Certificate

This is to certify that the work contained in the B.Tech. Project report titled "**Implementing Authentication and Authorization in NSIS Protocol Suite**", submitted by Dinesh Gurnani (Y3110) and Prateek Gupta (Y3229), has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Dhiraj Sanghi 7/5/07

Dr. Dheeraj Sanghi
Professor
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
May, 2007

Implementing Authentication and Authorization in NSIS Protocol Suite

Dinesh Gurnani (Y3110), Prateek Gupta (Y3229)
CS499: B.Tech Project Report,
IIT Kanpur

Abstract. Due to evolution of Web 2.0 and increased usage of multimedia services, the internet infrastructure has seen a phenomenal increase in real time data traffic. This calls for an extensible and secure signaling protocol. NSIS (Next Steps in Signaling) [2] [3] is a major initiative by the Internet Engineering Task Force(IETF) in this direction. It is a generic IP based signaling protocol which provides means to establish and manage network control states along a data path between two nodes communicating on the Internet. The modification of control states along a data path allowed by NSIS introduces the need for authentication and authorization. This report discusses our work done in CS498 and CS499 leading to the implementation of authentication and authorization object in NSIS.

Key words. NSIS Signalling Authentication Authorization SIP NSLP GIST NatFW

1 Introduction

Signaling in communication networks is the exchange of information between the nodes to establish, renew and remove the control states on them. In telephone networks, signaling is used for call setup and termination. In circuit switched networks, the creation and termination of circuits is done by signaling messages. For example, in a telephone network, from the dialtone to the first spoken words, the telephone device sets up a chain of events in the network for call set-up. Signaling started with the beginning of telephone networks and has since evolved to cope up with the IP-based networks. Initially, signaling was used only for circuit-switched networks but because IP-based networks are mostly packet switched, it developed for packet switched networks too. In IP-based networks, it is mostly used for QoS reservations, interaction with middle-boxes like NAT, Firewall and network diagnosis. In recent years, due to the increase in real time data over IP networks e.g. live video streaming, VoIP, IPTV etc. a need has been felt for a generic IP based signaling protocol suite.

NSIS is a modular framework for signaling applications being developed by IETF as part of its standardization efforts for an extensible and secure signaling protocol. A need for such a protocol was felt because of emerging applications that can benefit from network-layer signaling. NSIS consists of two protocol layers.
(1) NSIS Transport Layer Protocol(NTLP), a lower layer comprising mostly of General Internet Signaling Transport(GIST) protocol which provides means to detect signaling nodes and transports

signaling application layer messages between them.
(2) An upper layer of NSIS Signaling Layer Protocol(NSLP) which comprises of various signaling applications like QoS signaling and Firewall/NAT control[6].

2 Related Work

There are a few existing protocols like the Resource ReSerVation Protocol (RSVP), YESSIR, Boomerang etc. for IP Signaling but every protocol has its limitations. NSIS aims to address these limitations and provide a generic and extensible protocol suite. For example, none of these protocols provides signaling for NAT/Firewall while NSIS not only provides QoS and NAT/Firewall signaling, it also provides the extensibility to add new signaling applications if need arises in future. NSIS also avoids the unnecessary complexity of previous protocols like RSVP by not supporting features which are no longer used widely.
NSIS framework implementation is being developed at the University of Goettingen (http://user.informatik.uni-goettingen.de/ nsis/). The project currently has implementations for GIST, QoS NSLP, NAT/FW NSLP, Ping Tool and a Diagnostics NSLP for testing. Most parts of the implementation are released under the open source General Public License(GPL) license. To allow implementation for proprietary NSLPs the API between GIST and the NSLPs is released under free LGPL licence. The project is being implemented in C++. In addition to University of Goettingen implementation, recently an effort for implementing it in Java has started at University of Coimbra (http://nsis.dei.uc.pt/). To the best of our knowledge the work on authentication and authorization is being done only at IIT Kanpur and Helsinki University of Technology.

3 NSIS vs Other Signaling Protocols

Most popular among other signaling protocols is Resource ReSerVation Protocol (RSVP). Here we will do a comparison of the two popular signaling protocols and decide why is NSIS better than RSVP [1]
There are a few key problems with RSVP which are addressed by the NSIS.
• RSVP does not support mobile nodes while NSIS does.
• RSVP supports IP multicast which adds a great deal of unnecessary complexity to it because multicast has not been widely deployed.
• RSVP does not allow fragmentation of messages and it supports only unreliable UDP transport or raw IP for carrying RSVP messages across the network.
• RSVP has combined discovery and signaling message delivery in a single protocol step which causes problems of security protection.
• RSVP does not adequately address the issues of authentication and key management. Further, the authorization aspects of RSVP do not interwork with the current authentication, authorization and accounting (AAA) infrastructure and in a roaming environment.

NSIS addresses these issues faced in RSVP as follows:
• NSIS separates signaling message transport from signaling applications i.e. it has a modular design.
• It decouples peer discovery from the signaling message transport.
• NSIS offers a session identifier to uniquely identify a signaling session and signaling state inde-

pendent of a flow identifier. This allows to update the flow to session mapping during the lifetime of the session, e.g. to add a new flow and delete the old one for the purposes of a make-before-break handover.

• NSIS supports signaling to hosts, networks and proxies; in other words, it supports end-to-end, edge-to-edge and end-to-edge signaling exchanges.

• NSIS does not have a support for multicast, which reduces complexity for the majority of applications.

• NSIS has been designed with security in mind from the beginning. The goal of the NSIS is to concentrate on path-coupled signaling [RFC4080], in which signaling messages are routed only through NSIS entities (NEs) that are on the data path.

4 Example Scenario

In this section we discuss the deployment scenario of NSIS and need for authentication and authorization in it. Consider a SIP scenario [5] shown in Figure 1 where the caller is behind a NAT and has a private IP address and the callee has public IP address and is outside the NAT.

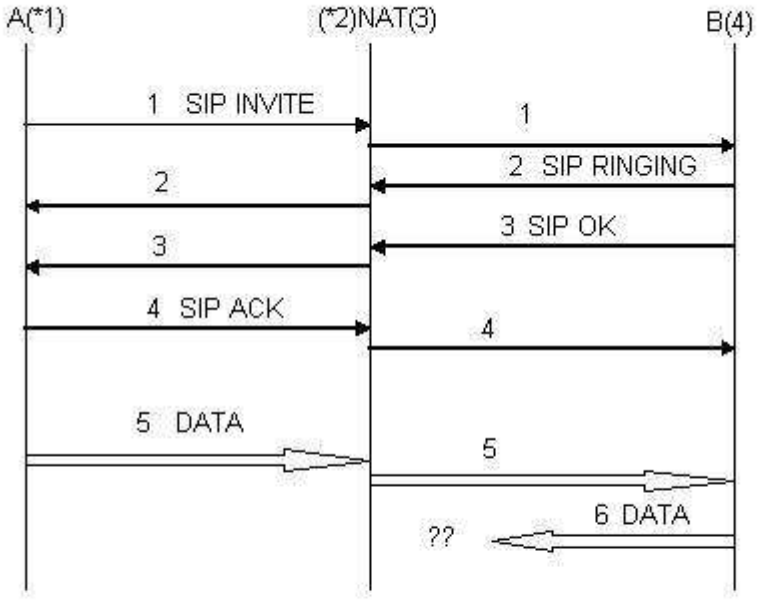

Figure 1.

The caller A has private IP address *1 and the internal interface of NAT has private IP address *2. The external interface has public IP address 3 and caller has public IP address 4. Following is the description of signaling message exchanges between the caller A and the callee B. The repre-

sentation of messages is of form X:Y -> P:Q, which means that the message is from box X, port number Y to box P, port number Q.

1: SIP INVITE (A:? -> B:SIP) : A is ready to receive data on IP address *1 and port x. The message passes through NAT which creates a NAT binding for address *1 to address 3.

2: SIP Ringing (B:SIP -> 3:?) : B is SIP aware and the device at B is ringing. When the message reaches NAT, it forwards it to A due to binding present on NAT.

3: SIP OK (B:SIP -> 3:?) : This OK message shows that B is ready to receive the data on the port y. Again the message is forwarded to A.

4: SIP ACK (A:? -> B:SIP) : All is fine, start transmitting the data. This message indicates that the ports negotiated initially have bee accepted and data can be transmitted to those ports.

5: DATA (A:? -> B:y) and #6: DATA (B:? -> *1:x) : Now the message 5 can be transmitted easily through since B has public IP address. But for message 6 the destination IP address is *1 which is private. Therefore the message will either not be routed or will be sent to a machine in local area network of B.

Now NSIS comes into picture here. If we deploy NSIS in the above scenario, following will be the communication that will take place:

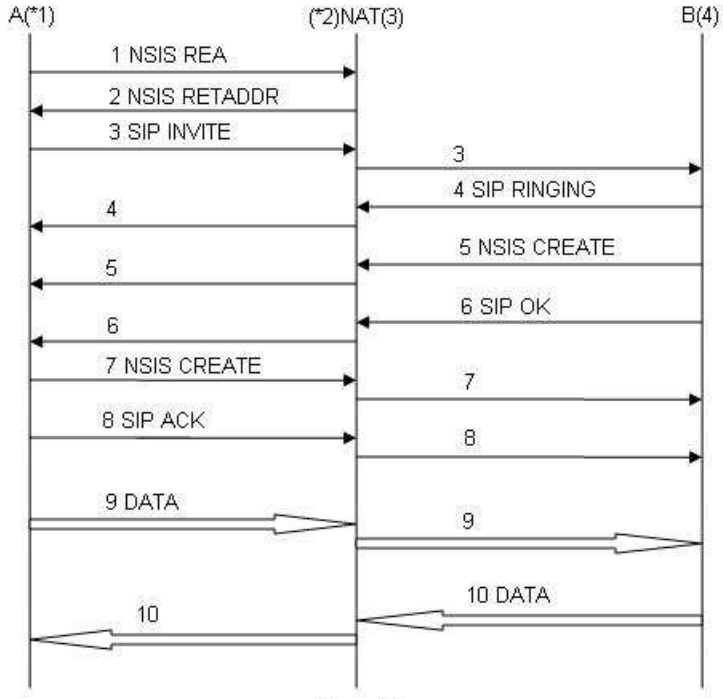

Figure 2.

Since A wants to call B, it first sends a RESERVE-EXTERNAL-ADDRESS (REA) NSIS message. If there is a NAT, as is the case, it will reply with the allocated public address, using a NSIS RETADDR RESPONSE message. If there was no NAT, A continues its normal operation after a

timeout. Normal SIP behavior follows, except that the source address in the SIP INVITE packet has been changed by the one provided by the NSIS RETADDR packet. When B receives the SIP INVITE message, it assumes there might be middle-boxes in the path, so it tries to open a path by sending an NSIS CREATE message to the address provided in the SIP INVITE which is were it will eventually send the voice stream.

The NSIS CREATE message reaches the NAT and activates the state that the NSIS RESERVE previously provided, and the message is forwarded inside; in case there where other middle-boxes that needed to be open. At this stage, B might or might not want to wait for the NSIS success RESPONSE message, issued by A as a reply to the NSIS CREATE. This message is not shown in Figure 2.

Once the path has been created, normal SIP behavior follows, and the communication succeeds. This scheme scales to several middle-boxes, since the NSIS REA messages reserve states in all the middle-boxes until an edge NAT is encountered. Now we clearly see that a normal user A and B are getting to change the control states on the on the middle-box NAT. Hence they need to authenticate themselves to the NAT before they can change the control states on the middle-boxes and reserve resources for their communication.

5 Problem Statement

NSIS protocol implementation currently does not have authentication and authorization functionality which is required by nodes requesting Firewall/Nat and QoS signaling. The aim of this work is to implement the NSIS authentication object and integrate the NSIS protocol stack and the authentication processing with an actual authentication server.

6 Authentication Authorization Model

As already discussed there is a need for authentication and authorization scheme in NSIS protocol suite. Authentication must be done by providing information in the NSLP message which may be used to verify request validity. Authorization can be done by having a session authorization policy element in the message which could be verified by the network.

The draft proposed by J. Manner et al [4] describes a generic NSLP layer session authorization object(AUTH_SESSION). NSLP message carries the the authorization information which is used for the verification of the network resource request. Using the information contained in the messages the network elements verify and process the request.

The session is described by a list of fields and the attributes contained in the AUTH_SESSION object. The AUTH_SESSION object can be used in the currently specified and future NSLP protocols.

The AUTH_SESSION object contains a list of fields which describes the session, along with other attributes. The object header follows the generic NSLP object header. The Object Structure is shown in Figure 3

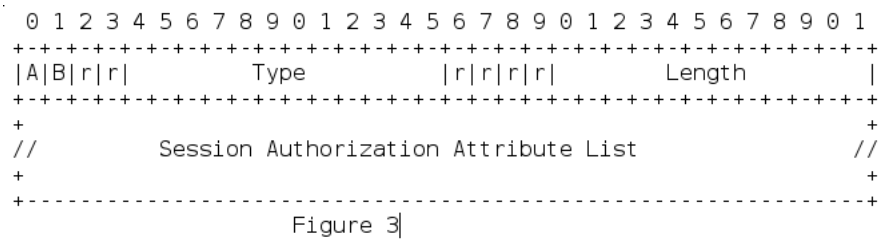

Figure 3

The *type* field marks the identity of the object and the Length field gives the length of the entire AUTH_SESSION (measured in units of 32 bit words) excluding the standard header length. A B are extensibility flags which have been hard-coded to "10" for the current set-up. The object header is followed by a Session Authorization Attribute List which follows the structure shown in Figure 4. The length field is two octets and indicates the actual length of the attribute (including Length, X-Type and SubType fields) in number of octets. The length does NOT include any bytes padding to the value field to make the attribute a multiple of 4 octets.

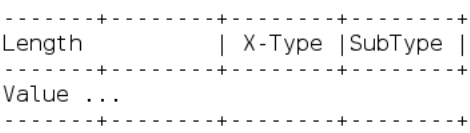

Figure 4, Session Authorization Attribute Structure

The following attributes exist in the AUTH_SESSION object:
AUTH_ENT_ID: The unique identifier of the entity which authorized the session.
SOURCE_ADDR: Address specification for the session originator.
DEST_ADDR: Address specification for the session end-point.
START_TIME: The starting time for the session.
END_TIME: The end time for the session.
AUTHENTICATION_DATA: Authentication data of the session authorization policy element.

Each of the attributes defined may have one or more sub-types associated with it. We give a brief description of the sub-types of attributes.
AUTH_ENT_ID:
The following subtypes are defined.
IPV4_ADDRESS: 32 bit IPv4 Address.
IPV6_ADDRESS: 128 bit IPv6 Address.
FQDN: Fully Qualified Domain Name.
ASCII_DN X 500: Distinguished Name according to RFC 2253.
UNICODE_DN X500: Distinguished Name according to RFC 2253.
URI: Universal Resource Identifier, as defined in RFC 2396.
KRB_PRINCIPAL: Fully Qualified Kerberos Principal name @ realm name (RFC 1510).
X509_V3_CERT: The Distinguished Name of the subject of the certificate as defined in (RFC 2253).
PGP_CERT: The PGP digital certificate of the authorizing entity as defined in (RFC 2440).

SOURCE_ADDR and **DEST_ADDR**
Both have the same sub-types:

IPV4_ADDRESS: IPv4 address represented in 32 bits.
IPV6_ADDRESS: IPv6 address represented in 128 bits.
UDP_PORT_LIST: list of UDP port specifications, represented as 16 bits per list entry.
TCP_PORT_LIST: list of TCP port specifications, represented as 16 bits per list entry.
SPI: Security Parameter Index represented in 32 bits.
In case multiple addresses have been authorised, multiple addresses types of this attribute can be included.

START_TIME and **END_TIME**
The prime reason for introducing this was to avoid replay attacks. It has just one subtype:
NTP_TIMESTAMP: NTP Timestamp Format.

AUTHENTICATION_DATA:
This is the last attribute to be included in the AUTH_SESSION policy element. Its contents are created by signing the data present in the other attributes. AUTH_ENT_ID value decides how is *AUTHENTICATION_DATA* computed. It is instrumental in preventing the integrity of the policy element in untrusted environments.

The AUTH_SESSION object is added to the NSLP message whenever some NSLP initiates a signaling session.

6.1 Authentication and Integrity

AUTH_ENT_ID value is extracted from the AUTH_SESSION object and used to identify the key used in creating the AUTHENTICATION_DATA. Using this key and the appropriate cryptographic algorithm (HMAC- MD5-128), a hash of the AUTH_SESSION object data before the AUTHENTICATION_DATA field is computed and compared with the octetString value of AUTHENTICATION_DATA. A string match insures integrity.

6.2 Authorization

Authorization is carried out by the follwing steps.
1. Retrieve the relevant data from the AUTH_SESSION object.
2. Verify the message integrity
3. Once the identity of the authorizing entity and the validity of the service request has been established, the authorizing router/PDP MUST then consult its authorization policy in order to determine whether or not the specific request is authorized.

7 Approach Adopted

Initially, we started with the background study of signaling, its applications and existing signaling protocols. Then, we moved on to study the problem background, along with studying the NSIS code-flow and also fixing the relevant bugs which would have affected our progress. The code-flow and a few bugs have been explained in the subsequent sections. This part took substantial amount of our time owing to the size and complexity of the NSIS code. After this we studied the draft

draft-manner-nsis-nslp-auth which describes the AUTH_SESSION object. This object has been explained in a previous section. Then we started with the major part of our work i.e. implementing the AUTH_SESSION object.

The implementation of the object mainly contains 2 classes: *NatFwAuthSessionObject* and *NatFwSessionAuthAttr*. The *NatFwAuthSessionObject* is a class representing the whole AUTH_SESSION object which consists of several Session Authorization attributes alongwith the Common Object Header. The *NatFwSessionAuthAttr* class represents single Session Authorization attribute. The latter follows the general structure of the other objects classes while the former follows the general structure of the NatFwMessage class. Both of these classes inherit the GenericObject class due to which they have the instance variables unsigned char * buffer and unsigned char * tail pointing to the start and end of the char string containing all the attributes to be transferred with the object. It also has relevant functions of getting (getBuffer()), setting (setBuffer()) the buffer, getting the length of the buffer (getLength()), extending the buffer by some bytes(extendBuffer()) in order to extend the buffer by given number of bytes. Additionally, the *NatFwSessionAuthAttr* has four attribute variables:

- unsigned char * lengthPtr
- unsigned char * xTypePtr
- unsigned char * subTypePtr
- unsigned char * octetStringPtr

The actual data-type of length is short, xType is char, subType is char and octetString is char *. All of them are defined as unsigned char * because have to added to the buffer of the corresponding object. Due to this constraint, there are functions defined in the class definition to retrieve these variables in their original data-types and are converted from original to unsigned char *. The constructor in this class takes the value of these attribute variables in their original data-types as arguments, extends the initially null buffer by the sum of the lengths of all arguments and converts them to unsigned char * and concatenates them to the buffer.

The *NatFwAuthSessionObject* class is the complete AUTH_SESSION object consisting of the Common Object Header and a list of Session Authorization attributes. This class has no attribute variables. The constructor of this class takes length as argument which specifies the length of the object to be created. In most scenarios, the length is given to be 0. The constructor constructs the Common Object Header with the length field as the length given in the argument. Initially, at the time of construction there are no Session Authorization attributes in the object. This creates a need to include a function which adds the Session Authorization attributes to the object with proper adjustment to the length in Common Object Header.

This *NatFwAuthSessionObject* is created along-with the other objects of NatFwMessage. Here, each Session Authorization attribute is created separately. Then a *NatFwAuthSessionObject* with length argument as 0 is created and each of the Session Authorization attribute is added to the object and the length of the header is adjusted accordingly. For generating the keyed-hash of all the attributes other than the last, we use the HMAC-MD5 algorithm of the OpenSSL library. This object is the last object to be added to the NatFwMessage which is transmitted over the network.

After completing this, our next major part was integrating it with an authentication server to authenticate the credentials of the user and authorizing his request for services. As Radius being the most widely used authentication server, our natural choice was it. But due to the complexity in setting-up and deploying Radius and lack of time, we decided to write our own code which will serve the purpose.

The authentication code accepts three arguments:
1. Authorizing entity identifier,
2. List of all Session Authorization attributes except the last one, and
3. OctetString attribute variable of the last Session Authorization attribute. It then looks up in a text file containing Authorization Entity identifiers and their corresponding keys or certificates. If not found, it sends an AUTH_REJECT message else it uses the corresponding key to compute the hash of second argument and compares it with the third argument. If they match, it sends an AUTH_ACCEPT message else sends an AUTH_REJECT message.

On receiving the AUTH_REJECT message the NatFw server drops the resource request and if it receives an AUTH_ACCEPT message it forwards or processes the request according to the requirement.

8 Code Structure and Flow

The uncompiled NSIS code has an uncompressed size of 6MB. A compilation of the code results in creation of several binaries such as nsis, nsis-natfw, nsis-natfwd, nsis-qos, nsis-qosd, nsis-ping, nsis-pingd, nsis-diag, nsis-diagd, where "d" ending binaries are daemons. On running "nsis", few daemons start alongwith the main "nsis" daemon depending on what has been specified in the configuration file. The configuration file also contains whether a particular host is a NAT or a Firewall or just a normal NSIS aware node. A GIST raw socket is created which listens for IP RAO enabled packets.
Now, we send a nsis-natfw CREATE message to create a firewall pinhole with some parameters. It sends this information in the GIST Object with the parameters as payload forwards the destination. The message is received on a raw socket. All the NSLPs listen for their respective messages by polling on a single gist UNIX socket. Now suppose that the message is received by a router on the way. If it is a firewall, it will make a pending rule in itself and forward the message to the next node. This will go on till the message reaches its destination i.e. NSIS Responder. The NSIS responder on receiving the message will send a RESPONSE message. All the firewalls on the path which had pending rules will remove that rule from the pending state and install it in iptables.

9 Bugs

The NatFwServer instance initiates 3 chains in the iptables namely natfwinput, natfwoutput and natfwforward. These rules are installed in these chains. One of the bugs which was found was that these chains were not initialized anywhere. The NatFwServer always installs ACCEPT policy rule in the firewall even if we specify "deny" in the request. The port numbers used in the request are

not used while installing the policy rule in the firewall.

10 Test Bed

The final goal/demonstration would be to show a functioning authentication/authorization system for the NatFw NSLP, which can easily be extended for other NSLPs because of the code modularity. To demonstrate NSIS with AUTH_SESSION support the following test-bed was chosen:

There are be four machines.(refer Figure 5.) Machine A is the client, machine B is the firewall, machine C is a server behind the firewall which provides some service and machine D is an authentication server. A, B and C have NSIS (with authentication system) installed on them. We will manually provide the authentication information of A in the packet. Using that auth. information, A will create a CREATE packet for creating a pinhole in B for connection A to C. When B receives that packet, it extracts the authentication information from it and sends it to D. Now, we will have some policies regarding authentication and authorization stored on D. Machine D will verify that authentication information with its policies and tell machine B whether it should allow A or not. Depending on D's response, B either ignores the request or processes it.

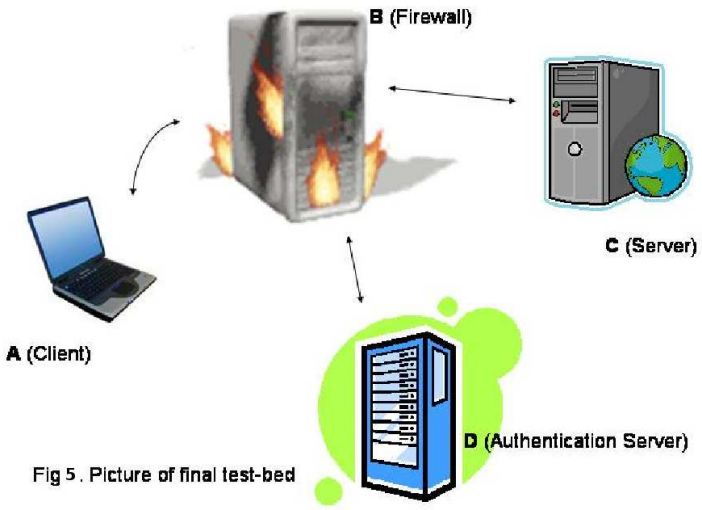

Fig 5. Picture of final test-bed

11 Future Work

Latest amendments in the draft [4] (IETF, March 2007) allow multiple AUTH_SESSION objects to be added for Authentication, Authorization. This has been done in order to support end-to-end signaling and request authorization from different networks. The current implementation provided can be tuned to accommodate this change.

12 Acknowledgements

We would like to thank Dr. Dheeraj Sanghi (IIT Kanpur), Prof Jukka Manner (HUT Finland) and the users of the NSIS help mailing list without whose help this would not have been possible.

References

[1] M. Brunner, *Requirements for Signaling Protocols*. RFC 3726, April 2004.

[2] X. Fu, H. Schulzrinne, A. Bader, D. Hogrefe, C. Kappler, G. Karagiannis, H. Tschofenig, and S. Van den Bosch, *NSIS: a new extensible IP signaling protocol suite*, IEEE Communications Magazine, 43 (2005), pp. 133–141.

[3] R. Hancock, G. Karagiannis, J. Loughney, and S. V. den Bosch, *Next Steps in Signaling (NSIS): Framework*. RFC 4080, June 2005.

[4] J. Manner, M. Stiemerling, and H. Tschofenig, *Authorization for nsis signaling layer protocols*. Internet Draft, March 2007. work-in-progress.

[5] M. Martin, M. Brunner, and M. Stiemerling, *Sip nsis interactions for nat/firewall traversal*. Internet Draft, July 2004. work-in-progress.

[6] M. Stiemerling, H. Tschofenig, C. Aoun, and E. Davies, *Nat/firewall nsis signaling layer protocol (nslp)*. Internet Draft, April 2006. work-in-progress.