ANTS - A Framework for Knowledge based NAT Traversal

Andreas Müller, Andreas Klenk and Georg Carle
Chair for Network Architectures and Services
Technische Universität München
{mueller, klenk, carle}@net.in.tum.de
http://www.net.in.tum.de

Please note that this is a preprint version of the paper. The full final paper was published at the IEEE Globecom 2009 Next-Generation Networking and Internet Symposium, Honolulu, Hawaii, USA, November 2009

***Abstract*—Today most home networks are connected to the Internet via Network Address Translation (NAT) devices. NAT is an obstacle for services that should be accessible from the public Internet. Especially applications following the peer-to-peer paradigm suffer from the existence of NAT. Various NAT Traversal methods emerged in research and standardization, but none of them can claim to be a general solution working in the heterogeneous environment of today's networks. This paper introduces the Advanced NAT Traversal Service (ANTS), a framework improving the communication of existing and future applications across NAT devices. The core idea of ANTS is to use previously acquired knowledge about NAT behavior and services for setting up new connections. We introduce the architecture of the extensible framework and propose a signaling protocol for the coordination of distributed instances. Finally, we compare the framework to ICE showing that ANTS is not only more flexible, but also faster due to the decoupled connectivity checks.**

I. Introduction

When Network Address Translation (NAT) was proposed in [1], it was only seen as a temporary solution to the shortage of IPv4 addresses and was therefore not standardized sufficiently. The purpose of NA(P)T (Network Address and Port Translation) is to share one public IP address among a number of private hosts by using ports for multiplexing. This worked well for years, but with the growing success of peer-to-peer applications and Voice over IP (VoIP), many services suffer from the existence of NAT and firewalls. Both types of middleboxes usually assume asymmetric connection establishment and only allow inbound packets as an answer to outbound packets. And although IPv6 provides a large address space with unique addresses for each host, firewalls will still be present and enforce asymmetric connection establishment. Standardization for IPv6 NAT is already underway [2] because some features of NAT, such as topology hiding and privacy for local area networks [3], will still be desirable.

According to [4] and [5], we can differentiate between four NAT Traversal problem domains: *Peer-to-Peer Applications* expect to be reachable from the public Internet, an operation not available without creating a mapping via an outbound connection first. Protocols, such as SIP, using *Realm Specific IP Addresses* in their payload fail because NAT does not operate above layer 4. *Bundled Session Applications*, such as FTP, carry realm specific addresses in their payload, which are used for establishing an additional connection (control and data session). The last category covers *Unsupported Protocols* where the layer 3 or 4 address is not available for translation (e.g. due to encryption or new protocols such as SCTP and DCCP). Consequently, establishing connections through NAT devices became a field of intensive research [6], [7], [8]. However, due to the non-standardized behavior of NAT itself, most of the existing NAT Traversal solutions only work with certain NAT devices and few protocols. Many solutions focus on UDP and SIP [9], other solutions target TCP [10] exclusively.

The Interactive Connectivity Establishment (ICE) [9] is an IETF draft describing "a protocol for NAT Traversal for UDP-based multimedia sessions". ICE exchanges a number of potential endpoints, which are then tested for connectivity. Since these tests have to be performed for each new connection request, ICE creates a large delay during connection setup. Furthermore, ICE only establishes connections between applications on two hosts that both execute an instance of ICE. As a result ICE cannot be seen as a solution for arbitrary applications.

We propose the Advanced NAT Traversal Service (ANTS) to address these issues. In this paper we only focus on NAT and firewall traversal, but ANTS can also be seen as a general architecture helping to set up connectivity when multiple options (e.g. for multihomed hosts) are available. ANTS separates the gathering of applicable NAT Traversal techniques from the connection establishment and applies its knowledge whenever connectivity should be established. The ANTS approach not only considers NAT behavior, but also the type of connectivity an application wants to obtain. When choosing a NAT Traversal technique, it makes a big difference if an application needs to be accessible by arbitrary clients in the Internet or if one single connection between two given hosts should be established.

This paper has three main contributions: 1) Knowledge based NAT Traversal as a method for efficient connectivity establishment across NAT devices. We give an example how ANTS establishes connectivity in critical constellations with symmetric NAT devices. 2) An extensible architecture of the

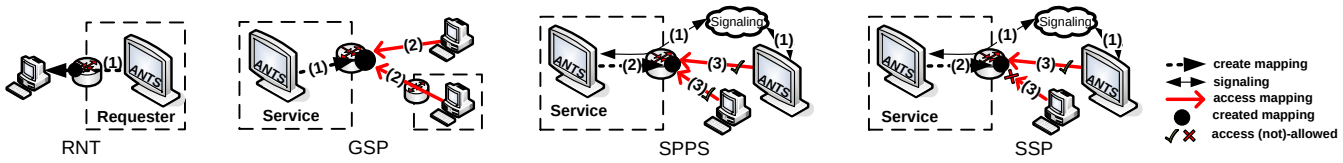

Fig. 1. NAT Traversal service categories for applications. The circle represents the allocated mapping and the arrows indicate who is allowed to access it.

ANTS framework for knowledge based NAT Traversal. The results from our extensive field test show that the implemented NAT Traversal techniques work in almost all cases. 3) Performance measurements show that knowledge based NAT Traversal is significantly faster and scales better in comparison to the traditional ICE approach.

This paper is organized as follows: Sec. II briefly introduces four different NAT Traversal service categories. Sec. III describes the concept of the ANTS framework. The technical details are described in Sec. IV and a reference example is given in Sec. V. In Sec. VI ANTS is evaluated followed by a survey of related work in Sec. VII and a conclusion in Sec. VIII.

II. NAT Traversal Service Categories

In [5] we introduced four NAT Traversal service categories, as presented in Fig. 1. Our categorization emphasizes that the applicability of many NAT Traversal techniques depends on the support of a combination of requester (the initiator of the connection), the responder (typically the service), globally reachable infrastructure nodes and the role of the application.

Our first category *Requester side NAT Traversal* (RNT) covers scenarios where only the requester side supports NAT Traversal (e.g. the application or the NAT itself). RNT helps applications that actively participate in the connection establishment and still suffer from the existence of NAT (e.g. SIP/SDP or FTP). The second category, *Global Service Provisioning* (GSP), assumes that the host which has NAT Traversal support wants to make its service globally accessible. This is done by creating and maintaining a NAT mapping which then accepts arbitrary connections from previously unknown clients. GSP is required for running a traditional server behind NAT. The last two categories assume support at both ends, the service and the requester, to allow more sophisticated NAT Traversal techniques such as hole punching with restricted filtering. While *Service Provisioning using Pre-Signaling* (SPPS) makes no assumptions about the accessibility of a service at all, *Secure Service Provisioning* (SSP) addresses scenarios that require authorization of the remote party before initiating the NAT Traversal process. The hereby established channel must only be accessible by the authorized requester. This restriction can be enforced at the host, at the NAT device itself, at a data relay or at a firewall.

III. Advanced NAT Traversal Service

The concept of ANTS is based on the idea of decoupling the discovery of working NAT Traversal techniques (the knowledge gathering process) from the utilization of such a technique. Instead of determining NAT behavior and detecting working endpoints for every application and connection separately, ANTS establishes knowledge once and re-uses this knowledge for a fast connection establishment.

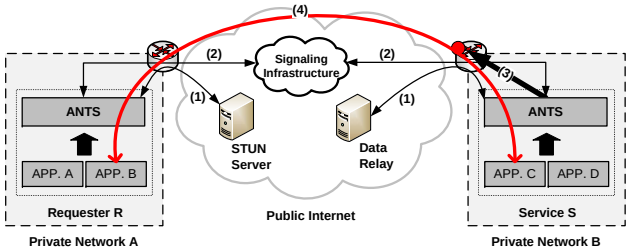

Fig. 2. Establishing a connection in ANTS depends on the available entities. If both hosts run the framework: (1) external servers are queried (2) connection specific information is exchanged (3) the service utilizes a NAT Traversal technique to create the mapping (4) a direct connection is established.

The framework is only installed once on each host and supports all applications requiring NAT Traversal support (see Fig. 2). The connectivity establishment process depends on the available entities such as STUN [11] servers, data relays and signaling infrastructure nodes. The modular architecture consists of three layers and five modules (Fig. 3). Whenever an application needs support, a privileged user registers it at the session manager located at the Input Module and assigns a service category (e.g. a web server needs to be globally reachable, thus GSP). The framework is then able to establish connectivity using one of its NAT Traversal techniques located at the NAT Traversal Module. The selection of a method depends on many factors such as the behavior of the NAT, potential requesters (not all requesters run ANTS) and the role of the application (see Fig. 1 and Fig. 2). To gain knowledge about these parameters, we implemented a small component, the NAT Tester, which is described in [12] and can also be found as a standalone component at http://nattest.net.in.tum.de. It is located at the Input Module and once ANTS is loaded, it performs a number of connectivity tests with a public test server. The Knowledge and Decision Module is then responsible for the selection and parameterization of an appropriate NAT Traversal technique. For GSP, the framework would automatically create a public endpoint using an available technique for GSP (e.g. UPnP), assign a dynamic DNS name to it and report both back to the user. Other connectivity scenarios (SPPS and SSP) require a coordination of ANTS instances on the requester and service. In this case two other modules are needed: the Signaling Module on both hosts for exchanging connection specific information before creating

the NAT mapping (we call this process pre-signaling), and the Application Interface on the requester for connecting applications to the created public endpoint. Sec. IV-B presents two possibilities: the ANTS socket API for newly developed applications and a TUN device for legacy applications.

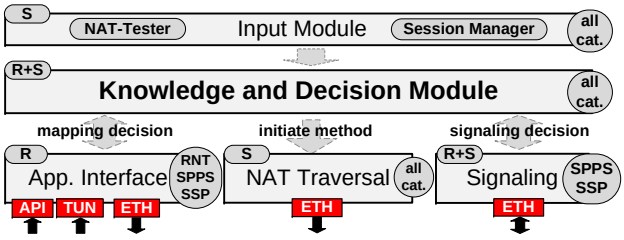

Fig. 3. Modular architecture of ANTS. R denotes that this module has to be available at the requester, S at the service and R+S at both. "all cat." means that this module is needed by all NAT Traversal service categories.

A. Integration of NAT Traversal Techniques

Since none of today's existing NAT Traversal techniques works in every situation, we implemented a number of different techniques to cover a large variety of connectivity scenarios. This section gives a brief introduction to the NAT Traversal techniques and includes the success rates that we observed in a public field test with more than 800 NAT devices (http://nattest.net.in.tum.de). The techniques described below are implemented at the NAT Traversal Module and then invoked by the Knowledge and Decision Module.

1) Universal Plug and Play: Using an UPnP controllable Internet Gateway Device (IGD) [13] is probably one of the most reliable methods for UDP and TCP. Unfortunately, UPnP has such strong security issues that it is often disabled and cannot be used with many NATs. In our test, UPnP was only enabled in 36.42% of all NAT devices. The advantage of UPnP over other techniques is the easy allocation of new mappings. An UPnP mapping does not need to be aware of the source of the connection, it is enough to specify which external port is forwarded to which internal port, thus making it an ideal candidate for GSP.

2) Hole Punching: Hole punching [14] allows direct connections between two hosts and is based on the fact that a NAT forwards all incoming packets if a mapping in its table exists. To create this mapping a packet has to be sent from the port the service is running at towards the source of the connection. The actual connection request then looks like a response to this packet and is forwarded by the NAT. Determining the source of the connection is not trivial because the source ports for applications are allocated dynamically for each connection. Thus, we have to find a way to capture the source port of an upcoming connection in order to send it to the service which is then able to create the mapping. But unfortunately this is not enough. If the requester is also behind a NAT it still needs to determine the external endpoint of the captured port. This is only possible when using an independent mapping strategy or if port prediction is possible. Finally, hole punching works in most cases for UDP (83.5% in our test), but tends to fail with stateful protocols such as TCP (51.8%).

3) Data Relay: With TURN [15] (or a relay in general), a host behind a NAT actively establishes a connection to request a public endpoint. The relay then forwards all traffic sent to this endpoint to the internal host by using the already established connection. Since the connection is always made towards the public Internet, relaying works as long as outgoing packets are not blocked by a firewall and thus can be used if no other technique is available. One of the drawbacks though is that the reliability and the performance depend on an external entity which constitutes a single point of failure.

4) Tunneling: UDP tunneling allows TCP or SCTP packets to travel directly from the source to the destination, encapsulated in UDP packets. Therefore, tunneling for unsupported protocols should be used if there is no other possibility for establishing a direct connection. The success rate of tunneling depends on the success rate of UDP traversal, which was possible in 88.05% (UPnP or UDP hole punching) in our field test.

5) Direct Addressable: If both hosts reside behind the same NAT device (same local NAT or same ISP NAT) or if ANTS is running on a host directly connected to the Internet, we can use the local addresses directly. In our test, 3.7% of the tested hosts were directly connected.

B. Knowledge based NAT Traversal

The main task of the Knowledge and Decision Module is to decide which NAT Traversal techniques are applicable in which situations. Knowledge about the NAT is obtained by the NAT Tester that is ran at boot time and also triggered by new events such as the detection of a new network interface or an IP address change. It answers the following relevant questions: 1) which NAT type is present 2) is port prediction possible 3) which external IP address(es) 4) is UPnP enabled 5) which protocols are supported by ALGs and 6) what other NAT Traversal techniques are supported (e.g. UDP or TCP hole punching). Currently, the Knowledge and Decision Module uses the following prioritization if the NAT Tester discovered more than one working technique: Direct Addressable, Hole Punching, UPnP, Tunneling and a Data Relay. The Knowledge and Decision Module has rules for each combination of a NAT Traversal technique and service category (e.g. UPnP is fine for GSP, but not for SSP). The rules are then needed to transform the knowledge about working techniques into decisions and to provide the dynamic input parameters for the chosen NAT Traversal technique.

- If an application is registered at the session manager with GSP or RNT, our framework allocates the appropriate mapping without any further signaling.
- On an incoming signaling message requesting access for a service on this host ANTS first queries the session manager if the application is authorized to receive connections. The framework then selects a working NAT Traversal technique for the registered application, allocates a mapping and reports it back to the requester.

- If a client application makes a connection request, the framework generates an appropriate signaling message.

A sample of a detailed decision tree can be found in Fig. 5.

IV. Implementation Details of ANTS

After describing the overall concept focusing on the integrated techniques and the Knowledge and Decision Module this section explains the Signaling Module, the Application Interface and the rather technical decisions we made when implementing ANTS.

A. ANTS Request Response Protocol

The purpose of the Signaling Module is to notify the service about an upcoming connection. This is important for two reasons: first it allows hole punching with restricted cone NATs and second it allows the service to only allocate a mapping if the requester is authorized to access it. Thus, signaling first informs the service about the source of the connection, while in the second step the service reports the created endpoint back to the requester.

In ANTS each host has a unique URI (e.g. a SIP-URI [16]) which might be assigned to a DNS name. Whenever a peer wants to establish a connection to a service it queries the DNS for the URI and sends the signaling messages to it. Multiplexing is then done via the service port that has to be included in the so called Service Request (see Fig. 4). More precisely, an URI identifies a host behind a NAT according to its layer 3 address. The Service Request asks for a specific service (port) on this host. The service is then responsible for allocating an appropriate NAT mapping and sends the public endpoint back to the requester. The main advantage of our signaling process is that for SPPS only two messages are needed. For SSP we use the well known Digest Access Authentication [17] which increases the number of signaling messages to four. Instead of exchanging a number of candidates and finding out which one actually works (ICE), ANTS uses its knowledge about the NAT, the application and the requester to allocate a working endpoint resulting in a very fast connection establishment.

The ANTS signaling protocol is based on XML and can be easily used with a number of signaling infrastructures such as SIP and XMPP (Jabber) [18]. The current ANTS implementation uses SIP messages for signaling. However, in the future we aim to use the decentralized P2P-SIP (http://www.p2psip.org/) approach for transporting.

B. ANTS support for applications

When connecting applications to ANTS there are two options: The ANTS socket API allows newly created programs to easily use the framework. The requester application issues the *connect* function with the URI identifying the service as the destination. ANTS then opens a real network socket to connect to the public endpoint created after exchanging information through signaling and finally translates between the sockets.

However, since legacy applications are not linked against the API yet, we implemented a TUN-based solution instead.

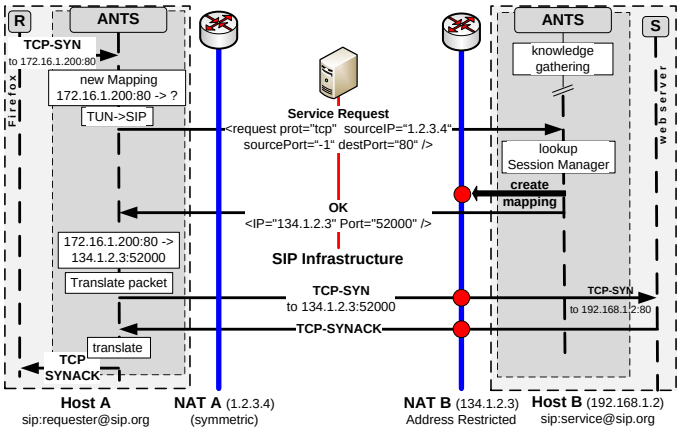

Fig. 4. Reference example for ANTS SPPS

A TUN device is a virtual network interface that delivers packets to the user space of a program and is well known from applications such as OpenVPN. With this approach the requester application only needs to send the packets to an IP address that represents the service and is routed to the TUN device. ANTS gets this packet, initiates signaling and translates the packet to the allocated public endpoint at the service. From now on, ANTS maps all packets coming from the TUN device to the public endpoint and vice versa. The next section gives an example for one specific connection establishment using the TUN- based implementation.

V. Reference Example for ANTS

To get an overall picture Fig. 4 gives an example for how the modules interact. It shows how an application on a host behind NAT (e.g. Firefox) is able to connect to a web server behind a different NAT. We assume that both hosts are registered with a SIP proxy. The service S also adds the combination of a SIP-URI and a DNS name (e.g. *service.nat*) to a DNS server and registers the local port 80 to its session manager. Furthermore, we assume that the requester R is behind a symmetric NAT and cannot predict any global port. Symmetric NATs are known to be the most challenging, because they use a connection dependent algorithm for creating new mappings (e.g. a random port for every connection).

First R launches Firefox and connects to *service.nat* which is resolved (by a DNS proxy in ANTS) to S's SIP-URI (sip:service@sip.org). ANTS allocates an available IP address (here 172.16.1.200) from the range that is routed to the TUN device and reports it back as an answer for *service.nat*. Please note that this IP address only needs to be unique on this host and does not represent the service in general (only the SIP-URI does). Firefox now sends the first TCP-SYN to this address and ANTS allocates a new mapping entry with a pending state. The packet is passed to the Knowledge and Decision Module of R which knows that it is behind a symmetric NAT and cannot predict its public port. However, since the NAT has only one public interface, the global IP address is known. Therefore, R creates a service request holding the source IP address and the

destination port 80 and sends it to S. S looks up its session manager and finds an entry for port 80. From the service request, S knows that the requester is behind a symmetric NAT where IP prediction is possible (indicated by the -1 in the sourcePort field according to Fig. 4). Its local NAT test showed that S is behind an address restricted NAT. According to Fig. 5 this means S has two options leading to a direct connection: UPnP and hole punching. Since S also knows that UPnP is not available but TCP hole punching works, it creates an appropriate mapping in the NAT. The global mapping for port 80 (52000) is looked up via STUN and sent in the OK message. R can now complete the mapping entry by adding this endpoint to the table and set it to available. The held back initial TCP-SYN from Firefox is translated and sent out to the real network interface. Since S's NAT now has a mapping for this packet, it forwards it directly to the web server. From now on every packet in both directions is rewritten according to R's mapping table.

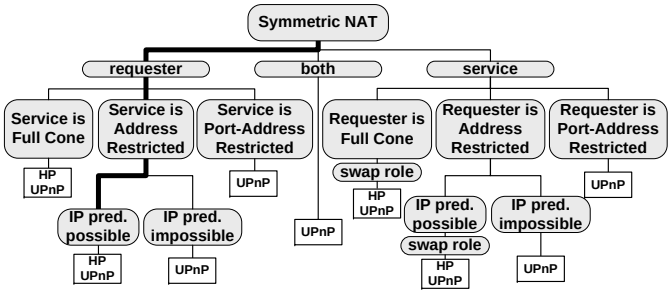

Fig. 5. The decision tree shows how ANTS establishes a direct connection although a symmetric NAT is involved. The highlighted path referes to the example given in the text.

In addition to the example given, Fig. 5 shows the decision tree for ANTS when a symmetric NAT is involved. As shown, a direct connection is still possible in many cases. *Swap Role* means that we simply turn around the connection establishment, the original service becomes the requester and establishes the connection. However, this is out of scope for this paper.

VI. Evaluation

Our field test (http://nattest.net.in.tum.de) with more than 800 NATs has shown that the success rate for a direct connection is mainly dependent on the constellation of the involved NATs. In 43% of the tested NAT devices both endpoints were of type port address restricted leading to a success rate of 94.1% for a direct TCP connection. When we then looked at other constellations, we observed that the success rate for a direct connection is independent of the NAT type as long as both of them implement an independent mapping strategy. As soon as a symmetric NAT is involved (6%), the success ratio dropped to 65.2%. It is important to note that in the remaining cases ANTS allocates an external data relay which worked for 100% of all tested NAT devices.

We then measured the delay introduced by our framework and compared it to ICE. For both frameworks, ANTS and ICE, we used SIP for signaling and STUN to determine the public transport address. We measured on the requester from the first candidate gathering until the first byte (e.g. a TCP-SYN) was sent out. In the initial test run, the RTT to all entities (STUN, SIP, R to S) was 2ms. Each test run was repeated 50 times and the largest standard deviation for ANTS was 5.9 and for ICE 18.8 with normal deviation.

For ANTS the measurement includes the following packets: the requester queries the STUN server (2 messages) and sends the service request via SIP (1 message). The service allocates the mapping using a hole punching message (1 message), queries the STUN server (2 messages) in parallel and returns the allocated public endpoint via SIP (1 message). Thus, the overall number of messages for this constellation is 7. For ICE we used the PJNATH implementation (http://www.pjsip.org) connected to the same STUN and SIP server. Here we measured the time from the gathering of candidates until the end of the pairing process including SIP signaling.

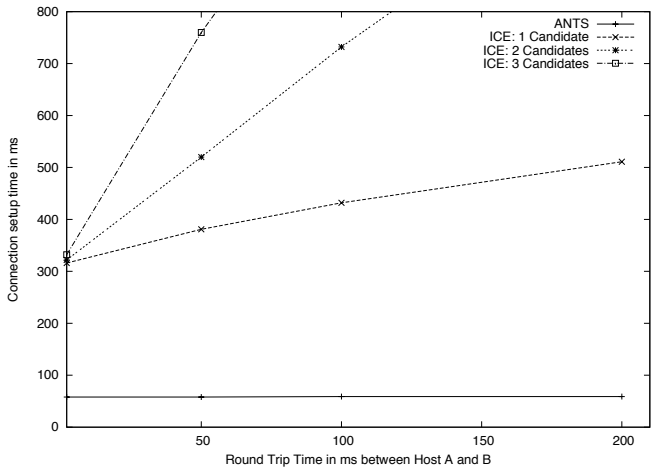

Fig. 6. Comparing the overall connection setup time of ANTS and ICE.

Conn.	Frmw.	2ms	σ	50ms	σ	100ms	σ	200ms	σ
R - S	ANTS	57,2	5,6	57,5	5,5	59,0	5,7	58,5	5,6
	ICE	316,2	12,4	381,6	14,1	432,8	11,2	511,9	13,2
R - STUN	ANTS	56,8	5,9	95,8	5,8	145,3	5,6	250,1	5,5
	ICE	315,4	14,6	534,0	13,4	968,0	14,8	1275,0	18,8
R - SIP	ANTS	58,1	4,4	109,6	5,6	157,3	5,4	257,8	4,6
	ICE	316,8	11,2	362,0	11,3	422,0	12,5	509,0	13,2

TABLE I
The Connection Setup Time (CST) in milliseconds depends on the Network Delay for the Signaling between the Hosts

Fig. 6 and Table I show the results of our measurements. Since ANTS signaling process does not include packets traveling from the requester to the service directly (only STUN and SIP is involved) the connection setup time (CST) is only dependent on the RTT to the STUN or SIP server (here we assume that both hosts use the same SIP server). Table I shows the impact of increasing the RTT on different paths. The connection setup time of ANTS can be described as follows

where $ANTS_{core}$ (XML processing, creation of mapping, packet translation) is roughly 50ms according to our test runs:

$$CST_{ANTS} = ANTS_{core} + RTT_{R-STUN} + RTT_{S-STUN} + RTT_{R-SIP} + RTT_{S-SIP} + RTT_{SIP_R-SIP_S}$$

ICE however performs STUN connectivity checks between the requester R and the service S for each candidate pair. Thus, the connection setup time is not only dependent on the RTT between R and S, but also on the number of candidates that have to be checked. Since every check consists of two STUN tests running in parallel, it increases the setup time by one RTT. Thus we can give the following formula where ICE_{core} includes the message processing and the delay to the STUN server used during the gathering process:

$$CST_{ICE} = ICE_{core+STUN} + RTT_{R-SIP} + RTT_{S-SIP} + RTT_{SIP_R-SIP_S} + candidates^2 * RTT_{R-S}$$

The measurements show the advantage of ANTS. The decoupling of knowledge gathering and connection setup dramatically decreases the number of messages and therefore the overall connection setup time while still reaching a success rate of almost 95 % for a direct connection.

VII. Related Work

Several other frameworks have been proposed to address the NAT Traversal problem. NATBLASTER [7] and NATTrav [8] both exchange public endpoints and use TCP hole punching as the only technique to traverse the NAT. In [6] and [19], the authors propose a new architecture called NUTSS. They describe the integration of a TCP NAT Traversal method called "Simple Traversal of User Datagram Protocol through NATs and TCP too" (STUNT). NUTSS uses SIP for signaling and "specially encoded host names" for identifying endpoints, whereas ANTS supports legacy IP addresses and DNS names to contact a service behind a NAT. When categorizing NUTSS into our service categories, it only supports SPPS and SSP. Furthermore, in our field test we observed a success rate of 52% for STUNT and a success rate of 95% for ANTS for a direct connection. Thus, NUTSS needs to allocate a data relay for almost 50% of all connections. The proprietary Skype (http://www.skype.com) VoIP application has a sophisticated probing technique [20] for NAT Traversal, which is not available to any other application. NAT Traversal is done using an unspecified algorithm that has similarities with the STUN and TURN protocols. If possible, a Skype node uses hole punching to establish a direct connection. If not, a super node acts as a data relay. ICE [9] aims to provide a solution flexible enough to work with all network topologies and was mainly designed for VoIP. Whenever a call is established, each phone gathers all possible transport addresses and exchanges them using SIP. Both clients then set up a local STUN server and go through the received candidate list. A client tries to connect to each candidate address using STUN binding requests and responses, resulting in working candidate pairs. Finally, ICE requires both peers to have an ICE implementation running. If one side does not, ICE cannot help.

VIII. Conclusion

This paper introduces the ANTS framework that allows a broad range of applications to benefit instantly from NAT Traversal. In contrast to related approaches it can be deployed at the service only, the requester only, or at both hosts. ANTS integrates many different NAT Traversal techniques to always utilize the best available method. Results from an extensive field test with NAT devices indicate that the implemented NAT Traversal techniques almost always enable direct connectivity. Furthermore, the ANTS approach decouples knowledge gathering about applicable NAT Traversal techniques from the connection establishment. Our experiments showed that the speedup for connection establishment in comparison with the evaluated ICE implementation is significant. Future work is to evaluate the use of a distributed signaling infrastructure (e.g. P2P-SIP) to facilitate the discovery of ANTS instances.

References

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

[2] M. Wasserman and F. Baker, "IPv6-to-IPv6 Network Address Translation (NAT66)," Internet Draft - work in progress, IETF, November 2008.

[3] G. V. de Velde et. al., "Local Network Protection for IPv6," RFC 4864, IETF, May 2007.

[4] M. Holdrege and P. Srisuresh, "Protocol Complications with the IP Network Address Translator," RFC 3027, IETF, January 2001.

[5] A. Müller, A. Klenk, and G. Carle, "Behavior and Classification of NAT devices and implications for NAT-Traversal," *IEEE Special issue on Middleboxes*, pp. 14–19, September 2008.

[6] P. Francis, S.Guha, and Y. Takeda, "NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity," In Proceedings of SIGCOMM Workshops, Portland, OR, August 2004.

[7] Andrew Biggadike et.al., "NATBLASTER: Establishing TCP connections between hosts behind NATs," in *ACM SIGCOMM Asia Workshop, Beijing, China*, 2005.

[8] J. Eppinger, "TCP Connections for P2P Applications," Carnegie Mellon University, Tech. Rep., 2005.

[9] J. Rosenberg, "Interactive Connectivity Establishment (ICE)," Internet Draft - work in progress, IETF, October 2007.

[10] Saikat Guha et. al., "NAT Behavioral Requirements for TCP," RFC 5382, IETF, October 2008.

[11] J. Rosenberg and R. Mahy et. al., "Session Traversal Utilities for NAT (STUN)," RFC 5389, IETF, October 2008.

[12] A. Müller, A. Klenk, and G. Carle, "On the Applicability of knowledge-based NAT-Traversal for future Home Networks," In Proceedings of IFIP Networking 2008, Springer, Singapore, May 2008.

[13] UPnP Forum, "Internet gateway device (IGD) standardized device control protocol," November 2001.

[14] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translation," MIT, Tech. Rep., 2005.

[15] J. Rosenberg, R. Mahy, and P. Matthews, "Traversal Using Relays around NAT (TURN)," Internet Draft - work in progress, IETF, February 2009.

[16] H. Schulzrinne et. al., "SIP: Session Initiation Protocol," RFC 3261, IETF, June 2002.

[17] J. Franks et. al., "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617, IETF, June 1999.

[18] E. P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP)," RFC 3920, IETF, October 2004.

[19] S. Guha and P. Francis, "Towards a Secure Internet Architecture Through Signaling," Cornell University, Tech. Rep., 2006.

[20] S. A. Baset and H. Schulzrinne, "An Analysis of the Skype P2P Internet Telephony Protocol," Columbia University, Tech. Rep., 2004.