Implementation and Evaluation of a NAT-Gateway for the General Internet Signaling Transport Protocol

Roland Bless and Martin Röhricht
Institute of Telematics
Karlsruhe Institute of Technology (KIT)
Zirkel 2, P.O.Box 6980, 76049 Karlsruhe, Germany
Email: {bless, roehricht}@kit.edu

Abstract—**The IETF's Next Steps in Signaling (NSIS) framework provides an up-to-date signaling protocol suite that can be used to dynamically install, maintain, and manipulate state in network nodes. In its two-layered architecture, the General Internet Signaling Transport (GIST) protocol is responsible for the transport and routing of signaling messages. The strong presence of Network Address Translation (NAT) gateways in today's Internet infrastructure causes some major challenges to signaling protocols like NSIS. The address translation mechanisms performed by common NAT gateways are primarily concerned with address information contained in the IP and transport layer headers. Messaging associations between two signaling peers do, however, rely on address information contained in GIST data units. If a non GIST-aware NAT gateway merely adapts addresses in the IP and transport headers, inconsistent state will finally be installed at the signaling nodes. In this paper we present the design, implementation, and evaluation of an *application level gateway* for the GIST protocol, that translates GIST messages in a way that allows for the establishment of messaging associations between any two GIST nodes across a NAT gateway.**

I. Introduction

Signaling protocols provide a useful set of tools to dynamically install, maintain, and manipulate state in network nodes. As a prominent example, the ReSource ReserVation Protocol (RSVP) was once designed to establish state in network routers for Quality-of-Service reservations on demand. In response to some inherent deficiencies of RSVP, the *Next Steps in Signaling* (NSIS) working group of the Internet Engineering Task Force (IETF) designed an up-to-date signaling framework that is not limited to a particular signaling application only [1]. The NSIS framework follows a two-layered architecture where the lower layer, called *General Internet Signaling Transport* (GIST) [2] protocol, is solely responsible for the routing and transport of signaling messages, whereas the upper layer, called NSIS Signaling Layer Protocol, implements the actual signaling application's logic, e.g. for Quality-of-Service resource reservations [3].

Network Address Translation (NAT) [4] was once introduced in order to mitigate the potential shortage of IPv4 addresses. NATs come in a variety of different flavors [5], mostly dealing not only with translation of IP addresses of different address realms, but also by mapping TCP or UDP transport protocol ports within a session (so called Network Address and Port Translation – NAPT, in the following we also use the term NAT for NAPT). The strong presence of NAT gateways in today's Internet infrastructure causes some major challenges to signaling protocols like NSIS. Not only that bindings must be established in these gateways in order to properly exchange messages with the actual signaling destination. Furthermore, NSIS signaling messages have to carry IP address information in their GIST payload that wouldn't be translated by an ordinary NAT gateway. Hence, in order to allow NSIS signaling sessions to be established even across NAT gateways, the NAT gateway must be GIST-aware and rewrite some of the addressing information within the signaling message's payload.

The GIST protocol specification already describes a dedicated *NAT traversal object (NTO)* that carries necessary translation information which can then be used by a GIST-aware NAT gateway. This NTO is designed in a modular way which allows for the traversal of a number of subsequent NAT gateways. It must, however, be created, inserted into a signaling message's payload, and later interpreted by a GIST-aware NAT gateway. In this paper we present the design, implementation, and evaluation of an application level gateway for the GIST protocol, which translates GIST messages in order to allow for the establishment of signaling associations between GIST nodes across a NAT gateway.

The rest of this paper is organized as follows. Section II gives an overview of GIST's protocol operation with all necessary protocol specific details and discusses related work. In Section III and IV we provide an analysis of the design and the implementation of a GIST-aware NAT gateway. Section V provides evaluations and performance measurements before we conclude in Section VI.

II. Background and Related Work

Within the Next Steps in Signaling framework the General Internet Signaling Transport (GIST) protocol is responsible to discover NSIS-capable nodes along a data flow's path, to establish messaging associations between two adjacent GIST nodes and to transport signaling messages along this route. In order to setup state between two nodes, GIST uses a three-way handshake, consisting of QUERY, RESPONSE, and CONFIRM messages. Subsequently exchanged messages from a particular signaling application are carried via GIST DATA messages.

GIST makes use of already present underlying transport protocols, like UDP, TCP, TCP with TLS, or SCTP (cf. Figure 1) and provides two modes of operation, namely a datagram mode (D-mode for UDP data) and a connection mode (C-mode for TCP and SCTP).

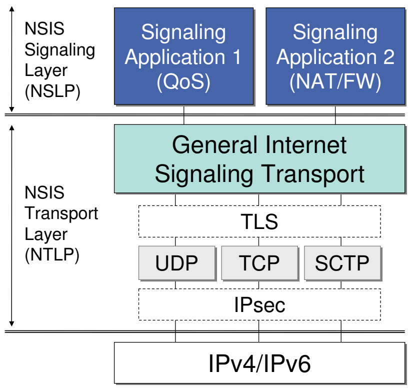

NSIS Signaling Layer (NSLP)
Signaling Application 1 (QoS)
Signaling Application 2 (NAT/FW)
NSIS Transport Layer (NTLP)
General Internet Signaling Transport
TLS
UDP
TCP
SCTP
IPsec
IPv4/IPv6

Figure 1. Layered Architecture of the NSIS Protocol Framework

QUERY messages are always sent in a so-called encapsulation mode (Q-mode) by the *Querying Node* and are intercepted by a *Responding Node*. In order to setup a messaging association between two adjacent GIST nodes, the initial QUERY contains a context-free flag (C-flag) which indicates that a new routing state must be established. The following RESPONSE indicates whether a CONFIRM must be sent by the Querying Node. In order to defend against denial of service attacks, the Responding Node can use a so-called *delayed state installation* mechanism where the installation of routing state is delayed until a final CONFIRM message arrives upon which a return routability check can be performed.

The way signaling messages are routed—e.g., strictly following the data path—is specified in a *Message Routing Method* (MRM). The MRM contains all necessary addressing information encapsulated in a *Message Routing Information* (MRI) object, e.g. the source and destination's IP addresses, as well as the transport protocol and port numbers. Multiplexing of messaging associations, i.e. the re-use of existing messaging associations for multiple flows and sessions, is controlled by a *Network Layer Information* (NLI) object. The NLI basically contains a unique peer identity and an interface address through which a signaling node can be reached. Due to the specific address information contained in the MRI and the NLI, these objects are of particular importance when it comes to address translations within a NAT gateway.

Common NAT traversal techniques, such as STUN [6] or TURN [7] do only operate on the address information contained in the IP header. Therefore, any protocol that uses address information in its payload needs to be explicitly supported by an intermediate NAT gateway. Raz et al. specified the construction of an SNMP-aware NAT gateway [8] and Han et al. proposed an application level gateway for the Session Initiation Protocol [9]. In recent work, Huang et al. even propose the use of a programmable NAT [10]. Even though different design aspects outlined in these papers are of particular interest in the context of this work, the specific solutions provided cannot directly be applied to an application level gateway for the GIST protocol.

In two Internet-Drafts Pashalidis and Tschofenig provide problem statements on a GIST NAT traversal [11] and a GIST legacy NAT traversal [12]. In order to traverse legacy NAT gateways, the authors propose the use of UDP tunnels for signaling and data traffic. However, this approach relies on static NAT bindings and does not differentiate between signaling and pure data traffic. Furthermore, the UDP tunnels add a significant level of complexity and overhead to the GIST peers. The proposal towards a GIST-aware NAT gateway on the other hand comes with a transparent and a non-transparent approach. In the transparent approach the GIST header fields are simply translated by the NAT gateway as it is done with the layer 3 and layer 4 address information. This approach allows the NAT gateways to be used completely transparent for the GIST peers participating in a signaling session, but it suffers from the restriction of not being applicable in case cryptographic protection of signaling messages is used. The non-transparent approach uses the aforementioned NAT traversal object which is included by the GIST-aware NAT gateway into initial QUERY messages and which is then subsequently echoed back by the GIST responder. This work is based on the non-transparent approach outlined in [11], but instead of storing the entire translation information in an NTO object, the translation information stored within our approach is split between the NTO object and the GIST header.

III. ANALYSIS AND DESIGN OF A GIST-AWARE NAT GATEWAY

In order to setup state for a signaling flow between two adjacent signaling peers, GIST messages must always carry addressing information in their header fields. A NAT gateway that performs address translations in IP and transport layer headers only, but not the GIST header, would create inconsistent states for signaling flows at the end-points. Furthermore, GIST handshake messages that setup state between any two signaling nodes, carry additional addressing information.

GIST-aware NAT gateways must therefore only modify signaling messages that are exchanged without any routing state installed. This applies to initial GIST QUERY messages that can be identified by the context-free flag (C = 1) in GIST's common header and subsequent RESPONSE messages that are used to set up routing state.

The GIST protocol specification [2] already introduced a so-called *NAT traversal object* that stores address information about translated objects and needs to be included in initial QUERY messages by each intermediate NAT gateway. Figure 2 depicts the object definition of an NTO. Its modular design allows to keep track of all necessary address information that has been replaced by NAT gateways along the path.

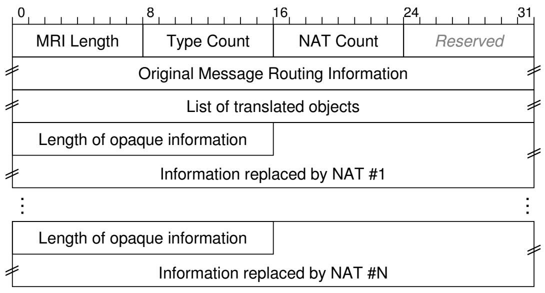

0
8
16
24
31
MRI Length
Type Count
NAT Count
Reserved
Original Message Routing Information
List of translated objects
Length of opaque information
Information replaced by NAT #1
Length of opaque information
Information replaced by NAT #N

Figure 2. Type definition of a NAT traversal object according to [2]

According to the GIST specification, a GIST-aware NAT gateway should only process QUERY messages that have the C-flag set as well as D-mode messages carrying the NAT traversal object. All remaining GIST messages, i.e., messages sent in C-mode or D-mode without NAT traversal object should be processed by the NAT gateway as ordinary data traffic. Since C-mode messages are carried inside a TCP transport connection they are not really visible to a NAT gateway anyway.

The reason for subsequent GIST messages, i.e. initial CONFIRM and subsequent DATA messages, not to be processed by the GIST-aware NAT gateway is, that messages after the initial QUERY and RESPONSE need to refer to a common MRI. Following this approach, this is the MRI of the Querying Node which must be exchanged via the NAT traversal object.

An exemplified GIST three-way handshake between a Querying Node (QN) and a Responding Node (RN) across a GIST-aware NAT gateway is illustrated in Figure 3. First of all, a GIST-aware NAT gateway must establish bindings for the signaling data flows, e.g., for subsequent C-mode signaling. Once an initial QUERY passes the gateway, it must create new MRI and NLI objects that reflect the translated address information and adds a NAT traversal object that lists all translated objects. The NTO may also carry NAT specific information that is useful for the NAT gateway, e.g., carrying state or state referral information. In case a NAT traversal object already exists, this object must be extended by additionally modified objects. After that, the message is encapsulated in Q-mode and forwarded further along the path.

The Responding Node installs routing state according to the information contained in the original MRI (QN) and the translated MRI (NAT) and NLI (NAT). The triple (MRI, NSLP-ID, Session-ID) is used as referral to routing states. The QUERY's NAT traversal object as being received by the Responding Node is copied into the RESPONSE. Furthermore, this RESPONSE's MRI uses the original and unmodified values of the Querying Node.

In any further GIST messages that cross the GIST-aware NAT gateway and that belong to a flow for which bindings already exist, only IP addresses and TCP/UDP ports are translated. Subsequently sent CONFIRM and DATA messages always carry the untranslated MRI and NLI objects of the

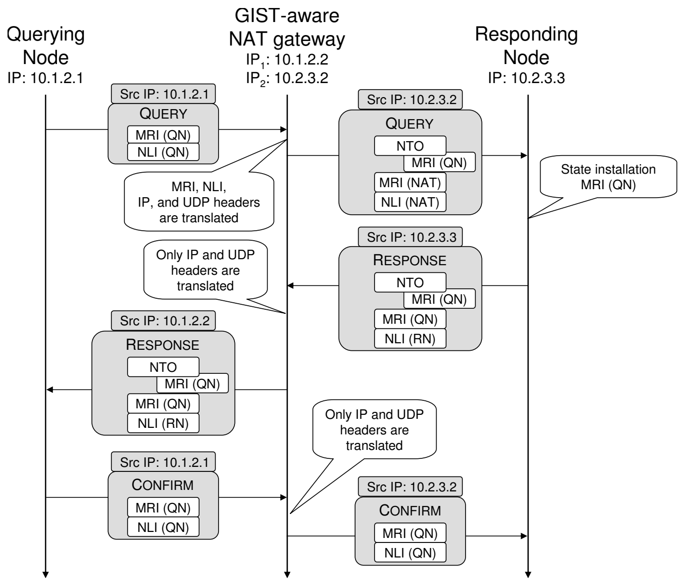


Figure 3. GIST handshake between a Querying Node (QN) and a Responding Node (RN) via a GIST-aware NAT gateway

Querying Node and are not processed by the NAT gateway with respect to addresses contained in GIST PDUs.

Special rules apply to the delayed state installation mechanism where a Responding Node does not install state before it received a final CONFIRM. As outlined above, CONFIRM messages do only carry the untranslated MRI and NLI objects, preventing the Responding Node from a correct routing state installation in this case. The GIST protocol specification leaves this issue open to the implementation. However, the specification suggests to use the Responder cookie, in which all of the translated objects that were received by the Responding Node can be carried securely. This Responder cookie is finally echoed back by the Querying Node's subsequent CONFIRM message, upon which the Responding Node receives the necessary information in order to properly install routing state.

IV. IMPLEMENTATION

In this section, we describe the design and implementation of our GIST-aware NAT gateway and explain how we perform NSIS signaling even across NAT gateways. The implementation can be basically divided into a kernel and a user-space part as depicted in Figure 4. The kernel part of the GIST-aware NAT gateway intercepts and filters GIST packets. In case a GIST packet's payload must be further modified, it is passed to a user space thread that performs the remaining packet translations before the modified packet is forwarded by the kernel.

Packet filtering is achieved in our implementation by means of the Linux netfilter framework [13]. Initial QUERY and subsequent RESPONSE messages are intercepted and passed into an `ip_queue` data structure. The communication between kernel and user space is realized on behalf of the Linux netlink messaging system [14].

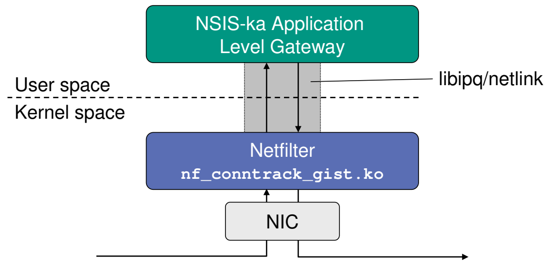


Figure 4. Architecture of an NSIS compatible Application Level Gateway

A. Implementation of the Kernel Module

The Linux netfilter framework provides a set of hooks that correspond to different positions of a packet on its way through the protocol stack. Hooks can be used to perform rules or actions, e.g., on incoming packets, on packets being forwarded, or on outgoing packets. We designed a GIST kernel module that is registered at the netfilter's *PRE_ROUTING* and *POST_ROUTING* hooks and intercepts GIST QUERY and RESPONSE messages.

Once a GIST QUERY enters the netfilter, the connection is tracked and a `conntrack` structure is initialized for a subsequent RESPONSE. In case the netfilter instance receives a RESPONSE, NAT rules are created depending on the RESPONSE message's payload. These rules can then be used to establish corresponding NAT bindings for IP and protocol port translations for any subsequent messages that do only rely on the functionality of a legacy NAT.

Initial QUERY and subsequent RESPONSE messages must, however, be further processed by the user space part of the application level gateway and are therefore passed to user space by means of the `ip_queue` data structure.

B. Implementation of the User Space Part

The user space part of the application level gateway is based on the already existing NSIS-ka implementation [15]. Note however, that it is not necessary to run the entire NSIS-ka suite on a GIST-aware NAT-gateway, so only some NTLP object classes were re-used.

A netlink listener, where messages enqueued by the kernel are received, builds the first part of the application level gateway. Once packets are received by the NSIS-ka application level gateway, the entire PDU is parsed, beginning with the IP and UDP headers, and transferred into a GIST PDU. Address information in the MRI and NLI of GIST QUERY messages must then be translated, according to the NAT processing rules. Furthermore, a NAT traversal object must be inserted right after GIST's common header and the source addressing mode flag must be set to one in a GIST QUERY. After that, the GIST PDU objects are serialized into byte code and IP and UDP checksums are re-calculated, before the packet is copied into the netlink's message buffer from where it is then sent back to kernel space.

The implementation of the kernel module consists of 420 lines of C code, whereas the GIST-aware NAT gateway consists of additional 680 lines of C++ code, but makes heavily use of already existing libraries and data structures of the NSIS-ka suite. The GIST implementation and its underlying protocol library, which we had to use for evaluation tests currently contain 40,692 physical source lines of code, mostly based on C++ (93.78%). The code of our GIST-aware NAT gateway implementation is publicly available at https://svn.ipv6.tm.uka.de/nsis/dist/nsis-ka/branches/20100602-gist-aware-nat-gw.

V. EVALUATION

We evaluated the implementation of our GIST-aware NAT gateway in a real testbed environment, consisting of four standard PCs being equipped with Intel Pentium 4 2.8 GHz CPUs, 4 GB DDR-400 RAM, and four 1000TX Ethernet cards. All four PCs ran Ubuntu 10.04 with Linux kernel 2.6.32. The topology is depicted in Figure 5 where two NSIS hosts that were equipped with the NSIS-ka framework exchanged signaling messages across two GIST-aware NAT gateways.

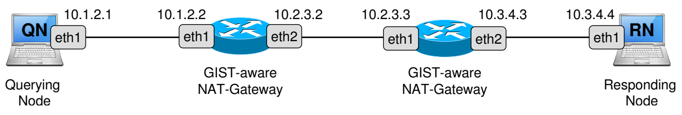


Figure 5. Evaluation Setup with two hosts communicating across two GIST-aware NAT gateways

The latency between the two endpoints was intentionally kept small (approximately 0.165 ms, measured by 100 ping tests) in order to concentrate measurements on the pure protocol and processing overhead.

First of all, we measured the time spent by each of the GIST-aware NAT gateways that were used to process QUERY, RESPONSE, CONFIRM, and DATA messages. In order to focus on GIST message processing time, the DATA messages carried only a simple artificial Echo-NSLP payload consisting of 16 additional bytes for a 132 byte long Ethernet frame. The measurement points for the GIST message processing and translation time correspond to the timestamps of tcpdump packet captures at the ingress and the egress interface.

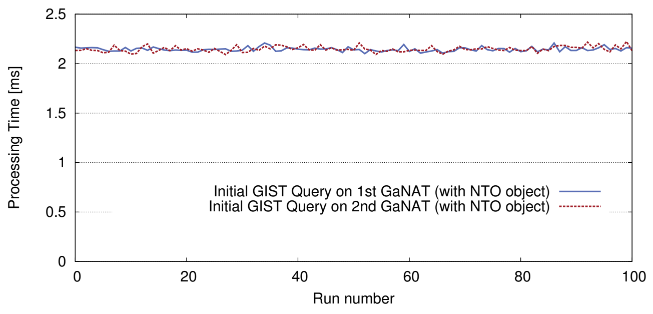
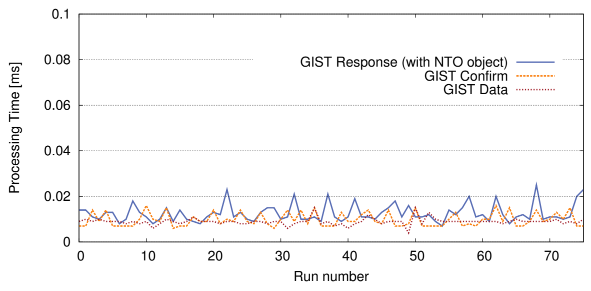
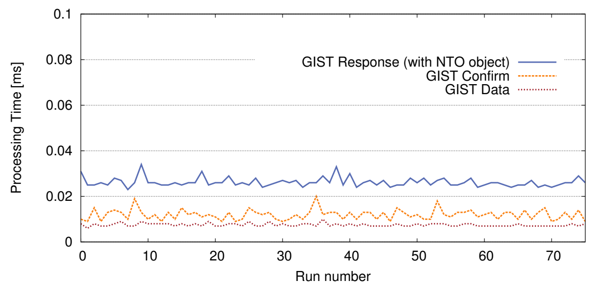


Figure 6. Processing time of initial QUERY messages when NAT traversal objects are included

Figure 6 shows the processing time for initial QUERY messages on both GIST-aware NAT gateways for 100 consecutive



(a) TCP Response, Confirm, and Data



(b) UDP Response, Confirm, and Data

Figure 7. Processing time for different GIST message types on the first GIST-aware NAT gateway

runs. As outlined above, these initial QUERY messages must be processed by a GIST-aware NAT gateway by translating MRI and NLI objects and including a new NAT traversal object that carries the original MRI of the Querying Node.

The measurement results show a fairly stable processing time of about 2.153 ms on average with a very small standard deviation of 0.15 ms (cf. Table I) and a 95% confidence interval in the range of (2.1476, 2.1542) ms. Note, that the results for the first and the second GIST-aware NAT gateway are also almost identical.

Measurement results for the processing time of 75 consecutive runs of the remaining GIST PDUs on the first GIST-aware NAT gateway are shown in Figure 7. While QUERY messages must always be sent in Q-mode encapsulation, i.e., by using UDP, subsequent GIST messages can be exchanged either via UDP or via TCP, depending on the negotiated protocol stack configuration data.

Note that the results of Figure 7 seem to indicate a rather unstable behavior, but it actually stems from the high resolution of the plotted data sets. The absolute time values for all of these three GIST message types are very small ranging from 0.008 ms and 0.026 ms on average and would otherwise not be visible compared to the processing time of initial QUERY messages.

Table I summarizes the results for the first GIST-aware NAT gateway. The small values of all standard deviations suggest a very stable behavior.

Processing time on the first GIST-aware NAT gateway	Avg [ms]	Median [ms]	StdDev [ms]
UDP Query (with NTO)	2.153	2.161	0.152
TCP Response (with NTO)	0.012	0.011	0.004
UDP Response (with NTO)	0.026	0.026	0.002
TCP Confirm	0.010	0.009	0.003
UDP Confirm	0.012	0.012	0.002
TCP Data	0.009	0.009	0.001
UDP Data	0.008	0.007	0.001

Table I
EVALUATION RESULTS FOR THE OVERALL PROCESSING TIME OF DIFFERENT GIST PDUS ON THE FIRST GIST-AWARE NAT GATEWAY

Besides measuring the pure processing costs induced on a GIST-aware NAT gateway, we also measured the duration of complete GIST handshakes between the two end points with one consecutive DATA message. We conducted tests for GIST handshakes with and without NAT gateways in-between, in order to obtain a resulting overhead. Figure 8 shows the results obtained for complete GIST handshakes in case C-mode was requested, i.e., when TCP connections were used. In this case we only picked traces from our data sets that established an entirely new TCP connection. Therefore, each trace consists of an initial GIST QUERY as starting point until the TCP acknowledgement for the first DATA message is received by the Querying Node.

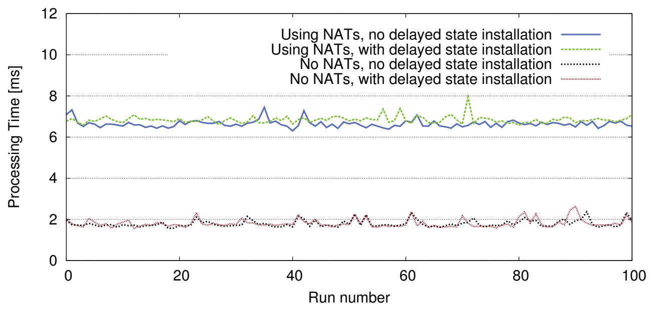


Figure 8. Duration of complete GIST handshakes with one subsequently sent DATA message measured on the Querying Node using TCP

The results are again fairly stable and show a time difference for the complete handshake duration of about 5 ms between a NAT-free setup (lower two curves) and the use of two GIST-aware NAT gateways on the path (upper two curves). Using delayed state installation induces a small processing overhead in case NAT gateways are used, but no difference can be observed by using no NAT gateways.

Figure 9 uses the same setups and measurements in case only D-mode, i.e. UDP, is used. In this case the first time stamp was again the initial QUERY message, whereas the second timestamp had to be the emitting point of the final DATA message. Again, we observe a very stable behavior and this time we can not detect a difference between using delayed state installation and using normal state installation. The GIST handshake duration is about 1 ms faster when using UDP instead of TCP.

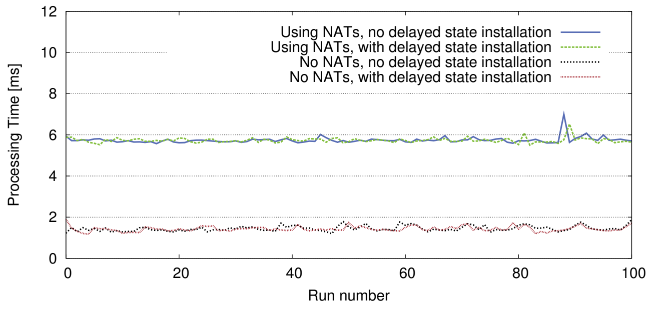


Figure 9. Duration of complete GIST handshakes with one subsequently sent DATA message measured on the Querying Node using UDP

Table II summarizes the results obtained for complete GIST handshakes for UDP and TCP connections, as well as by using delayed state installation (DSI) or by not using it.

GIST Handshake duration using TCP	Avg [ms]	Median [ms]	StdDev [ms]
Using NATs, with DSI	6.843	6.820	0.178
Using NATs, without DSI	6.659	6.630	0.182
No NATs, with DSI	1.816	1.746	0.210
No NATs, without DSI	1.797	1.732	0.176

GIST Handshake duration using UDP	Avg [ms]	Median [ms]	StdDev [ms]
Using NATs, with DSI	5.737	5.722	0.127
Using NATs, without DSI	5.744	5.720	0.154
No NATs, with DSI	1.432	1.413	0.124
No NATs, without DSI	1.449	1.407	0.136

Table II
EVALUATION RESULTS FOR THE DURATIONS OF GIST HANDSHAKES WITH ONE SUBSEQUENTLY SENT DATA MESSAGE

VI. CONCLUSIONS

In this paper we presented the design of a NAT application level gateway for the General Internet Signaling Transport protocol. Following this approach NSIS signaling messages can safely traverse such NAT gateways and routing state can be established even across NATs. The evaluation results show only a slight overhead for processing initial QUERY messages on a GIST-aware NAT gateway in the range of about 2.15 ms on average. All subsequent GIST messages show almost no processing overhead and do not exceed 0.026 ms on average.

Using GIST-aware NAT gateways has also only a small impact on the duration of complete GIST handshakes from end-to-end. While a GIST handshake and a subsequent DATA message can be exchanged in a NAT-free setup within at most 1.82 ms on average, the complete duration of a handshake with two GIST-aware NAT gateways on the path does not exceed 6.84 ms.

Furthermore, the measurement results showed that the use of GIST's delayed-state installation mechanism, which can be used as a denial-of-service attack prevention, does not induce a notable performance overhead, compared to a normal state installation.

ACKNOWLEDGMENT

We thank Yun Lin for his contributions to the conceptual elaboration of this work and the resulting implementation. Parts of this work were funded by the Federal Ministry of Education and Research of the Federal Republic of Germany (support code 01 BK 0809, G-Lab, http://www.german-lab.de/).

REFERENCES

[1] X. Fu, H. Schulzrinne, A. Bader, D. Hogrefe, C. Kappler, G. Karagiannis, H. Tschofenig, and S. V. den Bosch, "NSIS: A New Extensible IP Signaling Protocol Suite," *Communications Magazine, IEEE*, vol. 43, no. 10, pp. 133–141, Oct. 2005.

[2] H. Schulzrinne and R. Hancock, "GIST: General Internet Signalling Transport," http://tools.ietf.org/id/draft-ietf-nsis-ntlp, IETF, Jun. 2009, Internet Draft draft-ietf-nsis-ntlp-20.

[3] J. Manner, G. Karagiannis, and A. McDonald, "NSLP for Quality-of-Service Signaling," http://tools.ietf.org/id/draft-ietf-nsis-qos-nslp, IETF, Jan. 2010, Internet Draft draft-ietf-nsis-qos-nslp-18.

[4] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022 (Informational), Internet Engineering Task Force, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt

[5] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787 (Best Current Practice), Internet Engineering Task Force, Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4787.txt

[6] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," RFC 5389 (Proposed Standard), Internet Engineering Task Force, Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5389.txt

[7] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," RFC 5766 (Proposed Standard), Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5766.txt

[8] D. Raz, J. Schoenwaelder, and B. Sugla, "An SNMP Application Level Gateway for Payload Address Translation," RFC 2962 (Informational), Internet Engineering Task Force, Oct. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2962.txt

[9] J. C. Han, W. Hyun, S. O. Park, I. J. Lee, M. Y. Huh, and S. G. Kang, "An Application Level Gateway for Traversal of SIP Transaction through NATs," in *Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference*, vol. 3, Feb. 2006.

[10] T.-C. Huang, S. Zeadally, N. Chilamkurti, and C.-K. Shieh, "A Programmable Network Address Translator: Design, Implementation, and Performance," *ACM Transactions on Internet Technology*, vol. 10, no. 1, pp. 1–37, 2010.

[11] A. Pashalidis and H. Tschofenig, "GIST NAT Traversal," http://tools.ietf.org/id/draft-pashalidis-nsis-gimps-nattraversal, IETF, Jul. 2007, Internet Draft draft-pashalidis-nsis-gimps-nattraversal-05.

[12] ——, "GIST Legacy NAT Traversal," http://tools.ietf.org/id/draft-pashalidis-nsis-gist-legacynats, IETF, Jul. 2007, Internet Draft draft-pashalidis-nsis-gist-legacynat-02.

[13] P. McHardy, H. Welte, J. Kadlecsik, M. Josefsson, Y. Kozakai, and P. N. Ayuso, "Firewalling, NAT, and Packet Mangling for Linux," Jun. 2010. [Online]. Available: http://www.netfilter.org/

[14] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, "Linux Netlink as an IP Services Protocol," RFC 3549 (Informational), Internet Engineering Task Force, Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3549.txt

[15] Institute of Telematics, "NSIS-ka – A free C++ implementation of NSIS protocols," Jun. 2010. [Online]. Available: http://nsis-ka.org/