Scalable Computing: Practice and Experience
Volume 11, Number 4, pp. 329–343. http://www.scpe.org
ISSN 1895-1767

IMPLEMENTATION AND EVALUATION OF A NAT-GATEWAY FOR THE GENERAL INTERNET SIGNALING TRANSPORT PROTOCOL*

ROLAND BLESS† AND MARTIN RÖHRICHT†

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 network 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. Signaling sessions 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 only, 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 to establish signaling sessions between any two GIST nodes across a NAT gateway.

Key words: distributed systems, Internet, middleware, performance evaluation

1. Introduction. Network layer resource 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 limitations 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 Signalling 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 to map non-publicly usable ("private") addresses to public IP addresses. NATs mostly deal not only with the translation of IP addresses of different address realms, but also with the mapping of 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). Due to this additional port multiplexing, one public IP address can serve many private IP addresses and thus mitigate the potential shortage of IPv4 addresses. But the price is lost end-to-end transparency [8] complicating the design and life of transport or application protocols. Main problems are the missing address binding information for communication initiated from the public side of the NAT gateway as well as the IP address information contained in IP payloads. For the latter so-called Application Level Gateways (ALGs) inside the NAT try to help translating the application specific address information in the payload. Their implementation is, however, sometimes error-prone, immutable and does not cope with newer protocol versions. Moreover, a variety of different NAT types exist [5] that cause even more complications due to their different behavior, aside from configurations with nested NATs. One solution to prevent the use of ALGs is to avoid putting IP address information in the payload, which is, however, not always possible.

In this sense, the strong presence of NAT gateways in today's Internet infrastructure causes some major challenges to signaling protocols like such of the NSIS suite. Since NSIS protocols are controlling resources in the network layer, they have to carry IP address information similar to some transport protocols. Not only that bindings must be established in these gateways to properly exchange messages with the actual signaling destination. Furthermore, NSIS signaling messages have to carry IP address information in their GIST payload that would not 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.

*This work was supported by the Federal Ministry of Education and Research of the Federal Republic of Germany (support code 01 BK 0809, G-Lab, http://www.germanlab.de/. This is an extended and reorganized version of a paper of the same title that appears in the Proceedings of HPCC, Melbourne, Australia, September 2010, pp. 659-664

†Institute of Telematics, Karlsruhe Institute of Technology (KIT), Zirkel 2, P.O.Box 6980, 76049 Karlsruhe, Germany ({bless, roehricht}@kit.edu)

The GIST protocol specification [2] 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 2 gives an overview of GIST's protocol operation with all necessary protocol specific details and discusses related work. Section 3 illustrates GIST handshake message exchange procedures across a legacy NAT gateway. In Section 4 and 5 we provide an analysis of the design and the implementation of a GIST-aware NAT gateway. Section 6 provides evaluations and performance measurements before we conclude in Section 7.

2. Background and Related Work. Different from other signaling protocols, such as RSVP, the Next Steps in Signaling framework follows a two-layered architecture as depicted in Figure 2.1. This two-layer split allows for a flexible and extensible Internet signaling protocol suite, where the routing and transport of signaling messages is separated from the actual signaling application's logic.

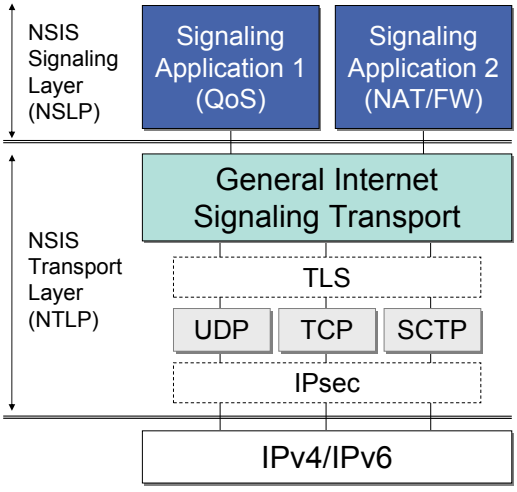

FIG. 2.1. *Layered Architecture of the NSIS Protocol Framework*

The NSIS framework therefore consists of a lower layer called *NSIS Transport Layer Protocol* (NTLP), whereas the signaling application is realized via specific *NSIS Signaling Layer Protocols* (NSLP).

The General Internet Signalling Transport (GIST) protocol is a concrete realization of an NTLP. It makes use of already present underlying transport protocols, like UDP, TCP, TCP with TLS, or SCTP 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). GIST 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 as depicted in Figure 2.2. Subsequently exchanged messages from a particular signaling application are carried via GIST DATA messages (not shown in Figure 2.2).

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*

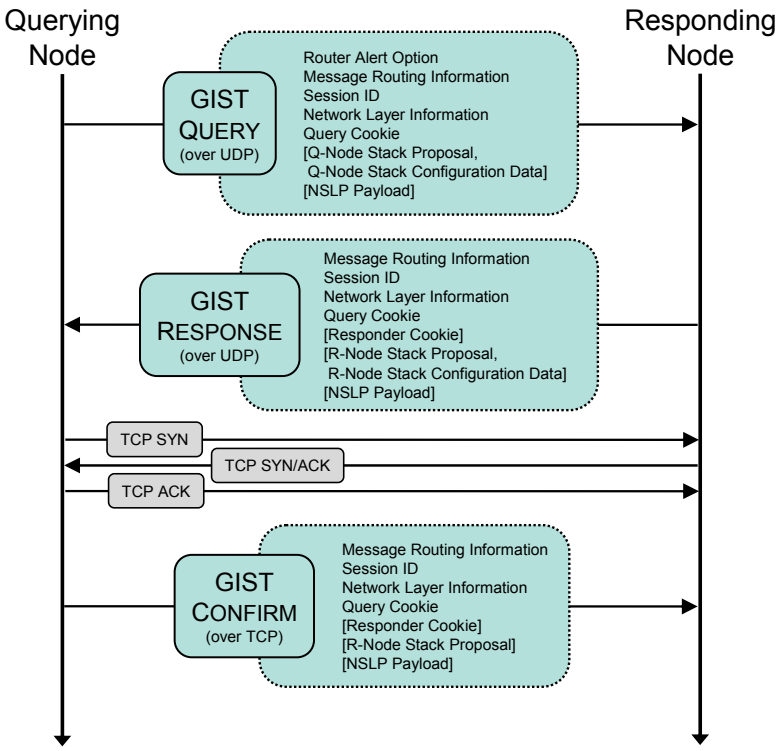

FIG. 2.2. *A detailed view of the message sequence of a GIST handshake for initial state setup, including a TCP-based messaging association*

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] require the presence of supporting servers and help applications to find out information about address translations so that they can put the correct address information into the payload. The dependency on other server and control protocols for network layer signaling protocols like NSIS is inappropriate at best. For instance, Raz et al. specified the construction of an SNMP-aware NAT gateway [9]. In recent work, Huang et al. even propose the use of a programmable NAT [11]. 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 [12] and a GIST legacy NAT traversal [13]. 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. However, if the signaling traffic is cryptographically protected between two GIST peers residing on either side of the NAT gateway and the NAT gateway is not actively participating as an NSLP entity (therefore terminating the cryptographic protection), the transparent approach cannot be applied. As cryptographic protection of NSIS signaling traffic is realized

either via IPsec or via TLS, signaling messages are encrypted above layers 3 or 4 and hence GIST data units cannot be modified following the transparent approach by a NAT gateway anymore.

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. Different from the transparent approach discussed above, the non-transparent approach does not suffer from cryptographic protection of signaling traffic as it only affects the initial QUERY and the corresponding RESPONSE message which must always both be sent unencrypted. The work presented in this paper is therefore based on the non-transparent approach outlined in [12], 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 Responder Cookie.

3. Analysis of Legacy NAT Handling. In order to setup state for a signaling flow between two adjacent NSIS signaling peers, GIST messages always carry addressing information in their header fields. However, once a GIST signaling message reaches a legacy NAT gateway, this gateway will only be concerned with address translations within the IP and UDP/TCP headers. In order to get a better understanding of the requirements of a GIST-aware NAT gateway (c.f. Section 4), this section provides an analysis of GIST's handshake message exchange across legacy NAT gateways.

The GIST specification introduced the notion of a *source addressing mode flag* (S-flag) in its common header which can optionally be used to indicate whether the flow source address corresponds to the signaling source address. By default, the S-flag is set to 0 but can also be used for legacy NAT detection during a GIST handshake in which case the S-flag must be set to 1. Depending on the locations of the NAT gateway and the Querying Node (QN) we must consider two different signaling message proceedings. The QN can either be located on the internal or the external side of the NAT gateway.

3.1. Querying Node on the Internal Side of the NAT Gateway. Figure 3.1 illustrates the case in which the QN is located on the internal side of the NAT gateway and the S-flag is set to 0. In this case, the QN is equipped with an IP address of a private subnet (10.1.2.1 in Figure 3.1) and sends an initial QUERY towards a signaling endpoint with IP address 141.3.71.56 in downstream direction. Therefore, the MRI of the QN holds the IP address of the QN as its source address and the IP address of the signaling endpoint as its destination address. The NLI of the QN is equipped with a unique interface peer-identifier of the QN and the corresponding interface address (in this case: 10.1.2.1).

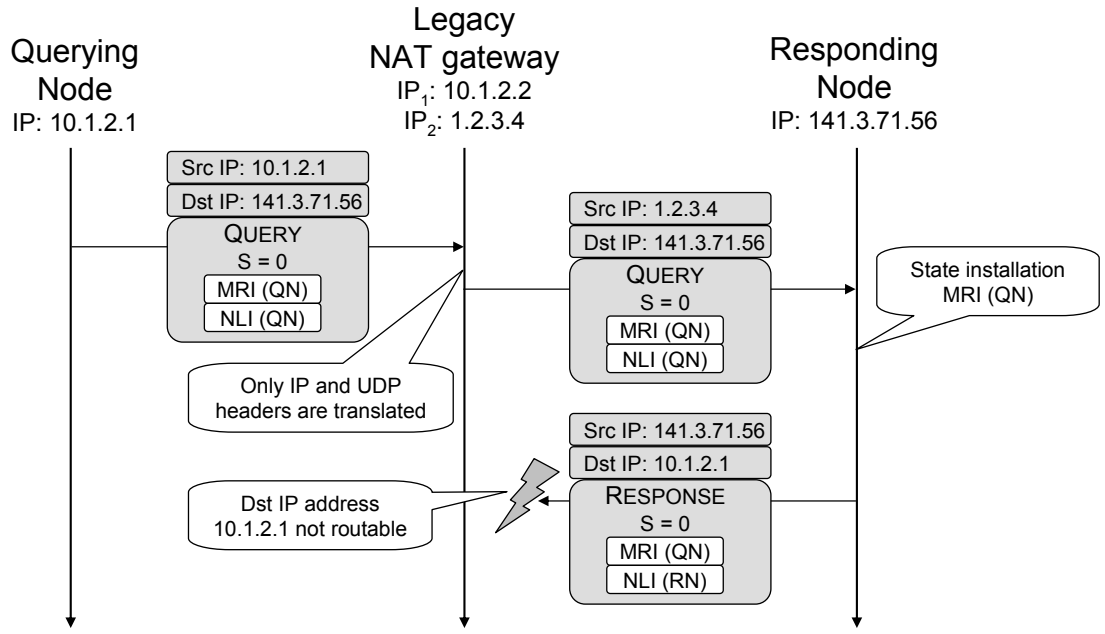

FIG. 3.1. *Legacy NAT traversal of a GIST handshake message exchange in case the Querying Node is on the* internal *side of the NAT gateway and the S-flag is not set (S = 0)*

The QUERY eventually reaches the legacy NAT gateway which only performs IP and UDP header address translations and is then forwarded towards the signaling destination endpoint. The Responding Node (RN) intercepts the message in order to establish a messaging association with the QN and therefore installs state according to the information contained in the MRI (QN). The RN generates a RESPONSE depending on the interface address given in the NLI (QN). This RESPONSE is, however, not routable and hence will not be delivered to the QN. The QN will then try to retransmit QUERY messages (not shown in the figure) and will eventually time out in case retransmitted QUERY messages keep S = 0.

In case a QUERY is (re-)transmitted with S = 1, the RN must check the source address provided in the IP header with the interface address in the NLI. As the address information contained in these two fields does not match, a legacy NAT can be assumed to be located on the path. Message routing methods, such as the default path-coupled MRM, need corresponding address translations which cannot be provided by legacy NAT gateways. As depicted in Figure 3.2, the RN returns an *Object Type Error* message including the MRI (QN) within its error subcode, indicating that no correct messaging association can be established based on this MRI, upon which the handshake message exchange is terminated.

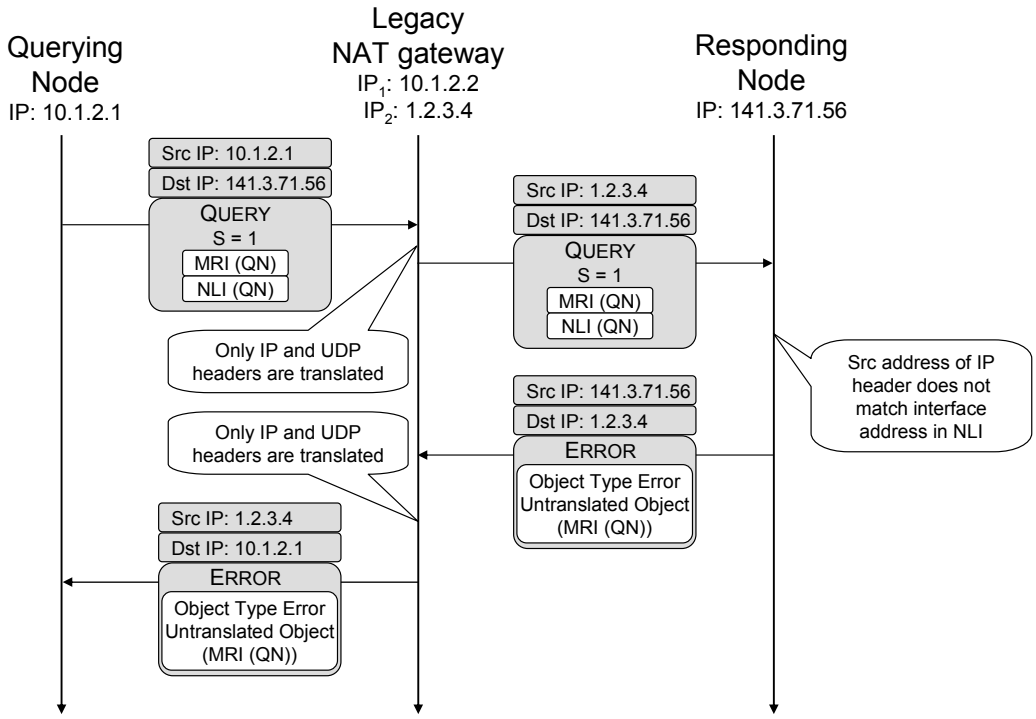

FIG. 3.2. *Legacy NAT traversal of a GIST handshake message exchange in case the Querying Node is on the* internal *side of the NAT gateway and the S-flag is set (S = 1)*

According to the GIST specification, the error message should use the destination address of the original IP datagram as its source address in the IP header. This makes the response more likely to be accepted by the NAT gateway and finally be forwarded towards the QN. In case this error message is blocked by the NAT, the QN will further retransmit QUERY messages until it eventually times out.

3.2. Querying Node on the External Side of the NAT Gateway. When the legacy NAT gateway is located on the side of the RN instead of the QN, GIST messages may only traverse the gateway in case static configurations have been carried out on the NAT gateway. As illustrated in Figure 3.3 we assume the QN to be equipped with a public IP address (1.2.3.4), whereas the RN has an IP address from the NAT gateway's private subnet (10.1.2.2).

The initial QUERY traverses the NAT gateway where IP and UDP header addresses are translated. Once this modified data packet reaches the RN, it sends a RESPONSE with S = 1 and an IP address derived from the NLI (QN), i. e., 1.2.3.4 in this case, back to the QN. The legacy NAT gateway forwards the RESPONSE as ordinary UDP traffic and only translates IP and UDP header information.

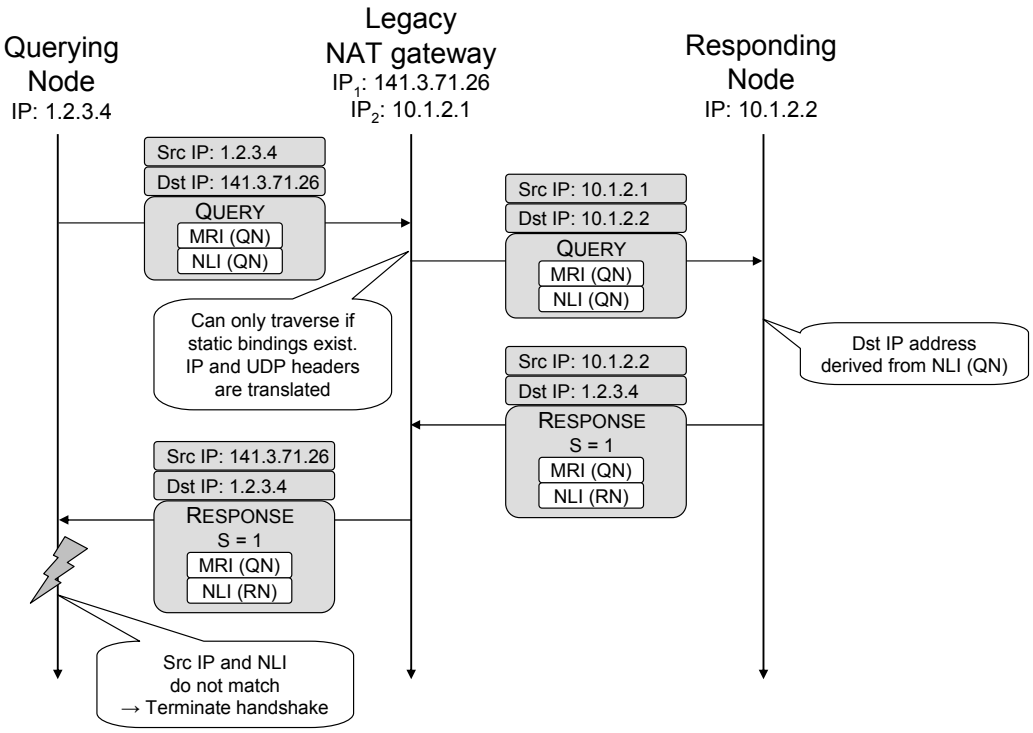

FIG. 3.3. *Legacy NAT traversal of a GIST handshake message exchange in case the Querying Node is on the* external *side of the NAT gateway and the S-flag is set (S = 1)*

Once the RESPONSE reaches the QN, it detects a mismatch between the IP header's source address (141.3.71.26 in Figure 3.3) and the NLI(RN) (10.1.2.2 in Figure 3.3) upon which the QN terminates the GIST handshake.

This section revealed that a legacy NAT gateway will always terminate a GIST message handshake and thus prevents GIST messaging associations to be set up, no matter if the legacy NAT gateway is located on the side of the Querying Node or on the side of the Responding Node. Therefore, NSIS signaling cannot be performed across a legacy NAT gateway.

4. Analysis and Design of a GIST-aware NAT Gateway. As outlined above, a legacy NAT gateway that performs address translations in the IP and transport layer headers only, but not in the GIST header, creates inconsistent states for signaling flows at the end-points which eventually leads to a termination of the signaling session at an early stage. In order for NSIS signaling to install routing state even across NAT gateways, such NAT gateways must be GIST-aware.

Note, however, that GIST-awareness does not mean that the NAT gateway must have the entire NSIS framework or any of the particular NSLPs installed. Instead, a GIST-aware NAT gateway must only modify GIST signaling messages that are exchanged without any routing state installed.

The GIST protocol specification [2] already introduced a so-called *NAT traversal object* (NTO) that stores address information about translated objects and needs to be included in initial QUERY messages by each intermediate NAT gateway. Figure 4.1 depicts the object definition of an NTO.

The modular design of an NTO allows to keep track of all necessary address information that has been replaced by NAT gateways along the path. The most important part of this object is the *Original Message Routing Information* element of the QN which is used among the signaling peers as referral to a common routing state. Furthermore, it contains a field where the number of NATs is stored that were traversed by the message and a list of translated objects which is of variable length and contains 16-bit wide fields for corresponding objects that have also been translated by intermediate NAT gateways. In particular, the Network Layer Information (NLI) object, which contains the outgoing interface address and peer-identity, as well as a modified Stack-Configuration-Data element, if present, should be included into the list of translated objects.

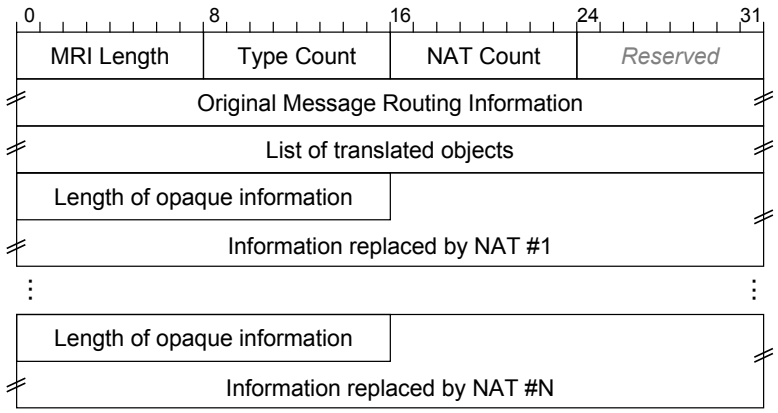

FIG. 4.1. *Type definition of a NAT traversal object according to [2]*

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 as outlined above.

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 4.2. 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 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.

5. 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 5.1. 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,

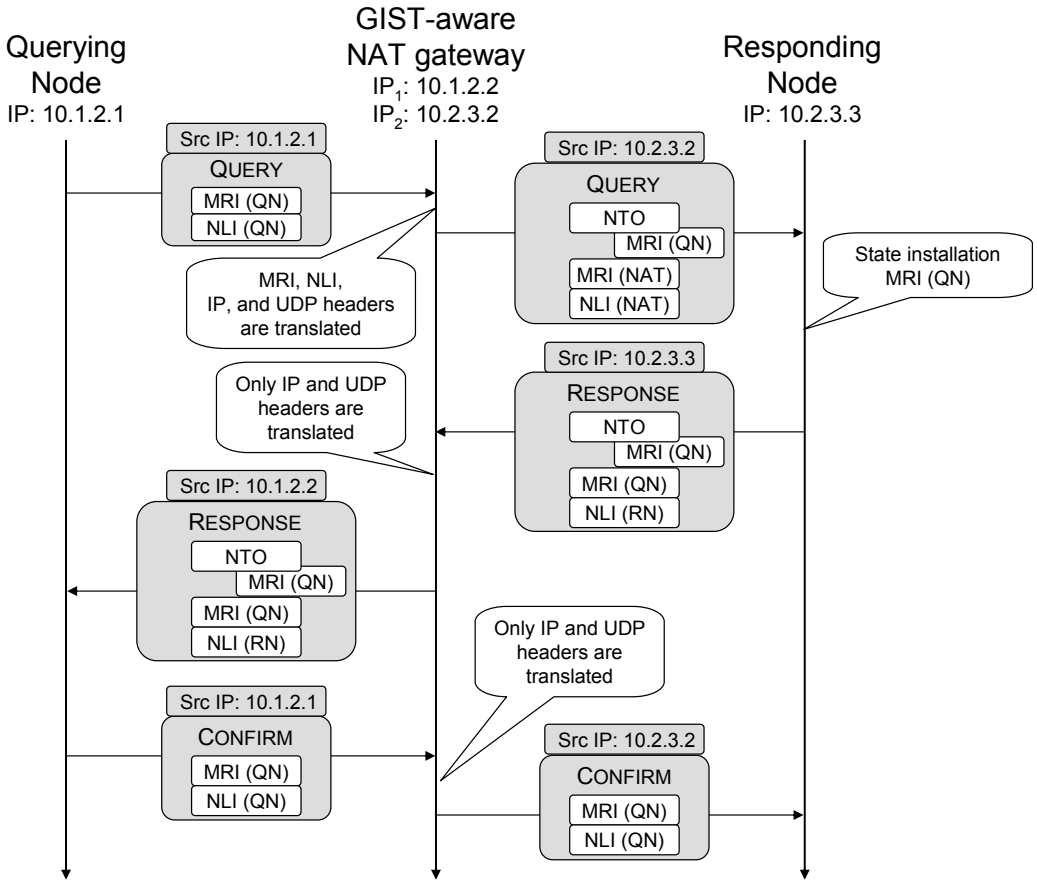

FIG. 4.2. *GIST handshake between a Querying Node (QN) and a Responding Node (RN) via a GIST-aware NAT gateway*

it is passed to a user space thread that performs the remaining packet translations before the modified packet is forwarded by the kernel.

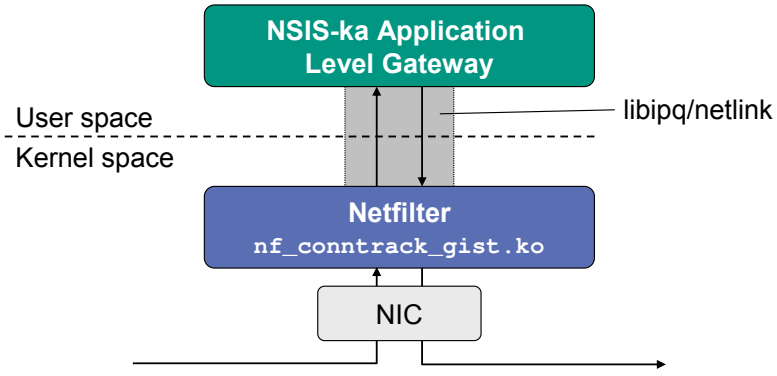

FIG. 5.1. *Architecture of an NSIS compatible Application Level Gateway*

Packet filtering is achieved in our implementation by means of the Linux netfilter framework [14]. 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 [15].

5.1. 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 registers itself 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.

5.2. Implementation of the User Space Part. The user space part of our application level gateway is based on the already existing NSIS-ka implementation [16]. 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 can be 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.

5.3. Interaction between Kernel and Userspace Parts. The resulting work flow of the kernel and userspace parts of the implementation are illustrated in Figure 5.2. Once a packet enters the kernel and its netfilter hooks, the implementation inspects whether the data packet is a valid GIST message, i. e., if the UDP datagram is addressed towards the GIST well-known port 270 and if the first 32 bits of the UDP payload match the specified GIST magic number (`0x4e04bda5`). In case it is a GIST message and it is an initial QUERY (`ip_conntrack_info` set to `IP_CT_NEW`) or a corresponding RESPONSE (`ip_conntrack_info` set to `IP_CT_IS_REPLY`) the data packet will be further processed, otherwise it is forwarded as normal data traffic. An initial QUERY is passed to the user space part via the `NF_QUEUE` return code.

After that the kernel module tests for an NTO in the GIST message. In case it is a RESPONSE and an NTO is present, the connection between the sender and the receiver has already been established. The kernel module then needs to define how subsequent traffic should be handled for this connection and stores the necessary address information in the RESPONSE. Finally the kernel module needs to ensure that the `iptables` rules for the network address translations were updated accordingly and passes the packet on to the user space part.

The user space part uses the API of the `libipq` library to read messages queued in `ip_queue` from kernel space. First of all it parses the packet to see if the Router Alert Option is set. After that, the UDP header is parsed for corresponding port information. In case the GIST message is a QUERY, the source address information contained in the IP and UDP header must be later used for translation. Then the UDP payload is deserialized and parsed, based on existing classes and functions from the NSIS-ka framework, in order to retrieve the corresponding GIST objects. The results are stored in a dedicated peer address structure.

The ALG translation takes one of the most important parts in this process, as GIST objects like the MRI and the NLI must be translated with the addresses stored in the peer address structure. After that, an NTO must be created and inserted in case the packet is a QUERY that does not yet have an NTO. Furthermore, the original NLI is stored in the NTO's opaque information field. If the QUERY already has an NTO, the NAT count is increased by one and the translated types of the NLI and the stack proposal are added to the opaque information field. In case it is a RESPONSE, the NAT count is decreased by one.

Once the ALG processing is done, the GIST message must be serialized into byte-code by means of the NSIS-ka framework, the S-flag and R-flag must be explicitly set, and the GIST hop count must be increased by one in the GIST common header. For a QUERY the C-flag must also be set. Then the IP and UDP checksums must be re-calculated and the `libipq` must be used again in order to send the packet back to kernel space and from there further along the path on the network.

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

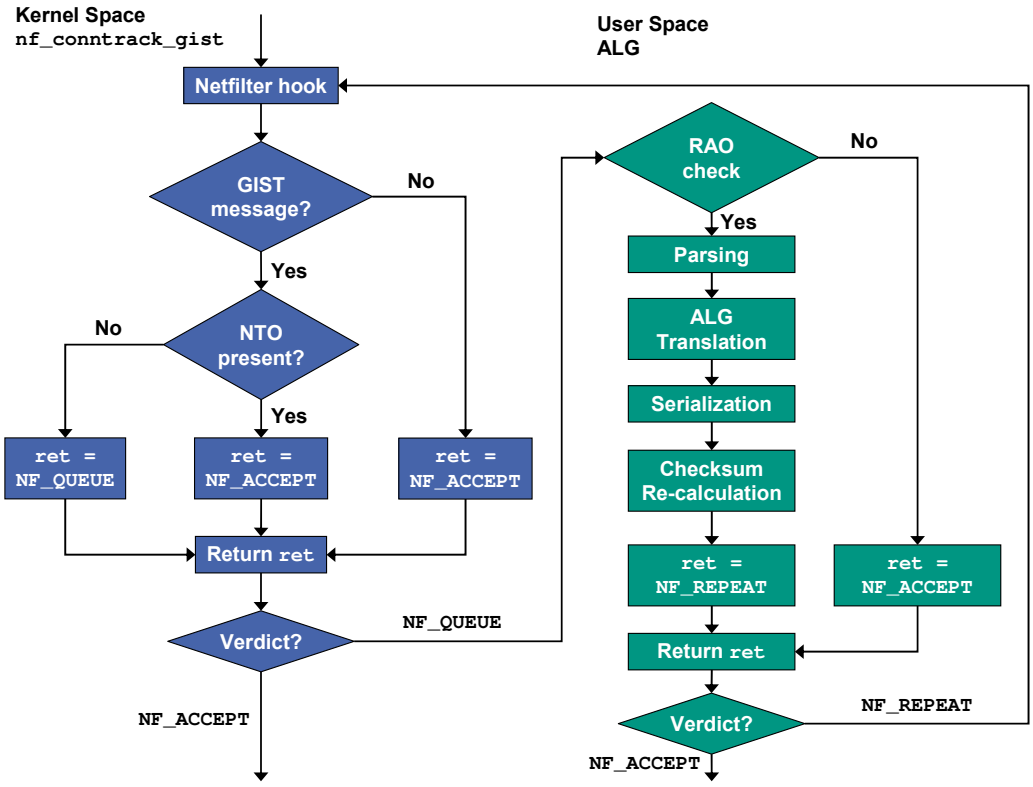

FIG. 5.2. *Illustration of the different processing steps in a GIST-aware NAT gateway between the kernel module and the userspace part*

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.tm.kit.edu/nsis/dist/nsis-ka/branches/20100602-gist-aware-nat-gw`.

6. 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 6.1 where two NSIS hosts that were equipped with the NSIS-ka framework exchanged signaling messages across two GIST-aware NAT gateways. For evaluation purposes, we assigned IP addresses from the private 10.0.0.0/24 subnet to the PC's interface cards and set up static routing tables accordingly.

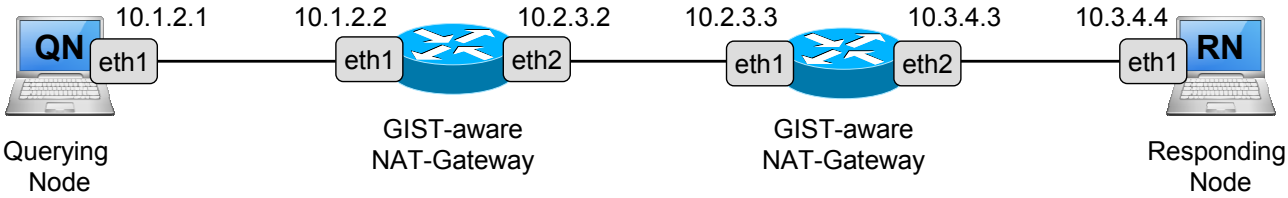

FIG. 6.1. *Evaluation Setup with two hosts communicating across two GIST-aware NAT gateways*

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.

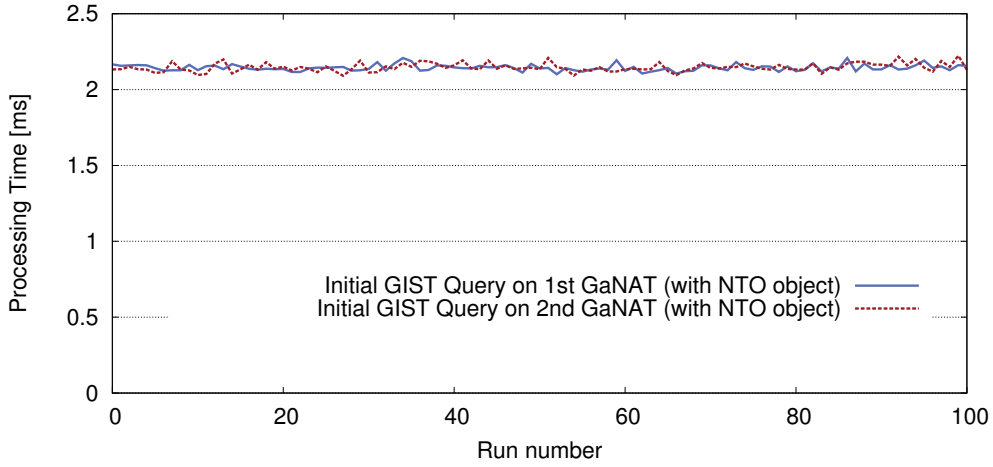

FIG. 6.2. *Processing time of initial* QUERY *messages when NAT traversal objects are included*

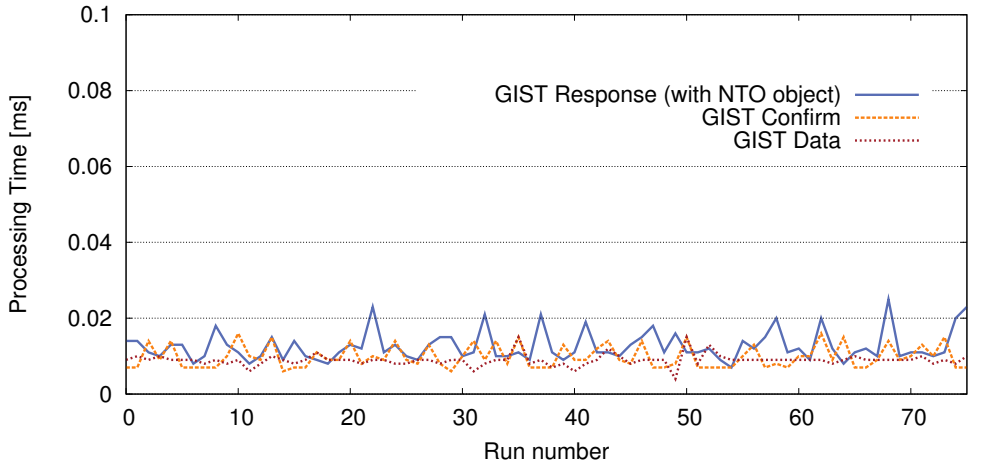

FIG. 6.3. *Processing time of GIST* RESPONSE, CONFIRM *and* DATA *messages on the first GIST-aware NAT gateway using TCP*

Figure 6.2 shows the processing time for initial QUERY messages on both GIST-aware NAT gateways for 100 consecutive 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 6.1) 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 Figures 6.3 and 6.4. 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 Figures 6.3 and 6.4 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.

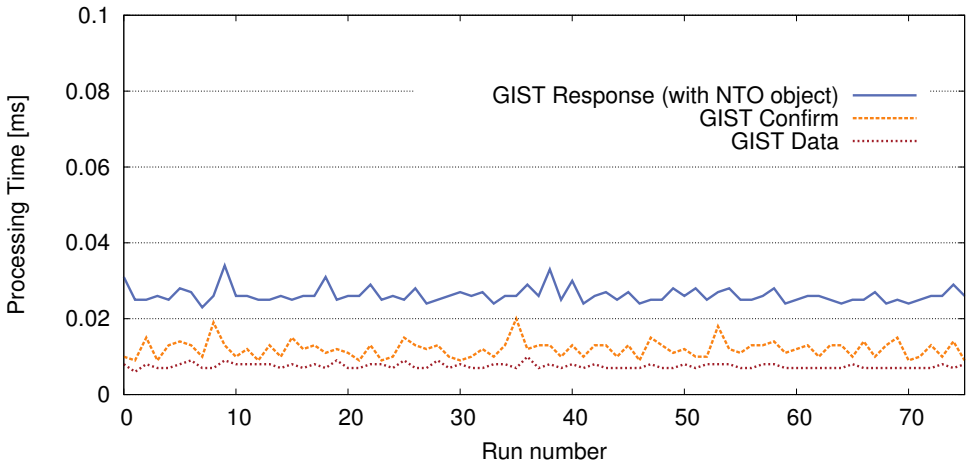

FIG. 6.4. *Processing time of GIST* RESPONSE, CONFIRM, *and* DATA *messages on the first GIST-aware NAT gateway using UDP*

Table 6.1 summarizes the results for the first GIST-aware NAT gateway. The small values of all standard deviations suggest a very stable behavior.

TABLE 6.1
Evaluation results for the overall processing time of different GIST PDUs on the first GIST-aware NAT gateway

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

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 6.5 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.

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 6.6 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 cannot 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.

Table 6.2 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.

We note that the results obtained for complete GIST handshakes using our evaluation setup are perfectly in-line with the results obtained for the processing overhead of single GIST PDUs. As outlined above, a UDP QUERY induces an overhead of about 2.15 ms on average, whereas subsequent RESPONSE, CONFIRM, and DATA

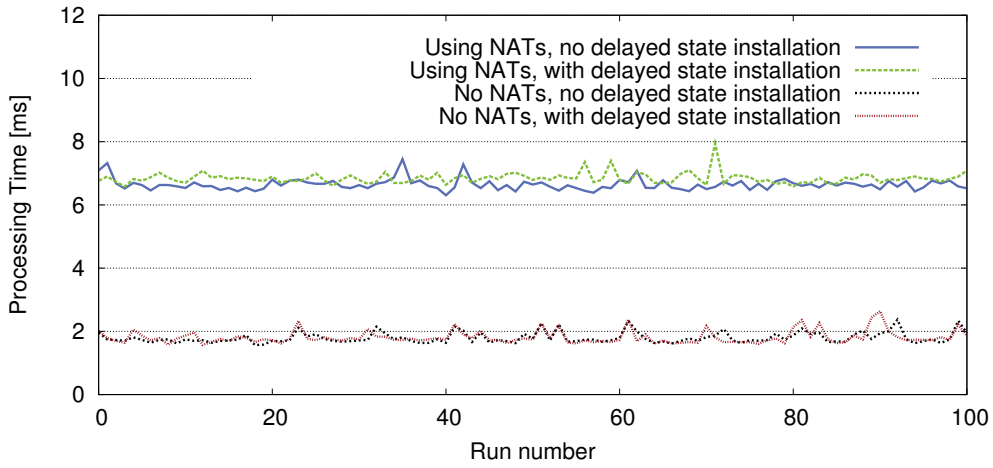

FIG. 6.5. *Duration of complete GIST handshakes with one subsequently sent* DATA *message measured on the Querying Node using TCP*

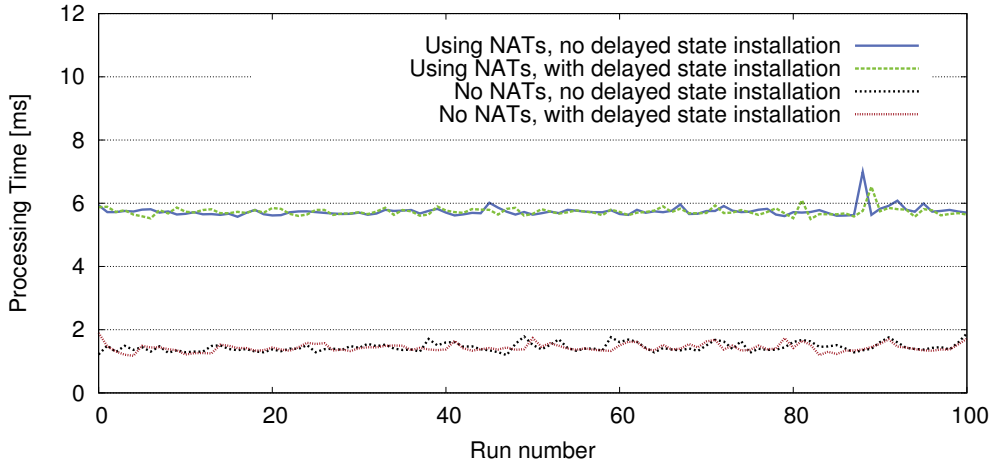

FIG. 6.6. *Duration of complete GIST handshakes with one subsequently sent* DATA *message measured on the Querying Node using UDP*

TABLE 6.2
Evaluation results for the durations of GIST handshakes with one subsequently sent DATA *message*

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

messages do only produce an overhead of (0.012 + 0.010 + 0.009) ms in case a TCP connection is used and (0.026 + 0.012 + 0.008) ms in case UDP is used (c.f. Table 6.1). As our setup uses two GIST-aware NAT gateways on the path, the plain processing overhead sums up to 2·2.15 ms+2·(0.012+0.010+0.009) ms = 4.3 ms+2·0.062 ms = 4.362 ms for a TCP connection and $2 \cdot 2.15\,\text{ms} + 2 \cdot (0.026 + 0.012 + 0.008)\,\text{ms} = 4.3\,\text{ms} + 2 \cdot 0.046\,\text{ms} = 4.392\,\text{ms}$ for UDP connections.

7. 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 design of the GIST-aware NAT gateway followed mostly the proposed design outlined in the GIST specification. This showed again how a design of a formal specification can be directly adopted in real-world implementations if it was well-defined.

The evaluation results of our implementation 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. The processing time of GIST messages could even be further lowered if the implementation runs in kernel-mode or is supported by the hardware chip itself. However, as NAT gateways are commonly used only at the very edge of the network, we would not expect the gateway to handle a big amount of concurrent signaling sessions.

Even though the authors do in general not support the idea of NATs—as they break the end-to-end principle of the Internet architecture—we note that the implementation of the presented approach follows only some basic design mechanisms, which is quite positive. Therefore, an application-level gateway support for NSIS signaling traffic may easily be incorporated into existing NAT implementations, especially the ones which are already based on Linux. Future research in this area may be directed towards the support of SCTP as an alternative transport layer protocol as opposed to UDP or TCP, as SCTP is currently, to the best of our knowledge, not supported by existing NAT gateways.

Acknowledgments. The authors thank Yun Lin for his contributions to the conceptual elaboration of this work and the resulting implementation.

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,* RFC 5971 (Experimental) Internet Engineering Task Force, Oct. 2010. [Online]. Available: `http://www.ietf.org/rfc/rfc5971.txt`

[3] J. Manner, G. Karagiannis, and A. McDonald, *NSLP for Quality-of-Service Signaling,* RFC 5974 (Experimental) Internet Engineering Task Force, Oct. 2010. [Online]. Available: `http://www.ietf.org/rfc/rfc5974.txt`

[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] B. Carpenter, *Internet Transparency,* RFC 2775 (Informational), Internet Engineering Task Force, Feb. 2000. [Online]. Available: `http://www.ietf.org/rfc/rfc2775.txt`

[9] 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`

[10] 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, Phoenix Park, Gangwon-Do, Republic of Korea, Feb. 2006.

[11] 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.
[12] A. Pashalidis and H. Tschofenig, *GIST NAT Traversal,* IETF, Jul. 2007, Internet Draft draft-pashalidis-nsis-gimps-nattraversal-05, `http://tools.ietf.org/id/draft-pashalidis-nsis-gimps-nattraversal`.
[13] ———, *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.
[14] 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/`
[15] 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`
[16] Institute of Telematics, *NSIS-ka—A free C++ implementation of NSIS protocols,* Dec. 2010. [Online]. Available: `http://nsis-ka.org/`

Edited by: Jemal H. Abawajy, Mukaddim Pathan, Al-Sakib Khan Pathan, and Mustafizur Rahman
Received: October 11th, 2010
Accepted: November 10th, 2010