekannter Schwachstellen eignen sie sich hervorragend dafür, die Ausbeutung bereits bekannter Sicherheitslücken zu dokumentieren. Da es einem Angreifer aufgrund der kontrollierten und limitierten Umgebung nicht möglich ist, ein System komplett zu übernehmen, besteht wenig Risiko, dass der Honeypot für weitere kriminelle Zwecke missbraucht werden kann.

In der Einfachheit liegen allerdings auch die größten Nachteile von Low-Interaction Honeypots.

Sie erfordern einen hohen Arbeitsaufwand an Entwicklung und Aktualisierung, denn die zu emulierenden Sicherheitslücken müssen ständig von Hand programmiert und implementiert werden.

Aufgrund ihrer Konzeption und der durch die Emulation einzelner Schnittstellen eingeschränkten Möglichkeiten sind sie eher weniger geeignet für das Sammeln sogenannter Zero-Day Exploits, also noch unbekannter bzw. erst seit kurzer Zeit bekannter Sicherheitslücken, für die noch kein Patch existiert.

Low-Interaction Honeypots geben lediglich vor, ein reales System darzustellen, verweigern dem Angreifer jedoch die mächtigen Optionen einer realen Root-Shell. Experten könnten den Täuschungsversuch durchschauen und frühzeitig abgeschreckt werden.

Aufgrund der Warteposition von Low-Interaction Honeypots müssen die Angreifer sowie mögliche Schadsoftware zwingend aktiv zum Honeypot gelangen. Da gewisse IP-Adressbereiche aber mehr oder weniger zufällig von Angreifern abgegrast werden, kann es passieren, dass der Honeypot gar nicht bzw. nur sehr selten attackiert wird.

Netzwerkes.

IV. HIGH-INTERACTION HONEYPOTS

A. *Konzept*

Der High-Interaction Honeypot repräsentiert ein vollwertiges System, z.B. in Form eines gewöhnlichen Standardrechners, Routers oder Switches. Er simuliert also nicht entsprechende Schnittstellen bzw. Sicherheitslücken, sondern präsentiert sich in Form eines realen, wenn auch sehr schwach bis ungeschützten Betriebssystems, welches sich so auch im täglichen Gebrauch finden lässt. High-Interaction Honeypots bieten Angreifern damit die gesamte Bandbreite an Interaktionsmöglichkeiten, also die Möglichkeit, sämtliche bekannte, aber auch noch unbekannte Sicherheitslücken ausbeuten zu lassen. Sie liefern ein breiteres Datenspektrum und ermöglichen somit tiefere Erkenntnisse bezüglich Vorgehensweise, verwendeter Programme und Absichten der Eindringlinge.

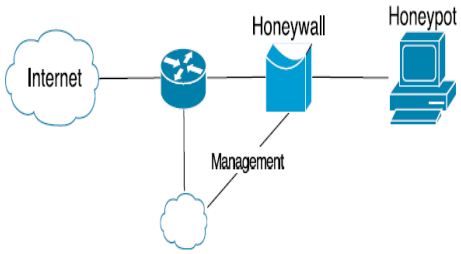


Abbildung 3: High-Interaction Honeypot geschützt durch Honeywall [7]

High-Interaction Honeypots müssen sehr stark überwacht werden, um Angreifer frühzeitig erkennen zu können und eventuellem Missbrauch gezielt vorzubeugen. Hierbei bietet sich das Konzept der sogenannten „Honeywalls“ an [13] (siehe Abbildung 3). Ein prominentes Beispiel bildet die Honeywall des Honeynet Projects, welche als Teil von GenIII und GenIV Honeynets konzipiert wurde. Die Honeywall bildet dabei eine Art zentrales Gateway - einem oder mehreren Honeypots vorgeschalten. Sie erlaubt das Sammeln von Daten, also sämtlicher Aktivitäten innerhalb des Honeynets bzw. eingehender und ausgehender Kommunikation, ohne Kenntnis des Angreifers. Darüberhinaus enthält sie auch Konzepte eines Network Intrusion Detection Systems, indem sie auffälligen ein- oder ausgehenden Traffic analysiert und eventuell kompromittierte Honeypots blockiert. Um das Risiko von DoS Attacken gegen andere Rechner zu minimieren, lassen sich per Netfilter Firewall auch gezielt spezielle Ports erlauben bzw. sperren (z.B. Port 22 für ssh-Verbindungen) sowie die Anzahl ausgehender Verbindungen limitieren. Hierbei muss jedoch die Balance gewahrt werden. Der Angreifer soll in der Lage sein, sich mit anderen Systemen im Internet zu verbinden, allerdings in eingeschränktem Rahmen, sodass von dem kompromittierten Rechner kein Bedrohungspotential ausgehen kann. Normalerweise wird die Honeywall als transparente Bridge realisiert, welche auf der Datenschicht fungiert. Ihr ist auf beiden Interfaces keine IP-Adresse zugewiesen, weshalb ein potentieller Angreifer nicht so leicht erkennen kann, dass ihm ein weiteres Netzwerkgerät vorgeschalten wird.

B. *Abgrenzung*

High-Interaction Honeypots unterscheiden sich von Low-Interaction Honeypots primär im bereitgestellten Funktionsumfang. Sie bieten, auf Basis alltäglicher Betriebssysteme, im Wesentlichen den gleichen Funktionsumfang, der Angreifern auch bei anderen ungeschützten Rechnern im Internet zur Verfügung stünde. Auf diese Weise lieferten sie beispielsweise bereits Erkenntnisse über strategisches Verhalten der Angreifer. In ihren Experimenten haben Alata et. al. [5] herausgefunden, dass die Vorgehensweise zumeist in zwei Schritten erfolgt. Zu Beginn wird, meist automatisch per Skript, per „dictionary attack“ versucht, anhand eines Wörterbuchs,

schwache Passwörter ausfindig zu machen. Ist ein solches Passwort gefunden, verbindet sich der Angreifer einige Tage später manuell, um entsprechende Kommandos auszuführen.

Die High-Interaction Honeypots sind ebenfalls passiv konzipiert und warten auf potentielle Angreifer. Bestimmte Arten moderner Schadsoftware bleiben wiederum außen vor.

C. Bewertung

Mit Hilfe eines High-Interaction Honeypots lässt sich prinzipiell der gesamte Kompromittierungsprozess, von der anfänglichen Auskundschaftung bis hin zur eigentlichen Übernahme, verfolgen:

- Wie geht der Angreifer bei der Auswahl seiner Ziele vor?
- Welche Techniken verwendet er, um mehr über ein bestimmtes System in Erfahrung zu bringen?
- Wie attackiert er den ausgewählten Zielrechner und welche Programme verwendet er?
- Welche Sicherheitslücken nützt er aus, um das System zu übernehmen?

Die sich ergebenden Erkenntnisse sowie die Vorgehensweise auf dem Rechner selbst erlauben eine Analyse der Methoden und Motive eines Hackers. Beispielsweise konnte so einiges über die typische Vorgehensweise bei Phishing Attacken und weiteren Formen des Identitätsdiebstahls herausgefunden werden. Der Prozess der Analyse ist allerdings zeitaufwendig und im Vergleich zu Low-Interaction Honeypots steigt der Ressourcen Verbrauch.

Der größte Nachteil liegt in der Gefahr der vollständigen Kompromittierung des Honeypots. Nachlässige Überwachung kann einem Angreifer Möglichkeit zu weiterem Missbrauch bieten. Der Honeypot mutiert zum Bot oder zur Schadsoftware- Schleuder und gefährdet nun weitere Rechner innerhalb oder außerhalb des eigenen Netzwerkes. Hier läuft der Honeypot Betreiber Gefahr, in rechtliche Schwierigkeiten zu geraten.

V. CLIENT HONEYPOTS

A. Konzept

Mit der steigenden Attraktivität und Benutzung des Internets, steigt auch die Anzahl von Sicherheitslücken in Internet Anwendungen, z.B. in Internet Browsern wie dem Microsoft Internet Explorer. Der typische Angreifer wird über eine Sicherheitslücke versuchen, eine Art Malware auf dem kompromittierten Rechner zu installieren. Diese Attacken erfordern keinerlei Aktionen von Seiten der Benutzer, lediglich der Besuch der Webseite reicht für eine Infektion aus. Unter Umständen erlangt der Angreifer somit die volle Kontrolle über das infizierte System und kann nach Belieben damit verfahren. Mögliche Optionen wären die Installation eines IRC-Bots, um den Rechner in ein bestehendes Botnetz einzureihen bzw. die Installation von Spyware oder einem Keylogger, um die Daten des Opfers (insbesondere Kreditkarteninformation, Bankdaten) auszuspähen.

Um sich vor diesen Gefahren zu schützen, bedarf es zwangsweise einer Ausweitung des klassischen Honeypot Prinzips: kein passives Warten, bis der Honeypot attackiert wird, sondern aktive Suche nach entsprechenden Schadsoftware-Distributionsstätten. Gemäß diesem Konzept arbeiten Client Honeypots, je nach Verwendungszweck auf Basis emulierter Schnittstellen (siehe Low-Interaction Honeypot) oder realem Betriebssystem (siehe High-Interaction Honeypot).

Die Aufgabe eines Client Honeypots ist das Auffinden, das Abspeichern und das Analysieren von Webseiten, die speziell von Angreifern mit Schadsoftware präpariert wurden, um den Internet Browser zu infizieren.

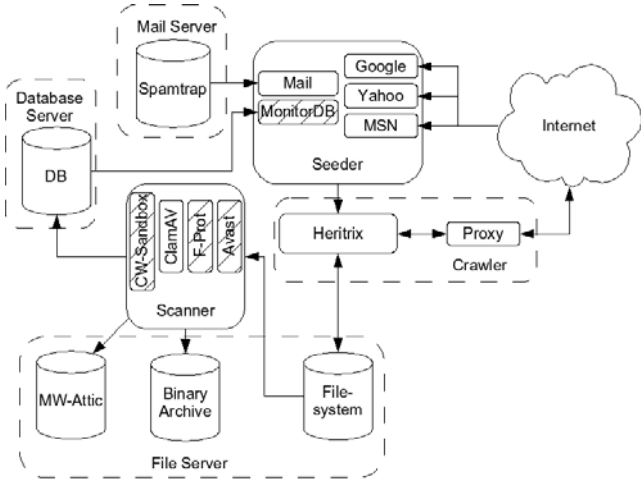


Abbildung 4: Schematischer Überblick des Monkey-Spider Konzepts [1]

Ikinci et. al. [1] beschreiben in ihrem Artikel „Monkey-Spider“ einen Client Honeypot Ansatz auf Basis des Low-Interaction Konzepts (siehe Abbildung 4). Sogenannte Web-Seeder durchforsten dabei entweder gängige Suchmaschinen bzw. extrahieren einschlägige URLs aus Spam-Mails. Eine Alternative wäre die Benutzung spezieller Blacklists, die eine Übersicht entsprechend einschlägiger Internetadressen enthalten. Gemäß Ikinci et. al. kommt dann für den Besuch der jeweiligen extrahierten URLs meist eine angepasste Version eines Web-Crawlers zum Einsatz, wie er in abgewandelter Form auch Suchmaschinenbetreibern zur Webindexierung dient. Der Crawler speichert die Webseite dann in einer Datei auf dem Server, die anschließend zur weiteren Analyse bereitsteht. Diese kann sowohl zentral als auch dezentral, weltweit verteilt, erfolgen. Ziel ist es, eine umfangreiche Datenbank über gefundene Schadsoftware zu erstellen.

Wang et. al. [15] beschreiben in „Automated Web Patrol with Strider HoneyMonkeys” einen Client Honeypot Ansatz basierend auf dem High-Interaction Prinzip in virtueller Umgebung. Hauptziel ist das Aufzeichnen sämtlicher Veränderungen, die nach erfolgreichem

Ausbeuten einer Sicherheitslücke, auf dem Honeypot stattfinden. Analog zu Ikinci et. al. müssen dabei zunächst entsprechende Website URLs extrahiert werden, ebenfalls unter Zuhilfenahme von Suchmaschinen, Spam-Mails oder Blacklists. Anschließend werden die Webseiten, im Gegensatz zur Low-Interaction Variante, mit vollständigen Browsern (je nach Konzept un- bzw. gepatcht) angesteuert. Hierbei gilt es, menschliches Surfverhalten so gut wie möglich nachzuempfinden. Bei erfolgreicher Infektion wird dem Angreifer eine gewisse Zeitspanne gewährt, in der er auf dem System Veränderungen vornehmen kann. Anschließend generiert HoneyMonkey eine XML-Datei mit einer Gesamtübersicht über sämtliche Veränderungen am vormaligen System. Enthalten sind darin Informationen über neu erstellte bzw. modifizierte ausführbare Dateien, über gestartete Prozesse, über neu angelegte oder modifizierte Registry-Einträge (im Falle von Windows), über die ausgebeutete Sicherheitslücke sowie über die vom Browser eventuell besuchten Redirect-URLs. Anschließend wird das infizierte System automatisch komplett neu aufgesetzt um somit für weitere Angreifer erneut in unberührtem Zustand zur Verfügung zu stehen.

B. Abgrenzung

Von den serverseitigen Honeypots, also Low- und High-Interaction Honeypots, unterscheiden sich die Client Honeypots vor allem in ihrer Konzeption. Sie warten nicht darauf, angegriffen zu werden, sondern suchen aktiv nach Angreifern. Trotzdem basieren Client Honeypots auf den Grundkonzepten der Low- und High-Interaction Honeypots, fangen per emulierter Umgebung Schadsoftware ein oder lassen eine erfolgreiche Infektion zu.

C. Bewertung

Client Honeypots reagieren auf die sich langsam wandelnde Distribution von Schadsoftware. Sie erlauben es, gezielt selbst nach Angreifern zu suchen und verzichten dabei auf das u.U. langwierige Warten serverseitiger Honeypots. Die Ausbeute an Schadsoftware ist so in der Regel um einiges größer.

Hauptproblem der Server Honeypots bleibt die Tatsache, dass sie mit bestimmten Varianten moderner Schädlinge nicht in Berührung kommen, da sich diese vom Konzept her anders verbreiten. Client Honeypots füllen diese Lücke, indem sie aktiv (einschlägige) Webseiten abgrasen und dort nach potentieller Schadsoftware fahnden.

High-Interaction Client Honeypots eignen sich aufgrund ihrer Konzeption in Form eines vollständigen Betriebssystems auch sehr gut dafür, sogenannte Zero-Day Exploits aufzudecken.

Basierend auf den Grundkonzepten von Low- und High-Interaction Honeypots übernehmen Client Honeypots allerdings auch deren jeweilige Schwachstellen. Low-Interaction Client Honeypots sind aufgrund emulierter Systemumgebung und Sicherheitslücken ebenfalls weniger geeignet, Zero-Day Exploits zu sammeln. High-Interaction Client Honeypots benötigen eine stärkere Überwachung.

VI. AKTUELLE IMPLEMENTIERUNGEN

Im Folgenden sollen kurz aktuelle Konzepte vorgestellt werden.

Der Nepenthes Plattform [14], aus dem Bereich der Low-Interaction Honeypots, liegt ein Framework zugrunde, dessen Hauptziel es ist, sich selbst verbreitende Schadsoftware im großen Stil zu sammeln. Nepenthes besteht dabei aus austausch- und erweiterbaren Honeypot Modulen. Der Hauptvorteil liegt somit in der flexiblen Erweiterbarkeit und damit einhergehenden Zukunftsfähigkeit. Allerdings kann Nepenthes nicht das gesamte Ausmaß eines Angriffs bestimmen und es ist nur Kontakt mit sich automatisch verbreitender Schadsoftware bzw. aktiven Angreifern möglich.

SGNET [3] gelingt es, die vielleicht wichtigste Einschränkung von Nepenthes - den Prozess der manuellen Anpassung und Erweiterung emulierter Schnittstellen - aufzuheben, durch automatisch dazu lernende Sensoren, die auf verschiedene IP-Adressbereiche weltweit verteilt sind.

Alata et. al. [5] experimentierten mit einem High-Interaction Honeypot auf virtueller Basis. Ihr Ziel: die Analyse der Aktivitäten eines Angreifers, sobald dieser sich erfolgreich Zugang zum Honeypot verschafft hatte. Angriffspunkt waren hierbei schwache ssh-Passwörter. Dabei konnte einiges über die allgemeine Vorgehensweise, den Angreifertypus (z.B. Mensch oder Maschine) sowie die generellen Fähigkeiten der Angreifer in Erfahrung gebracht werden. Die Ergebnisse sind zu deuten unter Berücksichtigung der bewussten Fokussierung auf die ssh-Schwachstelle. Andere Sicherheitslücken wurden ausgeblendet und nicht mit einbezogen.

Freiling et. al. [7] verwendeten ein Honeynet auf Basis ungepatchter, virtueller Windows 2000 und XP Versionen. Zur Aufzeichnung des Netzwerkverkehrs und zum Schutz vor vollständiger Kompromittierung befanden sich sämtliche Honeypots hinter einer Honeywall. Alle 24 Stunden wurden die Honeypots zudem komplett neu aufgesetzt.

Im Bereich der Low-Interaction Client Honeypots agieren Ikinci et. al. mit dem MonkeySpider Projekt sowie Seifert et. al. mit HoneyC.

MonkeySpider [1] konzentriert sich auf das Sammeln von Schadsoftware, die sich per Webseiten verbreitet. Der Such- und Sammelprozess verläuft dabei recht zügig, neue Bedrohungen lassen sich aber nicht in vollem Umfang ausfindig machen. Prinzipiell wird zwar der Gesamtprozess beschleunigt, die Analyse der gespeicherten Schadsoftware erfordert jedoch hohen Zeitaufwand, da diese nicht auf dem jeweiligen Honeypot selbst sondern mithilfe weiterer Tools auf anderen Rechnern ausgeführt werden muss. Aufgrund der sich ständig wandelnden Internetumgebung bleibt es, trotz aktiver Suche, unmöglich sämtliche Bedrohungen zu erfassen.

HoneyC [2] besteht im Wesentlichen aus drei modular aufgebauten Bausteinen, welche zum Datenaustausch untereinander auf das XML-Format (in Form von HTTP

Requests und Responses) setzen. Die „Queuer“-Komponente liefert die zu besuchenden URLs. Die „Visitor“-Komponente arbeitet diese ab und leitet Informationen an die „Analysis Engine“ (basierend auf Snort Rules). Letztere untersucht anschließend, inwieweit die betroffenen Webseiten Bedrohungspotenzial enthalten. Die flexible Erweiterbarkeit und damit einhergehende Zukunftsfähigkeit machen die Stärke von HoneyC aus. Allerdings ist HoneyC nicht gegen „False Positives“ gefeit.

Der von Wang et. al. [15] beschriebene Client Honeypot Ansatz ermöglicht das Aufzeichnen sämtlicher Veränderungen, die nach erfolgtem Ausbeuten einer Sicherheitslücke, auf dem Honeypot stattfinden, sowie die Möglichkeit, das Ausbeuten neuer, unbekannter Sicherheitslücken festzustellen. Den Autoren gelang es unter anderem, den ersten Zero-Day Exploit auf Basis der javaprxy.dll Schwachstelle ausfindig zu machen.

VII. ZUSAMMENFASSUNF

Aufbau und theoretische Funktionsweise von Low-Interaction Honeypots, High-Interaction Honeypots und Client Honeypots wurden in der vorliegenden Arbeit erläutert. Die unterschiedlichen Vor- und Nachteile der einzelnen Konzepte belegen deren individuelle Daseinsberechtigung. Trotzdem lässt sich wohl prognostizieren, dass die Zukunft den aktiveren und flexibleren Konzepten aus dem Bereich der Client Honeypots gehören wird.

Denkbar sind zudem kreative Kombinationen – basierend auf den Grundkonzepten der Low- und High-Interaction Honeypots – zu sogenannten hybriden Honeypot Architekturen [9].

Als wichtigste Herausforderung zeichnet sich ab, neuen Methoden der Internetkriminalität schnell wirksam begegnen zu können. An diesem Kriterium werden bestehende wie zukünftige Honeypot Konzepte gemessen werden.

LITERATUR

[1] Ali Ikinci, Thorsten Holz, Felix Freiling (2008), ‘Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients‘, in Proceedings of Sicherheit 2008, Gesellschaft für Informatik.

[2] Christian Seifert, Ian Welch, Peter Komisarczuk (2006), ‘HoneyC - The Low-Interaction Client Honeypot’, http://homepages.mcs.vuw.ac.nz/~cseifert/blog/images/seifert-honeyc.pdf, zugegriffen: 11.09.2009.

[3] Corrado Leita, Marc Dacier (2008), ‘SGNET: A Worldwide Deployable Framework to Support the Analysis of Malware Threat Models’, in Proceedings of the Seventh European Dependable Computing Conference (EDCC’07), IEEE Computer Society.

[4] E. Alata, Marc Dacier, Y. Deswarte, M. Kaâniche, K. Kortchinsky, V. Nicomette, Van H. Pham, Fabien Pouget (2005), ‘CADHo: Collection and Analysis of Data from Honeypots’, in Proceedings of the 5th European Dependable Computing Conference (EDDC'05).

[5] E. Alata, V. Nicomette, M. Kaâniche, M. Dacier, M. Herrb, (2006), ’Lessons learned from the deployment of a high-interaction honeypot’, in Proceedings of the Sixth European Dependable Computing Conference (EDCC'06), IEEE Computer Society.

[6] Evan Cooke, Farnam Jahanian, Danny Mcpherson (2005), ‘The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets’, in Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI).

[7] Felix C. Freiling, Thorsten Holz, Georg Wicherski (2005), ‘Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks’, in Proceedings of 10th European Symposium on Research in Computer Security, (ESORICS).

[8] Marc Dacier, Fabien Pouget, Hervé Debar (2004), **‘**Honeypots: Practical Means to Validate Malicious Fault Assumptions’, in Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04), IEEE Computer Society.

[9] Michael Bailey, Evan Cooke, David Watson, Farnam Jahanian (2004), ‘A hybrid honeypot architecture for scalable network monitoring’, in Technical Report CSE-TR-499-04.

[10] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage (2005), **‘**Scalability, fidelity, and containment in the potemkin virtual honeyfarm’, Source ACM SIGOPS Operating Systems Review, vol. 39, no. 5, Dec. 2005

[11] Mohamed Kaâniche, Y. Deswarte, Eric Alata, Marc Dacier, Vincent Nicomette (2005), ‘Empirical analysis and statistical modeling of attack processes based on honeypots’, in Workshop on empirical evaluation of dependability and security (WEEDS).

[12] Niels Provos (2003), ‘A Virtual Honeypot Framework’, in Proceedings of the 13th USENIX Security Symposium.

[13] Niels Provos, Thorsten Holz (2007), ‘Virtual Honeypots: From Botnet Tracking to Intrusion Detection’, Addison Wesley Professional.

[14] Paul Baecher, Markus Koetter, Maximillian Dornseif, Felix Freiling (2006), ‘The nepenthes platform: An efficient approach to collect malware’, in Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID), Springer.

[15] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King (2006), ‘Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites that Exploit Browser Vulnerabilities’, in Proceedings of NDSS 2006.

Analyse von Würmern und Bots

Cornelius Diekmann
Betreuer: Lothar Braun
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: diekmann@in.tum.de

***Kurzfassung*— Die Bedrohung durch Würmer und Bots im Internet nimmt stetig zu. Sowohl im akademischen, als auch im kommerziellen Umfeld werden Wurm- und Botdateien über verschiedene Honeypots gesammelt. Um die Bedrohung, die von ihnen ausgeht, zu verstehen und zu bekämpfen, stellt sich die zentrale Frage, was diese Dateien auf dem Hostcomputer und im Netzwerk machen. In dieser Arbeit wird gezeigt, wie man gesammelte Würmer und Bots statisch und dynamisch analysieren kann. Es zeigt sich, dass ein Großteil der Analyse automatisch durchgeführt werden kann, jedoch ist zumindest für die Auswertung und Nutzung der Ergebnisse ein menschlicher Operator nötig.**

***Schlüsselworte*— Würmer, Bots, Botnet, Bot-Netz, statische Analyse, dynamische Analyse, Malware Analyse**

I. Einleitung

Die Anzahl und Artenvielfalt von Würmern und Bots steigt seit geraumer Zeit rapide an. Längst sind Würmer und Bot-Netze kein Spielzeug von interessierten Jugendlichen mehr, sondern eine Bedrohung, die aus dem professionellen Umfeld mit kriminellen Absichten betrieben wird. Im Internet vergeht beispielsweise kein Tag ohne eine sogenannte „DDoS Attacke“, wie McIntyre in [1] zeigt. Durch Honeypot Systeme wird versucht, ein Exemplar dieser Schädlinge zu erhalten. Die große Anzahl bekannter Schädlinge[1] zeigt, dass diese Malware dann so schnell wie möglich analysiert werden muss, um der Bedrohung entgegenzuwirken. Die Analyse sollte aufgrund des hohen Malwareaufkommens automatisch geschehen.

Der folgende Abschnitt II beschäftigt sich mit der Begriffserklärung von Würmern und Bots. Nachdem geklärt wurde, welche möglichen Funktionen diese Programme aufweisen und welches Verhalten von ihnen bei der Analyse zu erwarten ist, wird in den darauffolgenden Abschnitten III und IV auf die Untersuchung von unbekannten Wurm- und Bot-Binärdateien eingegangen. Dieser Punkt untergliedert sich in die statische und dynamische Analyse. Schließlich fasst Kapitel V zusammen, wie die gesammelten Ergebnisse im Kampf gegen Bedrohungen eingesetzt werden können.

II. Überblick über Würmer und Bots

Als Wurm wird laut Kohring und Schafmeister in [3] ein Computerprogramm bezeichnet, dass sich selbst von einem Computer auf einen anderen kopiert. Dabei ist es nicht auf die aktive Hilfe des Benutzers angewiesen. Nach dieser Definition ist ein Wurm keine Schadsoftware, jedoch werden Würmer in den meisten Fällen nur entwickelt, um Schadcode auf den betroffenen Computern zu installieren. Als Bot-Netz bezeichnet man ein großflächiges Netzwerk von Computern, welche eine spezifische Bot-Software ausführen. Diese Computer werden „Zombies“ genannt. Durch die Bot-Software lassen sich die befallenen Computer fernsteuern, wodurch ein Bot-Netz-Operator seine Armee von Zombies über einen oder mehrere Command-and-Control (C&C) Server kontrollieren kann [4]. Heutzutage verschwimmt jedoch die klare Trennung zwischen Würmern und Bots, denn beide Gattungen werden unter dem Begriff Malware zusammengefasst. In dieser Arbeit werden deshalb die Wörter Bot und Wurm weitgehend als Synonym verwendet.

Für einen Wurm gibt es verschiedene Möglichkeiten, sich zu verbreiten. Während Bots von sogenannten „Scriptkiddies“ sich bevorzugt automatisch über Sicherheitslücken auf den niederen OSI-Schichten verbreiten [5], verbreiten sich kommerzielle und professionelle Bot-Netze bevorzugt auf der Anwendungsschicht, beziehungsweise direkt über Social Engineering [1], [4]. Eine einfache Methode hierfür ist der Massenversand von präparierten Emails, mit dem Wurm im Anhang oder Links auf die Malware. Beide Formen sind heutzutage anzutreffen.

Damit Würmer ihre Arbeit ungestört und dauerhaft erledigen können, nisten sie sich meist sehr tief auf dem befallenen System ein. Dazu gehört beispielsweise, dass die Malware bei jedem Systemstart ausgeführt wird und meistens dauerhaft als Hintergrundprozess aktiv bleibt. Des Weiteren können Würmer versuchen, ihre Präsenz auf dem System vor dem Benutzer und vor Sicherheitssoftware zu verstecken. Damit Virenscanner die Malware nicht finden können, kann die Binärdatei polymorph[2] sein. TimeLock Puzzles, wie sie von Nomenumbra in [6] vorgestellt werden, stellen eine weitere mächtige Möglichkeit dar, um vor Virenscannern unentdeckt zu bleiben. Würmer haben auch die Möglichkeit, laufende Schutzsoftware zu deaktivieren. Rootkits hingegen nisten sich sehr tief in das Betriebssystem ein und können dafür sorgen, dass Systemfunktionen in bestimmten Fällen falsche Antworten geben. So kann

[1] Die Datenbank von Symantec's „Endpoint Protection 11“ enthält am 15.09.09 insgesamt 4568898 bekannte Bedrohungen und Risiken [2]

[2] Polymorphie bedeutet in diesem Zusammenhang, dass ein Programm unterschiedliche Binärrepräsentationen annehmen kann, semantisch dabei jedoch gleich bleibt.

beispielsweise die Anfrage an das Betriebssystem, ob sich eine bestimmte Datei auf der Festplatte befindet, oder ein bestimmter Prozess läuft, von dem Rootkit abgefangen werden, um die Existenz der Malware zu verbergen. Durch Bootkits [7] kann sogar sichergestellt werden, dass die Malware bereits vor dem Systemstart ausgeführt wird und somit die absolute Kontrolle über das laufende Betriebssystem hat. Hinzu kommt, dass Bot-Netze ihr C&C Protokoll verschlüsseln können und zur Kommunikation Protokolle wie HTTP verwenden können, wodurch Bot-Netz-Traffic in einem Netzwerk sehr schwer zu erkennen ist. Es ist auch denkbar, dass Bots ihr Host-System nach Debuggern absuchen und eine vm[3]-Erkennung durchführen, um der automatischen Analyse zu trotzen. Durch automatische Updates der Botsoftware können die Bot-Netz-Operatoren dem Benutzer immer einen Schritt voraus sein.

Die Hauptaufgabe einer Malware ist jedoch meistens eine Andere, einige mögliche Funktionen seien hier kurz aufgelistet:

- Zerstörung von Daten
- Zerstörung von Hardware
- Erpressung der Benutzer, zum Beispiel durch Verschlüsselung ihrer Daten
- Diebstahl sensibler Daten
- Installation und Transport von Viren
- Eingliederung des betroffenen Rechners in ein Bot-Netz

Besonders auf den letzten Punkt wird diese Arbeit genauer eingehen, da dieser Angriff sehr weit verbreitet ist und weitreichende Auswirkungen für die Opfer haben kann.

Bot-Netze werden unter Anderem für die folgenden Aufgaben eingesetzt [5]: Durch Distributed Denial-of-Service(DDoS) Attacken wird versucht, den Dienst eines Ziels in die Knie zu zwingen. Dies kann durch eine einfache Flut von Paketen und der daraus resultierenden Überladung der Bandbreite des Opfers geschehen, oder durch spezielle Attacken, die alle Rechenressources des Opfers aufbrauchen.

Der Versand von ungewollten Massenemails (Spamming) erfolgt fast ausschließlich über Bot-Netze. In [4] wird gezeigt, dass circa 79% aller Spamnachrichten von nur sechs verschiedenen Bot-Netzen verschickt werden.

Durch Traffic Sniffing, Keylogging, und Man-in-the-Browser Attacken [8] können sensible Informationen wie Benutzernamen, Passwörter und Kontodaten gestohlen werden, oder Überweisungen auf fremde Konten umgelenkt werden. Weitentwickelte Malware wie ZeuS kann sogar Screenshots aller Benutzereingaben erstellen, um speziell Loginvorgänge beim Onlinebanking abzufangen, wie Kurtenbach in [9] aufgedeckt hat. Bot-Netze können eingesetzt werden, um Identitätsdiebstahl im großen Umfang zu betreiben. Ein Zombie-PC lässt sich auch zum Betreiben einer Phishing-Seite missbrauchen.

Oft nutzt ein Bot-Netz-Operator sein Bot-Netz, um neue Zombies in sein Netz einzugliedern und somit noch mächtiger zu werden.

[3] virtuelle Maschine

Viele Bot-Netze haben die Funktionalität, beliebige Programme nachzuladen und auszuführen. So kann ein Bot-Netz-Operator nicht nur Updates für seine Zombies verteilen, er kann seinem Netz befehlen, eine spezielle Seite zu besuchen und dort Werbung anzuklicken. Die Werbeeinnahmen erhält somit der Bot-Netz-Operator. Zusätzlich kann auch Werbung auf den Zombie-PCs installiert werden, wodurch der Bot-Netz-Operator eine weitere Einnahmequelle hat. Im Speziellen kann auch ein Werbedienstleister (wie beispielsweise Google AdSense) missbraucht werden, um gezielt Werbung eines konkurrierenden Unternehmens anzuklicken. Somit wird der Klick-Zähler künstlich erhöht und das angegriffene Unternehmen muss entweder hohe Werbekosten an Google zahlen, oder das Kontingent wird aufgebraucht und es wird keine Werbung mehr für dieses Unternehmen im laufenden Monat angezeigt.

Da jeder Zombie eine eigene IP-Adresse hat, eignen sich Bot-Netze auch hervorragend für die Manipulation von Online-Umfragen/Spielen.

Oft werden Bot-Netze von ihren Operatoren verkauft oder vermietet. Viele Bot-Netze bieten die Möglichkeit, beliebige ausführbare Dateien nachzuladen und sind damit für jeden erdenklichen Zweck einsetzbar. Wie das Beispiel ZeuS [9] zeigt, werden Bot-Netze vermietet, verkauft und gestohlen. Es existiert sogar ein ganzer Bereich der sich nur mit Support, wie dem Einrichten eines neuen Bot-Netzes, dem Verkauf eines Bot-Netzes und dem Mieten eines Bot-Netzes beschäftigt. In [9], [10] wird gezeigt, dass Bot-Netze von jedem – sogar ohne technische Vorkenntnisse – betrieben werden können.

Während Bacher et al. in [5] zu dem Schluss kommen, dass viele Bot-Netze von sogenannten „Scriptkiddies“ geleitet werden, zeigen die Studien [4], [9], dass Bot-Netze im großen Stil zu kommerziellen Zwecken genutzt werden. Dabei muss häufig zwischen den technisch-fachkundigen Malwareautoren und den oft technisch unwissenden, kriminellen Nutzern des Bot-Netzes unterschieden werden.

A. Kommunikation mit dem C&C Server

Es gibt verschiedene Möglichkeiten, wie ein Bot mit seinem C&C Server Kontakt aufnehmen kann. Seit den Anfangszeiten der Bot-Netze verbinden sich die Zombies zu einem IRC-Server und erhalten dort ihre Befehle. Modernere Bots nutzen Protokolle wie HTTP oder SMTP, da so der Bot-Traffic schwerer zu erkennen ist. Die meisten Bots heutzutage verbinden sich zu einem zentralen C&C Server, wobei die Möglichkeiten der Bots diesen Server zu finden, sehr unterschiedlich sind. Einfache Bots haben die IP-Adresse ihres C&C Server statisch in ihrem Code eingebettet, während sehr weit entwickelte Ansätze, wie DNS Fast Flux und algorithmisches Durchprobieren mehrerer DNS-Namen zu finden sind. Auch p2p Netzwerke zur Bot-Netz-Steuerung sind anzutreffen [4]. Einen detaillierten Überblick über die Command und Control Strukturen eines Bot-Netzes bieten D. und S. Dietrich in [11].

III. Statische Analyse

Die statische Analyse versucht eine Whitebox-Sicht auf den Code und die Logik der Malware zu erlangen, das bedeutet,

man versucht die interne Arbeitsweise der Malware zu verstehen. Der Reverse-Engineering-Prozess kann in zwei Phasen unterteilt werden: Disassemblierung und Dekompilierung. Die Aufgabe der Disassemblierungsphase besteht darin, aus der binären Repräsentation des Programms eine symbolische Darstellung in Form von Assembler Code zu gewinnen. Dekompilierung ist der Prozess, aus diesem Code die Logik und die semantischen Strukturen (und sogar den Quellcode in einer menschenlesbaren Hochsprache) zu extrahieren [12].

A. Disassemblierung

Die Disassemblierung kann heutzutage zu großen Teilen automatisiert mit Hilfe von Programmen durchgeführt werden. Ein sehr bekannter Disassembler ist IDA Pro [13]. Jedoch versuchen Programmautoren seit vielen Jahren ihre Binärdateien vor einer automatischen Disassemblierung zu schützen. Beispielsweise werden sogenannte „Junk Bytes“ an nicht erreichbare Stellen der Binärdatei eingefügt, wodurch viele Disassembler verwirrt werden. Durch die variable Länge der Maschinenbefehle auf x86-Rechnern, wird der komplette darauf folgende Maschinencode falsch interpretiert.

```
Adresse     Befehl
00000000    jmp ZIEL
00000002    <Junk Bytes>
00000004    Rücksprungadresse:
            weiter im Programm

——— Das Disassemblierungsergebnis ———
00000000    jmp ZIEL
00000002    AND
00000004    Parameter 1 für AND
00000006    Parameter 2 für AND
00000008    Falschinterpretation ab hier
```

Abbildung 1. Der obere Teil zeigt einen mit Junk Bytes beispielhaft präparierten Code. Der untere Teil zeigt, wie die Disassemblierung dabei fehlschlagen kann.

Abbildung 1 zeigt im ersten Teil des Beispiels einen durch zwei Junk Bytes präparierten Code. Die Junk Bytes sollen der Binärrepräsentation des Maschinenbefehls AND entsprechen. Im originalen Programm werden die Junk Bytes niemals angesprungen. Ein schlechter Disassembler kann dies jedoch nicht immer erkennen und interpretiert diese Bytes als Maschinenbefehl, wodurch die darauf folgenden Bytes als Parameter interpretiert werden und die Disassemblierung fehlschlägt. Jedoch gibt es auch automatisierte Techniken, die dies erkennen und behandeln können [12].

B. Dekompilierung

Allerdings wird der Einsatz eines Menschen nötig, wenn der Code der Malware verschleiert[4] ist. Dabei liefert der Disassembler einen Code, der korrekt, jedoch nicht zu gebrauchen ist. Das Beispiel des Rustock Bots [14] zeigt, dass eine Malware sich auf verschiedene Arten verschleiern kann, um ihre eigentliche Logik zu verbergen. Rustock nutzt einerseits verschachtelte Schleifen, um den Code zuerst zu entschleiern bevor er ausgeführt wird. Zusätzlich ist ein zweiter Teil mit XOR verschleiert und muss erst entschlüsselt werden, bevor Rustock ihn ausführt. Eine weitere beliebte Methode von Malwareautoren ist es, den eigentlichen Payload mit einem Packer zu verändern und dadurch zu verschleiern. Vor der Ausführung entpackt sich die Malware selbst. Durch die Verschleierung gelingt es einer Malware auch, verschiedene semantisch äquivalente Versionen zu verbreiten, die jedoch komplett unterschiedliche binäre Repräsentationen haben. So können viele signaturbasierte Virenscanner ausgehebelt werden.

Für einen Menschen ist es normalerweise nicht möglich, aus dem verschleierten Code nützliche Informationen herauszulesen. Deswegen bietet es sich an, mit Emulatoren wie [15], die Malware so weit laufen zu lassen, bis sie sich selbst entschlüsselt hat und mit dieser entschlüsselten Binärdarstellung des Payloads die statische Analyse von vorne zu beginnen.

Wenn nun die Malware in ihrer reinen Form vorliegt, lassen sich sehr schnell sehr viele Informationen herauslesen. Eine Suche nach Zeichenketten kann Aufschluss geben, welche Kommandos beispielsweise ein Bot unterstützt. Die Liste der importierten und exportierten Funktionen gibt Hinweise auf die Arbeitsweise der Malware, manchmal lässt sich sogar ein geheimer Schlüssel, der zu der Kommunikation mit einem C&C Server genutzt wird, extrahieren.

C. Vorteile der statischen Analyse

Durch die statische Analyse kann man eine vollständige Sicht auf die Malware erhalten. Selbst Codefragmente der Malware, die in Testläufen nicht ausgeführt werden, können entdeckt werden. Zusätzlich besteht keine Gefahr für andere Computer im Netzwerk.

D. Nachteile der statischen Analyse

Die Analyse ist sehr aufwendig und bei verschleierten Binärdateien in den meisten Fällen nicht automatisierbar. Der resultierende Assemblercode ist sehr schwer zu verstehen und eine Rückführung in einen möglichen Quellcode ist sehr zeitraubend wenn nicht unmöglich. Oftmals ist die Aussagekraft der statischen Analyse auch nicht groß genug, da die Malware zum Beispiel dynamisch neue Befehle von ihrem C&C Server erhält, oder Stromchiffren wie RC4 zur Verschlüsselung verwendet (bsp. Rustock, ZeuS), bei denen sich der Schlüssel erst zu Ausführungszeit extrahieren lässt.

IV. Dynamische Analyse

Bei der dynamischen Analyse wird die Malware in einer Testumgebung ausgeführt. Dabei wird der Schadcode auf einem jungfräulichen Host-System installiert. Während und nach der Laufzeit des Wurms wird versucht, alle Aktionen, die die Malware durchführt, zu protokollieren. Das folgende Kapitel IV-A beschäftigt sich mit der Analyse der Handlungen der Malware auf dem Host, Kapitel IV-B mit der Analyse der Tätigkeiten im Netzwerk. Auf diese Weise können Rückschlüsse auf die Arbeitsweise der Malware getroffen werden. Außerdem kann

[4] Verschleierung - von dem Englischen ’obfuscation’

ein Profil der Malware erstellt werden, wodurch infizierte Computer schnell erkannt werden, Firewallregeln aufgestellt werden und Statistiken und Gefahrenanalysen erstellt werden können. Das Kapitel V geht genauer auf die Verwendung der Ergebnisse ein.

A. Analyse der Aktionen auf dem Host

Eine sehr einfache Möglichkeit um Rückschlüsse der Aktivitäten der Malware auf dem Host zu erhalten, ist es, die Malware auf einem frisch installierten System auszuführen und die Änderungen am System vor und nach der Ausführung zu vergleichen. So lässt sich leicht herausfinden, wie weit die Malware sich ins System integriert und ob sie sogar ein Rootkit installiert. Außerdem kann die Malware keine Auffälligkeiten – wie Analysesoftware – im laufenden System finden und verhält sich somit garantiert wie auf einem normalen System. Diese Analyse ist jedoch sehr aufwendig und liefert keine Aussagen darüber, was die Malware während der Laufzeit macht.

Eine weitaus bessere Möglichkeit stellt spezielle Analysesoftware wie CWSandbox dar [16].

CWSandbox führt die verdächtige Datei in einer kontrollierten Win32 Umgebung aus. Es werden alle System Calls, Änderungen am Dateisystem und der Registry, gestarteten Prozesse, sowie Netzwerkaktivitäten aufgezeichnet. Dabei kann das Ergebnis sowohl in menschenlesbarer als auch in maschinenlesbarer Form ausgegeben werden. Abbildung 2 zeigt einen Ausschnitt eines Analyseprotokolls im XML-Format. Zur Protokollierung der Aktionen der Malware nutzt CWSandbox die Techniken API Hooking[5] und DLL Injection. Das bedeutet, CWSandbox übernimmt die Kontrolle über alle Aufrufe der Windows-API, die die Malware während ihrer Laufzeit macht.

CWSandbox besteht im Wesentlichen aus zwei Teilen: cwsandbox.exe und cwmonitor.dll. Der erste Teil ist die eigentliche Analyseplatform. Er startet die zu untersuchende Malware im suspended mode, führt als Urlader alle nötigen API Hooks aus und bereitet die DLL Injection vor. Danach wird die Malware gestartet und cwmonitor.dll in den Adressraum der Malware injiziert. cwmonitor.dll enthält alle nötigen Hook Funktionen um alle Windows API Aufrufe der Malware zu protokollieren und an cwsandbox.exe zu senden. Zusätzlich kann sich cwmonitor.dll in jeden neu erzeugten Prozess und Thread der Malware selbstständig injizieren.

Abbildung 3 zeigt beispielhaft einen gehookten Aufruf einer Windows API-Funktion der Malware. Beim Start der Malware – und jedem neuen Prozess, den die Malware startet – werden alle Aufrufe der Windows API im Speicher der Malware gehookt: Die ersten sechs Bytes jeder API Funktion werden mit einen Sprung auf den jeweiligen Hook Code überschrieben (1). Hier können nun alle API Aufrufe und Parameter protokolliert werden und das Ergebnis direkt an die Analyseplatform cwsandbox.exe geliefert werden (2). Danach werden die überschriebenen Anweisungen der Windows

[5]Unter API Hooking versteht man, dass bei jedem Aufruf einer API-Funktion – anstatt der originalen Funktion – eigener Code ausgeführt wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This analysis was created by CWSandbox (c) CWSE GmbH / Sunbelt Software-->
<analysis cwsversion="2.1.17" time="06.10.2009 16:20:54" file="c:\7z465.exe" md5="
        fcd1b1472302fc7283147f4df471f402" sha1="
        c36012e960fa3932cd23f30ac5b0fe722740243a" logpath="c:\cwsandbox\log\7z465.exe
        \run_1\" analysisid="885548" sampleid="762367">
<calltree>
<process_call index="1" pid="2656" filename="c:\7z465.exe" starttime="00:01.422"
        startreason="AnalysisTarget"/>
<process_call index="2" pid="984" filename="C:\WINDOWS\system32\svchost.exe"
        starttime="00:03.328" startreason="DCOMService"/>
</calltree>
<processes>
<process index="1" pid="2656" filename="c:\7z465.exe" filesize="939956" md5="
        fcd1b1472302fc7283147f4df471f402" sha1="
        c36012e960fa3932cd23f30ac5b0fe722740243a" username="Administrator"
        parentindex="0" starttime="00:01.422" terminationtime="00:13.078" startreason
        ="AnalysisTarget" terminationreason="NormalTermination" executionstatus="OK"
        applicationtype="Win32Application">
<com_section>
<com_create_instance inprocserver32="%SystemRoot%\system32\browseui.dll" clsid="{00
        BB2765-6A77-11D0-A535-00C04FD7D062}" interfaceid="{00000000-0000-0000-C000
        -000000000046}"/>
<com_create_instance inprocserver32="%SystemRoot%\system32\browseui.dll" clsid="{03
        C036F1-A186-11D0-824A-00AA005B4383}" interfaceid="{00000000-0000-0000-C000
        -000000000046}"/>
<com_create_instance inprocserver32="%SystemRoot%\system32\browseui.dll" clsid="{00
        BB2763-6A77-11D0-A535-00C04FD7D062}" interfaceid="{EAC04BC0-3791-11D2-BB95
        -0060977B464C}"/>
<com_create_instance inprocserver32="shell32.dll" progid="lnkfile" clsid="
        {00021401-0000-0000-C000-000000000046}" interfaceid="{000214EE-0000-0000-C000
        -000000000046}" quantity="2"/>
</com_section>
<dll_handling_section>
<load_image filename="c:\7z465.exe" successful="1" address="$400000" end_address="
        $434000" size="212992"/>
<load_dll filename="C:\WINDOWS\system32\ntdll.dll" successful="1" address="$7
        C910000" end_address="$7C9C9000" size="757760"/>
<load_dll filename="C:\WINDOWS\system32\kernel32.dll" successful="1" address="$7
        C800000" end_address="$7C908000" size="1081344"/>
<load_dll filename="C:\WINDOWS\system32\USER32.dll" successful="1" address="$7
        E360000" end_address="$7E3F1000" size="593920"/>
<load_dll filename="C:\WINDOWS\system32\GDI32.dll" successful="1" address="$77
        EF0000" end_address="$77F39000" size="299008"/>
<load_dll filename="C:\WINDOWS\system32\SHELL32.dll" successful="1" address="$7
        E670000" end_address="$7EE91000" size="8523776" quantity="4"/>
<load_dll filename="C:\WINDOWS\system32\ADVAPI32.dll" successful="1" address="$77
        DA0000" end_address="$77E4A000" size="696320"/>
<load_dll filename="C:\WINDOWS\system32\RPCRT4.dll" successful="1" address="$77
        E50000" end_address="$77EE2000" size="598016"/>
<load_dll filename="C:\WINDOWS\system32\Secur32.dll" successful="1" address="$77
        FC0000" end_address="$77FD1000" size="69632"/>
<load_dll filename="C:\WINDOWS\system32\msvcrt.dll" successful="1" address="$77
        BE0000" end_address="$77C38000" size="360448"/>
<load_dll filename="C:\WINDOWS\system32\SHLWAPI.dll" successful="1" address="$77
        F40000" end_address="$77FB6000" size="483328"/>
<load_dll filename="C:\WINDOWS\WinSxS\X86_Microsoft.Windows.Common-
        Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\" successful="1"
        address="$773A0000" end_address="$774A3000" size="1060864" quantity="3"/>
<load_dll filename="C:\WINDOWS\system32\ole32.dll" successful="1" address="$774
        B0000" end_address="$775ED000" size="1298432" quantity="3"/>
<load_dll filename="C:\WINDOWS\system32\VERSION.dll" successful="1" address="$77
        BD0000" end_address="$77BD8000" size="32768"/>
<load_dll filename="C:\WINDOWS\system32\IMM32.DLL" successful="1" address="
        $76330000" end_address="$7634D000" size="118784" quantity="2"/>
<load_dll filename="C:\WINDOWS\system32\oleaut32.dll" successful="1" address="$770
        F0000" end_address="$7717B000" size="569344"/>
<load_dll filename="C:\WINDOWS\system32\pstorec.dll" successful="1" address="$5
        E490000" end_address="$5E49D000" size="53248"/>
<load_dll filename="C:\WINDOWS\system32\ATL.DLL" successful="1" address="$76AD0000"
        end_address="$76AE1000" size="69632"/>
<load_dll filename="C:\WINDOWS\system32\uxtheme.dll" successful="1" address="$5
        B0F0000" end_address="$5B128000" size="229376" quantity="3"/>
<load_dll filename="C:\WINDOWS\system32\RichEd20.dll" successful="1" address="$74
        DB0000" end_address="$74E1D000" size="446464"/>
<load_dll filename="C:\WINDOWS\system32\msctfime.ime" successful="1" address="$11
        F0001" end_address="$11F0001" size="0" quantity="2"/>
<load_dll filename="C:\WINDOWS\system32\msctfime.ime" successful="1" address="
        $75250000" end_address="$7527E000" size="188416" quantity="2"/>
<load_dll filename="C:\WINDOWS\system32\browseui.dll" successful="1" address="$75
        F20000" end_address="$7601D000" size="1036288" quantity="3"/>
<load_dll filename="C:\WINDOWS\system32\ntshrui.dll" successful="1" address="
        $76940000" end_address="$76966000" size="155648"/>
<load_dll filename="C:\WINDOWS\system32\NETAPI32.dll" successful="1" address="$597
        D0000" end_address="$59825000" size="348160"/>
<load_dll filename="C:\DOKUME~1\ADMINI~1\LOKALE~1\Temp\nss5.tmp\InstallOptions.dll"
        successful="1" address="$10000000" end_address="$10009000" size="36864"/>
<load_dll filename="C:\WINDOWS\system32\MSCTF.dll" successful="1" address="$746
        A0000" end_address="$746EC000" size="311296"/>
</dll_handling_section>
```

Abbildung 2. Ausschnitt eines Reports der CWSandbox im XML-Format

API ausgeführt und es wird auf die originale API Funktion zurückgesprungen (3). Der Hook Code befindet sich in der cwmonitor.dll, welche durch DLL Injection in den Speicher der Malware geladen wurde.

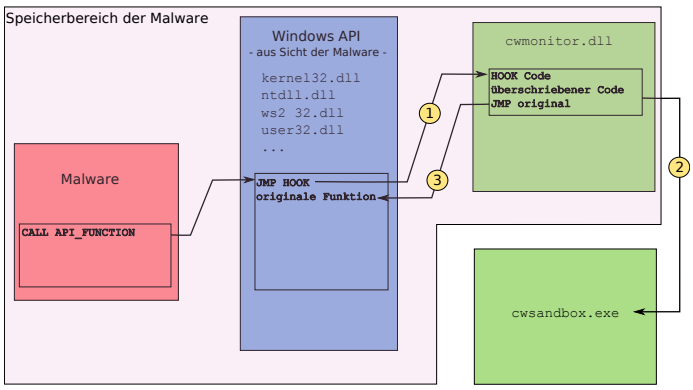


Abbildung 3. CWSandbox Funktionsweise

Die aktuelle Version von CWSandbox kann jedoch keine direkten Zugriffe auf den Kernel erkennen. CWSandbox braucht auch einen menschlichen Operator, um gegebenenfalls das Analysefensters manuell zu setzten, falls die Malware erst nach einer gewissen Zeit aktiv wird oder Techniken wie TimeLock Puzzles [6] einsetzt. Auch für die finale Auswertung der Ergebnisse ist oft ein Mensch von Nöten.

Die Verwendung von virtuellen Maschinen erleichtert die Arbeit des Wiederherstellens des ursprünglichen Systemzustands. Jedoch existiert Malware, die das laufende System untersucht und gegebenenfalls ihre Aktivitäten einstellt oder sich auf einem Analysesystem anders verhält. Dieses Verhalten ist beispielsweise bekannt, falls die Malware einen laufenden Debugger erkennt, oder feststellt, dass sie in einer virtuellen Maschine ausgeführt wird. Zu diesem Zweck muss getestet werden, ob die Malware eine vm-Erkennung durchführt. Dies kann zum Beispiel durch den Vergleich der „network fingerprints“ der Malware (siehe IV-B.2) auf einem realen und einem virtuellen System erfolgen.

B. Analyse der Aktionen im Netz

Für diese Analyse müssen alle Nachrichten, die die Malware über das Netzwerk empfängt und sendet, protokolliert werden. Dafür genügt es, einen Paketsniffer in einer Man-in-the-Middle Position zwischen dem Analysesystem und dem Netzwerk zu installieren.

1) Analyse der ausgehenden Verbindungen: Über die ausgehenden Verbindungen eines Bots erhält man direkte Informationen über die Aktionen des Bots. Oft wird anhand der verwendeten Protokolle und des Traffic-Aufkommens bereits ersichtlich, ob ein Bot an einer DDos-Attacke teilnimmt oder Spam versendet. Andere Aktivitäten erfordern oft eine genauere Analyse des Traffics. Ein Bot verbindet sich jedoch unweigerlich mit seinem C&C-Server um Instruktionen zu erhalten. Wenn diese Kommunikation mitgeschnitten wurde, kennt man einen C&C-Server zu dem jeweiligen Bot-Netz. Durch gezielte Firewallregeln ist es auch möglich, den Bot an der Kommunikation mit seinem C&C Server zu hindern, um so herauszufinden, nach welchem Algorithmus der Bot versucht, einen neuen Server zu finden. Umgekehrt kann man so einem Bot, der inaktiv wurde, weil er keinen C&C Server finden konnte, die Existenz eines solchen Servers vorspielen um ihn in Aktion zu erleben. Jedoch muss dazu das verwendete C&C-Protokoll bekannt sein.

2) Klassifizierung der Malware: Viele Würmer und Bots existieren in vielen Ausführungen und können deshalb meistens nicht automatisch von Virenscannern als Duplikate erkannt werden, da ein Virenscanner nur versucht herauszufinden *ob* es sich bei einer Datei um eine Malware handelt, nicht *um Welche*. Die Universität von Washington nutzt deshalb in ihren Experimenten mit Botlab [4] einen sogenannten „network fingerprint“. Dieser wird wie folgt gebildet: Sei X die Menge aller ausgehenden Verbindungen eines Programms, wobei jede Verbindung durch das Tupel

`<Protokoll, IP Adresse, DNS Adresse, Port>`

dargestellt wird. Damit zufällige Verbindungen ausgeschlossen werden, wird die Datei mehrfach ausgeführt und der network fingerprint aus der Schnittmenge über alle X gebildet. Experimentell konnte für spamorientierte Bot-Netze festgestellt werden, dass zwei Bots mit ihren network fingerprints N_1 und N_2 zu einer sehr großen Wahrscheinlichkeit identisch sind, falls gilt:

$$\frac{N_1 \bigcap N_2}{N_1 \bigcup N_2} > 0.5$$

3) Analyse von Spam: Ein großer Teil aller weltweit versandten Emails besteht aus Spamnachrichten, welche von Bot-Netzen erzeugt wurden. Um diese zu analysieren, bietet es sich an, den Spam, der auf den eigenen Mailservern eingeht, sowie der Spam, der durch die lokal untersuchte Malware versendet wird, in Beziehung zu setzen. So lässt sich Spam sehr akkurat einem Bot-Netz zuordnen und dadurch lassen sich Rückschlüsse auf seine Größe ziehen. Es sei jedoch vermerkt, dass es keine Seltenheit ist, dass mehrere Bot-Netze an der gleichen Spamkampagne teilnehmen [4]. Dies ist ein Indiz für die starke Kommerzialisierung der Internetkriminalität.

4) Verhinderung von Blacklisting: Um zu verhindern, dass die eigene IP Adresse über kurz oder lang von den Bot-Netz-Operatoren auf die schwarze Liste gesetzt wird, ist es ratsam, diese nicht preiszugeben. Dies kann erreicht werden, indem man jeden Traffic, der ins Internet gelassen wird, über ein Anonymisierungsnetzwerk wie TOR [17] leitet.

5) Gefahren der dynamischen Analyse: Da bei der dynamischen Analyse die Schadsoftware tatsächlich ausgeführt wird, besteht eine Gefahr für alle an das das Netzwerk angeschlossenen Computer. Es gilt deshalb das Risiko für angrenzende Systeme zu minimieren und dabei trotzdem gute Analyseergebnisse zu erzielen. Eine Möglichkeit besteht darin, den Rechner komplett vom Netz zu trennen, worunter die Analyseergebnisse jedoch stark leiden. Viele Malware testet ihre Konnektivität bevor sie aktiv wird, die meisten Bots stellen sogar alle Aktivitäten ein, falls sie sich nicht zu ihrem C&C Server verbinden können.

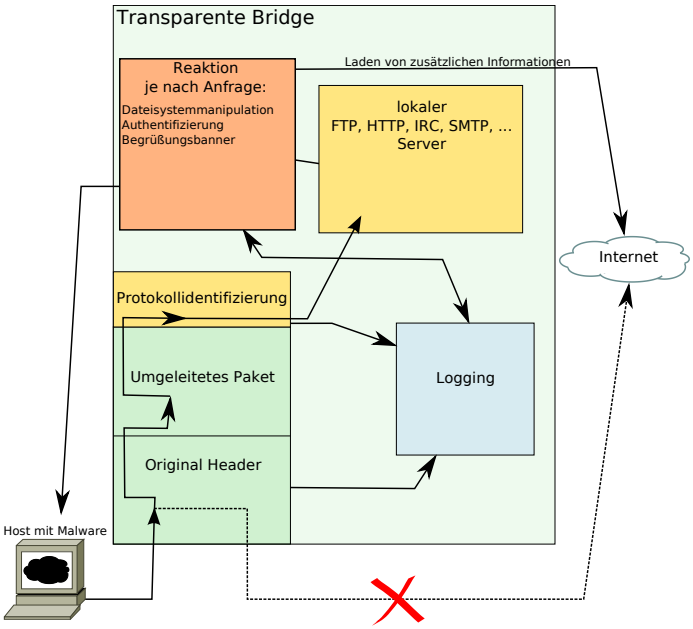


Abbildung 4. TrumanBox Schema

Einen erfolgreichen Ansatz um der Malware so viel Freiraum wie nötig und so wenig Zugang zum Netzwerk wie möglich zuzugestehen, bieten Systeme wie Honeywalls oder die TrumanBox [18]. Die TrumanBox wird wie eine transparente Bridge – und damit für die Malware unsichtbar – zwischen dem Host und dem restlichen Netzwerk eingerichtet. Sie kann Verbindungsversuche der Malware auf generische, lokale Dienste transparent umleiten und spielt der Malware so eine echte Internetumgebung vor, ohne einen fremden Rechner zu gefährden. Sie arbeitet auf allen ISO/OSI Schichten um so die Analyseergebnisse zu optimieren. Ein solches System muss in der Lage sein, eine Dienstanfrage auch zu erkennen, wenn sie auf einem nicht standardisierten Port gestellt wird. Außerdem muss der emulierte Dienst bei Protokollen, bei denen der Server dem Client zuerst eine Nachricht schickt, die richtige Nachricht ausliefern (beispielsweise die Begrüßungsbanner bei FTP oder SMTP). Des Weiteren sollte das System sich so verhalten, wie es die Malware erwartet, dies beinhaltet unter anderem die Dateisystemstruktur auf dem Server bei Diensten wie FTP und HTTP. Abbildung 4 zeigt diesen schematischen Aufbau. Alle Anfragen der Malware an das angrenzende Netzwerk (hier: Internet) werden transparent auf lokale Dienste umgeleitet. Es werden alle wichtigen Daten protokolliert, sodass der lokale Dienst den Erwartungen der Malware entsprechend reagieren kann. Falls nötig, können sogar Informationen von dem real angefragten Server im Internet nachgeladen werden.

Da sich das hier beschriebene System in einer Man-in-the-Middle Position befindet, können alle relevanten Informationen mitgeloggt werden.

C. Vorteile der dynamische Analyse

Nur durch die dynamische Analyse ist es möglich, in relativ kurzer Zeit, automatisiert und mit relativ geringem Aufwand, sehr viel Malware sehr effizient zu analysieren.

D. Nachteile der dynamische Analyse

Die gewonnenen Ergebnisse müssen nicht vollständig sein, wie Kapitel III-C bereits gezeigt hat. Der bekannte Wurm Conficker aus den Jahren 2008/09 enthielt beispielsweise ein Feature, welches bis zum 1. April 2009 im Wurm deaktiviert war und so durch die dynamische Analyse nicht vor diesem Datum erkannt werden konnte. Noch schwieriger gestaltet sich die Analyse, wenn ein Bot verschlüsselt kommuniziert. Aus dem Mitschnitt der versendeten Nachrichten wird nicht klar, welche Informationen der Bot gesendet und empfangen hat. Da Malware sehr individuell arbeitet, ist ein automatisches System zum Schutz der angrenzenden Systeme allein nicht ausreichend. Teilweise ist eine manuelle Filterung der ausgehenden Verbindungen nötig, um bessere Analyseergebnisse zu erreichen oder den ganzen Funktionsumfang der Malware auszulösen.

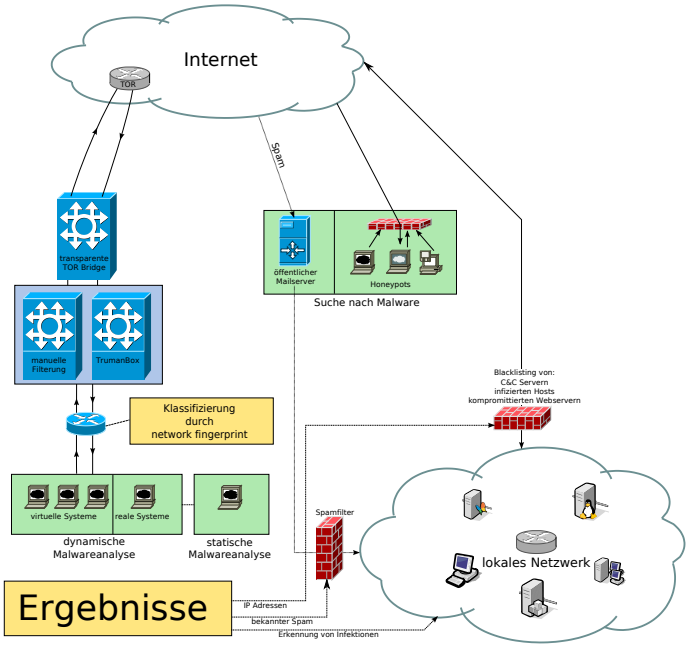


Abbildung 5. Darstellung einer kompletten Analyseumgebung.

V. Verwendung der Ergebnisse

Die gewonnenen Ergebnisse können vielseitig genutzt werden. Einerseits erhält man wichtige Erkenntnisse über die Arbeits- und Funktionsweise von Würmern und Bots und über die Vorgehensweise von Bot-Netz-Operatoren. Andererseits erhält man aktuelle Informationen über neuartige Bedrohungen. Aus der Analyse lässt sich sehr oft der C&C Server eines Bots bestimmen, oft lassen sich auch kompromittierte Webserver aufdecken, die weitere Infektionen verbreiten. Teilweise können sogar infizierte Clients durch bestimmte

Charakteristika entdeckt werden. Mit diesen Informationen lässt sich in der Firewall des lokalen Netzes (beispielsweise Firmennetz/Campusnetz) eine Blacklist mit allen bedrohlichen IP Adressen einrichten. Die dynamische Analyse eines Bots zeigt bereits, ob er allein durch Blacklisting seiner C&C Server deaktiviert werden kann. Des Weiteren lässt sich durch die Untersuchung von Spam die Trefferquote des lokalen Spamfilters verbessern. Außerdem ist es sogar möglich, infizierte Hosts im lokalen Netzwerk zu erkennen. Aufgrund des Wissens über die Infektion, durch die Analyse der Aktionen auf dem Host, lässt sich der betroffene Rechner schnell bereinigen.

Abbildung 5 zeigt ein Beispiel einer kompletten Analyseumgebung, wie sie in dieser Arbeit theoretisch beschrieben wurde. Diese Analyseumgebung wurde nach dem Beispiel von Botlab [4] entworfen. Durch verschiedene Arten von Honeypots wird Malware gesammelt. Auch eingehender Spam wird auf Links mit Malware untersucht. Die Analyse erfolgt bevorzugt auf virtuellen Systemen, da diese leichter Wiederherzustellen sind, im Einzelfall werden jedoch auch echte Systeme benutzt. Die dynamische Analyse wird bevorzugt, jedoch ist manchmal eine statische Analyse nötig. Ein Schaden an anderen an das Internet angeschlossenen Systemen, wird automatisch durch die TrumanBox verhindert, allerdings ist auch eine manuelle Filterung – falls nötig – vorgesehen. Der Traffic der Malware wird durch TOR geleitet, um ein Blacklisting der eigenen IP auf Seiten der Bot-Netz-Operatoren zu verhindern. Mit den Ergebnissen werden die Firewalls und Spamfilter eines lokalen, zu schützenden Netzwerks verbessert.

VI. Zusammenfassung und Ausblick

Wir haben die grundlegenden Funktionen und Vorgehensweisen von Würmern und Bots kennengelernt. Mit diesem Wissen haben wir uns einen Überblick verschafft, wie man unbekannte Bot- und Wurm-Dateien analysieren kann. Die Möglichkeiten, aber auch die Grenzen von automatischer Analysesoftware wurden vorgestellt. Die Qualität und Benutzerfreundlichkeit dieser Software kann sich in Zukunft noch verbessern, genauso wie die Bedrohung durch Würmer und Bots nicht geringer werden wird. Der Bedarf nach einem menschlichen Operator wird jedoch stets vorhanden sein.

Literatur

[1] S. McIntyre, "Teh Internetz are pwned - How I learned to stop worrying and love teh Internetz," in *Hacking at Random*, 2009. [Online]. Available: https://har2009.org/program/events/171.en.html

[2] Symantec, "Virendefinitionen und Sicherheits-Updates," September 2009. [Online]. Available: http://www.symantec.com/de/de/norton/security_response/definitions.jsp

[3] A. Kohring and C. Schafmeister, "Viren und Würmer: Mit welchen Mitteln gelangen Viren und Würmer in den Computer, und wie kann man sich gegen sie schützen?" 2009.

[4] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy, "Studying Spamming Botnets Using Botlab," *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*, 2009.

[5] P. Bacher, T. Holz, M. Kotter, and G. Wicherski, "Know Your Enemy: Tracking Botnets," *The Honeynet Project*, 2005. [Online]. Available: http://www.honeynet.org/papers/bots

[6] Nomenumbra, "Countering behavior-based malware analysis through TimeLock Puzzles," in *Hacking at Random*, 2009. [Online]. Available: https://har2009.org/program/events/57.en.html

[7] P. Kleissner, "Stoned Bootkit - The Rise of MBR Rootkits & Bootkits in the Wild," in *Hacking at Random*, 2009. [Online]. Available: https://har2009.org/program/events/86.en.html

[8] D. Bachfeld, "Mit guten Karten - Sicher bezahlen im Internet," *c't*, vol. 19, pp. 92–95, 2009.

[9] C. Kurtenbach, "The ZeuS evolution - A malware case study.." in *Hacking at Random*, 2009. [Online]. Available: https://har2009.org/program/events/40.en.html

[10] BBC, "Click - Gaining access to a hacker's world," TV, Intenet, 03 2009. [Online]. Available: http://news.bbc.co.uk/2/hi/programmes/click_online/7938201.stm

[11] D. Dittrich and S. Dietrich, "command and control structures in malware," *USENIX ;login:*, vol. 32, 2007.

[12] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static Disassembly of Obfuscated Binaries," 2004.

[13] "IDA Pro," 08 2009. [Online]. Available: http://www.hex-rays.com/idapro/

[14] K. Chiang and L. Lloyd, "A Case Study of the Rustock Rootkit and Spam Bot," in *The First Workshop in Understanding Botnets*, 2007.

[15] C. Eagle, "The x86 Emulator plugin for IDAPro," 2008. [Online]. Available: http://ida-x86emu.sourceforge.net

[16] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[17] [Online]. Available: http://www.torproject.org

[18] C. Gorecki, "TrumanBox-Transparente Emulation von Internetdiensten."

Attacks and exploits targeting BitTorrent and other P2P file sharing networks

Andreas Hegenberg
Betreuer: Benedikt Elser
Seminar Future Internet WS 09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: andreas.hegenberg@in.tum.de

***Abstract*— This proceeding deals with various attacks and exploitations P2P networks have to cope with. In the last few years BitTorrent became famous for spreading big amounts of data to many people in a short time. BitTorrent will therefore be the main focus of this work. This proceeding serves basically as overview about different types of attacks. It shows attacks that are able to prevent files from being downloaded or at least delay the downloading process enormously. For some of the attacks countermeasures are shown. Furthermore the proceeding discusses so-called free riding which allows a peer to download without reciprocating. Understanding how the different attacks and exploits work is a important step to design more resilient protocols in the future.**

***Keywords*— P2P, attacks, BitTorrent,free riding**

I. Introduction

The amount of fast Internet connections increases steadily and so does the need for cheap but yet fast content distribution. Peer to Peer (P2P) file sharing systems are very popular for quickly spreading high amounts of data to many people without much effort or cost. Unfortunately there are several groups which are interested in attacking or exploiting those networks for different reasons. For example the music, film and television industries already (successfully) attacked various P2P networks because they hoped to stop the illegal distribution of their assets. But also regular users can cause problems if they try to download without reciprocating in any form to the community, this is called free riding and could cause the entire network to slow down extremely.
This proceeding will deal with the question how resilient today's overlay P2P systems are against a variety of possible attacks and/or exploitations. Therefore I will describe some potential types of attacks and their impact to harm P2P systems. Because BitTorrent has evolved to the currently most used P2P network and because it is available as open source this work will concentrate on it.
The first chapter will give you a very basic introduction to the BitTorrent protocol. The second chapter will be about different types of attacks that prevent files from being downloaded. The third chapter will then deal with the incentives system in BitTorrent and how to trick it. Finally followed by a short discussion about future work and other P2P file sharing protocols.

II. BitTorrent

BitTorrent is an open source peer to peer file-sharing protocol which became very popular during the last few years. Today BitTorrent is responsible for a huge amount of traffic on the internet. The 'Internet Study 2008/2009' released by the german company Ipoque at the beginning of the year estimated the traffic produced by BitTorrent users at 27-55% of all internet traffic, depending on the geographical location. Meanwhile BitTorrent serves as blueprint for many applications, e.g. for streaming software. Even the media industries checks if BitTorrent could be used for the commercial distribution of movies, music, tv-shows etc.. [1]

A. BitTorrent Characterization

In BitTorrent a file is divided into many pieces (usually each pieces size is 256KB) those pieces are subdivided into blocks (usually 16KB). The information how a file is split gets saved in a metafile, which also holds SHA1 hashes for every piece (but not for the blocks) and the URL of a tracker server. The tracker is a centralized entity which is responsible for managing the peers that belong to a specific file and for assisting their communication. Therefore the tracker maintains a list of currently active peers and delivers a random subset of these to clients, upon request. (see figure 4) [2]. Usually trackers are also the platforms where users can download the metafiles via standard HTTP. As you see BitTorrent in its original specification is not pure P2P but has some client-server attributes too.
Peers (clients) that want to download a single file or already got that file are grouped into swarms - as long as their client is running. A swarm usually consists of seeders and leechers whereby a seeder is an user who already possesses all the pieces of a file.
A leecher uploads blocks he already got to other leechers that request those blocks from him. The other leechers are able to determine which pieces he has based on bit-field messages they exchange [3]. Peers chose the pieces they request using a local rarest first policy [4], this means they chose those pieces which are least replicated among the peers they are connected to. This shall ensure a balanced piece availability.

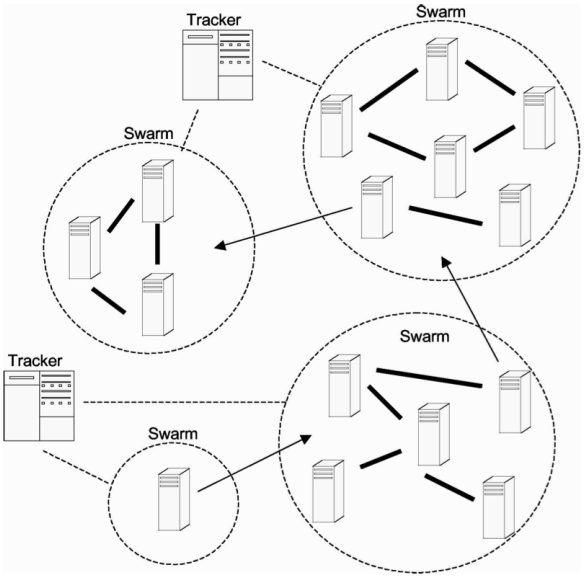


Fig. 1. Portion of a BitTorrent system, arrows show how users could jump from one swarm to another or even be in both swarms at the same time. [5]

The decision which other leechers will get one of his n upload slots (usually there are 4-5 slots) is made using an incentive mechanism. [4]. The standard mechanism works periodically, one period typically takes 10 seconds. In a period the client sorts the list of peers he has currently downloaded from by the speed they offered to him. In the next period he unchokes a specific number (n-1) of peers that achieved highest rates and chokes the other peers. Unchoking means, that the selected peer gets one of the n upload slots and choking means he will lose his upload slot (but stay connected) see figure 2.

This behaviour is called Óbandwidth firstÓ. Additionally he selects another random peer for an optimistic unchoke every third period. [6] This makes finding new, maybe better peers possible and also allows new clients to gather some data so they can become uploaders fast. Through this tit-for-tat system a fair trading shall be ensured. Later in this work we will see if it works well.

If a leecher has all parts of a file he becomes a seeder. Seeders can not consider any download speed for unchoking (because they download anymore) thus the upload speed is watched. The peers to which a seed can upload fastest are then chosen in a round robin fashion. [3]

Because BitTorrent is open source, the protocol is implemented in many slightly different ways from various clients. A closer look will show that some of those implementations are considerably more prone to specific attacks than others.

If you need a more specific description of the BitTorrent protocol take a closer look on the Bittorrent protocol specification. If needed we will describe some specific parts of the protocol in the course of this work.

III. Attacks on BitTorrent

The authors of [7] [8] [1] [4], describe various attacks on BitTorrent swarms or parts of a swarm which could make it impossible to download a file or slow down the whole process enormously. This chapter addresses various attacks of this type.

A. Attacks against Seeders

Targeting seeds is a very natural approach because without a seed it clearly becomes hard to complete the download for any leecher. However the attack has to take place in an early stage of distribution, if all the parts are already spread to leechers it is often too late. [7] [8] In this section we will discuss different seed attacks. In almost every seed attack identifying the initial seeders as fast as possible is a necessary precondition to render the attack successful. Unfortunately this is not too hard as [7] shows. Because most big torrent web sites give their files an incrementing index number new files can easily be watched. Also some trackers offer public RSS feeds of all new files.

1) Early stage DoS attacks on seeders : Early stage Denial of Service attacks which aim on cutting off the initial seeder from the network seem to be an effective way to take down a whole swarm. The results of [8] confirm this. Their measurements show that if a seeder is cut off from the network due to some sort of DoS attack while the average download progress is below 20% almost always the leechers can not complete their download . They also show that it is hard to predict what happens if the average download ratio is between 20% and 60%. But it is obvious that the probability of completing the download increases with the average download ratio. Above 60% the download could always be completed for every leecher.

In [8] is not described how the DoS attack is executed. The simplest approach would be to launch massive requests at the seeder. However flooding DoS attacks need quite a lot resources especially if they are directed to many popular torrent files or if the seeder is a host with a fast internet connection. So instead of running this simple kind of DoS attack it could be more efficient to exploit specific characteristics of the BitTorrent protocol [1] Unfortunately it is hard to circumvent those attacks. Newer BitTorrent clients support a feature called super seeding [9] which intends to spread a file faster. As a sideline this feature hides that the client is a seeder. Maybe this is a good start for hampering seeder attacks, but there definitely is much more room for improvements and further investigations.

2) Bandwidth attacks on seeders: The authors of [7] describes two attacks against seeders, the first we will discuss is called bandwidth attack. The approach is quite simple, the attacker attempts to consume the majority of the seeds upload bandwidth [7].

For understanding how seeders assign their upload slots to leechers the authors examine two actual client implementations of seeding algorithms, the one of Azureus version 3.1.1.0 (meanwhile called Vuze) and the one of BitTornado (unknown version).
According to [7] Azureus uses a variation of the original bandwidth first seeding algorithm. This algorithm prefers users who have a high download speed and at the same time low amount of data downloaded from the seed. The other client, BitTornado uses a pure bandwidth first algorithm when seeding [7].
Both implementations perform optimistic unchokes.
Two cases of bandwidth allocation between seeders, leechers and attackers were considered during the experiments. In the first case the seeders upload bandwidth per upload slot is lower than the leechers download bandwidth and this is lower than the attackers download bandwidth. This means both, the leechers and the attackers can only download at the same speed (limited by the seeder) and thus the attackers have no real advantage by their higher bandwidth.
The second case where seeders achieve higher upload speed per slot than leechers can download but lower than attackers can download seems to be more harmful if a bandwidth first algorithm is used.
As the measurements in [7] show Azureus' implementation ensures a high resilience against bandwidth attacks in every case because even if the attackers download very fast they will soon have a high amount of data downloaded from the seed and thus get choked. During their experiments the delay ratio was always less than 3 compared to the same download without attackers.
Surprisingly BitTornado performed only slightly worse. This is because in the second bandwidth allocation case the upload bandwidth of the seeder is higher. Thus even if the attackers obtain all regular upload slots, the optimistic unchoke from time to time makes sure a friendly peer downloads at high rate.
Unfortunately you will see that the seeding algorithm used by Azureus has a major disadvantage if you read on.

3) Connection/Eclipse attacks on seeders: Every BitTorrent client has a maximal number of connection slots that can be filled by other peers (usually about 50). The so-called connection attack, also known as eclipse attack (the second attack described in [7]) aims on filling the vast majority of those slots with malicious peers, so that no friendly peers can connect anymore. This shall be done using as low bandwidth as possible.
As the investigations in [7] show Azureus is higly vulnerable to this attack, in the authors experiments. BitTornado performs better but also is affected. Their attack environment consists of a single seed sharing a file, 30 leechers, a single tracker and a number of attack peers [7]. They try to simulate a flash crowd effect by starting 5 leechers first, then after those are connected to the seed all the attackers are launched, and finally the rest of the leechers.

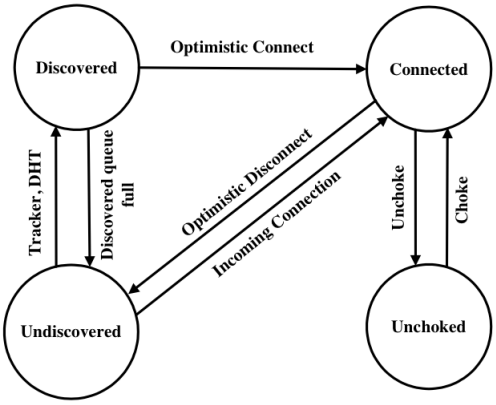


Fig. 2. State Diagram for Connection Management in Azureus. [7]

For understanding what happens we now have to take a look at the connection algorithms.
Azureus accepts received connection requests immediately unless a specific limit is reached (about 50). As shown in figure 2 the new connected peer will change its state from undiscovered to connected. Connected peers occupy a connection slot but do not receive any data until they are unchoked. Furthermore Azureus maintains a queue called 'Discovered Peers' which only contains peers the client discovered by itself using various methods. [7] If the connection list has free slots the oldest peers in the discovered list get included into it. This is called 'Optimistic Connect'. When seeding Azureus keeps track of the last time data was sent to a peer. In order to make room for new peers, every 30 seconds Azureus checks if one of the peers has not received data for more than 5 minutes. If this is the case the peer is disconnected and enters the undiscovered state.
The connection attack on Azureus causes all regular downloads to fail. The reason for this is a combination of the seeding algorithm and the connection algorithm used. Remember: a seeding peer prefers leechers who download at high speed but also have downloaded low amount of data. Because the attacking clients do not download data at all, the second criterion allows them to grab some of the seeders upload slots. As you know the connection algorithm disconnects leechers that have not got data for more than 5 minutes. Because the attackers outnumber the leechers, all the leecher disconnect one after another. Even if attackers get disconnected their chance of getting connected again is high because in contrast to friendly leechers they will steadily try to reconnect.
BitTornado is more robust because it uses a pure bandwidth first algorithm. Therefore the attackers can not get upload slots and fail. Only if the seeds upload bandwidth per slot is very low (so that the download speed of leechers and attackers is limited by the seed) the attack is moderately successful but then it is more like a bandwidth attack and the only advantage the attackers have is their quantity .

B. Attacks On Leechers Or The Whole Swarm

This section will describe attacks that are not specifically targeted on seeders. Still they aim on stopping the distribution of a file completely. This section will start with the sybil attack which can be combined with some other attacks.

1) The Sybil Attack: Usually you would think a peer exists exactly once per swarm he maybe a seeder, a leecher or something else but he only exists once. The Sybil attack [10] [4] tries to take over a swarm by faking multiple identities. In BitTorrent this is possible because identities are generated autonomously by peers, so there usually is no central entity for controlling them although a tracker could do this. Faking an identity is not expensive thus the number of multiple identities could be very high, sybils could even represent the vast majority of a swarm. However just by creating multiple identities BitTorrent is not harmed too much,thus the following attacks described by the authors of [4] try to combine sybils with other techniques. In Chapter 4 of this work sybils will be used for achieving higher download rates.

2) Lying Piece Possession Attack: This attack described in [4] tries to spread false information about piece availability. The goal is to exploit the local rarest first policy used in BitTorrent. Therefore malicious peers announce they have pieces they do not really have. If there are many of those malicious peers all other peers will delay downloading the pieces because they think it is not very rare. The funny thing about this is that in contrast to the peers belief the pieces become increasingly rare and possibly even disappear completely. To make this attack efficient it should be combined with the sybil attack as described above. The effect of this approach is affected by various parameters e.g. the number and real dissemination of pieces lied about, the number of malicious peers and the file size. [4]. The impact evaluation discussed in [4] shows an increasing effectiveness of the attack the higher the number of malicious peers is. Although they note that they could not tell if this is caused by the piece lying or the peer eclipsing effect (as described below). As a matter of fact the lying piece possession attack can be very harmful and may be able causing BitTorrent swarms to fail.
The authors of [11] show a possible countermeasure with low overhead. They therefore introduce an algorithm called 'PeerRotation' which tries to determine malicious peers that seem to be uninterested in exchanging data and replace them with other peers.

3) Eclipse Attack: In A we already discussed the connection attack on seeders, in [4] a similar approach is used to attack a whole swarm using sybils which try to snatch as many connection slots of a peer as possible. As a result of this the friendly peer is eclipsed, he can not connect to other friendly peers anymore because the attackers can mediate most or all communication [4] and so he starves. Figure 3 shows this in a very simplified way. Remember that peers usually have a connection limit of about 50. Attacking peers in contrast can open as many connections as their resources allow them. This means, that with a relatively low number of malicious peers a nearly unlimited number of friendly peers can be eclipsed.

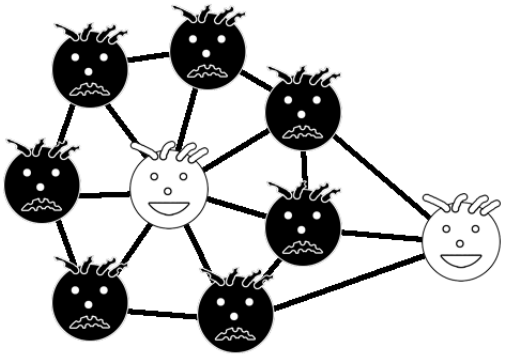


Fig. 3. Eclipse Attack: Friendly peers eclipsed by many malicious sybils

The measured results in [4] show that this attack is highly efficient.

4) Piece attack: The piece attack described in [1] is similar to the Lying Piece Possession attack discussed before. The goal of the piece attack is to slow down the download process. As you know a BitTorrent metafile contains SHA1 hashes for every piece. Those are used for integrity checking. So if a malicious peer would attempt to send fake data the integrity check would fail and the piece would be downloaded again from another peer. The problem in doing so is, that pieces in BitTorrent are split into blocks and blocks from one piece can be received from various peers. As you may remember single blocks can not be integrity checked because hashes are only available for the whole piece (everything else would make the metafiles way too big). So if a malicious peer sends fake blocks the integrity check will not fail until the peer gathered all blocks of that piece. A blacklist of malicious peers could solve this problem, unfortunately detecting who sent the fake block is not possible. P.Dhungel et al. [1] calculate the probability of downloading a clean piece. According to them if n is the number of neighbors that claim to have the piece and m denotes the number of attackers and k denotes the subset of n which is selected by the peer for downloading the blocks, the probability of downloading a clean file is approximately $(1 - \frac{m}{n})^k$. Thus for a successful attack the fraction $\frac{m}{n}$ has to be high. Naturally this value will be higher when peers try to download a rare file. They also show that in the endgame of a torrent the piece attack is most efficient because there only rare pieces are requested.
Countermeasures against this attack are introduced in [1] and [11]. Thereby in [11] a reputation system is suggested which raises a peers reputation if he contributed to correct pieces and lowers his reputation if he contributed to corrupted pieces. If a threshold is undershot the peer will be moved to quarantine. Their results show, that this approach is efficient because it does not introduce any overhead if there is no attack.
The countermeasures suggested in [1] are based on three heuristics. The first heuristic only measures the amount of bad pieces which an IP has contributed to. The second heuristic measures the ratio between good pieces and bad pieces and

the third heuristic measures the ratio of bad pieces a peer participated in and the total number of blocks he contributed. Attackers will try to minimize their bandwidth cost and thus may only send one corrupted block per piece to a peer. Therefore a high ratio in heuristic three could convict them. Measurements in [1] show, that heuristic three is best suited for practical use but it could be combined with heuristic two.

IV. Selfish Behavior In BitTorrent

So far we only discussed attacks with the target to harm the BitTorrent network and prevent files from being downloaded or slow down the downloading process. Now let us shift attention to seemingly contrary topics, namely free riding and selfish peers. As described before BitTorrent uses a tit-for-tat incentive mechanism to ensure fair sharing. Free riding tries to trick that mechanism, so downloading can be achieved without contributing. There are different reasons why one would aspire this. E.g. in some countries downloading copyrighted stuff is not chargeable but uploading is. Slow internet connections with low upload speed could be another reason. [12]
Unfortunately free riding can be very harmful for a P2P system because like the attacks in the last chapter it slows down the whole network. Also users are selfish or just rational, so they want to download at highest possible speed. This chapter is going to show you some attempts to achieve free riding or cheat to achieve higher download rates than usual. Different types of selfish behavior or free riding are described in [2] [12] [6] [5] [3] [13]. this work will refer to some of them. Peers that try to achieve free riding will be called selfish peers.

A. The Large View Exploit

Remember, that if a BitTorrent client connects to a tracker and requests peers the tracker delivers a random subset of peers known to him. Also a peer only holds connections to a maximum of about 50 other peers. Further recall the optimistic unchoking BitTorrent implements and the round robin uploading seeders do.
The large view exploit as described in [12] and used in the BitThief implementation of Locher et al. [13] mainly tries to exploit these points. Therefore the authors of [13] make modifications to an existing BitTorrent client. E.g. their client never uploads, it mimics the behavior of a new client and it requests new peer lists from the tracker every 15 seconds. Also it has an unlimited number of connection slots and thus connects to as many peers as possible. Figure 4 demonstrates the amount of peers known to the tracker and the peers known to a regular client. The large view exploit tries to gather as many of the tracker known peers as possible. The goal is to get often optimistically unchoked and so download at fast rates. If the client is connected to more seeders he also has a higher chance in benefiting from their round robin unchoking. The results of [13] show that with this approach even higher download rates than with a honest client can be reached if the swarm is large enough.

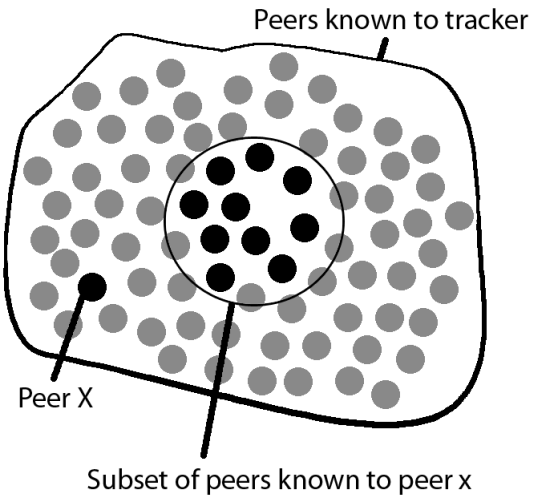


Fig. 4. Peers known to a specific peer vs peers known to tracker

B. Other Optimizations For Achieving Higher Speed

This section will describe a few possibilities to achieve higher download speeds than with regular clients. Some of those approaches could even be combined with the large view exploit to achieve better free riding.

1) Do not download rarest first: The developers of BitThief, a free riding BitTorrent client described in [13] implemented piece selection algorithm that does not apply the rarest first policy but fetches whatever pieces he can get. So it never leaves an unchoke period unused. [13]. This could help to download slighty faster.

2) Only Interact With The Fastest Peers: The authors of [3] describe three exploits for achieving higher download speeds, one of them is to identify the fastest peers and only communicate with them. This also includes to do no optimistic unchoking thus slow peers can not interact with us in any way. The BitTorrent protocol specifies that, every peer should send an advertisement when he finished downloading a piece. Through observing those advertisements it is possible to approximately find the fastest downloading peers of a swarm. In most cases high download speed comes with relatively high upload speed, so those peers are selected. Seeders do not finish any pieces so they do not send advertisements, and thus every pieces from seeds are requested not regarding their speed.

3) Use Sybils: Another good-sounding approach is the use of multiple identities aka sybils. If a single client is connected to a tracker with multiple identities and the tracker delivers those sybils to other peers. The chance of being unchoked could raise. Unfortunately the examinations of [14] show, that this does not result in a higher download speed.

4) Upload Garbage: Because of the tit-for-tat incentives system uploading is rewarded by other peers. Unfortunately we can only upload pieces we already got thus we can not upload to all other peers because they may have those pieces already. To exploit this we could attempt advertising

rare pieces we do not really have (and if requested upload random garbage). This is very similar to the piece attack described in chapter 3. Because the integrity check fails not until all blocks of a piece are gathered by the downloading peer, we could try to upload only some blocks per piece. Additionally in [3] is suggested to not advertise all pieces because then other peers would think we are a seeder and thus not letting us download from them. In theory the peers should not recognize us as malicious peer and so reward us for uploading to them. As the evaluation in [3] and [13] shows this really works for the official implementation. Unfortunately the official implementation is not used much and clients like Azureus are not as easy to fool. E.g. Azureus uses a pretty nice mechanism for preventing this attacks: if an integrity check fails, it looks who contributed with the most blocks to it. Then Azureus requests the missing blocks from that peer. If the peer refuses to answer or sends garbage his IP gets banned.

V. Comparison And Future Work

This work was based entirely on BitTorrent and some of the described problems are very specific. You might be surprised but still BitTorrent is one of the most resilient P2P overlay networks today.

Former networks like FastTrack (used by KaZaA) which was very popular have suffered massive attacks or law suits and thus became nearly unusable or totally disappeared. All of the P2P file sharing networks left today are prone to many different kinds of attacks and selfish peer behavior.

Future protocols have to learn from the mistakes of previous ones. Today there is much scientific research done on the development of robust, yet scalable P2P networks and many of the discussed attacks could be prevented.

During the last years BitTorrent got quite some additions to the original protocol, e.g. distributed hash tables (DHT) were introduced for trackerless communication.

Self-organizing, scalable and robust DHT may play a big role in P2P systems of the future because through their use P2P networks can be freed of trackers or other central servers which are always prone to attacks or law suits. Although DHTs bring many new problems, e.g. how can sybil attacks be prevented if there is no central entity that can assign identities? In papers like [15] fundamental design principle for such DHT based networks are discussed, they will help in further development.

VI. Conclusion

The contribution of this work mainly was to summarize and describe different types of attacks and exploitations that could harm BitTorrent and other P2P file-sharing overlay networks. Some of them are already used massively by attackers. It has been found that BitTorrent is vulnerable and exploitable in many different ways. Still today BitTorrent is one of the most robust public used P2P networks (if not the most robust) and has the potential power to stay alive for a long time. Therefore it is important to reveal and understand possible attacks. Protocol additions like DHT and different implementations of the BitTorrent protocol already exist. To become more robust the official protocol should include workarounds for known attacks, some possible countermeasures were broached in the course of this work. The designers of new P2P overlay protocols (at least those intended for file sharing) should consider all shown problems and learn from them.

Literatur

[1] P. Dhungel, D. Wu, and K. W. Ross, "Measurement and mitigation of bittorrent leecher attacks," *Computer Communications*, July 2009. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2009.07.006

[2] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent," in *In NSDI'07*, 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.112.6384

[3] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting bittorrent for fun (but not profit)," 2006.

[4] M. A. Konrath, M. P. Barcellos, and R. B. Mansilha, "Attacking a swarm with a band of liars: evaluating the impact of attacks on bittorrent," *Peer-to-Peer Computing, IEEE International Conference on*, vol. 0, pp. 37–44, 2007.

[5] D. Hales and S. Patarin, "How to cheat bittorrent and why nobody does," in *European Conference on Complex Systems*, 2006.

[6] B. Cohen, "Incentives build robustness in bittorrent," 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1911

[7] P. Dhungel, X. Heiz, D. Wu, and K. W. Ross, "The seed attack: Can bittorrent be nipped in the bud?" Tech. Rep., 2009.

[8] S. Rouibia, J. Vayn, O. Beauvais, and G. Urvoy-Keller, "Early stage denial of service attacks in bittorrent: An experimental study," in *WETICE '08: Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 141–142.

[9] Z. CHEN, C. LIN, Y. CHEN, V. NIVARGI, and P. CAO, "An Analytical and Experimental Study of Super-Seeding in BitTorrent-Like P2P Networks," *IEICE Trans Commun*, vol. E91-B, no. 12, pp. 3842–3850, 2008. [Online]. Available: http://ietcom.oxfordjournals.org/cgi/content/abstract/E91-B/12/3842

[10] F. Pontes, F. Brasileiro, and N. Andrade, "Bittorrent needs psychiatric guarantees: Quantifying how vulnerable bittorrent swarms are to sybil attacks," *Dependable Computing, Latin-American Symposium on*, vol. 0, pp. 65–72, 2009.

[11] M. P. Barcellos, D. Bauermann, H. Sant'anna, M. Lehmann, and R. Mansilha, "Protecting bittorrent: design and evaluation of effective countermeasures against dos attacks," in *27th International Symposium on Reliable Distributed Systems (IEEE SRDS 2008)*, October 2008.

[12] M. Sirivianos, J. H. Park, R. Chen, and X. Yang, "Free-riding in bittorrent networks with the large view exploit," in *IPTPS*, 2007.

[13] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *In HotNets*, 2006.

[14] J. Sun, A. Banerjee, and M. Faloutsos, "Multiple identities in bittorrent networks," 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.9160

[15] B. Awerbuch and C. Scheideler, "Towards a scalable and robust dht," 2006.

Group Management in Peer-to-Peer VPNs

Florian Fuchs
Betreuer: Benedikt Elser
Seminar Future Internet WS09/10
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München
Email: f.fuchs@mytum.de

***Abstract*— The importance of Virtual Private Networks (VPNs) is still increasing at a progressive rate. Apart from the use in companies for intranets and so called home offices, VPNs are already used in the private area for file-sharing or for the secure access to home networks. Besides a centralized organization of the network, which companies usually prefer, there is also a decentralized approach through a peer-to-peer (P2P) network imaginable. P2P networks are highly scalable, extremely flexible and lack a static infrastructure like servers which makes them affordable. Peer-to-peer VPNs are therefore an easy way for private users to build their own private network. Yet with peer-to-peer networks new kinds of problems arise. The administrative tasks, in centralized networks performed by servers, have to be handled by the peers themselves. This implicates designing an identification and authentication process as well as maintaining established connections and handling the frequently changing amount of users.**

This proceeding presents and compares different group styles and current implementations in regard to how they address these problems. There are theoretical concepts of groups and applications which attempt to provide an user-friendly solution. The focus of the proceeding is mainly identifying the characteristics and the limits of the various group styles and presenting the related challenges one might face when realizing a specific style. The proceeding concludes with a short discussion about the anonymity problem which occurs when more than one VPN is set up.

***Index Terms*— P2P, VPN, group management**

I. Introduction

A virtual private network is a secure logical network which manages tunneled connections between different parties through another, usually insecure, network like the internet [1]. The purpose is to provide exclusive services for the members of the private network. For companies an apparent adoption of this idea is an intranet. Employees all over the world have access to one common site and thus to the provided applications. This enables so called home offices where the employee has no more need to be present at the office while still having the ability to use all internal services of the company over the network connection.

Besides the use of VPNs in companies there are multiple possibilities for using them in the private area. Imagine sharing photos, music or videos through just storing them in a public folder, instant messaging without the need of an external service provider or playing computer games with friends using a local machine as server.

The impulse for using a peer-to-peer network instead of the classical client-server architecture is normally the amount of disposable resources distributed over the totality of its members. Peers release available resources or provide services for other peers and in exchange use resources from these peers. A common goal usually unites all peers, for example sharing files in large P2P networks like Gnutella.

Nonetheless there is a big difference between the employment of virtual private networks based on an existing infrastructure or on a peer-to-peer network. A VPN of a company for instance is highly centralized, meaning there is a static organization which provides the administrative services needed for establishing and maintaining the network. Contrary, in the private area there is normally not an adequate infrastructure and the members of a network might change more frequently. Both, the non-availability of a central server and the demanding administrative effort, have to be faced when designing a peer-to-peer VPN.

How to identify users for instance appears as a new challenge due to the lack of a central entity where peers can easily register themselves. New, until now unknown, users want to become a member and for security reasons there has to be an authentication process. What happens when active members quit their membership since they may have performed tasks for the group and provided a part of the collective knowledge?

All mentioned problems can be summarized as the management of a group, while the group denotes all current members of the decentralized network. In the following proceeding these problems will be addressed. The first part describes general characteristics and challenges of peer-to-peer VPNs while in a second part different group styles will be introduced. In the third part different implementations of peer-to-peer VPNs will be presented and how the group management is handled in each of them. As examples serve the SocialVPN application [2], the IgorVPN application [3], the Layer Two Peer-to-Peer VPN approach by Deri et al. [1] and the ELA approach by Aoyagi et al. [4]. Finally, there is a short discussion about the anonymity problem which is highly related to decentralized networks.

II. Characteristics of P2P VPNs

There are different types of networks depending on the degree of decentralization. Segmenting the types in three

parts, namely the centralized, the semi-decentralized and the decentralized network seems easy and works fine for the presented concepts. The main difference between them is the support through an external infrastructure. The centralized network was already mentioned in the introduction when talking about intranets and is not of our interest now. The distinction between the semi-decentralized and the decentralized network is quite useful since even the smallest support from an external entity might make things a lot easier. The terms decentralized network and peer-to-peer network will be used synonymously. Furthermore challenges related to decentralized networks can be classified in two categories.

Firstly, users of decentralized networks have to discover each other and subsequently a connection between them has to be established. In general, there is no need to know the identity of the other users in private networks as long as the members trust each other. In centralized networks a user trusts the superior authority and the authority itself is the contact point for new users. In contrast, in a decentralized network both, locating and the authentication of the members, have to be organized somehow and by someone. There may be more than one single response point for the users of a network as we will see later in detail.

Secondly, once a network is established, it has to be maintained. New users probably want to join the network and this raises the issue by who and in which manner new users are approved or denied. What happens in a situation where a member should be excluded from the network? Since there is usually neither a trained administrator nor a distinctive intention of the users to manage the peer-to-peer network, the administrative tasks should be minimized somehow.

Due to the lack of a centralized technical infrastructure in peer-to-peer networks, there are normally no fix costs. Therefore a P2P network is usually really cheap to maintain and fast to establish. The potential members just need to identify each other and then establish a connection, assuming there is a proper software providing this service. For a P2P VPN the minimum amount of members is obviously two whereas theoretically no limit for the amount of participants exists. In practice huge numbers of peers do matter due to the data overhead which will probably result when the number of peers increases [5], [6]. This shows the urgent need for an adequate management of the peers in a group. This flexibility makes a P2P network indeed easily scalable, a fact large P2P networks like BitTorrent or Gnutella are using [7]. It is important to notice that the VPNs intend to establish whole networks and not only supporting single services like file-sharing.

In a company the potential members of a VPN are easy to identify, the distribution of keys seems obvious and usually there is already a technical infrastructure which can be used for the network. Technical infrastructure suggests not only the presence of servers and available IP addresses yet furthermore technical workforce with the necessary knowledge.

The downside of using P2P connections in VPNs is their vulnerability. There is not only a risk from outside the network like in centralized networks yet also from the inside. One peer might have a hidden intention and, depending on the organization of the group, also the power to impair other peers [8]. This might leave a feeling of insecurity between the members of a group. Hence, the solution of this challenge is essential when one considers using a P2P network as fundament for a VPN. Thus the group management has to include mechanisms which are able to handle these threats.

There are already theoretical models about trust management [8]. The different approaches range from ranking other members to applying the context of the situation to every trust decision. Trust can thereby be represented by a simple decision or by different values on a whole scale of trustability. The trust decisions can furthermore be kept secretly or spread into the group while adopting the trust decision of another member also implies that one trusts this member totally. There is also the overall question how much every node in the network knows about the other nodes. If only direct neighbors are known and the whole communication is handled by them, the trust management should be really simple. Yet with a growing number of connections the required knowledge about others increases and the management activities may use unjustified resources. This can be countered through simply collecting information about bad behavior, namely creating a black-list, and distributing the information over the network. The decision should always be based on the success of a previous action. The question by whom the outgoing of an action is determined is indeed still unsolved. Should the decision be an automatically proceeding process or a process requiring user interaction? Implementation of different approaches of trust management depend extremely on the characteristics of the group and the decision should be made individually in every situation.

While the trust management can be classified as an internal problem of groups, peer-to-peer VPNs also face several impediments from the technical infrastructure. Since the IPv6 standard is not fully implemented yet, applications have to handle connections using the older and with a smaller address space equipped IPv4 standard with all related problems. Most computers are behind firewalls now and already in a local network managed by a router which handles the network adress translation (NAT). The routers may block due to security issues every incoming connections while allowing outgoing. Even though there are already technical solutions to these problems, the group management part of an application plays a major role [9].

Most users of P2P VPNs are no network administrators and the configuration of an application should therefore be as easy as possible. For that reason most applications use zero configuration networking (zeroconf) where the network is set up automatically. Once the network is deployed, the user should also not be overstrained with decisions affecting the group management. In the following section different group types will be presented and some of them require the user to be active. The implementor should be aware that not every user wants to be that active in a group.

III. Group styles

When defining group styles, the point of view has to be determined. This paragraph cares mainly about the management part of a group. This means how the group can be described as an organizational entity. Hence the access to the group and how it can be controlled seems to be a promising approach due to the fact that also the leaving of a member, especially when the member is active in a hierarchy, has an influence to the approval process of new members.

In this part different groups styles will be presented, based on the work of Salzeder [3]. Beginning with the easiest one, occurring problems will be identified and the need for more robust ones will emerge. The first challenge a group of peers which decide to create a P2P VPN might face is how to seperate themselves from other users of the same network they are all in.

A. Paradise

To start with the simplest style, the peers build a group by simply choosing a name for that group or more clearly an unique identifier. A new user becomes a member of the group by adopting the name. This fact is in the context of VPNs impractical and in general strongly idealized. Only in a network where every user trusts everyone else and all users stand to the rules the so-called ”Paradise” seems to fit. Figure 1 shows a group as a part of a set of peers. While in this group all users are connected among themselves the single peers interact with different other peers outside the group. A more expedient way needs security against unwanted peers for the group.

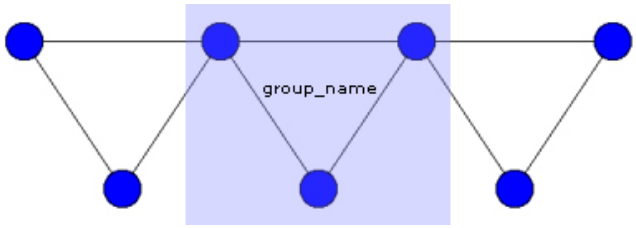


Fig. 1. Paradise: Forming a group

B. Password protection

Let’s ignore internal threats for the time being and focus on security against other users. The apparent solution for securing the network against unwanted members is to define a password for the group. Therefore the group is still identified by the group name yet the password is needed to enter it. The password can either be set by the founder of the group or in collaboration between all potential members. The security issue about the access to the group seems fixed. New users need only the password to become a member of the group as Figure 2 depicts and use the password to identify themselves.

Once a peer has the password it appears difficult to exclude or ban a member from the group. A new password has to be set and spread between all members except the one which should be excluded. Since there is usually no central entity in a P2P network the delivery of the new passwords is the duty of several members. All these members need to be aware which member is designated to be banned. Otherwise this member might be in the position of getting the new password through which the whole action fails. In conclusion, a password protection seems only helpful when the group has a static consistency and is not durable. For this reason the type is also called a ”temporary group”.

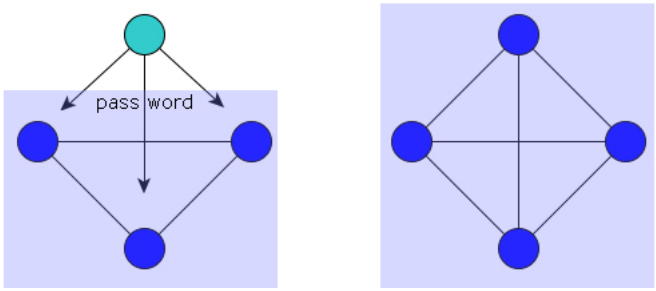


Fig. 2. Password protection: Using a password to enter the group

C. Monarchy

One might think about defining one or more entities inside the group performing tasks like setting new passwords, respectively banning members from the group. Assuming one member is assigned for this kind of tasks the group is called ”Monarchy”. The rights of the monarch might vary depending on the responsibility. If the monarch also maintains the right of approving new members to the group even the password is not needed anymore. How such a situation may look like is indicated by Figure 3. Still two challenges remain unsettled. At first, what happens if the monarch leaves the group? Is there a hierarchy which member inherits the position? And if so, how is it designed? Secondly, hypothesize the monarch itself is the one to be banned from the group. The latter seems unsolvable since there would be a need for complex control mechanisms.

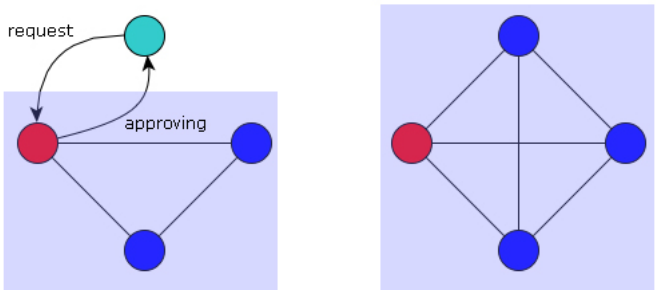


Fig. 3. Monarchy: admission process

D. Supporters

Two solutions might address these challenges. Next to the monarchy a whole hierarchy of responsibilities could

be implemented. Yet this seems to fix only the succession problem and is therefore not otherwise specified. Imagine instead separating the members of a group in different sub-groups. The easiest case are two groups, divided through the distribution of rights. One group contains users as they are also in the prior types and the other group contains members called ”supporters”. Figure 4 illustrates an example of two sub-groups. How many connections between the members exist depends on the purpose of the group. Nonetheless it appears helpful if all supporters keep connections among themselves. A non-supporter needs theoretically only a single connection to a supporter to keep connected to the group.

Supporters are similar to monarchs apart from the fact that there are several supporters while there is only one monarch. The rights of one supporter may also vary depending on the group and it seems useful that every supporter maintains the same rights. Rights could be approving new users, banning existing members or appointing new supporters. When setting up a group with supporters problems like how many supporters are needed to provide support at any time and how to become a supporter have to be solved. Yet the central question of this style is whether a supporter has independent rights or does his decisions rely on the approval of other supporters?

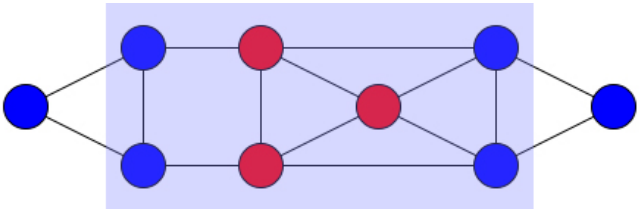


Fig. 4. Supporters: red circles indicate the sub-group supporters

E. Voting

As already discussed, decisions can be made independently or in common. If voters make a decision together, it is not as important to differentiate between supporters or normal members. One of the central issues of this style, called ”Voting”, is instead how the base should be designed since in networks not all users are available at all times. One apparent example is the time difference between regions. If one decides to count only online users there might be a situation where a small group of members gets unjustified power over other users. In contrast, if one requires too many members for a valid decision there might be a lack of decisions when not enough members are available.

Nevertheless this style is extremely flexible. Decisions can be made on a relative or an absolute basis. Relative means that a determined minimum of approvals are needed to accredit a task. Figure 5 shows a group controlled by supporters with a voting mechanism. A peer wants to become a member and therefore sends a membership request to a supporter of the group. The supporter forwards this request to other supporters and counts the responses. Once a minimum of approvals or denials is counted, the original supporter gives a reply to the potential new member. In the example the applicant has no majority and the access to the group is therefore denied [10].

As already mentioned this system is flexible and this example serves only as one of many. Yet it reveals one more problem, the need for a lot of traffic, especially if there are no supporters. The more democratic the voting system is, the more messages need to be sent between various members. Hence, the whole system gets slow and complex if not just one member manages the process. This might happen since in a large group not every member maintains a direct connection to every other member.

In the paper by Saxena et al. [7] about Threshold Cryptography using certificates for access control is proposed. A threshold results if some of the members play against the rules of the group. Therefore the cryptographic keys are distributed among the members of the group and decisions are made by mutual consent. Since every member possesses only a share of the key a specified minimum of group members is needed to sign messages for the group by contributing their share of the key. This prevents a single member from cheating.

This system can also be used for the access to a group. New members ask trusted members of the group for a part of the shared secret. Every member which approves admission sends this part to the potential member. Once the applicant has enough parts he is able to reconstruct the membership certificate. Using the certificate the new member is able to sign messages and prove membership to the rest of the group. Nonetheless in this method the question how to exclude a present member is also apparent. The approach does not provide a solution for this kind of problem yet.

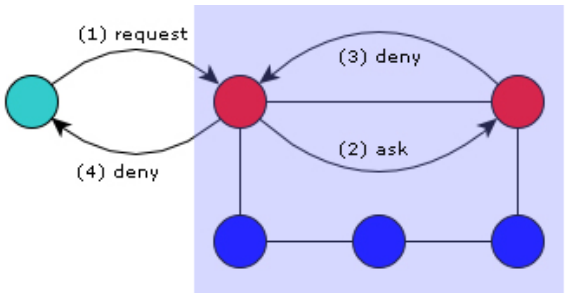


Fig. 5. Voting: Denying a request based on supporter voting

F. Anarchy

As seen in the last section, a member may have the intention of cheating. Therefore the group management should support mechanisms to prevent situations where risks for other members emerge. In a real network it will probably not be possible to suppress all misbehaviors. Large amounts of members also intensify the problem since the complexity increases according to the group strength. With cheating every violation of group rules by one or multiple group members is denoted.

A violation can be as simple as the deliberate rejection of a connection to another honest group member. Figure 6 shows

the situation where the peer labeled with an ”A” blocks every connection to the peer labeled ”B”. From a technical point of view this is easily realizable through an packet filter on the ethernet level. Nevertheless it destroys a part of the group structure and it seems hard for other members to detect this disruption of the network. Other peers might still rely on the existence of the connection while it is already blocked for a long time.

We call this situation ”Anarchy” yet it is obviously more naturally emerging than an proposed group style. It is proposed since it may arise very fast. As already mentioned peer-to-peer networks are characterized through a frequent change in memberships. For this reason new connections between peers are established and existing connections terminate generally with a higher frequency. Thus the group has usually a dynamic character and the behavior is hard to predict.

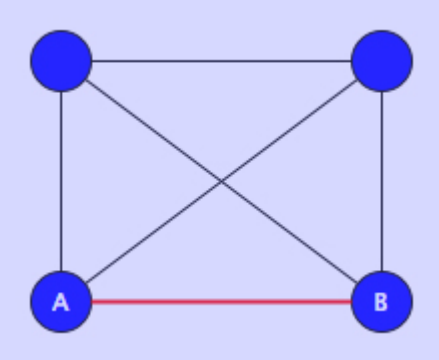


Fig. 6. Anarchy: Suppression of a connection between two peers

G. Web of trust

Until now the trust decision was either made completely by every peer itself or by a higher authority, namely a monarch or a member of the supporter sub-group. It is also imaginable that the authentication process is designed in a way that either the user itself or another peer certifies other peers. Therefore a peer has a group of other peers he trusts completely. This means he trusts in their decisions concerning, for instance, the approval of new group members. This adds transitivity to the trust model and is called a ”Web of Trust” [11] . Figure 7 depicts this graphically. Peers A and B trust the other peers and accept their certification of peer C. For that reason they establish a connection although they have never authenticated peer C by themselves. The certified connection is displayed by the dotted line.

While transitivity knows no limits, it is advisable to employ it for only one level. That ensures that only direct neighbors are able to certify for the peer and no net of complex linkages arises. Nonetheless users of a Web of Trust should be aware that a chain is only as strong as its weakest member and that they may thus suffer from failings of their trusted neighbors. Once a user has the certification of one trusted member it is easy for him to get additional certifications. Following this pattern the dishonest member may reach uncontrollable influence in the group and finally impair other peers using his achieved power. Even more dangerous is a situation where this member acts as a trusted node for other peers. Then he is also able to introduce new unwanted member to the group.

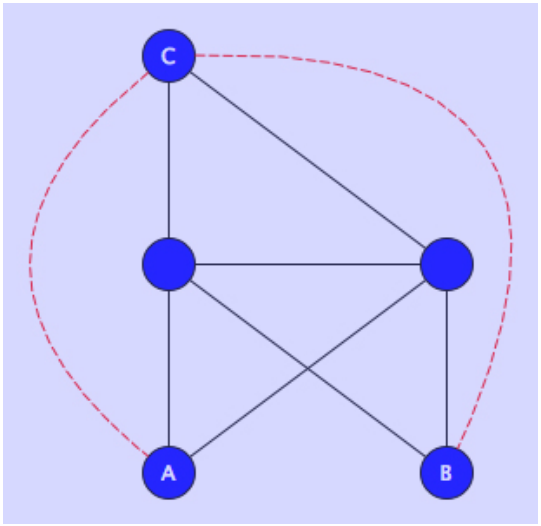


Fig. 7. Web of trust: Adding transitivity to the trust model

IV. Implementations

There are several different implementations of peer-to-peer VPNs next to the established, yet centralized, VPN applications like the Cisco VPN clients or OpenVPN. In the following, four examples will be presented which handle the raised challenges in different ways. The first example is the SocialVPN application [2], followed by the IgorVPN application [3]. Other promising approaches provide Deri et al. with the Layer Two Peer-to-Peer VPN [1] and Aoyagi et al. with the Everywhere Local Area network [4].

A. SocialVPN

The SocialVPN application [2] uses a social network as communication point to identify groups. It is therefore apparently not a fully decentralized P2P network and at the moment only the Facebok API serves as example for the support through a social network. For the selection of networks the ability to authenticate users, the possibility to query relationships and exchanging the cryptographic certificates is necessary. All this features are already provided by the Facebook API which made it the favorite option.

A group in the application is represented by the relationships between the user of the application and the peers on the other side of every relationship. Thus, the group is exactly tailored to every single user. Peer-to-peer networks, and especially P2P VPNs, can therefore be seen as an entity representing social structures and as a gate for fulfilling the needs members of the group have.

Using one group per user makes the SocialVPN application even more interesting since the center of an actual group is the user itself. Every user has its own group: the relationships

transferred from the social network. That derives from the fact that at every end-point of the network a virtual IP namespace exists managed by a virtual VPN router. The used technique for this is Brunet for the connectivity between the peers even behind NATs and IP over P2P (IPOP) which provides the fundament for the application to communicate with local IP addresses over a virtual network interface [9].

Which group the user of the application belongs to is defined by all relationships in the social network. At first it might look like the implementation of a Web of Trust. In a Web of Trust there would be only one group yet in the SocialVPN every user has its own group. An overlapping is theoretically only possible if two peers maintain exactly the same relationsships. Nevertheless the peers act in different virtual name spaces.

To establish a connection to another user obviously both have to use the same software. There is no chance to become a member of the group as long as there is no corresponding relationship in the social network. This seems to solve the security issue to the inside since the identification of users is shifted to the social network. It is implicated that the user trusts the users to which he maintains relationships in a social network and the social network is a convenient way of managing these relationships.

The security against breaking into a group is obviously only as strong as the security of the social network. The Facebook API is also used to exchange certificates in a secure way and it is therefore absolutly inevitable that the user trusts the social network. The established connections between the members in the network are then protected by IPsec and public key cryptography (PKI).

Every user of the application has also the possibility of blocking other users. This means that they are not able to use any services provided by the user even though they are in the group. The application supports this by a simple checkbox for every single member of the group.

The application is also able to support multiple computers of one user. A group may therefore contain various devices by every user. Virtual name spaces are used to alleviate the communication between users as much as possible. The name consists of the names of the machine, the users first and last name, the social network and ipop as suffix. An example would be *homepc.lisa.a.facebook.ipop* or *homepc.lisa.adams.facebook.ipop* depending on whether the use of an initial for the last name is already unique.

Figure 8 shows some aspects of the SocialVPN application graphically.

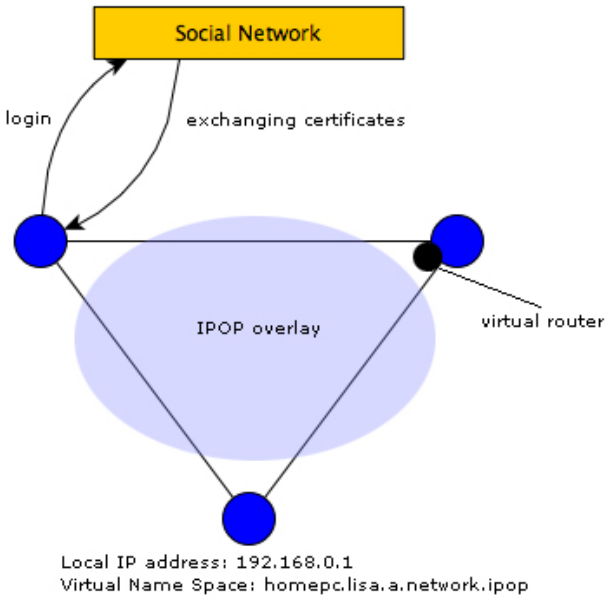


Fig. 8. SocialVPN

B. IgorVPN

The IgorVPN application [3] is a fully decentralized peer-to-peer VPN. Three different group management handler are implemented, namely for the described cases ”Paradise”, ”Temporary Group” and ”Monarchy”. Every type needs a group ID and has the optional element of a group name.

The identity keys of the group members are stored in a distributed hash queue (DHQ) which enables the communication between peers. In *Paradise* a new member needs only to be included in the DHQ to become a member of the group. Then the keys get regularly polled by the members of the group. Thereby all members are able to find everyone else in the group. The drawbacks of that simple mechanism were already described and the author recommends the use only for demonstration purposes.

In *Temporary groups* the group is identified by the group name, respectively the group ID, and a password. The password is set by the founder of the group and it is used to encrypt the entries in the DHQ. In every other aspect the temporary group is similar to a paradise group.

In *Monarchy* one peer has to found the group and acts therefore as the monarch. The only way to enter the group is by an invitation of the monarch. The invitation creates an entry in the subset through which the invitee is able to join the group. The monarch also has the exclusive right to ban members from the group. This is exerted by removing the entry from the subset. An implemented enhancement of the monarchy style is the possibility to assign trust points. This alleviates the occurring delay of distributing updates about banned members since returning members contact always the most trusted member before revealing their own identity to all group members.

C. N2N: Layer Two Peer-to-Peer VPN

The network-to-network (N2N) [1] approach is a fully decentralized peer-to-peer VPN as well. The application aims at providing full network support with all its advantages instead of application support like file-sharing. The members of the network are referred to as a community yet the meaning is the same as the term group used in this paper.

There are two types of members in the group, called Edge Nodes and Super Nodes, as described in Figure 9. This sounds

similar to the concept of supporters, presented as one of the group styles. The Super Node acts thereby as a mediator. Every Edge Node has to register itself at a Super Node. The Super Node has now the chance to manage the connections between Edge Nodes where NAT hinders a direct connection setup between the nodes. Even though a peer is protected by a NAT router he has the chance to get an arranged connection by the Super Node which registers him to the other peer. This detour also makes sense in a way that the trust issue is forwarded to the Super Node which may have more information about the members of the community than a single Edge Node.

A list of solid paths to Edge Nodes is saved and used to help asking Edge Nodes to communicate to other Edge Nodes. This enables the Super Nodes to act like a virtual network infrastructure. The original data is not sent through the super node since it has only an administrative character. This prevents this approach from excessive data overhead.

The N2N approach provides no information about the access control at the moment, except for the fact that there is an invitation process. Yet it shows how support from the inside of a group could be implemented and how supporters have to be designed to perform such support. Another feature of this approach is that every user can be a member of various groups since the application favors the establishment of connections more than the direct offering of services like storing data by every group member. Therefore each host maintains a virtual network interface for every network it is connected to.

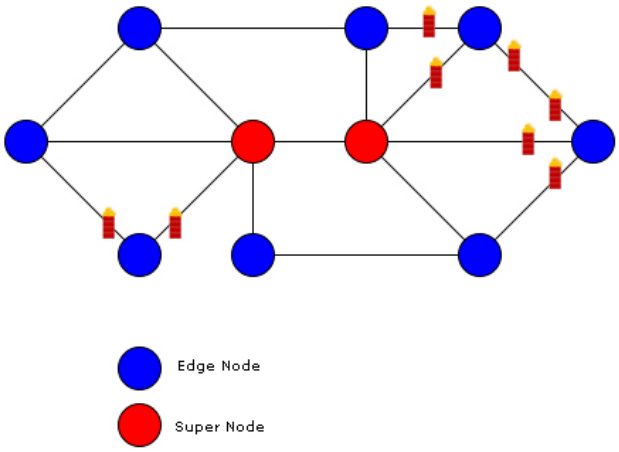


Fig. 9. N2N: Layer Two P2P VPN

D. Everywhere Local Area network

Like the IgorVPN application and the N2N approach the Everywhere Local Area network (ELA) is a fully decentralized peer-to-peer VPN as well. The approach also distinguishes its members in two groups yet with another differentiation. The two groups are called Core-Group and Edge-Group and the members are either denoted as Edge-Node or Core-Node depending on to which group they belong. The separation is based on wether the node is able to handle TCP and UDP connections or only TCP connections as tunnels. The use of UDP may be limited due to NAT routers or firewalls in the local network of the node.

To become a member of the network the new node must know at least one existing member of the group. After sending a request to this node the new peer gets classified and a randomly distributed IP address by the Network Pseudo Device which is a part of the ELA-VPN application.

While Core-Nodes are connected among one another, the Edge-Nodes maintain only one active connection to a Edge-Node. For backup there is a second connection to a different Core-Node yet it will be active only in the case the first Core-Node terminates its service. Hence, every Edge-Node has only one direct contact point to the inside of the group. This looks similar to the supporter group style with a full range of power for the core-nodes since they control the access to the rest of the group for the Edge-Nodes.

This concept is similar to the approach about information retrieval in third generation peer-to-peer architectures described in [6]. There a separation between leaf nodes and ultrapeers enables an efficient content distribution. Leaf nodes are connected to an ultranode while ultranodes are also connected among themselves. While lead nodes can only be a member of one group, called a club in the paper, an ultranode is used to connect the groups among each other.

In the ELA-VPN the whole routing process is based on the Core-Nodes which means that the Core-Nodes have to care about the destination of IP packets. The End-Node just forwards their packets to the Core-node.

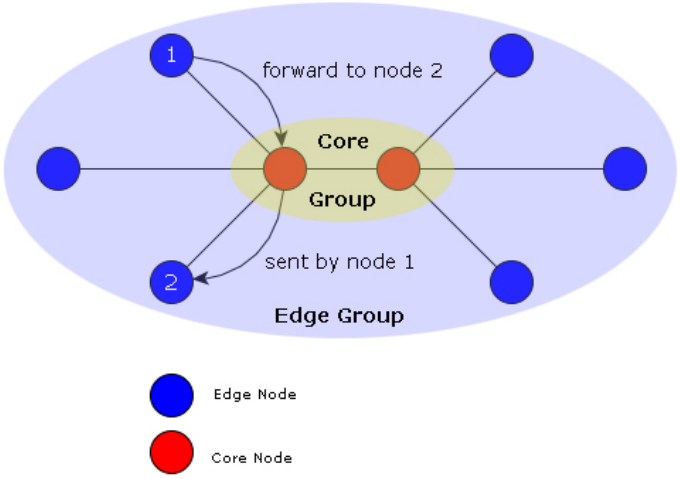


Fig. 10. Everywhere Local Area network

V. Anonymity

The prior section described several implementations of peer-to-peer VPNs. The focus of the presentation was on how they implement groups and how the groups are managed. Thereby the view was mainly defined by the existence of only one peer group and one virtual private network.

Imagine a peer is a member of more than one group like the situation described in Figure 11. The arising challenge is separating the traffic and the services provided by each single group. Otherwise there are high security risks for all members of all groups since they do not know anything about the members of the other groups. Therefore the group management of an application needs mechanisms to protect the user against the risks of this situation. As this is really hard most applications do not support multiple group membership yet.

In the SocialVPN [2] application this problem is solved quite elegantly. Since one user has only one group of relationships there is no need for a second group. At the moment the application supports only the Facebook API and it is not clear how it will be handled mixed up with other social network sites. Yet maintaining only one group even if it is mixed through multiple social networks seems a durable solution for this problem. Since every user manages his own local group the origin of a connection is only important for security issues.

The N2N application supports the membership in various groups as already mentioned. In the IgorVPN application the support is not implemented yet while there is no information at the moment how the ELA-VPN application handles multiple group membership.

Two different challenges have to be kept apart in the context of anonymity. *Firstly*, looking at a specific member of the group, there are services which the peer offers to other members of the group. This services might be providing public folders for file-sharing, a web server or other applications like games to which other users can connect. Two different problems may arise in this context. The first is that one service is only for one group while another is only for a different group. In general, the services have different addressees. The second problem is that one service should be offered to both groups yet with different content. *Secondly*, members of the same group might be in the position of asking for services. Therefore, the user has to assure that this user is not able to observe all provided services yet only the ones which are intended for his group.

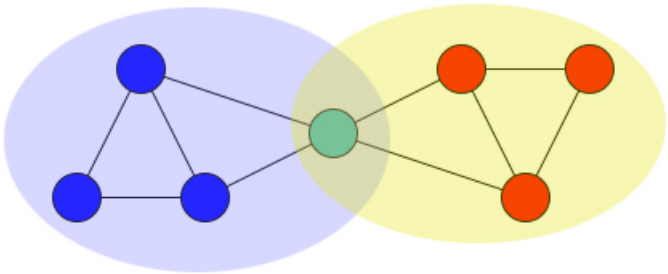


Fig. 11. Anonymity

VI. Conclusion

The contribution of this proceeding was the detailed presentation of different group styles and the evaluation of related implementations. As inspection depicted there are already sound applications which use a variety of techniques to solve the raised challenges. While most applications address all the related technical problems important group mechanisms remain unsettled. The SocialVPN application appears the most promising approach even though it is only a semi-decentralized approach. There is a clear authentication process supported by a trusted external entity and still the ability to block users. This seems an easy way for private users without network skills. The other presented concepts provide interesting approaches to the issued challenges related to group mechanisms. Also the Threshold Cryptography [7] as a concept to implement voting in a group appears promising. Yet an application which implements this technique in a robust technical environment is still missing. The development of peer-to-peer based virtual private networks seems still to be in its infancy. Most concepts are more a prototype for a specific problem than an ready-to-use application. Only the SocialVPN application seams already suitable for non-technical users.

References

[1] L. Deri and R. Andrews, "N2n: A layer two peer-to-peer vpn," in *AIMS '08: Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 53–64.

[2] R. J. Figueiredo, O. P. Boykin, P. St. Juste, and D. Wolinsky, "Social vpns: Integrating overlay and social networks for seamless p2p networking," June 2008. [Online]. Available: http://byron.acis.ufl.edu/papers/cops08.pdf

[3] M. J. Salzeder, "Using fully decentralized peer-to-peer technologies for virtual private networks," Diplomarbeit in Informatik, April 2009.

[4] S. Aoyagi, M. Takizawa, M. Saito, H. Aida, and H. Tokuda, "Ela: A fully distributed vpn system over peer-to-peer network," in *SAINT '05: Proceedings of the The 2005 Symposium on Applications and the Internet*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 89–92.

[5] D. Bin, W. Furong, and T. Yun, "Improvement of network load and fault-tolerant of p2p dht systems," in *Information Technology: Research and Education, 2006. ITRE '06. International Conference on*, Oct. 2006, pp. 187–190.

[6] A. Asvanund, R. Krishnan, M. D. Smith, and R. Telang, "Interest-Based Self-Organizing Peer-to-Peer Networks: A Club Economics Approach," *SSRN eLibrary*, 2004.

[7] N. Saxena, G. Tsudik, and J. H. Yi, "Threshold cryptography in p2p and manets: The case of access control," *Comput. Netw.*, vol. 51, no. 12, pp. 3632–3649, 2007.

[8] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*. New York, NY, USA: ACM, 2001, pp. 310–317.

[9] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "Ip over p2p: Enabling self-configuring virtual ip networks for grid computing," 2006, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2006.1639287

[10] T. Bocek, D. Peric, F. Hecht, D. Hausheer, and B. Stiller, "Towards A Decentralized Voting Mechanism for P2P Collaboration Systems," Department of Informatics, University of Zurich, Tech. Rep. ifi-2009.02, March 2009.

[11] A. Datta, M. Hauswirth, and K. Aberer, "Beyond "web of trust": enabling p2p e-commerce," in *E-Commerce, 2003. CEC 2003. IEEE International Conference on*, June 2003, pp. 303–312.

ISBN 3-937201-08-4

ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)