Implementation and Performance Study of a New NAT/Firewall Signaling Protocol

Niklas Steinleitner, Henning Peters, Xiaoming Fu
Institute for Informatics, University of Göttingen, Germany
Email: {steinleitner,hpeters,fu}@cs.uni-goettingen.de

and Hannes Tschofenig
Siemens AG, Munich, Germany, Email: hannes.tschofenig@siemens.com

Abstract

The NAT/Firewall NSIS Signaling Layer Protocol (NAT/Firewall NSLP) is a path-coupled signaling protocol for explicit Network Address Translator and firewall configuration within an extensible IP signaling framework currently being developed by the IETF Next Steps in Signaling (NSIS) working group. This new protocol allows end hosts to signal along a path to configure NATs and firewalls according to the data flow needs. In this paper we present a first open source implementation and performance evaluation of NAT/Firewall NSLP. The performance study shows that our implementation scales well and is able to support firewall signaling for up to tens of thousands of flows in parallel even in a low-end PC testbed environment. The overall performance bottleneck is found to lie in the utilized firewall implementation, not depending on the NAT/Firewall NSLP implementation.

1 Introduction

Middleboxes, such as firewalls (FW) and Network Address Translators (NAT), have been used throughout the Internet for many years and it is expected that they will remain present for a foreseeable future [1]. Firewalls are used to protect networks against certain types of threats, and NATs provide, among restricting inbound connections, a virtual extension of the depleting IP address space.

In most instances both types of devices only allow inbound traffic generated and destined to desired entities specified before. These entities have to follow specific rules or a limited set of supported applications to traverse them, typically some given protocols with relatively predetermined and static properties, e.g., port numbers. Applications which have more dynamic properties can traverse firewalls and NATs with assistance of application layer gateways (ALGs). In practice this leads to the fact that although the traffic of many applications is able to traverse firewalls or NATs, a direct connection to a peer behind a middlebox is not possible without further assistance.

Several implicit middlebox configuration (triggered by data traffic) approaches to enable end-to-end connectivity used for peer-to-peer applications have been proposed, e.g., P2PNAT [2] using STUN [3]. It is assumed that all middleboxes between the sender and the receiver behave well, otherwise such an implicit approach is not supported on a path. In contrast, explicit middlebox configuration (triggered by signaling traffic) approaches, such as middlebox communication (MIDCOM) [4] or NAT/Firewall NSLP [5], can rely on an open and standardized protocol behavior.

Signaling for a specific data flow as needed in Quality of Service (QoS) signaling is preferably done with a *path-coupled* approach, i.e., the data packets follow the same path as the signaling traffic. This idea is also adopted for NAT and firewall signaling, because there may be several middleboxes on the path between the sender and the receiver, interaction with all of them is needed. Therefore, a client-server approach as in MIDCOM [4] is of limited in a cascaded or symmetrical NAT environment.

RSVP [6] is an example of a path-coupled QoS signaling protocol. Roedig et al. [7] proposed the use of RSVP as firewall signaling protocol without NAT support. Unfortunately, they did not provide implementation details or a performance evaluation.

In this paper, we describe the implementation and performance study of NAT/Firewall NSLP [5], a new path-coupled signaling protocol for NAT and firewall configuration within the NSIS (Next Steps in Signaling) framework [8] developed by IETF NSIS working group [9]. NSIS is an extensible IP signaling framework based upon a two-layered approach. The lower layer protocol GIST [10] provides path-coupled signaling transport, whereas the upper layer is solely concerned with signaling. The layer splitting allows path-coupled signaling applications to reuse functionality and to avoid unnecessary protocol design complexity. Although proposals for middlebox traversal exist, their implementations are rarely reported and, as far as we know, there is no evaluation of such an implementation available.

The basic focus of this paper is to prove that the current draft of NAT/Firewall NSLP signaling protocol is implementable and technically feasible. Furthermore, we aim to know how a path-coupled NAT and firewall configuration protocol stresses the involved middleboxes. These aspects might prove applicability of the protocol in existing contexts.

The organization of this paper is as follows. Section II provides an overview of the NSIS framework and the NAT/FW NSLP, Section III presents the system design and implementation. Then our performance study is given in Section IV. Section V points out the open issues and aspects for further discussion. Section VI concludes this paper.

2 An Introduction to NSIS and the NAT/FW NSLP

2.1 The NSIS Signaling Concept

The NSIS framework has been developed with the goal of supporting different signaling applications which need to install and manipulate states in the network. These states belong to a certain data flow and are installed and manipulated on all NSIS Entities (NEs) along the signaling flow whereas not every node has to be a NE. Two NSIS entities (or called NSIS hops) are in a peer relationship if they communicate directly with each other. Neighboring nodes do not necessarily have to be NSIS-aware, because an NSIS- unaware node will be ignored at the peer discovery process. The discovery process is triggered when there is no GIST-peering between the peers yet. Under a soft-state approach, states are removed when the state timer expires. Neighbouring nodes store information about each other, but it is unnecessary to establish a long-term signaling connection between them. This basic concept can support the development path-coupled signaling protocols.

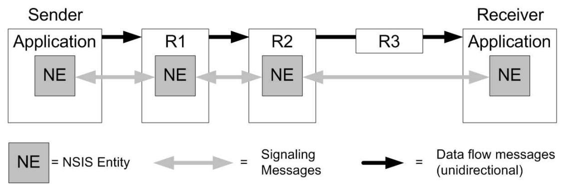

Figure 1. Signaling and Data Flow Example

Figure 1 shows a possible signaling scenario. A data flow is sent from an application at the sender via several routers to the receiver. The end hosts and two of the routers support the NSIS implementation. R3 does not support NSIS and performs only IP layer forwarding. The signaling message exchange is possible in both directions.

2.2 NSIS Layered Model Overview

In order to meet the modular requirements for an extensible and generic signaling protocol, the design of the NSIS protocol suite separates the transport functionalities for signaling message transport from signaling applications. Thus, the NSIS protocol is structured into two protocol layers [8] as illustrated in Figure 2:

- The NSIS Transport Layer Protocol (NTLP), which is responsible for transporting signaling application layer messages, called General Internet Signaling Transport (GIST) [10]. The NTLP layer is independent of the signaling application and runs over standard transport.
- The NSIS Signaling Layer Protocols (NSLPs) implement application specific signaling functionality. Examples for NSLPs are the QoS NSLP for resource reservation signaling [11] and the NAT/FW NSLP [5].

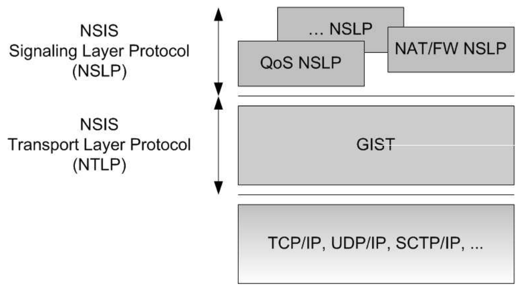
NSIS Signaling Layer Protocol (NSLP)
NSIS Transport Layer Protocol (NTLP)
... NSLP
QoS NSLP
NAT/FW NSLP
GIST
TCP/IP, UDP/IP, SCTP/IP, ...

Figure 2. Separation between Signaling Transport and Signaling Application

2.3 NAT/FW NSLP Overview

The main goal of NSIS NAT/FW NSLP signaling is to enable communication between two endpoints across IP networks in existence of NATs and firewall middleboxes. Firstly, it is assumed that firewalls will be configured in such a way that NAT/FW NSLP messages are accepted for local processing at any time. Then, the NAT/FW NSLP is used to dynamically install additional policy rules in all NAT/FW NSLP-aware middleboxes along the path. NATs will be configured to translate data packets according to the

NAT bindings which are installed and maintained by the NAT/FW NSLP signaling. Firewalls will be configured to process data packets according to the previously installed policy rules.

The signaling traffic of an application behind a middlebox must traverse all middleboxes along the data path to establish communication with a corresponding application on the other end host. To achieve middlebox traversal, the application triggers the local NSIS entity to signal along the data path. If the local NSIS entity supports NAT/FW NSLP signaling, the knowledge of these application is used to establish policy rules and NAT bindings at all middleboxes along the path, which allows the data to travel from the sender to the receiver. Clearly, it is necessary for intermediate NATs and firewalls to support NAT/FW NSLP, but not necessary for other intermediate nodes to support NAT/FW NSLP or even GIST.

NAT/FW NSLP works in different scenarios, such as cascaded NATs and firewalls, NATs with private network on sender side, NATs with private network on receiver side, both end hosts behind a NAT and/or firewall. Our implementation supports all currently specified scenarios, for both NAT and firewall cases. The performance study presented in this paper only concerns firewalls, while NATs are expected to exhibit similar behavior, given that both share many properties. Figure 3 shows a common topology for the use of NAT/FW NSLP. This network is separated into two distinct administrative domains, namely "Domain A" and "Domain B", and used for the performance study.

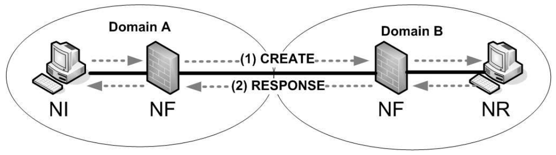

Figure 3. A Firewall Traversal Scenario

The NSLP Initiator (NI) sends NSIS NAT/FW NSLP signaling messages along the data path to the NSLP Responder (NR). Along the path, the signaling messages traverse intermediate NSLP Forwarders (NF). NFs process the messages and additional policy rules or NAT bindings might be installed for the upcoming data traffic.

3 Implementation

The NAT/FW NSLP daemon is implemented in user-space using C++. The code builds upon a GIST daemon that was developed at the University of Göttingen, both implementations are freely available in a single release [12] for Linux. GIST daemon offers an API for NSLPs to use its generic transport services via UNIX sockets. NAT/FW NSLP daemon itself also offers an API to upper layers to allow applications to trigger signaling flows, such as accepting inbound connections at an edge firewall. As depicted in Figure 4, the implementation consists of six main parts: (1) server core connecting to GIST-API and delegating callbacks to the other components, (2) NAT/FW engine API, (3) protocol behavior defined in a finite state machine, (4) message parsing and construction, (5) security policy table and (6) APIs to firewall and NAT. NAT/FW-engines, which enforce the NAT bindings and FW policy rules for corresponding data traffic, are connected to the NAT/FW NSLP daemon via the NAT/FW engine-API.

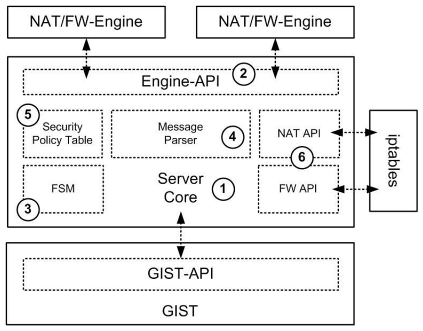

Figure 4. NAT/FW NSLP Architecture

We chose to use Linux kernel netfilter [13] module and its iptables front end as NAT and firewall, because of its availability and complete coverage of needed features. The use of the low-level iptables API `libiptc` is still discouraged by its developers because lack of robustness and missing documentation. To avoid problems and incompatibility with different iptables versions we chose to use `system()` call to invoke an iptables process with according parameters although this approach is known to be inefficient. NAT/FW NSLP imposes only a small set of requirements on the used firewall and NAT as it supports only hardly more than the smallest subset of any possible firewall or NAT implementation. Replacing the currently used firewall and NAT can be done easily.

It was shown during GIST development that having an efficient finite state machine in source code that represents similar sets of states, transitions and actions as in state machine specification simplifies the understanding of the code without sacrificing performance. A C++ template was written to allow reusability among GIST and NSLP daemon development, enabling a mapping between the definition of a finite state machine, including states, transitions and actions to corresponding variables, function pointers and executable code.

The NAT/FW NSLP state machine [15] lists three possible initial states, a host being in an initiator, a forwarder or a receiver idle state. The decision whether a message has to be forwarded or delivered can not be made solely on the destination address in GIST Message Routing Information (MRI) as in NAT case it is being rewritten, similarly as IP headers in NATs. Moreover, it depends on the current NAT configuration (or alternatively, often called *reservation*) status at the host where a message is received. The idea of the state machine giving an high-level overview for protocol understanding misses some aspects, such as locator rewrite and reservation dependency, that were fundamental aspects during implementation.

Incoming message are processed by GIST and delivered to an NSLP if the NSLP is supported on that node. The NSLP decides whether to accept the message or forward it. In the NAT/FW NSLP there are messages that are meaningful either just for NAT or just for firewall. The daemon configuration allows to set flags whether a NAT/FW NSLP host is running a firewall, a NAT or both. After a message is accepted, basic validity checks are performed and the daemon will try to associate an existing state with the incoming message based on the session ID carried in NSLP payload. If there is no state installed yet, a new state machine object with a new session ID needs to be created. As mentioned above, it must be distinguished whether the host is the NR of the signaling path or whether it is a NF. Depending on the initial state, the protocol behavior for this session is different. In contrast, if the action is triggered via API the host is always the NI and a new state machine object with a new session ID is created. Now, as the state machine object is created and the initial state is determined, the transition is applied on it. Transitions are modeled as a set of states, events and function pointers on the state machine object. According to a given state and an incoming event, a function is called. This keeps the function call overhead very small. The function bodies contain all relevant code that defines the protocol behavior, such as state manipulation, NAT and firewall interaction, message parsing and construction.

4 Performance Study

Our motivation is to evaluate the performance impact on our implementation under different number of sessions. We choose this approach, since the maximum supported flows is likely a bottleneck for the deployment of firewalls in heavy- load environments. Furthermore, NAT/FW NSLP might be not deployed in the network core, but rather in the edge. Considering large access network, such as corporate or campus ones where border firewalls protect thousands of nodes, maximum session support is the crucial performance aspect. However we decided to perform the tests without background traffic since we would examine rather the utilized firewall implementation as our NAT/FW NSLP implementation thereby.

4.1 Testbed Setup and Tools

The testing experiments were running Debian on standard PCs with Linux Kernel version 2.6.12.1. We decide to use middle- to low-end hardware as many firewalls are not equipped with high-end hardware and often run with middle-rate CPUs and minimal memory.

Our machines are equipped with the following hardware:

- Via Eden CPU 533 MHz
- 3 Realtek 100 Mb/s NICs
- 256 MB SD-RAM PC 133

Figure 3 depicts how we connected the nodes for our experiments. Both NSLP forwarders run firewalls. Most experiments were performed with this topology, while other experiments were performed with only one NSLP forwarder, namely for the session setup time measurement. The source code was extended by timestamps to measure the session setup time. The same approach is used to calculate the processing time for the NAT/FW NSLP implementation and for netfilter/iptables. A simple script similar to Linux process monitoring tool `top` was used to monitor CPU and memory consumption in regular intervals.

The tests were performed using a simple test application. This application was developed to run on the NI and creates a NAT/FW NSLP session every $\frac{3}{4}$ second. The delay of $\frac{3}{4}$ second between two sessions is a result of the utilized firewall implementation and will become more clearly in the following section. Each session has the same NR, but different source and destination ports. Thereby we can assure that the NF must create a new pinhole for each session. Each test generates up to 15,000 sessions between the NI and the NR without deleting or tearing down existing sessions during the tests. GIST is configured to use C-mode with a session refresh rate of 180 seconds and the NAT/FW NSLP is configured to use a session refresh rate of 90 seconds. [14] have shown that 180 seconds as NTLP session refresh rate and 90 seconds as NSLP session refresh rate are the probably best value for session refresh rate.

4.2 Performance Study

4.2.1 Analysis of the Processing Time

First tests showed us that the maximum session number is limited by a certain threshold and that an increased amount of sessions rapidly increases the session setup time for each new session. This motivated us to examine the processing time distribution of the individual implementation components more closely and to determine the performance bottleneck in detail. Therefore, timestamps were inserted in the source code and the same test was executed again. Several timestamps allow the individual measurement of the processing time for the NAT/FW NSLP implementation and the processing time of netfilter. The netfilter processing time can only be measured on firewall nodes. Hence, measurement was run on a NF and thus the NAT/FW NSLP processing time represents only the processing times on a NF.

Figure 5 shows the processing time for the NAT/FW NSLP implementation with and without netfilter calls. As it can be seen, the processing time of the NAT/FW NSLP implementation is almost stable between 40-50 ms, independent of the number of sessions. In contrast, the processing time for adding a new rule to netfilter may be acceptable for up to 6,000 sessions; with more than 6,000 sessions it increases rapidly and reaches nearly 400 ms with 15,000 sessions. This shows that the netfilter system-call to open a pinhole is a bottleneck in our experiments.

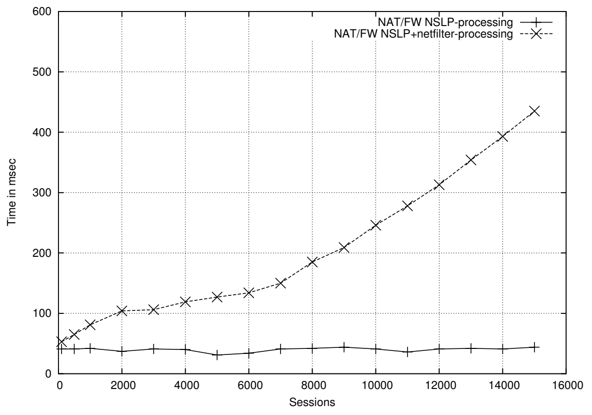

Figure 5. Average Processing Time

These high values may be caused by the machines used in our testbed, which are, due to their hardware, only comparable with middle or low-end firewall devices. However, a more crucial contributing factor for the limited performance in the case with netfilter calls is that netfilter implements the data structure for rule sets using a linked list. Netfilter uses a packet classification algorithm which traverses the rules in a chain linearly per packet until a matching rule is found. Thus, this approach lacks efficiency. Furthermore, the iptables approach is not optimized for chains with many rules as the cost for inserting rules and the processing time per packet increases rapidly with increasing numbers of rules in the rule set [16, 17].

Another critical aspect is that netfilter stalls the packet processing during chain updates, such as inserts. In order to study the performance of the implementation with larger numbers of sessions, netfilter would have to be replaced by a more scalable implementation.

Kadlecsik and Pásztor [16] and Hoffmann et al. [17] have investigated this behaviour of the netfilter implementation. They demonstrated that a high-end firewall needs above 400 seconds to add 16,384 rules into the rule set and that the cost to add a rule increases the more rules already exist. Several alternative solutions were proposed to solve this problem, e.g. ipset [18] or nf-HiPAC [19].

[16] have shown that both solutions reduce the number of memory look-ups per packet. They are applicable for environments with large rule sets or high bandwidth networks and outperform netfilter independent of the number of rules. Thereby, they do not impose any decisive overhead, even for very small rule sets. They also can update their rule sets dynamically without stalling packet processing.

4.2.2 Analysis of the Session Setup Time

The next performance test examines the session setup time of the implementation to answer the following question: how long does it take to establish a NAT/FW NSLP session between NI and NR? The experiment is performed using only three nodes of the testbed, one NI, one NR and one NF. We limited the number of nodes to simplify the measurement. The results can be used to calculate setup times for scenarios with more than one intermediate NF as the processing is almost equal among all NFs when considering a two-way message exchange.

As stated above, the netfilter implementation is the bottleneck of our experiments. Instead of using a different firewall implementation, the firewall was configured with an "*allow-all*" policy. Furthermore, the implementation was changed to ignore any netfilter calls as our primary goal is to study only the NAT/FW NSLP implementation performance and to validate protocol design and operation efficiency. This experiment approach is very likely to be a good approximation to some realistic scenarios as [16] and [17] show that the overhead of more scalable firewall implementations can be negligible.

Figure 6 shows the average session setup time under different number of sessions in both scenarios: with and without performing netfilter-calls.

The first observation is that both demonstrate a nearly linear increase of the session setup time with an increased number of sessions. When serving for approximately 15,000 sessions and performing the netfilter-calls, the NAT/FW NSLP implementation seemed heavily overloaded with a setup time of about 520 ms. Without performing the netfilter-calls, it can be seen that the NAT/FW NSLP implementation is now able to handle up to 35,000 sessions.

The figure also shows that the session setup time without the netfilter-call increases slower. With 35,000 sessions the session setup time remains under 150 ms.

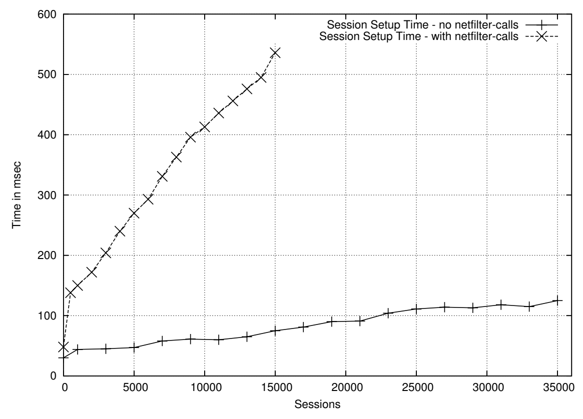

Figure 6. Average Session Setup Time

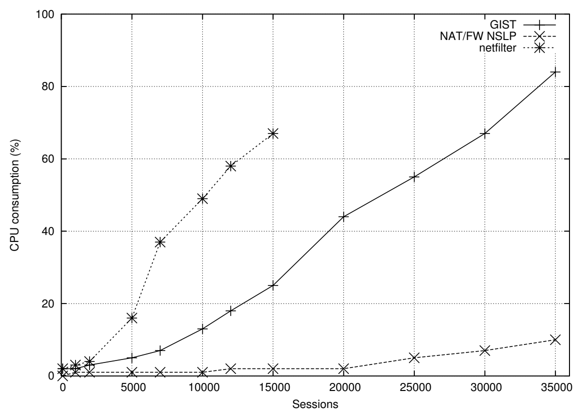

Figure 7. Average CPU Consumption

4.2.3 CPU and Memory Consumption

Another experiment was to measure the CPU consumption of the NAT/FW NSLP implementation. As described above we were only able to perform tests for less than 15,000 sessions if netfilter is running (and less than 35,000 sessions if netfilter is not running).

Figure 7 shows the CPU usage of the NSLP forwarder under different numbers of sessions. The first observation is that the CPU consumption of the netfilter implementation increases rapidly if it has to handle more than 2,000 sessions. The figure also confirms that the netfilter implementation is the bottleneck in the experiments. Furthermore, we can see that the NAT/FW NSLP implementation does not impose high overhead. We can also see that the maximum number of 35,000 sessions depends on the GIST implementation. That is due to the fact that the used GIST implementation currently uses an insufficient timer approach , which brings a large overhead to the GIST implementation [14].

Figure 8 presents the impact of the number of sessions on the memory utilization of the GIST and NAT/FW NSLP implementation at an NSLP forwarder. The memory utilization of the GIST implementation increases nearly linear. The memory utilization of the NAT/FW NSLP implementation also increases nearly linear, however more slowly.

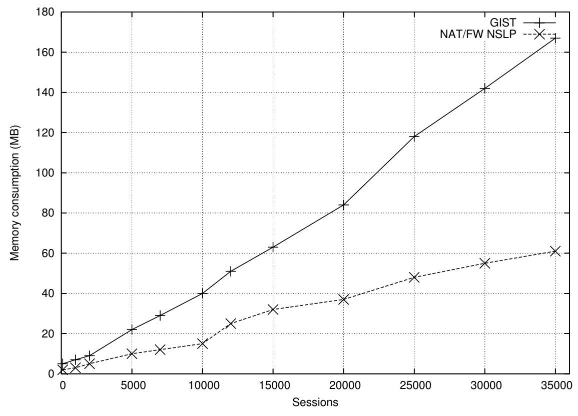

Figure 8. Average Memory Utilization

5 Open Issues

Our NAT/FW NSLP implementation supports signaling for both firewall and NAT configuration. The behavior of simultaneously using both features needs further investigation. Performance evaluation of the NAT/FW NSLP protocol using other firewall/NAT implementations than netfilter is another interesting topic which we plan to work in the next phase.

The implementation has been mostly concerned with enabling and maintaining fixed hosts session-based firewall (and NAT) configurations in middleboxes. It does not deal with node mobility case yet. This will be a very important step for enabling mobile support for the Internet and is currently being studied. Lastly, the performance study is currently only performed for firewalls, not NATs. We believe it is representative enough for the fundamental protocol operations but the run-time performance for NAT case will be of interest for further study.

6 Conclusion

This paper presented the first open source implementation and performance study of a recently proposed middlebox traversal approach which is currently under the development in the IETF. The implementation proves that the path-coupled NAT/FW NSIS Signaling

Layer Protocol [5] is technically feasible and behaves well. The performance results show that the NAT/FW NSLP implementation is scalable even in a low-end hardware environment and the selection of an appropriate firewall implementation has a big impact on the performance; here netfilter [13]. However, it should be noted that the session setup time includes a path discovery and path setup that is likely to be more time-consuming than any implicit peer-to-peer approach. Still, the session setup time might be negligible for the user experience in most cases and the standardized solution of NAT/FW NSLP offers more vendor independence and better interoperability.

To the best of our knowledge, there have been no performance results of any explicit middlebox signaling solution available and the results presented in this paper will be regarded as the first contribution in the field.

Due to the low-end hardware we decided to use in our test environment, it was impossible to create more than 15,000 sessions when netfilter is enabled. This is also caused by the high processing time of netfilter to insert a new rule into a large rule set. With more than 6,000 sessions the processing time increases rapidly and reaches nearly 400 ms with 15,000 sessions. This shows that the bottleneck in the experiments is the netfilter system-call to open a pinhole. Without the netfilter system call the implementation is able to handle 35,000 sessions with nearly stable session setup and processing time. As the NAT/FW NSLP protocol is based on the IETF GIST protocol [8] for signaling transport, the maximum session number of the NAT/FW NSLP implementation is also limited by the utilized GIST implementation. Nevertheless, to be able to handle 15,000 session or 35,000 sessions with a more scalable firewall implementation might be enough. This is even more obvious in larger networks with more powerful hardware. Therefore we assume that our NAT/FW NSLP implementation is applicable for most networks.

Previous firewall performance tests in [16] and [17] showed that firewall implementations also can perform well with a large rule set if the firewall rules are managed efficiently enough. In our implementation, the decision of using netfilter implementation resulted in a limited performance for the firewall traversal implementation as netfilter stores the maintained rule sets using linked lists. Thus, further study with other firewall implementations will be necessary.

Acknowledgment

We would like to thank Christian Dickmann and Bernd Schlör for their work on the GIST implementation. We would also like to thank members of the IETF NSIS working group, including Cedric Aoun, Martin Stiemerling and Elwyn Davies.

References

[1] B. E. Carpenter and S. Brim, “Middleboxes: Taxonomy and Issues”, Internet Engineering Task Force, RFC 3234, Feb. 2002.

[2] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-Peer Communication Across Network Address Translators”, in *Proc. of the 2005 USENIX Annual Technical Conference (USENIX ’05)*, Anaheim, California, April 2005.

[3] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, “STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)”, RFC 3489, March 2003.

[4] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, A. Rayhan, “Middlebox Communication Architecture and Framework”, RFC 3303 (Informational), August 2002.

[5] M. Stiemerling, H. Tschofenig, C. Aoun, and E. Davies, “A NAT/Firewall NSIS signaling layer protocol (NSLP)”, Internet draft (draft-ietf-nsis-nslp-natfw-10), work in progress, March 2006.

[6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, “Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification”, RFC 2205, Sept. 1997.

[7] U. Roedig, M. Goertz, M. Karsten, R. Steinmetz, “RSVP as firewall Signalling Protocol”, *Proc. of the 6th IEEE Symposium on Computers and Communications*, Hammamet, Tunisia, pp. 57–62, IEEE Computer Society Press, July 2001.

[8] R. Hancock, G. Karagiannis, J. Loughney, and S. Van den Bosch, “Next Steps in Signaling (NSIS): Framework”, Internet Engineering Task Force, RFC 4080, June 2005.

[9] “The IETF Next Steps in Signaling (NSIS) Working Group”, http://www.ietf.org/html.charters/nsis-charter.html.

[10] H. Schulzrinne and R. Hancock, “GIST: General Internet Signaling Transport”, Internet draft (draft-ietf-nsis-ntlp-09), work in progress, February 2006.

[11] J. Manner, G. Karagiannis, and A. McDonald, “NSLP for Quality-of-Service signaling”, Internet draft (draft-ietf-nsis-qos-nslp-10), work in progress, March 2006.

[12] ”An Implementation of the Next Steps in Signaling (NSIS) Protocol Suite at the University of Göttingen”, http://user.informatik.uni-goettingen.de/˜nsis/.

[13] “Netfilter, firewalling, NAT, and packet mangling for Linux”, http://www.netfilter.org.

[14] X. Fu, H. Schulzrinne, H. Tschofenig, C. Dickmann, D. Hogrefe, “Overhead and Performance Study of the General Internet Signaling Transport (GIST) Protocol”, in *Proc. of INFOCOM 2006*, Barcelona, Spain, IEEE, April 2006.

[15] C. Werner, X. Fu, H. Tschofenig, C. Aoun, N. Steinleitner, “NAT/FW NSLP State Machine”, Internet draft (draft-werner-nsis-natfw-nslp-statemachine-02), work in progress, March 2006.

[16] J. Kadlecsik, G. Pásztor, “Netfilter Performance Testing”, 2004, http://people.netfilter.org/kadlec/nftest.pdf.

[17] D. Hoffmann, D. Prabhakar, P. Strooper, “Testing iptables”, Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, pp. 80–91, IBM press, 2003.

[18] “IP sets,”http://www.ipset.netfilter.org.

[19] “nf-HiPAC: High Performance Firewall for Linux Netfilter”, http://www.hipac.org.