International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.362
(11/2006)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

IPCablecom

IPCablecom2 control point discovery

ITU-T Recommendation J.362

ITU-T Recommendation J.362

IPCablecom2 control point discovery

Summary

This Recommendation defines an IP-based protocol that can be used to discover a control point for a given IP address. The control point is the place where QoS operations, lawful intercept (LI) content tapping operations, or other operations may be performed.

Source

ITU-T Recommendation J.362 was approved on 29 November 2006 by ITU-T Study Group 9 (2005-2008) under the ITU-T Recommendation A.8 procedure.

CONTENTS

ITU-T Recommendation J.362

IPCablecom2 control point discovery

1 Scope

This Recommendation defines an IP-based protocol that can be used to discover a control point for a given IP address. The control point is the place where QoS operations, lawful intercept (LI) content tapping operations, or other operations may be performed.

2 References

None.

3 Definitions

This Recommendation defines the following terms:

3.1 control point: Within the context of this Recommendation, control point refers to a point in the network that can be used to apply a function for a media flow that flows through that point. Functions described here are:

- QoS (IPCablecom multimedia [b-ITU-T J.179] or IPCablecom DQoS [b-ITU-T J.163]).
- Replication, encapsulation and transmission for the purposes of LI content tapping.

3.2 control point discovery: The act of discovering information (IP address, protocol) concerning a control point in order to allow a requestor to apply a specific controlling function.

3.3 requestor: The requestor in this context is the controller that wishes to control the control point and hence needs to discover the necessary information to do so.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

CMS	Call Management Server
CMTS	Cable Modem Termination System
COPS	Common Open Policy Service
CPD	Control Point Discovery
CR	Control Relationship
DF	Delivery Function
DNS	Domain Name Service
DQoS	Dynamic Quality of Service
ICE	Interactive Connectivity Establishment
IP	Internet Protocol
LI	Lawful Intercept
MIB	Management Information Base
NAT	Network Address Translation
NE	Network Element

NLS	Network Layer Signalling
NLS-TL	Network Layer Signalling Transport Layer
PS	Policy Server
QoS	Quality of Service
SDP	Session Description Protocol
STUN	Simple Traversal of UDP through NAT
TURN	Traversal Using Relay NAT
UDP	User Datagram Protocol

5 Conventions

Throughout this Recommendation, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word means that the item is an absolute requirement of this Recommendation.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this Recommendation.
"SHOULD"	This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.
"MAY"	This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

6 Technical overview

The general approach for control point discovery is illustrated by the reference model in Figure 6-1. A requestor that knows the IP address of the media endpoint sends a control point discovery (CPD) message toward that endpoint (reference point pkt-qos-2). A control point in the path between the requestor and the endpoint recognizes the CPD message and responds back with the IP address to use for the particular application. The requestor can then make the necessary request for that application.

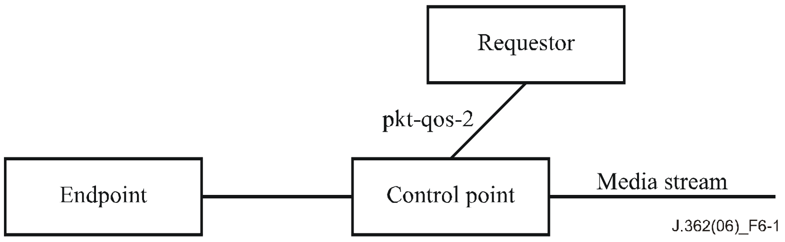

Figure 6-1 – Control point discovery architecture

In addition to supplying the IP address to use, the CPD response indicates the protocol to use and can optionally supply which subnet the destination address of the media endpoint is contained within.

Some of the components that need to make use of CPD are as follows:

- IPCablecom call management server (CMS), in order to determine the IP address of the CMTS for making a DQoS request.
- IPCablecom multimedia policy server (PS), in order to determine the IP address of the CMTS for requesting IPCablecom multimedia [b-ITU-T J.179].
- IPCablecom delivery function (DF), in order to determine the IP address of the CMTS or aggregation device for performing a lawful intercept content tap of the media stream.

CPD is not limited to QoS and LI. It may be used for other applications as well.

The CMS, PS and DF in the above examples need to determine which CMTS or aggregation device (the IP addresses they need to control) will handle the media stream based on the information these components have (the IP address of the endpoint which was obtained via SDP).

Several approaches have been suggested, as follows:

1) Provisioning of subnet versus control point information within each device.
2) Provisioning within DNS [b-IETF RFC 4183].
3) Collecting subnet information over the COPS DQoS or IPCablecom multimedia interface.

Approaches 1 and 2 above are provisioning solutions and as such present operational difficulties. Approach 3 is an application-specific solution that uses a policy management protocol for an unintended purpose, and cannot be reused to address other requirements (e.g., LI). The approach described here is a generic approach that can be used for all three of the above applications and other applications as well.

6.1 Assumptions

The following basic assumptions are used in defining the interface specification in clause 7:

- For the majority of cases, the media endpoint is single-homed behind the control point (i.e., the media route will go through a specific CMTS, media gateway or aggregation device). For possible exceptions to this assumption, the requestor can be provisioned with alternates (i.e., when the requestor receives a CPD response with the IP address of one control point, it is able to determine the IP address of control points for alternative media paths for that media endpoint).
- For some applications (e.g., LI), it is important that the CPD message does not reach a media endpoint outside the provider's network. The assumption is that ACL mechanisms will be in place to drop CPD packets at edge devices that do not support the CPD protocol.

7 Interface description

CPD uses the network layer signalling protocol (NLS). As illustrated in Figure 7-1, NLS consists of an application layer that sits on top of an NLS transport layer (NLS-TL) protocol as defined in Annex A.

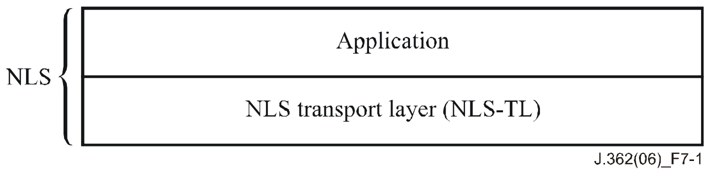

Figure 7-1 – NLS protocol

One such application is the network-based control point discovery application defined in this Recommendation.

7.1 Network-based control point discovery application

The application payload format within NLS consists of an NLS type, length, value (TLV) that consists of a 16-bit application ID followed by a payload that is opaque to the NLS transport layer. Per clause 14.2 of Annex A, the application ID for the control point discovery (CPD) application is "1".

The CPD message format is illustrated in Figure 7-2. For the network-based CPD application, a CPD messages consist of:

- A 4-bit version field. The 4-bit version field is set to "0" for the version of the CPD protocol described in this Recommendation.
- A 12-bit CPD message type. Only two CPD message types are defined:
  - CPD request: CPD message type = 1.
  - CPD response: CPD message type = 2.
- A 16-bit control relationship (CR) type. The CR TYPE identifies the type of control relationship. These values may be provisioned. The requestor MUST set the CR TYPE to one of the following values:
  - CR TYPE = 1: Lawful intercept content tap.
  - CR TYPE = 2: DQoS.
  - CR TYPE = 3: IPCablecom multimedia.
- A 16-bit control relationship ID (CR ID).
- A 32-bit transaction ID.
- CPD message contents for the particular CPD message type.

0
0 1 2 3 4 5 6 7 8 9 1
0 1 2 3 4 5
	6 7 8 9	2
		0 1 2 3 4 5 6 7 8 9 3
		0 1

Application ID	Version	CPD message type
CR TYPE	CR ID	
Transaction ID		
CPD message contents		

Figure 7-2 – CPD message format

The CR TYPE and CR ID uniquely define a specific control relationship between a group of controllers and the network elements (NEs) they control. If more than one NE along a CPD request path can respond to a CPD message with a given CR TYPE, they would have different CR IDs based on the provisioned CR ID value within the NE.

It MUST be possible to provision the control point with a CR ID. The requestor MUST set the CR TYPE to one of the following values:

- For CR TYPE = 1 (lawful intercept):
  - CR ID = 1: CMTS.
  - CR ID = 2: Aggregation router or switch in front of a CMTS.
  - CR ID = 3: Aggregation router or switch in front of media services (e.g., voice-mail).
  - CR ID = 4: Media gateway.
  - CR ID = 5: Conference server.
  - CR ID = 6: Other.
- For CR TYPE = 2 (DQoS) and CR TYPE = 3 (IPCablecom multimedia):
  - CR ID = 1: Default value.

The transaction ID is used by the requestor to relate a CPD response to a given CPD request. It is up to the requestor to pick transaction IDs that do not repeat within a time-frame that would prevent this correlation from occurring.

7.2 NLS-TL header parameters

For all network-based CPD messages, the requestor and control point MUST set the NLS-TL flags as follows:

- HOP-BY-HOP = 0;
- BUILD-ROUTE = 0;
- TEARDOWN = 0;
- BIDIRECTIONAL = 0.

The use of the AX_CHALLENGE and AX_RESPONSE flags are described separately in clause 9.

The requestor MUST set the flow-ID to a random number and the same value MUST be used by the control point in the response.

7.2.1 NLS-TL TLVs

The following NLS-TL TLVs are not used:

- NAT_ADDRESS;
- TIMEOUT;
- IPV4_HOP;
- IPV6_HOP.

7.3 CPD request

The CPD request message contents simply consists of a 32-bit number that contains some request control flags.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Flags								Reserved																							

Figure 7-3 – CPD request message contents

The contents of the CPD request message are illustrated in Figure 7-3. The fields are as follows:

- Flag definitions:
  - 0x01: Subnet information request.
  - 0x02: Forward if not supported.
  - 0x04-0x80: Reserved and set to "0".
- The remaining 24 bits of the message contents are reserved and set to "0".

If the "forward if not supported" flag is set to "1", and the control relationship consisting of the CR TYPE and CR ID is not supported on this device, the control point MUST attempt to continue to forward the CPD packet towards its destination. Otherwise it MUST respond with a response code "control relationship not supported" as indicated in clause 7.5. This mechanism is helpful in several cases. A couple of examples are as follows:

- This allows for an LI to have a different control point to that used for QoS. So, for example, LI could be done on an aggregation device in front of the CMTS with the control point for QoS being on the CMTS. A QoS CPD request would be sent with the "forward if not supported" flag set to "1", while an LI CPD request would normally have this flag set to "0".
- There may be two control points along the path that are of the same CR TYPE but different CR IDs. Again, if the "forward if not supported" flag is set to "1" and the CPD request arrives at a router/switch with the correct CR TYPE but wrong CR ID, the control point MUST forward the message. If the flag is set to "0" and that control relationship is not supported, the control point MUST respond with response code "control relationship not supported".

If the "subnet information request" flag is set to "1", this is an indication to supply the value for the subnet that contains the destination IP address. Otherwise this information will not be returned in the response.

A CR ID of "0" in a CPD request message is considered to be a wild-card, i.e., it is a request to any device with any CR ID value that supports the CPD request message for the CR TYPE specified. When a wild-card value is used for CR ID in the CPD request, the control point MUST include the actual value of the CR ID assigned to the control point in the CPD response.

7.4 CPD response

This clause describes the message content format for the CPD response message.

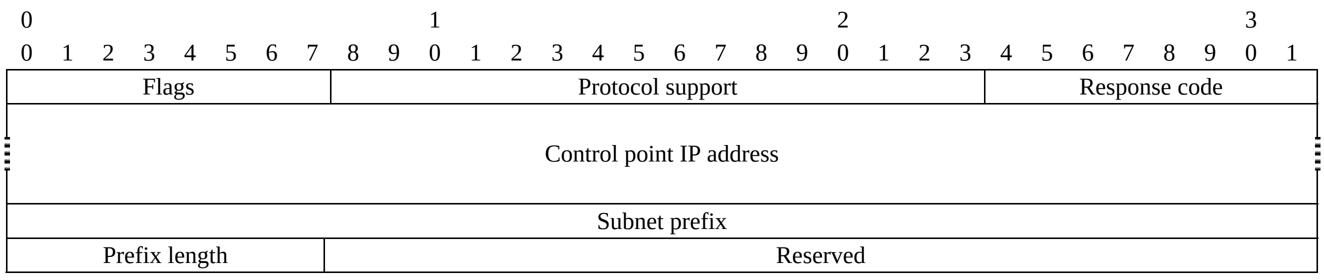

Figure 7-4 – CPD response message contents

The contents of the CPD response message are illustrated in Figure 7-4. The fields are as follows:

- Flags (8 bits) include the following:
  - 0x01: Subnet information included.
  - 0x02: IP version of the control point: "0" for IPv4, "1" for IPv6.
  - 0x04-0x80: Reserved and set to "0".

  Other flag bits are not used and MUST be set to "0".
- Protocol supported flags (16 bits): interface/protocol supported by the control point for that CR type include the following:
  - 0x01: DQoS over COPS.
  - 0x02: IPCablecom multimedia over COPS.
  - 0x04: Li Tap-MIB with SNMPv3.

  Other flag bits are not used and MUST be set to "0".

  Note that the same values can be re-used for different protocols for different CR types. Example: the 0x01 flag for DQoS could be used to describe some other protocol for some other CR type.
- Response code (8 bits) is an unsigned integer with the value "0" for a normal response. For error responses, refer to clause 7.5.
- Control point IP address is the IPv4 or IPv6 address of the control point.
- Subnet prefix is a 4-byte field representing the address prefix of the IPv4 subnet. If the "subnet information included" flag is "0", then this field as well as the prefix length field will be "0".
- Prefix length is a 1-byte unsigned integer representing the number of network bits in the subnet prefix.
- The remaining 24 bits of the message contents are reserved and set to "0".

7.5 Error responses

If an error occurs at the NLS transport level, either the IPv4_ERROR_CODE or IPv6_ERROR_CODE will be returned. If an error occurs at the CPD application level, a non-zero response code will be returned in the CPD response message. The following error codes are defined:

- Response code = 1: Control relationship not supported: this will be returned to the sender if the network element receiving the CPD request message does not support the control relationship (i.e., the CR type and CR ID specified) and the "forward if not supported" flag is set.
- Response code = 2: Poorly formed message (e.g., invalid CPD message type or invalid flags in the request).

8 Procedures

This clause provides a brief description of requestor and responder (control point) procedures.

8.1 Requestor

A requestor in this context is the controller that wants to apply some function to a control point within a network element such as a CMTS, aggregation router or media gateway. As such, it needs to obtain the IP address and protocol needed in order to apply some control over the media stream that passes through the control point.

The requestor receives information as to the IP address of the media stream. This is typically obtained via the session description information (SDP) for the media stream, i.e., the IP address is included in the "c=" line of the SDP. If NAT traversal [b-ITU-T J.160] is not used (i.e., there are no additional candidate addresses), then the requestor MUST send the CPD request to that media address as specified in the "c=" line of the SDP.

The requestor MUST send the CPD request to all candidate addresses except for a TURN server address in the case where it is able to determine that one of the candidates is a TURN server address. It may determine that a candidate is a TURN server address via one of the following mechanisms:

- by keeping a table of TURN server addresses; or
- by the existence of the "local-TURN" SDP attribute as defined in [b-ITU-T J.366.4].

When building the CPD request, the requestor MUST send the message to the well-known UDP port for NLS-TL (to be registered – see IANA considerations in Annex A). The requestor MUST send the CPD request message with CR type and CR ID set for the control point of interest. The requestor MAY also use the wild-card value for the CR ID.

Transaction ID selection: The requestor MUST pick a transaction ID that is not presently in use, which means that either:

- A response has already been received for a request with that transaction ID; or
- A response has been outstanding for some provisioned amount of time (default value, 600 seconds).

8.2 Control point

Network elements containing control points that support CPD will respond, based on the CR type that they support. A given NE may support multiple CR types. A CR ID may also be provisioned or simply left with its default value.

The NE containing the control point recognizes the NLS-TL message, based on the well-known UDP port (to be registered – see IANA considerations in Annex A). It then passes the application payload to the particular NLS application based on the application ID (in this case CPD). The CPD application looks at the CR type and CR ID to see if this control relation is supported. If not, the control point MUST respond as described in clause 7, based on the value of the "forward if not supported" flag.

If the control point does support the control relationship, it MUST respond to the source IP address of the request with a CPD response. The CPD response MUST contain the same CR type, CR ID, flow-id (NLS-TL) and transaction ID as was included in the CPD request message. The CPD application within the application MUST respond with either the IPv4 and IPv6 IP address depending on which is supported. The control point MUST also specify the version in the IP version flag. The protocol(s) supported MUST also indicate for the particular CR type.

The subnet information MUST be provided if requested. In the case where the subnet is provided, the control point MUST also indicate this in the "subnet information included" flag, corresponding to the "subnet information request" flag in the CPD request.

If the CR ID is set to a wild-card value (i.e., "0") in the request, the control point MUST respond with the actual CR ID value for that CR type.

9 Security considerations

The threat associated with illegitimate requests for control point information is that an attacker will have some additional information about the network element that provides that capability, as well as the networks attached to the control point. An attacker could perform some level of network discovery by requesting subnet information from control points.

The control point MUST be able to authenticate CPD requests before responding. The control point SHOULD only respond to CPD requests from authenticated sources. The NLS protocol itself provides an optional authentication mechanism that MUST be used. This uses a challenge mechanism as described in clause 13 of Annex A. This approach requires either pre-shared keys or some group keying mechanism for sharing a secret between the requestor and the control point. Since it does use a challenge-response mechanism, it will result in an additional round-trip delay.

Annex A

Network Layer Signaling: Transport Layer

(This annex forms an integral part of this Recommendation)

```
Network Working Group                                           M. Shore
Internet-Draft                                                 K. Biswas
Expires: May 8, 2006                                           D. McGrew
                                                           Cisco Systems
                                                        November 7, 2005
```

Network-Layer Signaling: Transport Layer
draft-shore-nls-tl-01.txt

Status of this Memo

Copyright Notice

Abstract

The RSVP model for communicating requests to network devices along a datapath has proven useful for a variety of applications beyond what the protocol designers envisioned, and while the architectural model generalizes well the protocol itself has a number of features that limit its applicability to applications other than IntServ. We are developing a modernized version that, among other things, is based on

a "two-layer" architecture that divides protocol function into transport and application. This document describes the transport protocol.

Table of Contents

1. Introduction

RSVP is based on a "path-coupled" signaling model, in which signaling messages between two endpoints follow a path that is tied to the data path between the same endpoints, and in which the signaling messages are intercepted and interpreted by RSVP- capable routers along the path. While RSVP was originally designed to support QoS signaling for Integrated Services [rfc1633], this model has proven to generalize to other problems extremely well. Some of these problems include topology discovery, QoS signaling, communicating with firewalls and NATs, discovery of IPSec tunnel endpoints, test applications, and so on.

This document describes the core protocol for an updated version of RSVP -- one that is not tied directly to IntServ and in which the protocol machinery itself is sufficiently generalized to be able to support a variety of applications (this protocol is referred to as "Network Layer Signaling", or "NLS"). What this means in practice is that there will be different signaling applications, all of which share a base NLS transport layer. This is similar to the concepts used in secsh, where authentication and connection protocols run on top of a secsh transport protocol (see [ylonen] for details).

The protocol machinery was originally based somewhat on RSVP [rfc2205] without refresh overhead reduction extensions [rfc2961], but in the process of generalization has lost many of the features that define RSVP, such as necessary receiver-oriented reservations and processing requirements at each node.

NLS differs from RSVP in several important ways. One of the most significant of these is that the transport protocol described in this document (NLS-TL) does not itself trigger reservations in network nodes. The NLS application will do that, and, indeed, some NLS applications may not carry reservation requests at all (discovery protocols, for example). Because of this NLS-TL does not support reservation styles (those would be also be attributes of an application). Another significant difference is that that reservations may be installed by a NLS application in either a forward (from the sender toward the receiver) or backward (from the receiver toward the sender) direction -- this is application-specific.

Other possibly significant differences include that NAT traversal support is integrated into the message transport, and that NLS allows an application to install reservations for paths that are bidirectional and asymmetric.

1.1 Transport layer

This document describes the transport layer. The NLS transport layer is as simple as we could make it, supporting two basic functions: routing and NAT traversal. The sources of complexity in signaling protocols tend to be the signaling applications themselves. Those applications have varying performance and reliability requirements, and consequently we feel that application-specific functions belong in the application layer.

The NLS transport layer is also relatively stateless. By "stateless" we mean that the transport layer does not itself create or manipulate state in participating nodes. By "relatively" we take exception to the previous assertion, in that the transport layer provides facilities for route identification and route pinning. This is an optimization, albeit a significant one, which allows NLS to be used without a separate route discovery process. Another source of state is in the case of NATs, where an NLS-TL request may trigger the creation of a NAT table mapping. However, this latter case does not create NLS-TL maintenance state.

An application may wish to support summary refreshes or other performance enhancements; that type of function is application-specific and requires no support from the transport layer.

2. NLS-TL Messages

2.1 Message Processing Overview

Unlike RSVP, NLS-TL has only one fundamental message type, and directionality is significant to the NLS application only. Three new attributes, HOP-BY-HOP, BUILD-ROUTE, and BIDIRECTIONAL, have been added in support of greater flexibility in the NLS application. For example, some applications which already know network topology or which run a separate routing protocol may choose to route hop-by-hop in a forward direction. Conversely, a topology discovery protocol may choose to route end-to-end in the return direction. Both of these would be departures from the Path/Resv message handling specified in RSVP.

The BUILD-ROUTE flag has been added to allow route discovery to be overloaded on top of basic messaging, much like the RSVP Path message. If the BUILD-ROUTE flag is present, NLS nodes store routing information carried in incoming HOP objects. They also overwrite routing information into the HOP TLV in outgoing NLS messages.

The BIDIRECTIONAL flag may be used to indicate that the application for which this NLS-TL message carries a payload must be executed in each direction. It may be used in combination with the HOP-BY-HOP flag in some circumstances, but typically it will be used with the HOP-BY-HOP flag set to 0.

Even with these departures, the basic operation of the protocol may made be similar to RSVP with the appropriate use of the new attributes. For example, a message may be injected into a network by the sender towards a receiver, routed end-to-end with the receiver's address in the destination address in the IP header. If the BUILD-ROUTE bit is set in the NLS header, entities along the path the message traverses will intercept it, store path state, act on (or not) the application payload data, and forward the message towards its destination. In NLS-TL, "path state" refers specifically to the unicast IP address of the previous hop node along with the previous node's optional logical interface information.

When the message arrives at the receiver (or its proxy), the receiver may generate another NLS message in response, this time back towards the original sender. As with the message in the forward direction, this message may be routed either end-to-end or hop- by-hop, depending on the requirements of the application. In order to emulate an RSVP Resv message, the HOP-BY-HOP is set to 1 and the BUILD-ROUTE bit is set to 0.

BUILD-ROUTE and HOP-BY-HOP must not be set in the same NLS-TL message, and BUILD-ROUTE and TEARDOWN MUST not be set in the same NLS-TL message.

2.2 NAT Traversal Support

NAT traversal poses a particular challenge to a layered protocol like NLS. If we assume the use of discrete, opaque applications, one of which is NAT, interactions between other applications that make use of addresses (for example, firewall rules or QoS filter specs) and the NAT application are complicated. Either every application will need to be able to peek into NAT payloads and identify which address mapping is the one they need, or NATs supporting NLS will need to be able to parse and write into every application payload type. Neither approach is particularly robust, reintroducing a type of stateful inspection and constraining how applications can be secured.

Because of the desire to be able to have a variety of NLS applications successfully interact with NATs and because of the constraints described above, in NLS NAT is supported in the transport layer rather than in a separate application. Addresses needing translation are tagged and put in NLS-TL TLVs and passed to the appropriate application at each NLS node. Application identification is based on tag contents.

2.3 NLS-TL Message Format

NLS messages consist of an NLS-TL header followed by optional TLV fields followed by an optional application payload.

2.3.1 The NLS-TL Message Header

All NLS-TL messages (and by implication, all NLS messages) start with an NLS header. The header is formatted as follows:

```
        0             1              2             3
+-------------+-------------+-------------+-------------+
|   Version   | (Reserved)  |      Message Length       |
+-------------+-------------+-------------+-------------+
|            Flags          |         Checksum          |
+-------------+-------------+-------------+-------------+
|                         Flow ID                       |
+-------------+-------------+-------------+-------------+
```

Figure 1

where the fields are as follows:

Version: 8 bits. The protocol version number; in this case 0x01.
Message Length: 16 bits. The total number of octets in the message, including the NLS-TL header and complete payload.
Flags: 16 bits. Flag bits include

0x01	HOP-BY-HOP
0x02	BUILD-ROUTE
0X04	TEARDOWN
0x08	AX_CHALLENGE
0x10	AX_RESPONSE
0x20	BIDIRECTIONAL

Checksum: 16 bits. The one's complement of the one's complement sum of the entire message. The checksum field is set to zero for the purpose of computing the checksum. This may optionally be set to all zeros. If a message is received in which this field is all zeros, no checksum was sent.
Flow ID: 32 bits. This is a value which, combined with the source IP address of the message, provides unique identification of a message, which may be used for later reference for actions such as quick teardowns, status queries, etc. The mechanism used for generating the value is implementation-specific.

2.3.2 NLS-TL TLVs

NLS-TL carries additional transport-layer information and requests as type-length-value fields, which are inserted after the header and before the application payload. The TLV format is as follows:

```
       0             1              2             3
+-------------+-------------+-------------+-------------+
|          Length           |            Type           |
+-------------+-------------+-------------+-------------+
|                                                       |
//                        Value                         //
|                                                       |
+-------------+-------------+-------------+-------------+
```

Figure 2

where the fields are as follows:
Length: 16 bits. Total TLV length in octets. It must always be at least 4 and be a multiple of 4.

```
   Type:  16 bits.  The type of information or request.  Defined below.
   Value:  Variable length.  At least 4 octets and a multiple of 4
      octets).  The TLV semantic content.
```

2.3.2.1 NAT_ADDRESS, TYPE=1

```
      +-------------+------------+------------+-------------+
      |       Application ID     |    Flags   |    Proto    |
      +-------------+------------+------------+-------------+
      |                    Address ID Tag                   |
      +-------------+------------+------------+-------------+
      |                 Original IPv4 Address               |
      +-------------+------------+------------+-------------+
      |                  Mapped IPv4 Address                |
      +-------------+------------+------------+-------------+
      |      Original Port       |        Mapped Port       |
      +-------------+------------+------------+-------------+
```

```
   where the fields are as follows:
   Application ID:  16 bits.  This is the same as the value that's used
      for identifying application payloads.
   Flags:  16 bits.  Flag bits include
      0x01 = NO_TRANSLATE
      0x02 = NO_REWRITE

      NO_TRANSLATE indicates that a NAT device handling the packet
      should not create a NAT table entry for the original address.  If
      the NO_TRANSLATE bit is set, the NAT does nothing.

      NO_REWRITE indicates that when the reply message is being returned
      towards the sender, any NATs along the path MUST NOT overwrite the
      Mapped Address.
   Proto:  IP protocol for this translation (TCP, UDP, SCTP, etc.).
   Address ID:  32 bits.  An value that's unique within the set of
      Address IDs used with a particular Application ID; used to
      uniquely identify a particular address (i.e.,  provide a tag).
   Original IPv4 Address: The original address for which a translation
      is being requested.
   Mapped IPv4 Address:  The address created by the NAT -- i.e.,  the
      "external" address.
   Original Port:  The original port for which a translation is being
      requested
```

Mapped Port: The port number created by the NAT for this mapping.

2.3.2.2 APPLICATION PAYLOAD, TYPE=2

```
+-------------+------------+------------+-------------+
|       Application ID     |         Payload          |
+-------------+------------+------------+-------------+
|                                                     |
//                       Payload                     //
|                                                     |
+-------------+------------+------------+-------------+
```

The application payload TLV carries the NLS application data. It MUST follow any NAT TLVs. It consists of a 16-bit Application ID, which uniquely identifies the NLS application for which the TLV is intended, and the application payload itself. The application payload is transparent to the NLS Transport Layer.

2.3.2.3 TIMEOUT, TYPE=3

```
+-------------+------------+------------+-------------+
|                    Timeout Value                    |
+-------------+------------+------------+-------------+
```

The TIMEOUT TLV carries the number of milliseconds for which state associated with a particular flow should be retained, with the expectation that the state will be deleted when the timeout expires. "State" in this case refers to routing state and to NAT state; NLS application state will be managed by its application.

2.3.2.4 IPV4_HOP, TYPE=4

```
+-------------+------------+------------+-------------+
|                     IPv4 Hop Address                |
+-------------+------------+------------+-------------+
|                 Logical Interface Handle            |
+-------------+------------+------------+-------------+
```

The IPv4_HOP TLV carries the IPv4 address of the interface through which the last NLS entity forwarded the message. The logical interface handle may be used to distinguish between multiple interfaces on the same entity, or it may be set to all 0s.

2.3.2.5 IPv6_HOP, TYPE=5

```
      +-------------+-------------+-------------+-------------+
      |                                                       |
      +                                                       +
      |                                                       |
      +             IPv6 Next/Previous Hop Address            +
      |                                                       |
      +                                                       +
      |                                                       |
      +-------------+-------------+-------------+-------------+
      |                Logical Interface Handle               |
      +-------------+-------------+-------------+-------------+
```

The IPv6_HOP TLV carries the IPv6 address of the interface through which the last NLS entity forwarded the message. The logical interface handle may be used to distinguish between multiple interfaces on the same entity, or it may be set to all 0s.

2.3.2.6 IPv4_ERROR_CODE, TYPE=6

```
      +-------------+-------------+-------------+-------------+
      |            IPv4 Error Node Address (4 octets)         |
      +-------------+-------------+-------------+-------------+
      |    Flags    | Error Code  |        Error Value        |
      +-------------+-------------+-------------+-------------+
```

The IPv4_ERROR_CODE TLV carries the address of a node at which an NLS-TL error occurred, along with an error code and error value. When no Error Value is defined, the Error Value field MUST be set to 0 by its sender and ignored by its receiver.

If the high-order bit of the Error Code is not set, the TLV carries an error message. If it is set, the TLV carries an informational message. Therefore Error Codes with values between 0 and 127 contain error messages and Error Codes with values between 128 and 255 contain informational messages.

IPv4 Error Node Address: 4 octets. The IPv4 address of the
   interface on the node that generated the error.
Flags: 8 bits. None currently defined.

```
   Error Code:  8 bits.  The type of error or informational message,
      with values as follows:

         Error Code = 0: No error
         Error Code = 1: Bad parameters

            Error Value = 1: HOP-BY-HOP and BUILD-ROUTE both present
            Error Value = 2: BUILD-ROUTE present but no HOP TLV
            Error Code = 3: HOP-BY-HOP present but no local stored
            routing state
            Error Code = 4: Message length not a multiple of 4
         Error Code = 2: Unrecognized TLV

            Error Value = TLV number
         Error Code = 3: Unrecognized application

            Error Value = Application ID
         Error Code = 4: Non-NLS NAT detected in path
         Error Code = 128: No message
         Error Code = 129: Sending node has detected a route change
```

2.3.2.7 IPv6_ERROR_CODE, TYPE=7

```
      +-------------+-------------+-------------+-------------+
      |                                                       |
      +                                                       +
      |                                                       |
      +           IPv6 Error Node Address (16 octets)         +
      |                                                       |
      +                                                       +
      |                                                       |
      +-------------+-------------+-------------+-------------+
      |    Flags    | Error Code  |        Error Value        |
      +-------------+-------------+-------------+-------------+
```

The IPv6_ERROR_CODE TLV carries the address of a node at which an NLS-TL error occurred, along with an error code and error value.

"IPv6 Error Node Address:" 16 octets. The IPv6 address of the interface on the node that generated the error.
Flags: 8 bits. None currently defined.

The Error Code and Error value fields are the same as those used in the IPv4_ERROR_CODE.

2.3.2.8 AGID, TYPE=8

The AGID is the authentication group ID, used in the authentication dialogue to identify the group key.

```
      +-------------+------------+------------+-------------+
      |                           id                        |
      +-------------+------------+------------+-------------+
```

2.3.2.9 CHALLENGE, TYPE=9

The CHALLENGE TLV is used to carry a 16-octet random nonce to be used as an authentication challenge.

```
      +-------------+------------+------------+-------------+
      |                                                     |
      +                                                     +
      |                                                     |
      +                        Nonce                        +
      |                                                     |
      +                                                     +
      |                                                     |
      +-------------+------------+------------+-------------+
```

2.3.2.10 RESPONSE, TYPE=10

The RESPONSE TLV carries the response to the authentication challenge. It is a variable length TLV with the length dependent on the transform being used.

```
      +-------------+------------+------------+-------------+
      |                                                     |
      //                          HMAC                      //
      |                                                     |
      +-------------+------------+------------+-------------+
```

3. Sending NLS-TL Messages

When an endhost or its proxy wishes to initiate a NLS session, it creates an NLS-TL message. If the message is being sent end-to-end the destination address in the IP header is the address of the device interface that is expected to terminate the path along which signaling is expected to be sent. It may be a application peer host or terminal, or it may be a proxy. If the message is being sent hop-by-hop the destination address in the IP header is the address of the device interface that is the next hop along the path. That address will have been discovered either through a separate routing process or through RSVP-style soft-state messaging.

NLS-TL messages may be sent with the router alert bit set in IPv4 headers or with the IPv6 router alert option [rfc2711]. If the message is end-to-end and needs route discovery and pinning, the BUILD-ROUTE bit in the NLS-TL flags header MUST be set to 1 and the HOP-BY-HOP bit MUST be set to 0. If the message is being routed hop-by-hop, the HOP-BY-HOP bit MUST be set to 1 and the BUILT-ROUTE bit MUST be set to 0. (Note that there may be applications in which both the HOP-BY-HOP and the BUILD- ROUTE bit will be set to 0.)

If the NLS application wishes to support bidirectional reservations, the BIDIRECTIONAL flag must be set to 1, the BUILD-ROUTE flag should be set to 1, and the HOP-BY-HOP flag should be set to 0, at least in the initial message. If the application makes use of periodic refreshes it may optionally choose to route some number of them hop-by-hop along the discovered path before sending out another message to refresh the route state; that is an application design issue.

In this version of the protocol, each NLS message must fit in one datagram. An NLS-TL message originator should perform PMTU discovery in order to avoid exceeding path MTU size.

4. Messaging and state maintenance

Message handling and state maintenance are determined by the presence (or absence) of two flags in the NLS-TL header: the HOP-BY-HOP bit and the BUILD-ROUTE bit. They also involve, and are involved by, NAT processing.

4.1 BUILD-ROUTE

The BUILD-ROUTE bit in the flags field of the NLS-TL header allows NLS-TL to function as a discovery and routing protocol, much like the Path message described in RFC 2205.

If the BUILD-ROUTE flag is present in a NLS-TL message, upon receipt a NLS node MUST check for the presence of an IPv4_HOP or IPv6_HOP TLV in the NLS-TL payload. If one is not present, the message MUST be discarded and an error returned to the sender. If both are present, the message MUST be discarded and an error returned to the sender. Otherwise, if there is no installed soft state associated with the Flow ID_ID, the node stores the HOP information, Flow ID, and other state information it chooses to retain, and forwards the message towards the address in the destination field of its IP header. If there is installed soft state associated with the Flow ID, the node compares the contents of the HOP field with the installed state. If they are identical nothing needs to be done; if they are different the HOP information in the node is overwritten with the information in the current message. This allows the protocol to be responsive to route changes, endpoint mobility, and so on.

A NLS node MAY send notification of a routing change back to the sender.

4.2 HOP-BY-HOP

If the HOP-BY-HOP bit is set in the flags field of the NLS-TL header, a NLS node MUST forward the message to the address stored in associated local soft state. That is to say, the node MUST write the address in the local HOP information associated with the MESSAGE_IDFlow ID into the destination field in the IP header on the outbound message. This is like message processing in the Resv message in RFC 2205.

The HOP information may have been acquired using a routing process based on HOP-BY-HOP processing, but it may have been acquired using an external routing mechanism. If there is no HOP information stored locally, the node MUST drop the message and return an error to the sender.

4.3 BIDIRECTIONAL

If the BIDIRECTIONAL flag is set, the receiver must send the answering message to the sender (that is to say, the destination address in the IP header must be set to the address of the sender) with the BUILD_ROUTE flag set and the HOP_BY_HOP flag set to 0. As with the message sent from the sender to the receiver, the HOP TLV contains information used to install routing state. If the nodes are already authenticated to one another (they were already traversed in the forward direction) it is unnecessary for the authentication dialogue to be performed again. If the nodes are not already authenticated to one another then the route is asymmetric and the authentication dialogue must be performed.

Note that the sender and receiver should retain knowledge that the session is bidirectional, as it may affect subsequent messaging and error processing.

Because a complete authentication dialogue may take place in each direction, with each node being authenticated to its adjacent node (i.e., the dialogue takes care of authenticating both A to B and B to A), this proposal neither changes the authentication dialogue nor should it undermine the security of the protocol.

4.4 Path Teardown Messages

Receipt of a NLS message with the TEARDOWN bit set indicates that matching path state must be deleted. Note that this is independent of directionality, and the teardown message may be sent in either direction. The applications which have reservations that were installed by a message containing a matching Flow ID must be notified, and they are responsible for managing (in this case, deleting) their own flow-related state. TEARDOWN and HOP-BY-HOP MUST not be set in the same message.

Unlike RFC 2205, if there is no matching path state the teardown message must be forwarded. There may be path state in support of an NLS application that is not running on every node, and the teardown message must not be lost.

4.5 Network Address Translation

If there is one or more NAT_ADDRESS TLVs present, an NLS- capable NAT must process each one that has does not have the NO_TRANSLATE bit set in the flags field. Processing takes place as follows:

- o The originator (sender) of the message creates a NAT_ADDRESS TLV for each address/port/protocol tuple requiring NAT mappings. It also creates a random 32- bit tag, which is used to identify the address in application payloads and to tag the mapping in the NAT_ADDRESS TLV in the NLS-TL header. It also sets the TRANSLATE bit in the flags field and zeros the Mapped Address field.

- When an NLS-capable NAT receives a request, for each NAT_ADDRESS TLV in which the NO_TRANSLATE bit is not set and the Mapped Address is all nulls, it creates a NAT table mapping for the Original Address and Original Port and inserts the "external" address and port into the Mapped Address and Mapped Port fields.
- When an NLS-capable NAT receives a request, for each NAT_ADDRESS TLV in which the NO_TRANSLATE bit is not set and the Mapped Address is not nulls, it creates a NAT table mapping for the Mapped Address and Mapped port and overwrites those values with the new external addresses and ports.
- When an NLS-capable node receives a request, for reach NAT_ADDRESS TLV in which the Application ID matches an NLS application payload ID and the application is supported by the node, the TLV is passed to the application with the application payload, allowing the application module on the node to correlate and use the address based on the tag [and the Original Address?]

Note that this approach to NAT requires that participants be sensitive to directional issues in cases where ordering matters, such as the need to find the outermost NAT address. API support is required in order to turn the NO_TRANSLATE bit on and off as needed by a particular application.

Also note that in cases where the only function required is NAT table mapping requests, there may be no application payloads, or it may be desirable to create a rudimentary NAT NLS application that does nothing other than allow the receiver, or other nodes, to turn the NO_TRANSLATE bit on.

5. Application Interface

Application payloads are encapsulated within NLS-TL TLVs, and MUST follow any NAT TLVs.

The Application Payload TLV carries includes the Application ID field, which is used to vector the requests off to the correct application on the router upon receipt. It is also used to identify NAT_ADDRESS TLVs to be passed to the application. In a nutshell, if the Application ID in a NAT_ADDRESS TLV matches the Application ID in an Application TLV, the NAT_ADDRESS TLV must be passed to the application along with the application payload.

The Length field carries the total application payload length, excluding the header, in octets. The length must be at least 4 and be a multiple of 4. It may be necessary for an application to pad its payload to accomplish that.

Note that there is no identifier in the TLV other than the Application ID. If there is a need for an application-specific identifier for reservations or other applications requiring retained state, those must be added to the application payload.

6. NAT Interactions

NLS uses IP addresses for routing, both end-to-end and hop-by-hop. Given the applications which NLS-TL will be transporting, it is highly likely that those applications will be using payload-embedded addresses and there will be some interactions. The use of a NAT application together with other applications can mitigate this, but there will be problems transiting non-NLS-capable NATs.

When an NLS entity receives an TL message traveling in the forward direction, it writes the address in the IPv4_HOP or IPv6_HOP, as appropriate, from the packet into local per-session state and replaces the HOP data in the message with the address of the outgoing interface. When the entity is a NAT, it will write the translated-to address. Note that while it is usually the case that payload integrity protection breaks in the presence of NATs if embedded addresses are being rewritten, this is not substantially different from the rewriting of the HOP field which occurs within NLS anyway.

However, if an NLS message crosses a non-NLS-capable NAT, several problems may occur. The first is that if the message is being dropped in a raw IP packet, the NAT may simply drop the packet because it doesn't know how to treat it. Another is that the address in the HOP field will be incorrect. NLS and the applications it carries cannot be expected to function properly across non-participating NATs. Discovery of a non-NLS-capable NAT is described in section 8

7. Using NLS-TL as a stand-alone NAT traversal protocol

Using the NLS Transport Layer as a stand-alone NAT traversal protocol is straightforward -- simply use the TL without application payloads, but set the NO_REWRITE flag in the NAT_ADDRESS TLV to 1. This provides two functions: 1) installation of new NAT table mappings, and 2) allowing the sender to learn what the "external" mappings are. The Application ID field in the NAT_ADDRESS TLV must be set to 0.

The TL header flags in the forward direction must be

```
HOP-BY-HOP = 0
BUILD-ROUTE = 1
TEARDOWN = 0
```

The TL header flags in the reverse direction (i.e., in the response message) must be

```
HOP-BY-HOP = 1
BUILD-ROUTE = 0
TEARDOWN = 0
```

The NAT table mappings are kept fresh through the retransmission of the request every refresh period. The refresh messages are identical to the original request message.

When the NAT table mappings are no longer required, the sender must send a teardown message containing the Flow ID of the installed mappings and with the TL flags set to

```
HOP-BY-HOP = 0
BUILD-ROUTE = 0
TEARDOWN = 1
```

An acknowledgement response message is not required. If there has been no refresh message received prior to the expiration of the timeout period, the NAT table mappings must be deleted when the timeout period ends.

8. Discovery of non-NLS NATs, and recovery

This section describes a method of discovering non-NLS NATs in the path, and a recovery-mechanism if one is discovered.

When there are non-NLS-capable NATs in the path, they will only be able to process or modify the IP/UDP header of the NLS-TL message and will not be able to understand or modify the NLS-TL message itself (including the NAT_ADDRESS_TLV inside).

If there are non-NLS NATs in the path the sender needs to be made aware of this, and it should be able to fall back to processing without NLS, using any other mechanisms that may be available. Also, the NLS_ NATs in the path which have allocated the NAT mappings based on NLS NAT_ADDRESS_TLV processing, need to be able to release these mappings.

The following algorithm can be applied for non-NLS NAT detection by NLS nodes :

```
    if (NAT_TL NAT_ADDRESS_TLV's mapped_addr == 0) {
        This NLS_TL NAT is first NLS_TL NAT in path
        if (NLS_TL packet's source IP address != NAT_ADDRESS_TLV's
        original_address) {
            This NLS_TL NAT is not the first in the path, and
            some non-NLS_TL NAT has touched this packet;
            send NLS_TL error message back to the sender
            with NLS_TL error-code = 4 (non-nls-nat in path)
        } else {
            This NLS_TL NAT is the first in the path, and no non-
            NLS_TL NAT has touched this packet;
            proceed with NLS_TL processing.
        }
    } else {
        This NLS_TL NAT is not the first NLS_TL NAT in path.
        if (NLS_TL packet's source IP address != NAT_ADDRESS_TLV's
        mapped_address) {
            Some non-NLS_TL NAT has touched this packet, send
            NLS_TL error message back to the sender with NLS_TL
            error-code = 4 (non-nls-nat in path)
        } else {
            No non-NLS_TL NAT has touched this packet; proceed
            with regular NLS_TL processing.
        }
    }
```

The NLS_TL error message will be relayed back to the sender. Intermediate NLS nodes should not be processing the NLS error message, but let this NLS packet be routed back to the sender.

Once the sender sees an NLS_TL error-message with Error-Code = 4 (non-nls-nat in path), it should resend the same NLS_TL message as earlier with the NAT_ADDRESS_TLV's Original IPv4 Address/Port/ Protocol as earlier and the Mapped IPv4 Address/Port as NULL, but should set the TEARDOWN flag in the NLS-TL header.

The intermediate NLS NATs in the path, upon seeing an NLS_TL message with the TEARDOWN bit set, should delete its local NAT mapping corresponding to the Flow ID and send the message on towards the receiver, traversing other NLS-capable NATs along the path which will also process the TEARDOWN message.

9. Endhost Processing

9.1 Sending

When a host or its proxy wishes to send an NLS request, it puts together the application payload and encapsulates it in a transport layer packet.

If the application needs to request NAT service because of its use of addresses for reservations, etc., it must create a random 32-bit tag for use as an address token in the application payload, and it must create a NAT_ADDRESS TLV in which it inserts the address and port for which it is requesting NAT service, as well as the 32-bit tag.

For example, in a hypothetical QoS application that needed NAT services for the address 209.4.89.110, TCP port 6603 in the flow description, it would generate the random tag 0x24924924, use that in the application payload instead of an address, and create a NAT_ADDRESS TLV with the following values:

```
Application ID = QoS
Flags = TRANSLATE
Proto = TCP
Address ID = 0x24924924
Original IPv4 Address = 209.4.89.110
Original Port = 6603
```

The endpoint also needs to set the flags that determine how path establishment and routing are to be handled on intermediate nodes. In some cases the application requires no stored state in NLS nodes or it simply requires a single NLS pass. Examples of this kind of application include topology discovery, tunnel endpoint discovery, or diagnostic triggers. In this case, in the NLS-TL header both the HOP-BY-HOP flag and the BUILD-ROUTE flag are set to 0.

If an application is establishing per-node state and wants the NLS transport layer to establish and pin NLS routing for it, as might be the case with a QoS application or a firewall pinholing application, the sending endpoint must set the BUILD-ROUTE flag to 1 and the HOP-BY-HOP flag to 0.

The endhost then UDP encapsulates the NLS-TL packet, and transmits it either as a raw IP packet or as a UDP packet.

9.2 Receiving

An NLS node "knows" that it's an endpoint or proxy when the following conditions are satisfied:

```
if (IP destination address == my address)  {
    if (HOP_BY_HOP)
        if (next hop data available)
            forward it on;
        else
            it's mine;
}
```

When an endpoint receives a packet and identifies it as terminating there, it demultiplexes the payload and passes the payload and associated NAT_ADDRESS data to the appropriate application.

If an application in the payload is not supported by the endpoint, the endpoint must return a message to the sender with an ERROR_CODE TLV with the error value set to 3 (Unrecognized application).

10. Intermediate node processing

The processing of NLS-TL packets at intermediate nodes is substantially the same as processing at endpoints. Upon the arrival of a request, the node demultiplexes the packet contents and vectors the application payloads off to their respective applications.

One major difference from endpoint processing is the handling of NAT requests by NAT intermediate nodes. When an NLS-capable NAT receives an NLS request, it checks for the presence of NAT_ADDRESS TLVs. For each NAT_TLV, it executes the process described in Section 4.5.

For state maintenance and forwarding, the node must follow the processes described in Section 4.1, Section 4.2, and Section 4.4.

11. Using NLS-TL to support bidirectional reservations

When an application that uses NLS-TL to transport reservation requests (for example, QoS reservations or firewall pinholes) and it wishes to make the request for a bidirectional data stream, the reservations should be made when the message is received in the "forward" direction. Note that this is a significant departure from the model used in RSVP and assumed in previous versions of NLS-TL. The reason for this should be apparent -- if the route between the sender and receiver is asymmetric, it is possible that a device traversed by a PATH message may not be traversed by a RESV message, and vice-versa.

It may be desirable to have different characteristics for the reservation in one direction than for the other. In this case the NLS application designer should make provision for identifying reservation specifications to be used in each direction.

It should also not be assumed, as is done in RSVP, that error messages will traverse all affected nodes unless care is taken by the sender, or the "owner" of the reservation, to ensure that error messages are propagated correctly. So, for example, if a reservation fails at a particular node, it may not be sufficient to return the error message towards the sender.

An application that manages reservations may wish to refresh application state more frequently than it wishes to refresh route state. In that case it should send the message with the BIDIRECTIONAL and HOP_BY_HOP flags set, and the BUILD_ROUTE flag set to 0.

12. Proxy Considerations

13. Security Considerations

13.1 Overview

This section describes a method for providing cryptographic authentication to the Network Layer Signaling (NLS) transport layer protocol. The method incorporates a peer discovery mechanism. Importantly, there is no provision for confidentiality. This fact simplifies the protocol, and removes the need for export control on products implementing it. NLS applications which require confidentiality may provide it themselves.

This mechanism provides both entity and message authentication along a single hop. In other words, the device on each end of the hop is assured that the identity of the other device, and the content of the message from that device, are correct. These security services are provided only on a hop-by-hop basis. That is, there are no cryptographic services provided across multiple hops, and each hop can independently use or not use authentication. In the following, we restrict our discussion to a single hop along an NLS path.

In order to support authentication, we introduce an optional two-message exchange into NLS called the Authentication Exchange, or AX. This exchange is needed in order to carry the challenge-response information.

13.2 Security Model

Authenticated NLS-TL provides both authorization and entity authentication using a group model. Authorizations correspond to particular applications. An Authorization Group (AG) is a set of network interfaces that share the following information:

- o a list of NLS Application IDs; these correspond to applications which the group is authorized to use,
- o a group authentication key,
- o a Message Authentication Code (MAC) algorithm type

Note that AGs are associated with interfaces and not devices since in many situations there are different trust levels associated with different interfaces.

For each device implementing Authenticated NLS-TL, each interface is associated with a list of Application IDs, each of which is associated with:

- o a list of AGIDs that authorize the corresponding application, or

- the symbol ALLOW, which indicates that the application has been explicitly allowed on the associated interface, or
- the symbol DROP, which indicates that the application has been explicitly disallowed on the associated interface.

In this model, finer grained authorizations are impossible. For example, it is impossible to authorize VoIP traversal of a Firewall while still disallowing telnet across the firewall. The model can be expanded to accommodate finer grained authorizations, but this issue is not considered further in this draft. Sensitive applications, such as firewall pinholing, must provide their own authentication and authorization.

13.3 Cryptography

Authenticated NLS-TL uses a single cryptographic function: a pseudorandom function that accepts arbitrary-length inputs and produces fixed-length outputs. This function is used as a message authentication code (MAC). [Note: in the future, it might be used as a key derivation function (KDF).]

The default function is HMAC SHA1. When used as a MAC, its length is truncated to 96 bits.

13.3.1 Keys

Authenticated NLS-TL uses group keys, in order to reduce the amount of protocol state and to mitigate the peer-discovery problem.

Implementations MUST provide a way to set and delete keys manually. However, they SHOULD also provide an automated group key management system such as GDOI [rfc3547], so that efficient revocation is possible.

13.4 Datatypes

An NLS-TL message MSG has the following format:

```
MSG :== HDR OPT* APP SEC*
```

where HDR, OPT, APP, and SEC are as follows:

```
HDR is the NLS header
OPT is an NLS optional TLV
APP is the Application Object
SEC is an AGID, A_CHALLENGE, A_RESPONSE, B_CHALLENGE, or
B_RESPONSE.  These datatypes are defined below.
```

The security TLVs are always last in order to avoid data-formatting issues with the inputs to the message authentication codes, and to minimize the amount of data movement needed during the Authentication Exchange.

Authorization Group Identifier (AGID): The AGID TLV identifies a particular group key. The Value field carries an identifier; there is no defined format. The length of this field is variable, and MUST be a multiple of four octets. If it is generated at random, the it SHOULD be at least 16 octets.
A_CHALLENGE: The A_CHALLENGE contains a 16-octet random nonce. This TLV is put into a message whenever outbound authentication is desired. When this TLV is received, then the next message sent MUST contain either an A_RESPONSE TLV or an error message indicating that no authentication is possible. The value MUST be generated either by using a strong random or pseudorandom source, or by the method described in Section X.Y.
B_CHALLENGE: The B_CHALLENGE contains a 16-octet random nonce. This TLV is put into a message whenever inbound authentication is desired. When this TLV is received, then the following message MUST contain either a B_RESPONSE TLV or an error message indicating that no authentication is possible. The value MUST be generated either by using a strong random or pseudorandom source.
A_RESPONSE: The A_RESPONSE TLV is sent in response to a message containing an A_CHALLENGE TLV. It contains a message authentication code (MAC) value computed over the complete NLS message containing the A_CHALLENGE, including the NLS header.
B_RESPONSE: The B_RESPONSE is sent in response to a message containing a B_CHALLENGE TLV. It contains a message authentication code (MAC) value computed over the complete NLS message containing the IN_CHALLENGE, including the NLS header.

13.5 The Authentication Exchange (AX)

Two new NLS flags are defined:

0x0008 AX_CHALLENGE, which is set for all messages carrying an A_CHALLENGE TLV.
0x0016 AX_RESPONSE, which is set for all messages carrying an A_RESPONSE TLV.

In the following, we consider only the SEC TLVs.

```
1.  A -> B : AGID*, B_CHALLENGE
2.  B -> A : AGID, A_CHALLENGE, B_RESPONSE
3.  A -> B : AGID, A_RESPONSE
```

Message 1: Device A includes in the message each AGID that is associated with the Application ID in the NLS message to be sent to B. Device B checks its local policy to determine which AGIDs are associated with the Application ID in the message, and determines which AGIDs are associated with that value. Device B then checks to see if the AGID set in the message intersects with the locally derived AGID set. If they intersect, then one of the AGID values is chosen to be 'active'; this choice is arbitrary. Otherwise, the AX cannot be successfully completed, and an error message is returned. A also constructs a B_CHALLENGE TLV and sends it to device B.

Message 2: Device B constructs Message 2 by replacing the AGID list of Message 1 with the active AGID and an A_CHALLENGE TLV, as well as a B_RESPONSE TLV, and sends it to device A. The rest of the NLS message is unchanged from Message 1, except that the AX_CHALLENGE flag is now set. Device A processes Message 1 by

Verifying that the AGID in the message is associated with the Application ID in the NLS message. If it is not, then the AX cannot be successfully completed, and an error message is returned.
Computing its own value of B_RESPONSE, using as input the key associated with the AGID in the message, and a reconstruction of Message 3 created using the locally cached value of the A_CHALLENGE TLV. If the locally constructed B_RESPONSE matches that in Message 2, then the message is rejected, and an error message is returned.
Looking up the key associated with the AGID. If it cannot find an associated key, then the AX cannot be successfully completed, and an error message is returned.

If those steps succeed, then the A_RESPONSE TLV is computed, using Message 2 and the key associated with the active AGID as its input.

Message 3: Device A constructs Message 3 by replacing the A_CHALLENGE TLV with the A_RESPONSE TLV computed in the preceding step and a randomly generated B_CHALLENGE TLV. The rest of the NLS message is identical to that of Message 1, except that the AX_RESPONSE flag is set. Device B processes Message 3 by

Verifying that the AGID in the message is associated with the Application ID in the NLS message. If it is not, then the AX cannot be successfully completed, and an error message is returned.
Computing its own value of A_RESPONSE, using as input the key associated with the active AGID, and a reconstruction of Message 2 created using the locally cached value of the A_CHALLENGE TLV. If the locally constructed A_RESPONSE matches that in Message 3, then the message is rejected, and an error message is returned.

14. IANA Considerations

There are several parameters for which NLS-TL will need registry services. These include

- o a registry for NLS Application IDs (NLS Application Identifiers) and for
- o NLS-TL TLV identifiers (NLS TLVs).

Initial values are given below. Future assignments are to be made through expert review.

14.1 NLS Application Identifiers

NAME	VALUE	DEFINITION
Control Point Discovery	1	See ?
Firewall Traversal	2	See ?

14.2 NLS TLVs

NAME	VALUE	DEFINITION
NAT_ADDRESS	1	See section 2.3.2.1
APPLICATION_PAYLOAD	2	See section 2.3.2.2
TIMEOUT	3	See section 2.3.2.3
IPV4_HOP	4	See section 2.3.2.4
IPV6_HOP	5	See section 2.3.2.5
IPV4_ERROR_CODE	6	See section 2.3.2.6
IPV6_ERROR_CODE	7	See section 2.3.2.7
AGID	8	See section 2.3.2.8
CHALLENGE	9	See section 2.3.2.9
RESPONSE	10	See section 2.3.2.10

NLS-TL also requires a protocol number when operating in raw datagram mode and a port number when using UDP for transport.

15 References

[rfc1633] Braden, R., Clark, D. and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.

[rfc2205] Braden, R., Zhang, L., Berson, S. and S. Herzog, "Resource Reservation Protocol -- Version 1 Functional Specification", RFC 2205, September 1997.

[rfc2711] Partridge, C. and A. Jackson, "IPv6 Router Alert Option", October 1999.

[rfc2961] Berger, L., Gan, D., Swallow, G., Pan, P., Tommasi, F. and S. Molendini, "RSVP Refresh Overhead Reduction Extensions", RFC 2961, April 2001.

[rfc3547] Baugher, M., Weis, B., Hardjono, T. and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.

[ylonen]   Ylonen, T., "SSH Protocol Architecture", draft-ietf-secsh-architecture-15.txt (work in progress), October 2003.

Authors' Addresses

Melinda Shore
Cisco Systems
809 Hayts Road
Ithaca, New York 14850
USA

EMail: mshore@cisco.com

Kaushik Biswas
Cisco Systems
510 McCarthy Blvd
Milpitas, California 95035
USA

EMail: kbiswas@cisco.com

David A. McGrew
Cisco Systems
510 McCarthy Blvd
Milpitas, California 95035
USA

EMail: mcgrew@cisco.com

Appendix A. Acknowledgements

The authors would like to express their gratitude to Jan Vilhuber, Senthil Sivakumar, and Bill Foster for their careful review and feedback.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Annex B

QoS requirements

(This annex forms an integral part of this Recommendation)

The policy server within IPCablecom MAY support this interface Recommendation as a requestor.

For DQoS (CR TYPE = 2) and for IPCablecom multimedia (CR TYPE = 3), the CR ID is either the default value (set to "1") or whatever value has been provisioned for that CMTS.

Caching: In the case of QoS, the requestor may decide to cache responses and may request subnet information to avoid making further requests for endpoints within the same subnet. Cache invalidation can be determined when a control request is made (i.e., an error receives as a result of a gate-set). It is left to policy and implementation as to whether the requestor consequently clears its entire cache or if it does so only for that subnet.

The CMTS MUST support this interface Recommendation as a control point for both IPCablecom multimedia QoS and DQoS.

Appendix I

Open issues

(This appendix does not form an integral part of this Recommendation)

- IANA registration – Need to register a UDP port number with IANA. Clauses 8.1 and 8.2 and Annex A need to be updated once the port is assigned.
- Status of Annex A – Need to consider integrating the NLS protocol definition into the Recommendation to avoid having to reference an informative RFC.

Bibliography

[b-ITU-T J.160]	ITU-T Recommendation J.160 (2005), *Architectural framework for the delivery of time-critical services over cable television networks using cable modems*.
[b-ITU-T J.163]	ITU-T Recommendation J.163 (2005), *Dynamic quality of service for the provision of real-time services over cable television networks using cable modems.*
[b-ITU-T J.179]	ITU-T Recommendation J.179 (2005), *IPCablecom support for multimedia*.
[b-ITU-T J.366.4]	ITU-T Recommendation J.366.4 (2006), *IPCablecom2 IP Multimedia Subsystem (IMS): Session Initiation Protocol (SIP) and Session Description Protocol (SDP) – Stage 3 specification. (3GPP TS 24.229)*
[b-IETF RFC 4183]	IETF RFC 4183 (2005), *A Suggested Scheme for DNS Resolution of Networks and Gateways*.

Printed in Switzerland
Geneva, 2007