Next Steps in Signaling
Internet-Draft
Expires: August 13, 2006

H. Schulzrinne
Columbia U.
R. Hancock
Siemens/RMR
February 9, 2006

GIST: General Internet Signaling Transport
draft-ietf-nsis-ntlp-09

Status of this Memo

This Internet-Draft will expire on August 13, 2006.

Copyright Notice

Abstract

This document specifies protocol stacks for the routing and transport of per-flow signaling messages along the path taken by that flow through the network. The design uses existing transport and security protocols under a common messaging layer, the General Internet Signaling Transport (GIST), which provides a universal service for diverse signaling applications. GIST does not handle signaling application state itself, but manages its own internal state and the

configuration of the underlying transport and security protocols to enable the transfer of messages in both directions along the flow path. The combination of GIST and the lower layer transport and security protocols provides a solution for the base protocol component of the "Next Steps in Signaling" framework.

Table of Contents

1. Introduction

Signaling involves the manipulation of state held in network elements. 'Manipulation' could mean setting up, modifying and tearing down state; or it could simply mean the monitoring of state which is managed by other mechanisms.

This specification concentrates on "path-coupled" signaling, which involves network elements which are located on the path taken by a particular data flow, possibly including but not limited to the flow endpoints. Indeed, there are almost always more than two participants in a path-coupled signaling session, although there is no need for every node on the path to participate. Path-coupled signaling thus excludes end-to-end higher-layer application signaling (except as a degenerate case) such as ISUP (telephony signaling for Signaling System #7) messages being transported by SCTP between two nodes.

In the context of path-coupled signaling, examples of state management include network resource allocation (for "resource reservation"), firewall configuration, and state used in active networking; examples of state monitoring are the discovery of instantaneous path properties (such as available bandwidth, or cumulative queuing delay). Each of these different uses of path-coupled signaling is referred to as a signaling application.

Every signaling application requires a set of state management rules, as well as protocol support to exchange messages along the data path. Several aspects of this protocol support are common to all or a large

number of signaling applications, and hence can be developed as a common protocol. The NSIS framework given in [22] provides a rationale for a function split between the common and application specific protocols, and gives outline requirements for the former, the 'NSIS Transport Layer Protocol' (NTLP). The application specific protocols are referred to as 'NSIS Signaling Layer Protocols' (NSLPs), and are defined in separate documents.

This specification provides a concrete solution for the NTLP. It is based on the use of existing transport and security protocols under a common messaging layer, the General Internet Signaling Transport (GIST). GIST does not handle signaling application state itself; in that crucial respect, it differs from application signaling protocols such as SIP, RTSP, and the control component of FTP. Instead, GIST manages its own internal state and the configuration of the underlying transport and security protocols to ensure the transfer of signaling messages on behalf of signaling applications in both directions along the flow path.

The structure of this specification is as follows. Section 2 defines terminology, and Section 3 gives an informal overview of the protocol design principles and operation. The normative specification is contained mainly in Section 4 to Section 8. Section 4 describes the message sequences and Section 5 their format and contents (note that the detailed bit formats are given in Appendix A). The protocol operation is captured in the form of state machine language in Section 6. Section 7 describes some more advanced protocol features and security considerations are contained in Section 8. In addition, Section 9 gives the IANA considerations, Appendix B describes an abstract API for the service which GIST provides to signaling applications, and Appendix C provides an example message flow.

1.1. Restrictions on Scope

This section briefly lists some important restrictions on GIST applicability and functionality. In some cases, these are implicit consequences of the functionality split developed in the NSIS framework; in others, they are restrictions on the types of scenario in which GIST can operate correctly.

Flow splitting: In some cases, e.g. where packet-level load sharing

has been implemented, the path taken by a single flow in the network may not be well defined. If this is the case, GIST cannot route signaling meaningfully. (In some circumstances, GIST implementations could detect this condition, but even this cannot be guaranteed.)

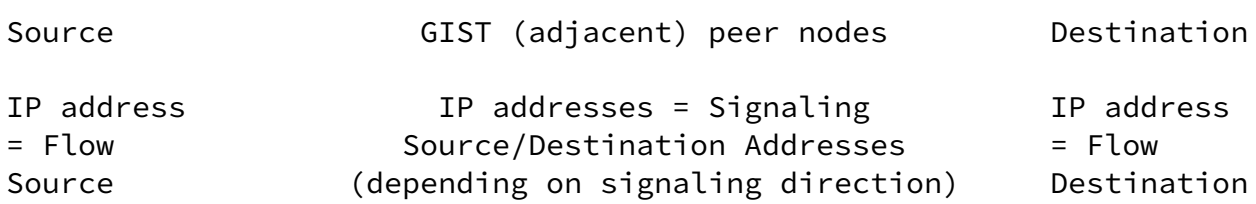
Multicast: GIST does not handle multicast flows. This includes 'classical' IP multicast and any of the 'small group multicast' schemes recently proposed.

Legacy NATs: GIST messages will generally pass through NATs, but unless the NAT is GIST-aware, any addressing data carried in the payload will not be handled correctly. There is a dual problem of whether the GIST peers either side of the boundary can work out how to address each other, and whether they can work out what translation to apply to the signaling packet payloads. The fundamental problem is that GIST messages contain 3 or 4 interdependent addresses which all have to be consistently translated, and existing generic NAT traversal techniques such as STUN [19] or TURN [20] can process only two. (Appropriate behaviour for a GIST-aware NAT is discussed in Section 7.2.)

2. Requirements Notation and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [2].

The terminology used in this specification is fully defined in this section. The basic entities relevant at the GIST level are shown in Figure 1.



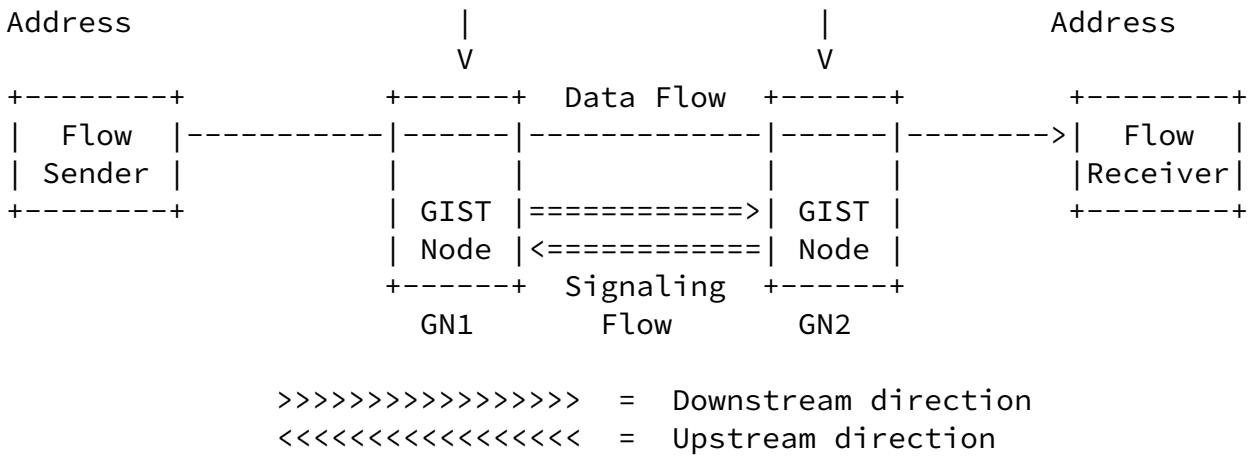


Figure 1: Basic Terminology

[Data] Flow: A set of packets identified by some fixed combination of header fields. Flows are unidirectional (a bidirectional communication is considered a pair of unidirectional flows).

Session: A single application layer flow of information for which some state information is to be manipulated or monitored. See Section 3.4 for further detailed discussion.

[Flow] Sender: The node in the network which is the source of the packets in a flow. Could be a host, or a router (e.g. if the flow is actually an aggregate).

[Flow] Receiver: The node in the network which is the sink for the packets in a flow.

Downstream: In the same direction as the data flow.

Upstream: In the opposite direction to the data flow.

GIST Node: Any node along the data path supporting GIST (regardless of what signaling applications it supports).

[Adjacent] Peer: The next node along the data path, in the upstream

or downstream direction, with which a GIST node explicitly interacts. The GIST peer discovery mechanisms implicitly determine whether two nodes will be adjacent. It is possible for adjacencies to 'skip over' intermediate nodes which decide not to take part in the signaling exchange at the NTLP layer; even if such nodes process parts of the signaling messages, they store no state about the session and are never explicitly visible at the GIST level to the nodes either side.

Datagram Mode: A mode of sending GIST messages between nodes without using any transport layer state or security protection. Datagram mode uses UDP encapsulation, with IP addresses derived either from the flow definition or previously discovered adjacency information.

Connection Mode: A mode of sending GIST messages directly between nodes using point to point "messaging associations" (see below). Connection mode allows the re-use of existing transport and security protocols where such functionality is required.

Messaging Association: A single connection between two explicitly identified GIST adjacent peers, i.e. between a given signaling source and destination address. A messaging association may use a specific transport protocol and known ports. If security protection is required, it may use a specific network layer security association, or use a transport layer security association internally. A messaging association is bidirectional; signaling messages can be sent over it in either direction, and can refer to flows of either direction.

Message Routing Method: There can be different algorithms for discovering the route that signaling messages should take. These are referred to as message routing methods, and GIST supports alternatives within a common protocol framework. See Section 3.3.

Transfer Attributes: A description of the requirements which a signaling application has for the delivery of a particular message; for example, whether the message should be delivered reliably. See Section 4.1.2.

3. Design Overview

3.1. Overall Design Approach

The generic requirements identified in the NSIS framework [22] for transport of path-coupled signaling messages are essentially two-fold:

"Routing": Determine how to reach the adjacent signaling node along each direction of the data path (the GIST peer), and if necessary explicitly establish addressing and identity information about that peer;

"Transport": Deliver the signaling information to that peer.

To meet the routing requirement, one possibility is for the node to use local routing state information to determine the identity of the GIST peer explicitly. GIST defines a 3-way handshake (Query/ Response/optional Confirm) which sets up the necessary routing state between adjacent peers, during which signaling applications can also exchange data; the Query message is encapsulated in a special way, depending on the message routing method, in order to probe the network infrastructure so that the correct peer will intercept it. If the routing state does not exist, GIST may be able to send a message anyway, with the same encapsulation as for a Query message.

Once the routing decision has been made, the node has to select a mechanism for transport of the message to the peer. GIST divides the transport problems into two categories, the easy and the difficult. It handles the easy cases internally, and uses well-understood transport protocols for the harder cases. Here, with details discussed later, "easy" messages are those that are sized well below the lowest MTU along a path, are infrequent enough not to cause concerns about congestion and flow control, and do not need security protection or guaranteed delivery.

In [22] all of these routing and transport requirements are assigned to a single notional protocol, the 'NSIS Transport Layer Protocol' (NTLP). The strategy of splitting the transport problem leads to a layered structure for the NTLP, of a specialised GIST 'messaging' layer running over standard transport and security protocols, as shown in Figure 2. This also shows GIST offering its services to upper layers at an abstract interface, the GIST API, further discussed in Section 4.1.

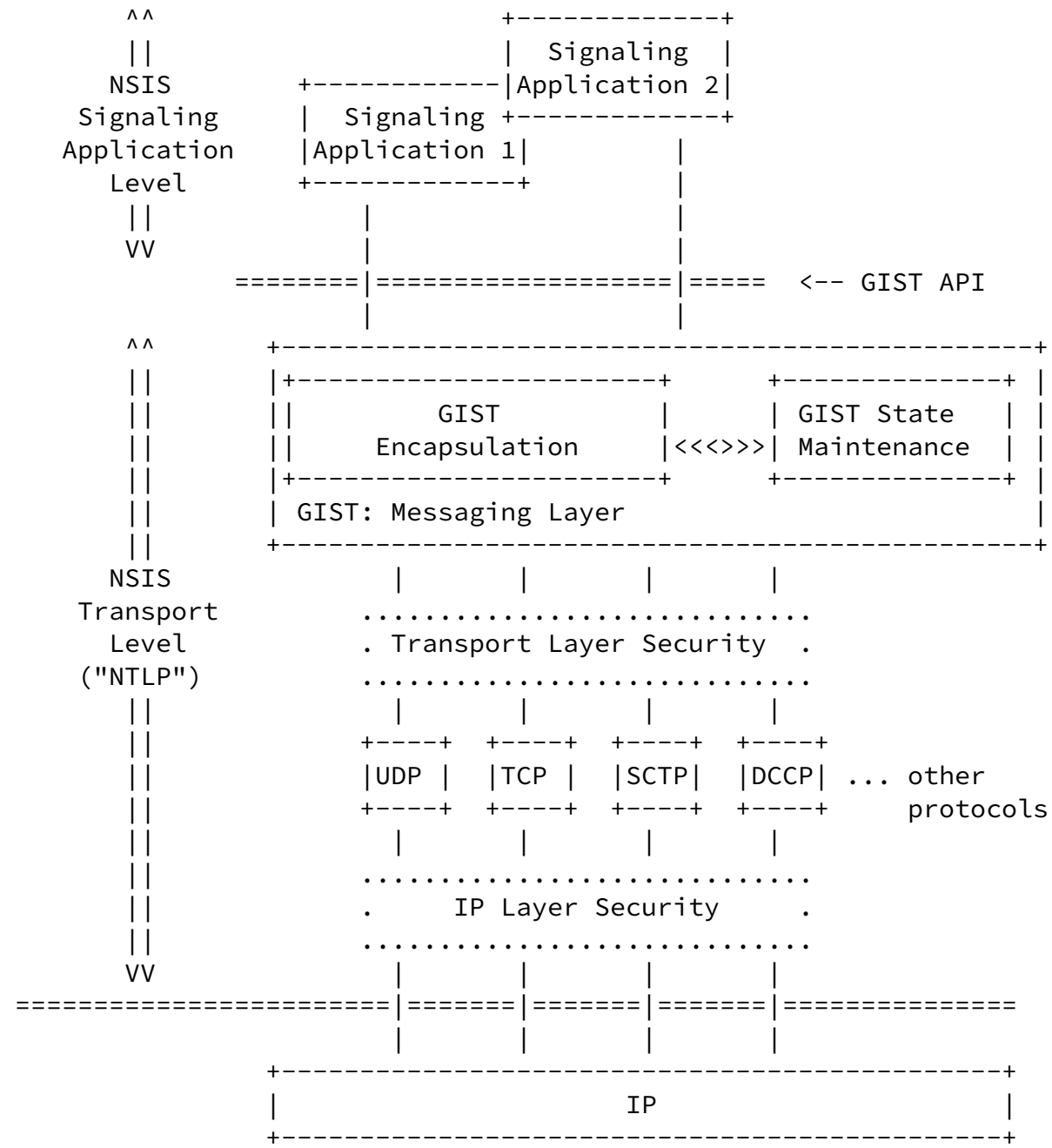


Figure 2: Protocol Stacks for Signaling Transport

3.2. Modes and Messaging Associations

Internally, GIST has two modes of operation:

Datagram mode ('D mode') is used for small, infrequent messages with modest delay constraints; it is also used at least for the Query message of the 3-way handshake.

Connection mode ('C mode') is used for larger data objects or where fast state setup in the face of packet loss is desirable, or where channel security is required.

Datagram mode uses UDP, as this is the only encapsulation which does not require per-message shared state to be maintained between the peers. The connection mode can in principal use any stream or message-oriented transport protocol; this specification defines TCP as the initial choice. It can in principal employ specific network layer security associations, or an internal transport layer security association; this specification defines TLS as the initial choice. When GIST messages are carried in connection mode, they are treated just like any other traffic by intermediate routers between the GIST peers. Indeed, it would be impossible for intermediate routers to carry out any processing on the messages without terminating the transport and security protocols used.

It is possible to mix these two modes along a path. This allows, for example, the use of datagram mode at the edges of the network and connection mode in the core of the network. Such combinations may make operation more efficient for mobile endpoints, while allowing multiplexing of signaling messages across shared security associations and transport connections between core routers.

It must be understood that the routing and transport decisions made by GIST are not independent. If the message transfer has requirements that require connection mode (e.g. the message is so large that fragmentation is required), this can only be used between explicitly identified nodes. In such cases, GIST carries out the 3-way handshake initially in datagram mode to identify the peer and then sets up the necessary connections if they do not already exist. It must also be understood that the signaling application does not make the D/C mode selection directly; rather, this decision is made by GIST on the basis of the message characteristics and the transfer attributes stated by the application. The distinction is not visible at the GIST service interface.

In general, the state associated with connection mode messaging to a particular peer (signaling destination address, protocol and port

numbers, internal protocol configuration and state information) is referred to as a "messaging association". There may be any number of messaging associations between two GIST peers (although the usual case is 0 or 1), and they are set up and torn down by management actions within GIST itself.

3.3. Message Routing Methods

The baseline message routing functionality in GIST is that signaling

messages follow a route defined by an existing flow in the network, visiting a subset of the nodes through which it passes. This is the appropriate behaviour for application scenarios where the purpose of the signaling is to manipulate resources for that flow. However, there are scenarios for which other behaviours are applicable. Two examples are:

Predictive Routing: Here, the intent is to signal along a path that the data flow may follow in the future. Possible cases are pre-installation of state on the backup path that would be used in the event of a link failure; and predictive installation of state on the path that will be used after a mobile node handover.

NAT Address Reservations: This applies to the case where a node behind a NAT wishes to reserve an address at which it can be reached by a sender on the other side. This requires a message to be sent outbound from what will be the flow receiver although no reverse routing state for the flow yet exists.

Most of the details of GIST operation are independent of which alternative is being used. Therefore, the GIST design encapsulates the routing-dependent details as a message routing method (MRM), and allows multiple MRMs to be defined. The default is the path-coupled MRM, which corresponds to the baseline functionality described above; a second MRM for the NAT Address Reservation case is also defined.

The content of a MRM definition is as follows, using the path-coupled MRM as an example:

- The format of the information that describes the path that the signaling should take, the Message Routing Information (MRI). For the path-coupled MRM, this is just the Flow Identifier (see

Section 5.8.1.1) and some additional control information. Specifically, the MRI always includes a flag to distinguish between the two directions that signaling messages can take, denoted 'upstream' and 'downstream'.

- A specification of the IP level encapsulation of the Query messages which probe the network to discover the adjacent peers. A downstream encapsulation must be defined; an upstream encapsulation is optional. For the path-coupled MRM, this information is given in Section 5.8.1.2 and Section 5.8.1.3.

- A specification of what validation checks GIST should apply to the probe messages, for example to protect against IP address spoofing attacks. The checks may be dependent on the direction (upstream/downstream) of the message. For the path-coupled MRM, the downstream validity check is basically a form of ingress

filtering, also discussed in Section 5.8.1.2.

- The mechanism(s) available for route change detection, i.e. any change in the neighbour relationships that the MRM discovers. The default case for any MRM is soft-state refresh, but additional supporting techniques may be possible; see Section 7.1.2.

In addition, it should be noted that NAT traversal almost certainly requires transformation of the MRI field in GIST messages (see Section 7.2). Although the transformation does not have to be defined as part of the standard, the impact on existing GIST-aware NAT implementations should be considered.

The MRI is passed explicitly between signaling applications and GIST; therefore, signaling application specifications must define which MRMs they require (they may use more than one, e.g. depending on the type of message). Signaling applications may use fields in the MRI in their packet classifiers; if they use additional information for packet classification, this would be carried at the NSLP level and so would be invisible to GIST. Any node hosting a particular signaling application MUST use a GIST implementation that supports the corresponding MRMs. The GIST processing rules enforce that nodes which do not host the signaling application are not forced to handle messages for it at the GIST level, so it does not matter if they support the MRM or not.

3.4. Signaling Sessions

GIST allows signaling applications to associate each message with a "signaling session". Informally, given an application layer exchange of information for which some network control state information is to be manipulated or monitored, the corresponding signaling messages should be associated with the same session. Signaling applications provide the session identifier (SID) whenever they wish to send a message, and GIST reports the SID when a message is received.

Most GIST processing and state information is related to the flow (defined by the MRI, see above) and signaling application (given by the NSLPID, see below). There are several possible relationships between flows and sessions, for example:

- The simplest case is that all messages for the same flow have the same SID.
- Messages for more than one flow may use the same SID, for example because one flow is replacing another in a mobility or multihoming scenario.

- A single flow may have messages for different SIDs, for example from independently operating NSLPIDs.

Because of this range of options, GIST does not perform any validation on how signaling applications map between flows and sessions, nor does it perform any validation on the properties of the SID itself. In particular, when a new SID is needed, logically it should be generated by the signaling application. (NSIS implementations could provide common functionality to generate SIDs for use by any signaling application, but this is not part of GIST.) GIST only defines the syntax of the SID as an opaque 128-bit identifier.

The SID assignment has the following impact on GIST processing:

- Messages with the same SID to be delivered reliably between the same GIST peers are delivered in order.

- All other messages are handled independently.

- GIST identifies routing state (upstream and downstream peer) by the triplet (MRI, NSLPID, SID).

Strictly, the routing state should not depend on the SID. However, if the routing state is keyed only by (MRI, NSLPID) there is a trivial denial of service attack (see Section 8.3) where a malicious off-path node asserts that it is the peer for a particular flow. Instead, the routing state is also segregated between different SIDs, which means that the attacking node can only disrupt a signaling session if it can guess the corresponding SID. A consequence of this design is that signaling applications should choose SIDs so that they are cryptographically random, and should not use several SIDs for the same flow unless strictly necessary, to avoid additional load from routing state maintenance.

3.5. Signaling Applications and NSLPIDs

The functionality for signaling applications is supported by NSIS signaling layer protocols (NSLPs). Each NSLP is identified by a 16 bit NSLPID, assigned by IANA (Section 9). A single signaling application (e.g. "resource reservation") may define a family of NSLPs to implement its functionality, for example to carry out signaling operations at different levels in a hierarchy (cf. [15]). However, the interactions between the different NSLPs (for example, to relate aggregation levels or aggregation region boundaries in the resource management case) are handled at the signaling application level; the NSLPID is the only information visible to GIST about the signaling application being used.

3.6. Example of Operation

This section presents an example of GIST usage in a relatively simple (in particular, NAT-free) signaling scenario, to illustrate its main features.

Consider the case of an RSVP-like signaling application which allocates resources for a single unicast flow. We will consider how GIST transfers messages between two adjacent peers along the path, GN1 and GN2 (see Figure 1). In this example, the end-to-end exchange is initiated by the signaling application instance in the sender; we

take up the story at the point where the first message is being processed (above the GIST layer) by the signaling application in GN1.

1. The signaling application in GN1 determines that this message is a simple description of resources that would be appropriate for the flow. It determines that it has no special security or transport requirements for the message, but simply that it should be transferred to the next downstream signaling application peer on the path that the flow will take.

2. The message payload is passed to the GIST layer in GN1, along with a definition of the flow and description of the message transfer attributes {unsecured, unreliable}. GIST determines that this particular message does not require fragmentation and that it has no knowledge of the next peer for this flow and signaling application; however, it also determines that this application is likely to require secured upstream and downstream transport of large messages in the future. This determination is a function of node-local policy interactions between GIST and the signaling application.

3. GN1 therefore constructs a GIST-Query message, a UDP datagram carrying the NSLP payload and additional payloads at the GIST level to be used to initiate a messaging association. The Query is injected into the network, addressed towards the flow destination and with a Router Alert Option included.

4. The Query message passes through the network towards the flow receiver, and is seen by each router in turn. GIST-unaware routers will not recognise the RAO value and will forward the message unchanged; GIST-aware routers which do not support the NSLP in question will also forward the message basically unchanged, although they may need to process more of the message to decide this.

5. The message is intercepted at GN2. The GIST layer identifies the message as relevant to a local signaling application, and passes

the NSLP payload and flow description upwards to it. The signaling application in GN2 indicates to GIST that it will peer with GN1 and so GIST should proceed to set up any routing state. In addition, the signaling application continues to process the

message as in GN1 (compare step 1), and this will eventually result in the message reaching the flow receiver.

6. In parallel, the GIST instance in GN2 now knows that it should maintain routing state and a messaging association for future signaling with GN1. This is recognised because the message is a GIST-Query, and because the local signaling application has indicated that it will peer with GN1. There are two basic possible cases for sending back the necessary GIST-Response:

   A. GN1 and GN2 already have an appropriate messaging association. GN2 simply records the identity of GN1 as its upstream peer for that flow and NSLP, and sends a GIST-Response back to GN1 over the association identifying itself as the peer for this flow.

   B. No messaging association exists. GN2 sends the GIST-Response in D mode directly to GN1, identifying itself and agreeing to the association setup. The protocol exchanges needed to complete this will proceed in the background.

7. Eventually, another NSLP message works its way upstream from the receiver to GN2. This message contains a description of the actual resources requested, along with authorisation and other security information. The signaling application in GN2 passes this payload to the GIST level, along with the flow definition and transfer attributes {secured, reliable}.

8. The GIST layer in GN2 identifies the upstream peer for this flow and NSLP as GN1, and determines that it has a messaging association with the appropriate properties. The message is queued on the association for transmission (this may mean some delay if the procedures begun in step 6.B have not yet completed).

Further messages can be passed in each direction in the same way. The GIST layer in each node can in parallel carry out maintenance operations such as route change detection (see Section 7.1).

It should be understood that several of these details of GIST operations can be varied, either by local policy or according to signaling application requirements. The authoritative details are contained in the remainder of this document.

4. GIST Processing Overview

This section defines the basic structure and operation of GIST. Section 4.1 describes the way in which GIST interacts with (local) signaling applications in the form of an abstract service interface. Section 4.2 describes the per-flow and per-peer state that GIST maintains for the purpose of transferring messages. Section 4.3 describes how messages are processed in the case where any necessary messaging associations and routing state already exist; this includes the simple scenario of pure datagram mode operation, where no messaging associations are necessary in the first place. Finally, Section 4.4 describes how routing state and messaging associations are created and managed.

4.1. GIST Service Interface

This section defines the service interface that GIST presents to signaling applications in terms of abstract properties of the message transfer. Note that the same service interface is presented at every GIST node; however, applications may invoke it differently at different nodes (e.g. depending on local policy). In addition, the service interface is defined independently of any specific transport protocol, or even the distinction between datagram and connection mode. The initial version of this specification defines how to support the service interface using a connection mode based on TCP; if additional protocol support is added, this will support the same interface and so the change will be invisible to applications (except as a possible performance improvement). A more detailed description of this service interface is given in Appendix B.

4.1.1. Message Handling

Fundamentally, GIST provides a simple message-by-message transfer service for use by signaling applications: individual messages are sent, and individual messages are received. At the service interface, the NSLP payload (which is opaque to GIST) is accompanied by control information expressing the application's requirements about how the message should be routed, and the application also provides the session identifier (see Section 3.4). Additional message transfer attributes control the specific transport and security properties that the signaling application desires for the message.

The distinction between GIST connection and datagram modes is not visible at the service interface. In addition, the invocation of GIST functionality to handle fragmentation and reassembly, bundling together of small messages (for efficiency), and congestion control

is not directly visible at the service interface; GIST will take

whatever action is necessary based on the properties of the messages and local node state.

4.1.2. Message Transfer Attributes

Message transfer attributes are used to define certain performance and security related aspects of message processing. The attributes available are as follows:

Reliability: This attribute may be 'true' or 'false'. For the case 'true', messages MUST be delivered to the signaling application in the peer exactly once or not at all; if there is a chance that the message was not delivered, an error MUST be indicated to the local signaling application identifying the routing information for the message in question. Messages with the same SID to the same peer MUST be delivered in order. For the case 'false', a message may be delivered, once, several times or not at all, with no error indications in any case.

Security: This attribute defines the security properties that the signaling application requires for the message, including the type of protection required, and what authenticated identities should be used for the signaling source and destination. This information maps onto the corresponding properties of the security associations established between the peers in connection mode. It can be specified explicitly by the signaling application, or reported by GIST to the signaling application (either on receiving a message, or just before sending a message but after configuring or selecting the messaging association to be used for it). This attribute can also be used to convey information about any address validation carried out by GIST (for example, whether a return routability check has been carried out). Further details are discussed in Appendix B.

Local Processing: An NSLP may provide hints to GIST to enable more efficient or appropriate processing. For example, the NSLP may select a priority from a range of locally defined values to influence the sequence in which messages leave a node. Any priority mechanism MUST respect the ordering requirements for reliable messages within a session, and priority values are not

carried in the protocol or available at the signaling peer or intermediate nodes. An NSLP may also indicate that reverse path routing state will not be needed for this flow, to inhibit the node requesting its downstream peer to create it.

4.2. GIST State

4.2.1. Message Routing State

For each flow, the GIST layer can maintain message routing state to manage the processing of outgoing messages. This state is conceptually organised into a table with the following structure.

The primary key (index) for the table is the combination of the information about how the message is to be routed, the session being signalled for, and the signaling application itself:

Message Routing Information (MRI): This defines the method to be used to route the message, the direction in which to send the message, and any associated addressing information; see Section 3.3.

Session Identification (SID): The signaling session with which this message should be associated; see Section 3.4.

NSLP Identification (NSLPID): This is an IANA assigned identifier associated with the NSLP which is generating messages for this flow. The inclusion of this identifier allows the routing state to be different for different NSLPs (e.g. because of different adjacencies).

The information for a given key consists of the routing state to reach the peer in the direction given by the MRI. For any flow, there will usually be two entries (for the upstream and downstream MRI). The routing state includes information about the peer identity (see Section 4.4.2), and a UDP port number (for datagram mode) or a reference to one or more messaging associations (for connection mode). All of this information is learned from prior GIST exchanges.

It is also possible for the state information for either direction to be null. There are several possible cases:

- The signaling application has indicated that no messages will actually be sent in that direction.
- The node is a flow endpoint, so there can be no signaling peer in one or other direction.
- The node is the endpoint of the signaling path (for example, because it is acting as a proxy, or because it has determined explicitly that there are no further signaling nodes in that direction).

- The node is using other techniques to route the message. For example, it can encapsulate it the same way as a Query message and rely on the peer to intercept it.

Each item of routing state has an associated validity timer for how long it can be considered accurate; when this timer expires, it MUST be purged if it has not been refreshed. Installation and maintenance of routing state is described in more detail in Section 4.4.

Note also that the routing state is described as a table of flows, but that there is no implied constraint on how the information is stored. However, in general, and especially if GIST peers are several IP hops away, there is no way to identify the correct downstream peer for a flow and signaling application from the local forwarding table using prefix matching, and the same applies always to upstream peer state because of the possibility of asymmetric routing: per-flow state has to be stored, just as for RSVP [9].

4.2.2. Messaging Association State

The per-flow message routing state is not the only state stored by GIST. There is also the state required to manage the messaging associations. Since these are typically per-peer rather than per-flow, they are stored separately, including the following information:

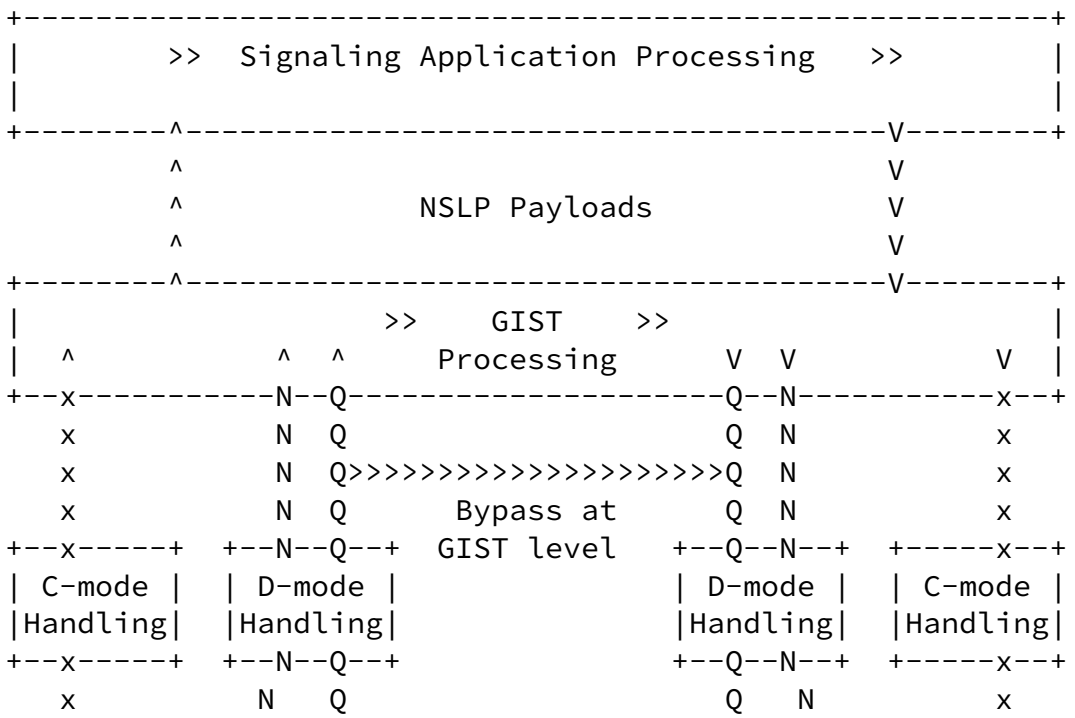
- messages pending transmission while an association is being established;

- a timer for how long since the peer re-stated its desire to keep the association open (see Section 4.4.3).

In addition, per-association state is held in the messaging association protocols themselves. However, the details of this state are not directly visible to GIST, and they do not affect the rest of the protocol description.

4.3. Basic Message Processing

This section describes how signaling application messages are processed in the case where any necessary messaging associations and routing state are already in place. The description is divided into several parts. Firstly, message reception, local processing and message transmission are described for the case where the node hosts the NSLPID identified in the message. Secondly, the case where the message is handled directly in the IP or GIST layer (because there is no matching signaling application on the node) is given. An overview is given in Figure 3.



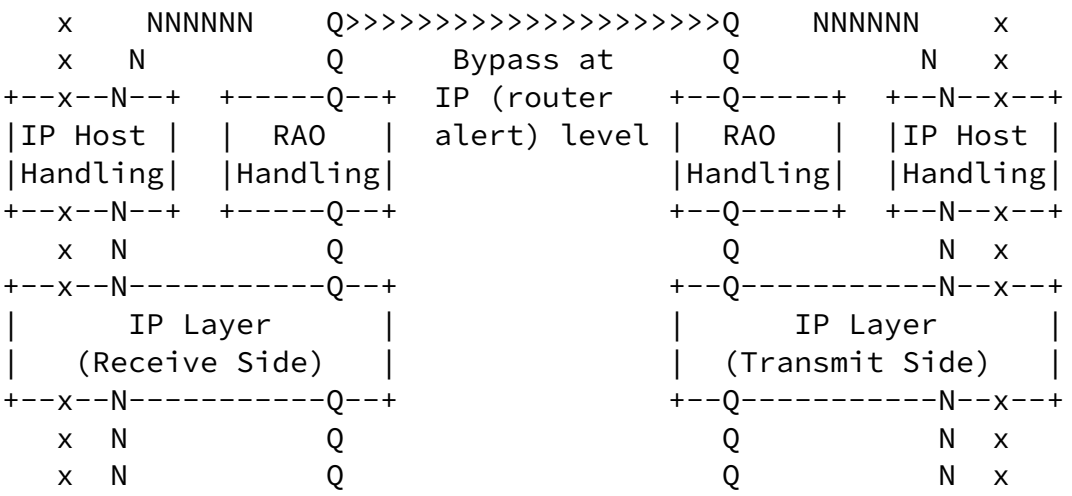


Figure 3: Message Paths through a GIST Node

4.3.1. Message Reception

Messages can be received in connection or datagram mode, and in the latter case with two types of message encapsulation.

Reception in connection mode is simple: incoming packets undergo the security and transport treatment associated with the messaging

association, and the messaging association provides complete messages to the GIST layer for further processing.

Reception in datagram mode depends on the message type. 'Normal' messages arrive UDP encapsulated and addressed directly to the receiving signaling node, at an address and port learned previously. Each datagram contains a single message which is passed to the GIST layer for further processing, just as in the connection mode case.

Where GIST is sending messages to be intercepted by the appropriate peer rather than directly addressed to it (in particular, Query messages), these are UDP encapsulated with an IP router alert option.

Each signaling node will therefore 'see' all such messages. The case where the NSLPID does not match a local signaling application at all is considered below in Section 4.3.4; otherwise, it is again passed up to the GIST layer for further processing.

4.3.2. Local Processing and Validation

Once a message has been received, it is processed locally within the GIST layer. The GIST hop count MUST be decremented immediately the message has been received. Further processing depends on the message type and payloads carried; most of the GIST payloads are associated with state maintenance and details are covered in Section 4.4.

In the case of a GIST-Query, there is an interaction with signaling application policy to determine which of two courses to follow:

1. The signaling application wishes to become a signaling peer with the Querying node. GIST MUST continue with the handshake process to set up message routing state, as described in Section 4.4.1. The application MAY provide an NSLP payload for the same NSLPID, which GIST will transfer in the GIST-Response.

2. The signaling application does not wish to set up state with the Querying node and become its peer. GIST MUST propagate the Query, similar to the case described in Section 4.3.4. No message is sent back to the Querying node. The application MAY provide an updated NSLP payload for the same NSLPID (which will be used in the Query message forwarded by GIST).

This interaction with the signaling application, including the generation or update of an NSLP payload, SHOULD take place synchronously as part of the Query message processing . In terms of the GIST service interface, this can be implemented by providing appropriate return values for the primitive that is triggered when such a message is received; see Appendix B.2 for further discussion.

For all other message types, the GIST payloads are processed as described in Section 4.4. The remainder of the GIST message consists of an NSLP payload, which is delivered locally to the signaling application identified by the NSLPID. The format of the payload is not constrained by GIST, and the content is not interpreted.

Delivery is subject to the following validation checks:

- if the message was explicitly routed (see Section 7.1.4) or is a Data message delivered without routing state (see Section 5.3.2), the payload is delivered but flagged to the receiving NSLP to indicate that routing state was not validated;

- else, if there is no routing state for the MRI/SID/NSLPID the message MUST be rejected with a "No Routing State" error message (Appendix A.4.4.5);

- else, if the message arrived on an association which is not associated with the MRI/NSLPID/SID combination given in the message, the message MUST be rejected with an "Incorrectly Delivered Message" error message (Appendix A.4.4.4);

- else, the payload is delivered as normal.

4.3.3. Message Transmission

Signaling applications can generate their messages for transmission, either asynchronously, or in response to a normal input message, and GIST can also generate messages autonomously. Regardless of the source, outgoing messages are passed downwards for message transmission. When a message is available for transmission, GIST uses internal policy and the stored routing state to determine how to handle it. The following processing applies equally to locally generated messages and messages forwarded from within the GIST or signaling application levels. (However, see Section 5.6 for special rules applying to the transmission of error messages by GIST.)

The main decision is whether the message must be sent in connection mode or datagram mode. Reasons for using the former are:

- signaling application requirements: for example, it has requested channel secured delivery, or reliable delivery;

- protocol specification: a message that requires fragmentation MUST be sent over a messaging association;

- local policy: for example, a node MAY send messages over a messaging association to benefit from adaptive congestion control.

In principle, as well as determining that some messaging association must be used, GIST MAY select between a set of alternatives, e.g. for load sharing or because different messaging associations provide different transport or security attributes.

If the use of a messaging association is selected, the message is queued on the association found from the routing state table, and further output processing is carried out according to the details of the protocol stacks used. If no appropriate association exists, the message is queued while one is created (see Section 4.4.1). If no association can be created, this is an error condition, and should be indicated back to the local signaling application.

If a messaging association is not required, the message is sent in datagram mode. The processing in this case depends on the message type and whether routing state exists or not.

- If the message is not a Query, and routing state exists, it is UDP encapsulated and sent directly to the address from the routing state table.

- If the message is a Query, then it is UDP encapsulated with IP address and router alert option determined from the MRI and NSLPID (further details depend on the message routing method).

- If no routing state exists, GIST can attempt to use the same encapsulation as in the Query case. If this is not possible (e.g. because the encapsulation for the message routing method is only defined for one message direction), then this is an error condition which is reported back to the local signaling application.

4.3.4. Nodes not Hosting the NSLP

A node may receive messages where it has no signaling application corresponding to the message NSLPID. There are several possible cases depending mainly on the encapsulation:

1. A Query-encapsulated message contains an RAO value which is relevant to NSIS but not to the specific node, but the IP layer is unable to recognise whether it needs to be passed to GIST for further processing or whether the packet should be forwarded just like a normal IP datagram.

2. A Query-encapsulated message contains an RAO value which is relevant to the node, but the specific signaling application functionality for the actual NSLPID in the message is not processed there.

3. A directly addressed message (in datagram or connection mode) is delivered to a node for which there is no corresponding signaling application. (With the current specification, this should never happen. While future versions might find a use for such a feature, currently this MUST cause an "Unknown NSLPID" error message, Appendix A.4.4.6.)

4. A Query-encapsulated message arrives at the end-system which does not handle the signaling application. This is possible in normal operation, and MUST be notified to the sender with an "Endpoint Found" informational message (Appendix A.4.4.7). The end-system includes the MRI and SID from the original message in the error message without interpreting them.

5. The node is GIST-aware NAT. See Section 7.2.

In cases (1) and (2), the role of GIST is to forward the message essentially unchanged, and it will not become a peer to the node sending the message. (Forwarding with modified NSLP payloads is covered above in Section 4.3.2.) However, a GIST implementation must ensure that the IP TTL field and GIST hop count are managed correctly to prevent message looping, and this should be done consistently independently of whether the processing (e.g. for case (1)) takes place on the fast path or in GIST-specific code. The rules are that in cases (1) and (2), the IP TTL MUST be decremented just as if the message was a normal IP forwarded packet; in case (2) the GIST hop count MUST be decremented as in the case of normal input processing, which indeed applies to cases (3) and (4).

A GIST node processing Query-encapsulated messages in this way SHOULD make the routing decision based on the full contents of the MRI and not only the IP destination address. It MAY also apply a restricted set of sanity checks and under certain conditions return an error message rather than forwarding the message. These conditions are:

1. The message is so large that it would be fragmented on downstream links (e.g. because the downstream MTU is very small). The error "Message Too Large" (Appendix A.4.4.8) SHOULD be returned to the sender, which SHOULD begin messaging association setup.

2. The GIST hop count has been exceeded. The error "Hop Limit Exceeded" (Appendix A.4.4.2) SHOULD be returned to the sender,

which MAY retry with a larger initial hop count if it is clear that a loop has not been formed.

3. The MRI represents a flow definition which is too general to be forwarded along a unique path (e.g. the destination address prefix is too short). The error "MRI Validation Failure"

(Appendix A.4.4.12) with subcode 0 ("MRI Too Wild") SHOULD be returned to the sender, which MAY retry with restricted MRIs, possibly starting additional signaling sessions to do so. If the GIST node does not understand the MRM in question it MUST NOT apply this check, instead forwarding the message transparently.

In the first two cases, only the common header is examined; in the third case, the MRI is also examined. The rest of the message MUST never be inspected or modified.

Note that the hop count is only intended to prevent message looping at the GIST level, and by default NSLPs must take their own measures to prevent looping at the application level. However, the GIST API (Appendix B) provides the incoming hop count to the NSLPs, which can preserve it on outgoing messages as they are forwarded further along the path. This provides a lightweight loop-prevention mechanism for NSLPs which do not define anything more sophisticated.

4.4. Routing State and Messaging Association Maintenance

The main responsibility of GIST is to manage the routing state and messaging associations which are used in the message processing described above. Routing state is installed and maintained by specific GIST messages. Messaging associations depend on the existence of routing state, but are actually set up by the normal procedures of the transport and security protocols that comprise them. Timers control routing state and messaging association refresh and expiration.

There are two different cases for state installation and refresh:

1. Where routing state is being discovered or a new association is to be established; and

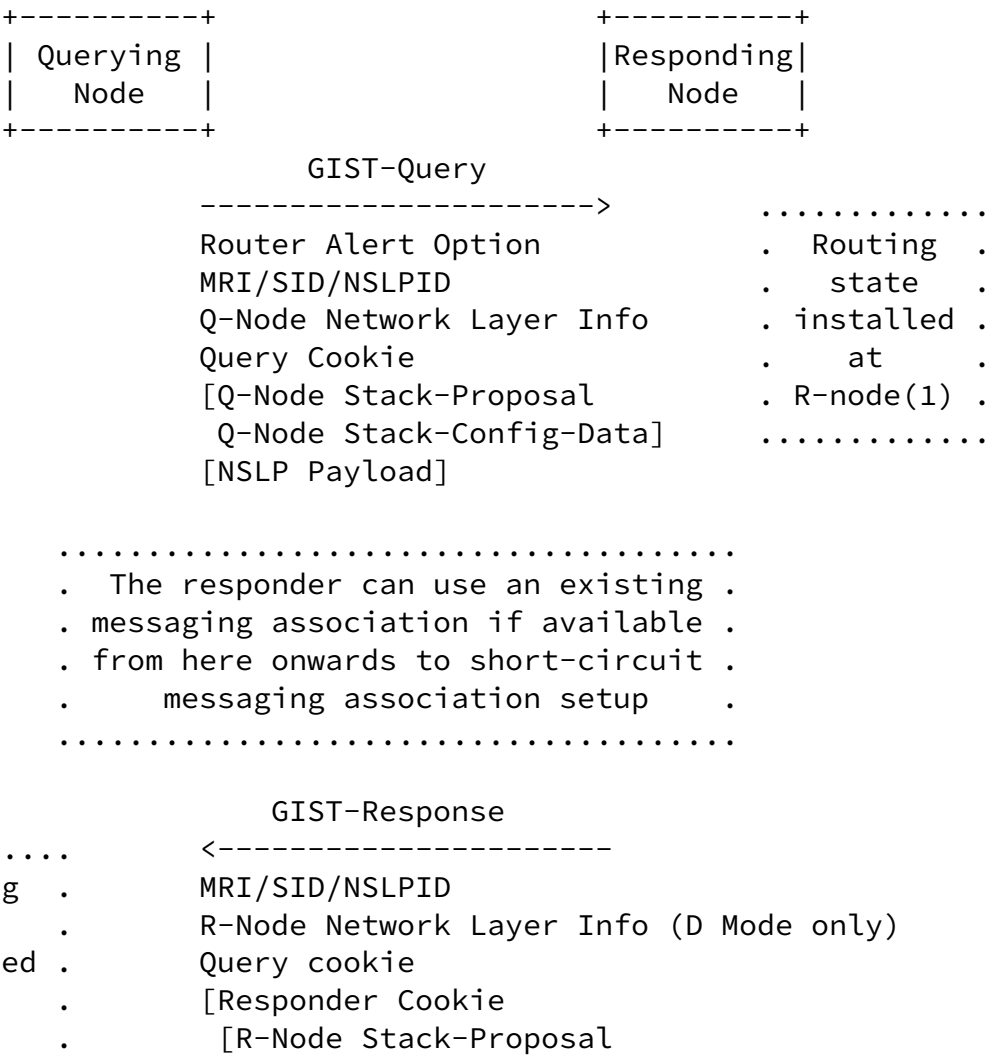
2. Where an existing association can be re-used, including the case

where routing state for the flow is being refreshed.

These cases are now considered in turn, followed by the case of background general management procedures.

4.4.1. State Setup

The complete sequence of possible messages for state setup between adjacent peers is shown in Figure 4 and described in detail in the following text. A concrete example is given in Appendix C.



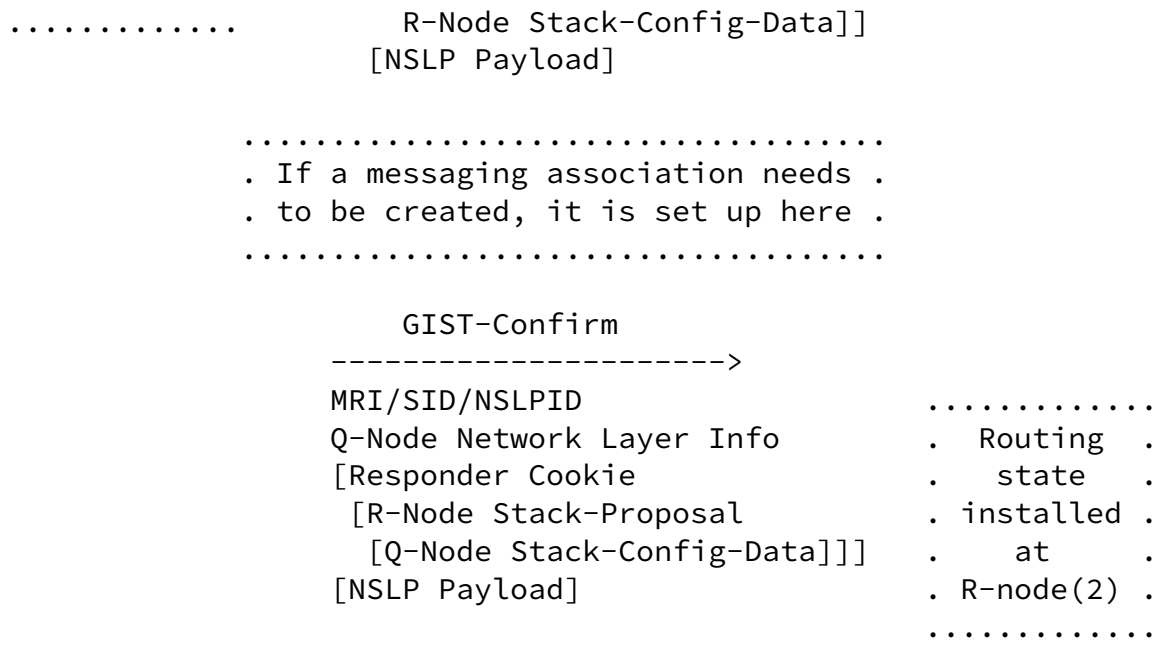


Figure 4: Message Sequence at State Setup

The initial message in any routing state maintenance operation is a GIST-Query message, sent from the querying node and intercepted at the responding node. This message has addressing and other identifiers appropriate for the flow and signaling application that state maintenance is being done for, addressing information about the node itself, and it MAY contain an NSLP payload. It also includes a Query Cookie, and optionally capability information about messaging association protocol stacks. The role of the cookies in this and subsequent messages is to protect against certain denial of service attacks and to correlate the various events in the message sequence.

Provided that the signaling application has indicated that message routing state should be set up (see Section 4.3.2), reception of a GIST-Query MUST elicit a GIST-Response message. This is a 'normally' encapsulated datagram mode message with additional payloads. It contains network layer information about the responding node, echoes the Query Cookie, and MAY contain an NSLP payload (possibly a response to the NSLP payload in the initial message). In case a messaging association was requested, it MUST also contain a Responder Cookie and its own capability information about messaging association protocol stacks. Even if a messaging association is not requested,

the Response MAY still include a Responder Cookie if the node's routing state setup policy requires it (see below).

Setup of a new messaging association begins when peer addressing information is available and a new messaging association is actually needed. The setup MUST be contemporaneous with a specific GIST-Query/Response exchange, because the addressing information used may have a limited lifetime (either because it depends on limited lifetime NAT bindings, or because it refers to agile destination ports for the transport protocols). The Stack-Proposal and Stack-Configuration-Data objects carried in the exchange carry capability information about what messaging association protocols can be used, and the processing of these objects is described in more detail in Section 5.7. With the protocol options currently defined, setup of the messaging association always starts from the Querying node, although more flexible configurations are possible within the overall GIST design. In any case, once set up the association itself can be used equally in both directions.

Finally, a GIST-Confirm MUST be sent if the GIST-Response requested it. If a messaging association is being used, the GIST-Confirm MUST be sent over it before any other messages for the same flow, and it echoes the Responder Cookie and Stack-Proposal from the GIST-Response. The former is used to allow the receiver to validate the contents of the message (see Section 8.5), and the latter is to prevent certain bidding-down attacks on messaging association security. The Confirm MAY also contain an abbreviated form of the

original Stack-Configuration-Data to finalise details of the messaging association configuration. The association can be used in the upstream direction for that MRI and NSLPID after the Confirm has been received.

The querying node MUST install the responder address as routing state information after verifying the Query Cookie in the GIST-Response. The responding node MAY install the querying address as peer state information at two points in time:

1. after the receipt of the initial GIST-Query, or

2. after a GIST-Confirm message containing the Responder Cookie.

The precise constraints on when state information is installed are a matter of security policy considerations on prevention of denial-of-service attacks and state poisoning attacks, which are discussed further in Section 8. Because the responding node MAY choose to delay state installation as in case (2), the GIST-Confirm must contain sufficient information to allow it to be processed identically to the original Query. This places some special requirements on NAT traversal and cookie functionality, which are discussed in Section 7.2 and Section 8 respectively.

4.4.2. Association Re-use

It is a design goal of GIST that, so far as possible, messaging associations should be re-used for multiple flows and sessions, rather than a new association set up for each. This is to ensure that the association cost scales only like the number of peers, and to avoid the latency of new association setup where possible.

However, re-use requires the identification of an existing association which matches the same routing state and desired properties that would be the result of a full handshake in D-mode, and this identification must be done as reliably and securely as continuing with the full procedure. Note that this requirement is complicated by the fact that NATs may remap the node addresses in D-mode messages, and also interacts with the fact that some nodes may peer over multiple interfaces (and so with different addresses).

Association re-use is controlled by the Network-Layer-Information (NLI) object, which is carried in GIST-Query/Confirm and optionally GIST-Response messages. The NLI object includes:

Peer-Identity: For a given node, this is an interface independent value with opaque syntax. It MUST be chosen so as to have a high probability of uniqueness between peers, and SHOULD be stable (at least between restarts). Note that there is no cryptographic protection of this identity (attempting to provide this would essentially duplicate the functionality in the messaging association security protocols).

Interface-Address: This is an IP address through which the signaling node can be reached. There may be several choices available for the Interface-Address, and further discussion of this is contained in Section 5.2.2.

By default, a messaging association is associated with the NLI object that was provided by the peer in the Query/Response/Confirm at the time the association was set up. There may be more than one association for a given NLI object (e.g. with different properties).

Association re-use is controlled by matching the NLI provided in a GIST message with those associated with existing associations. This can be done on receiving either a GIST-Query or GIST-Response (the former is more likely):

- If there is a perfect match to the NLI of an existing association, that association SHOULD be re-used (provided it has the appropriate properties in other respects). This is indicated by sending the remaining messages in the handshake over that association. This will only fail (i.e. lead to re-use of an association to the 'wrong' node) if signaling nodes have colliding Peer-Identities, and one is reachable at the same Interface-Address as another. (This could be done by an on-path attacker.)

- In all other cases, the full handshake MUST be executed in datagram mode as usual. There are in fact four possibilities:

  1. Nothing matches: this is clearly a new peer.

  2. Only the Peer-Identity matches: this may be either a new interface on an existing peer, or a changed address mapping behind a NAT, or an attacker attempting to hijack the Peer-Identity. These should be rare events, so the expense of a new association setup is acceptable.

  3. Only the Interface-Address matches: this is probably a new peer behind the same NAT as an existing one. A new association setup is required.

4. The full NLI object matches: this is a degenerate case, where one node recognises an existing peer, but wishes to allow the option to set up a new association in any case (for example to create an association with different properties).

4.4.3. State Maintenance Procedures

Refresh and expiration of all types of state is controlled by timers.

Each item of routing state expires after a validity lifetime which is negotiated during the Query/Response/Confirm handshake. The NLI object in the Query contains a proposal for the lifetime value, and the NLI in the Response contains the value the Responding node requires. The Querying node MUST generate a GIST-Query message before this timer expires, if it believes that the flow is still active; otherwise, the Responding node MAY delete the state. Receipt of the message at the Responding node will refresh peer addressing state for one direction, and receipt of a GIST-Response at the querying node will refresh it for the other. There is no mechanism at the GIST level for explicit teardown of routing state. However, GIST MUST NOT refresh routing state if a flow is known to be inactive, either because upstream state has expired, or because the signaling application has indicated via the GIST API (Appendix B.5) that the state is no longer required, because this would prevent correct state repair in the case of network rerouting.

Unneeded messaging associations are torn down by GIST, using the teardown mechanisms of the underlying transport or security protocols if available (for example, simply by closing a TCP connection). The teardown can be initiated by either end. Whether an association is needed is a combination of two factors:

- local policy, which could take into account the cost of keeping the messaging association open, the level of past activity on the association, and the likelihood of future activity (e.g. if there is routing state still in place which might generate messages to use it).

- whether the peer still wants the association in place. During messaging association setup, each node indicates its own MA-Hold-Time as part of the Stack-Configuration-Data. (Because the Responding node can choose not to retain state until a Confirm message, an abbreviated Stack-Configuration-Data object containing just this information MUST be repeated by the Querying node in the first Confirm sent on a new messaging association.) A node MUST NOT tear down the association if it has received traffic from its peer over that period. A peer which has generated no traffic but still wants the association retained SHOULD use a special 'null'

message (GIST-MA-Hello) to indicate the fact.

Messaging associations can always be set up on demand, and messaging association status is not made directly visible outside the GIST layer. Therefore, even if GIST tears down and later re-establishes a messaging association, signaling applications cannot distinguish this from the case where the association is kept permanently open. To maintain the transport semantics described in Section 4.1, GIST MUST close transport connections carrying reliable messages gracefully or report an error condition, and MUST NOT open a new association for a given session and peer while messages on a previous association may still be outstanding.

This specification defines precisely only the time at which state or messaging associations expire; it does not define when refresh transactions should be initiated. Implementations SHOULD select timer settings which take at least the following into account:

- The transmission latency between source and destination;
- The need for retransmissions (either explicitly or within the messaging association protocols);
- The need to avoid network synchronisation of control traffic (cf. [34]).

In most cases, a reasonable policy is (for example) to initiate the refresh process when between 1/2 and 3/4 of the appropriate validity time has elapsed since the last successful refresh. The actual moment is chosen randomly within this interval to avoid synchronisation effects.

5. Message Formats and Transport

5.1. GIST Messages

All GIST messages begin with a common header, followed by a sequence of type-length-value (TLV) objects. This subsection describes the various GIST messages and their contents at a high level; a more detailed description of the header and each object is given in Section 5.2.

The common header includes a version number, message type and size, and NSLPID. It also carries a hop count to prevent message looping and various control flags, including one to indicate if a reply of some sort is requested. The objects following the common header MUST be carried in a fixed order, depending on message type. Messages with missing, duplicate or invalid objects for the message type MUST be rejected with an "Object Type Error" error message with the appropriate subcode (Appendix A.4.4.9).

The following gives the basic syntax of GIST messages in ABNF [7]. Note that the NAT traversal mechanism for GIST involves the insertion of an additional NAT-Traversal object in certain messages; the rules for this are given in Section 7.2.

GIST-Message: The main messages are either one of the stages in the 3-way handshake, or a simple message carrying NSLP data. Additional types are allocated for errors and messaging association keepalive.

```
GIST-Message = GIST-Query / GIST-Response /
               GIST-Confirm / GIST-Data /
               GIST-Error / GIST-MA-Hello
```

GIST-Query: A GIST-Query MUST be sent in datagram mode. As well as the common header, it contains certain mandatory control objects, and MAY contain a signaling application payload. A stack proposal and configuration data MUST be included if the message exchange relates

to setup of a messaging association. The R flag MUST always be set (R=1) in a Query, since this message always elicits a Response.

```
GIST-Query = Common-Header
             Message-Routing-Information
             Session-Identification
             Network-Layer-Information
             Query-Cookie
             [ Stack-Proposal Stack-Configuration-Data ]
             [ NSLP-Data ]
```

GIST-Response: A GIST-Response may be sent in datagram or connection

mode (if a messaging association is being re-used). It MUST echo the MRI (with inverted direction), SID and Query-Cookie of the Query, and in D-mode carries its own Network-Layer-Information; if the message exchange relates to setup of a messaging association (which can only take place in datagram mode), a Responder cookie MUST be included, as well as its own stack proposal and configuration data. The R flag MUST be set (R=1) if a Responder cookie is present but otherwise is optional; if the R flag is set, a Confirm MUST be sent as a reply.

```
GIST-Response = Common-Header
                Message-Routing-Information
                Session-Identification
                [ Network-Layer-Information ]
                Query-Cookie
                [ Responder-Cookie
                  [ Stack-Proposal Stack-Configuration-Data ] ]
                [ NSLP-Data ]
```

GIST-Confirm: A GIST-Confirm may be sent in datagram or connection mode (if a messaging association has been re-used). It MUST echo the MRI (with inverted direction), SID, and Responder-Cookie if the Response carried one; if the message exchange relates to setup of a new messaging association or reuse of an existing one (which can only take place in connection mode), the message MUST also echo the Stack-Proposal from the GIST-Response so it can be verified that this has not been tampered with. The first message on an association MUST also repeat the Stack-Configuration-Data from the original Query in an abbreviated form (just containing the MA-Hold-Time).

```
GIST-Confirm = Common-Header
               Message-Routing-Information
               Session-Identification
               Network-Layer-Information
               [ Responder-Cookie
                 [ Stack-Proposal
                   [ Stack-Configuration-Data ] ] ]
               [ NSLP-Data ]
```

GIST-Data: A plain data message contains no control objects, but only the MRI and SID associated with the NSLP data being transferred. Network-Layer-Information MUST be carried in the datagram mode case and not otherwise.

```
GIST-Data = Common-Header
            Message-Routing-Information
            Session-Identification
            [ Network-Layer-Information ]
            NSLP-Data
```


GIST-Error: A GIST-Error message reports a problem determined at the GIST level. (Errors generated by signaling applications are reported in NSLP-Data payloads and are not treated specially by GIST.) The message includes a Network-Layer-Information object for the originator of the error message it if is being sent in datagram mode; all other information related to the error is carried in a GIST-Error-Data object.

```
GIST-Error = Common-Header
             [ Network-Layer-Information ]
             GIST-Error-Data
```

GIST-MA-Hello: This message MUST be sent only in C-Mode to indicate that a node wishes to keep a messaging association open. It contains only the common header, with a null NSLPID. The R flag MAY be set (R=1) to indicate that a reply is requested, thus allowing a node to test the liveness of the peer.

```
GIST-MA-Hello = Common-Header
```

5.2. Information Elements

This section describes the content of the various objects that can be present in each GIST message, both the common header, and the individual TLVs. The bit formats are provided in Appendix A.

5.2.1. The Common Header

Each message begins with a fixed format common header, which contains the following information:

Version: The version number of the GIST protocol.

Length: The number of 32 bit words in the message following the common header.

Upper layer identifier (NSLPID): This gives the specific NSLP that this message is used for.

GIST hop counter: A hop counter to prevent a message from looping.

Message type: The message type (Query, Response, etc.)

Source addressing mode: If set (S=1), this indicates that the IP source address of the message is the same as the signaling source address, in which case replies to this message can be sent safely to this address. It is cleared (S=0) if the IP source address was derived from the message routing information in the payload and

this is different from the signaling source address.

Response requested: A flag which if set (R=1) indicates that a message should be sent in response to this message.

Explicit routing: A flag which if set (E=1) indicates that the message was explicitly routed (see Section 7.1.4).

5.2.2. TLV Objects

All data following the common header is encoded as a sequence of type-length-value objects. Currently, each object can occur at most once; the set of required and permitted objects is determined by the message type and encapsulation.

Message-Routing-Information (MRI): Information sufficient to define how the signaling message should be routed through the network.

```
Message-Routing-Information = message-routing-method
                              method-specific-information
```

The format of the method-specific-information depends on the message-routing-method requested by the signaling application. It is provided by the NSLP in the message sender and used by GIST to select the message routing.

Session-Identification (SID): The GIST session identifier is a 128 bit, cryptographically random identifier chosen by the node which originates the signaling exchange. See Section 3.4.

Network-Layer-Information: This object carries information about the network layer attributes of the node sending the message, including data related to the management of routing state. This includes a peer identity and IP address for the sending node. It also includes IP TTL information to allow the hop count between GIST peers to be measured and reported, and a validity time for the routing state.

```
Network-Layer-Information = peer-identity
                            interface-address
                            RS-validity-time
                            IP-TTL
```

The use of the RS-validity-time field is described in Section 4.4.3. The peer-identity and interface-address are used for matching existing associations, as discussed in Section 4.4.2.

The interface-address must be routable, i.e. it MUST be usable as

a destination IP address for packets to be sent back to the node generating the signaling message (whether in datagram or connection mode). Where this object is carried in a GIST-Query or GIST-Confirm, the interface-address MUST specifically be set to an address bound to the interface associated with the MRI (e.g. the one carrying the outbound flow), to allow its use in route change handling, see Section 7.1. A node may have several choices for which of its addresses to use as the interface-address. For

example, there may be a choice of IP versions, or addresses of limited scope (e.g. link-local), or addresses bound to different interfaces in the case of a router or multi-homed host. However, some of these interface addresses may not be usable by the peer. A node SHOULD follow a default policy of using a global address of the same IP version as in the MRI, unless it can establish that an alternative address would also be usable.

The setting and interpretation of the IP-TTL field depends on the message direction (as determined from the MRI) and encapsulation.

* If the message is downstream, the IP-TTL MUST be set to the TTL that will be set in the IP header for the message (if this can be determined), or else set to 0.

* On receiving a downstream message in datagram mode, a non-zero IP-TTL is compared to the TTL in the IP header, and the difference is stored as the IP-hop-count-to-peer for the upstream peer in the routing state table for that flow. Otherwise, the field is ignored.

* If the message is upstream, the IP-TTL MUST be set to the value of the IP-hop-count-to-peer stored in the routing state table, or 0 if there is no value yet stored.

* On receiving an upstream message, the IP-TTL is stored as the IP-hop-count-to-peer for the downstream peer.

In all cases, the TTL value reported to signaling applications is the one stored with the routing state for that flow, after it has been updated (if appropriate) from processing the message in question.

Stack-Proposal: This field contains information about which combinations of transport and security protocols are available for use in messaging associations, and is also discussed further in Section 5.7.

```
Stack-Proposal = 1*stack-profile
```



```
stack-profile = 1*protocol-layer
```

Each protocol-layer field identifies a protocol with a unique tag; any additional data (e.g. higher-layer addressing or other options data) associated with the protocol will be carried in a MA-protocol-options field in the Stack-Configuration-Data TLV (see below).

Stack-Configuration-Data: This object carries information about the overall configuration of a messaging association.

```
Stack-Configuration-Data = MA-Hold-Time
                           0*MA-protocol-options
```

The MA-Hold-Time field indicates how long a node will hold open an inactive association; see Section 4.4.3 for more discussion. The MA-protocol-options fields give the configuration of the protocols to be used for new messaging associations, and they are described in more detail in Section 5.7.

Query-Cookie/Responder-Cookie: A Query-Cookie is contained in a GIST-Query message and MUST be echoed in a GIST-Response; a Responder-Cookie MAY be sent in a GIST-Response message, and if present MUST be echoed in the following GIST-Confirm message. Cookies are variable length (chosen by the cookie generator). See Section 8.5 for further details on requirements and mechanisms for cookie generation.

NSLP-Data: The NSLP payload to be delivered to the signaling application. GIST does not interpret the payload content.

GIST-Error-Data: This contains all the information to determine the cause and context of an error.

```
GIST-Error-Data = error-class error-code error-subcode
                  common-header
                  [ Message-Routing-Information-content ]
                  [ Session-Identification-content ]
                  0*additional-information
                  [ comment ]
```

The error-class indicates the severity level, and the error-code and error-subcode identify the specific error itself. A full list of GIST errors and their severity levels is given in Appendix A.4. The common-header from the original message is always included, as are the contents of the Message-Routing-Information and Session-Identification objects if they were successfully decoded. For some errors, additional information fields must be included

according to a fixed format; finally, an optional free-text comment may be added.

5.3. Datagram Mode Transport

This section describes the various encapsulation options for datagram mode messages. Although there are several possibilities, depending on message type, message routing method, and local policy, the general design principle is that the sole purpose of the encapsulation is to ensure that the message is delivered to or intercepted at the correct peer. Beyond that, minimal significance is attached to the type of encapsulation or the values of addresses or ports used for it. This allows new options to be developed in the future to handle particular deployment requirements without modifying the overall protocol specification.

5.3.1. Normal Encapsulation

Normal encapsulation MUST be used for all datagram mode messages where the signaling peer is already known from previous signaling. This includes Response and Confirm messages, and Data messages except if these are being sent without using local routing state. Normal encapsulation is simple: the complete set of GIST payloads is concatenated together with the common header, and placed in the data field of a UDP datagram. UDP checksums MUST be enabled. The message is IP addressed directly to the adjacent peer; the UDP port numbering MUST be compatible with that used on Query messages (see below), that is, the same for messages in the same direction and swapped otherwise.

5.3.2. Query Encapsulation

Query encapsulation MUST be used for messages where no routing state is available or where the routing state is being refreshed, in particular for GIST-Query messages. Query encapsulation is similar to normal encapsulation, with changes in IP address selection, IP options, and a defined method for selecting UDP ports.

In general, the IP addresses are derived from information in the MRI; the exact rules depend on the message routing method. In addition, the IP header is given a Router Alert Option ([1] and [4]) to assist the peer in intercepting the message depending on the NSLPID. Each NSLPID corresponds to a unique RAO value, but not necessarily vice versa; further details are discussed in [36].

The source UDP port is selected by the message sender as the port at

which it is prepared to receive UDP messages in reply, and a destination UDP port is allocated by IANA (see Section 9). Note that

GIST may send messages addressed as {flow sender, flow receiver} which could make their way to the flow receiver even if that receiver were GIST-unaware. These should be rejected (with an ICMP message) rather than delivered to the user application (which would be unable to use the source address to identify it as not being part of the normal data flow). Therefore, a "well-known" port is required.

5.3.3. Retransmission and Rate-Control

Datagram mode uses UDP, and hence has no automatic reliability or congestion control capabilities. Signaling applications requiring reliability should be serviced using C-mode, which should also carry the bulk of signaling traffic. However, some form of messaging reliability is required for the GIST control messages themselves, as is rate control to handle retransmissions and also bursts of unreliable signaling or state setup requests from the signaling applications.

Query messages which do not receive Responses MAY be retransmitted; retransmissions MUST use a binary exponential backoff, with an initial timeout of T1 up to a maximum of T2 seconds. Retransmitted Queries MUST use different Query-Cookie values. These rules apply equally to the message that first creates routing state, and those that refresh it. The values of T1 and T2 are implementation defined. Note that Queries may go unanswered either because of message loss (in either direction), or because there is no reachable GIST peer. Therefore, implementations should trade off reliability (large T2) against promptness of error feedback to applications (small T2). If the Query message carries NSLP data, it may be delivered multiple times to the signaling application. If the NSLP has indicated a timeout on the validity of this payload (see Appendix B.1), T2 SHOULD be chosen to be less than this value.

This algorithm is sufficient to handle lost Queries and Responses. The case of a lost Confirm is more subtle. Notionally, we can distinguish between two cases:

1. Where the Responding node is already prepared to store per-flow state after receiving a single (Query) message. This would

include any cases where the node has NSLP data queued to send. Here, the Responding node MAY run a retransmission timer to resend the Response message until a Confirm is received, since the node is already managing state for that flow. The problem of an amplification attack stimulated by a malicious Query is handled by requiring the cookie mechanism to enable the node receiving the Response to discard it efficiently if it does not match a previously sent Query.

2. Where the responding node is not prepared to store per-flow state until receiving a properly formed Confirm message.

In case (2), a retransmission timer should not be required. However, we can assume that the next signaling message will be in the direction Querying Node -> Responding Node (if there is no 'next signaling message' the fact that the Confirm has been lost is moot). In this case, the responding node will start to receive messages at the GIST level for a MRI/NSLP combination for which there is no stored routing state (since this state is only created on receipt of a Confirm).

Therefore, the error condition is detected at the Responding node when such a message arrives, without the need for a specific timer. Recovery requires a Confirm to be transmitted and successfully received. The mechanism to cause this is that the Responding node MUST reject the incoming message with a "No Routing State" error message (Appendix A.4.4.5) back to the Querying node, which MUST interpret this as caused by a lost Confirm; the Querying node MUST regenerate the Confirm purely from local state (e.g. in particular it needs to remember a valid Responder Cookie).

In all cases, Responses MUST be sent promptly to avoid spurious retransmissions. Nodes generating any type of retransmission MUST be prepared to receive (and match) a reply to any of them (not just the one most recently sent).

The basic rate-control requirements for datagram mode traffic are deliberately minimal. A single rate limiter applies to all traffic (for all interfaces and message types). It applies to retransmissions as well as new messages, although an implementation MAY choose to prioritise one over the other. When the rate limiter

is in effect, datagram mode messages are queued until transmission is re-enabled, or an error condition MAY be indicated back to local signaling applications. The rate limiting mechanism is implementation defined, but it is RECOMMENDED that a token bucket limiter as described in [26] be used.

5.4. Connection Mode Transport

Encapsulation in connection mode is more complex, because of the variation in available transport functionality. This issue is treated in Section 5.4.1. The actual encapsulation is given in Section 5.4.2.

5.4.1. Choice of Transport Protocol

It is a general requirement of the NTLP defined in [22] that it

should be able to support bundling (of small messages), fragmentation (of large messages), and message boundary delineation. Not all transport protocols natively support all these features.

TCP provides both bundling and fragmentation, but not message boundaries. However, the length information in the common header allows the message boundary to be discovered during parsing.

SCTP [12] satisfies all requirements.

DCCP [25] is message based but does not provide bundling or fragmentation. Bundling can be carried out by the GIST layer sending multiple messages in a single datagram; because the common header includes length information, the message boundaries within the datagram can be discovered during parsing. Fragmentation of GIST messages over multiple datagrams should be avoided, because of amplification of message loss rates that this would cause.

The bundling together of small messages is either built into the transport protocol or can be carried out by the GIST layer during message construction. Either way, two approaches can be distinguished:

1. As messages arrive for transmission they are gathered into a bundle until a size limit is reached or a timeout expires (cf.

the Nagle algorithm of TCP or similar optional functionality in SCTP). This provides maximal efficiency at the cost of some latency.

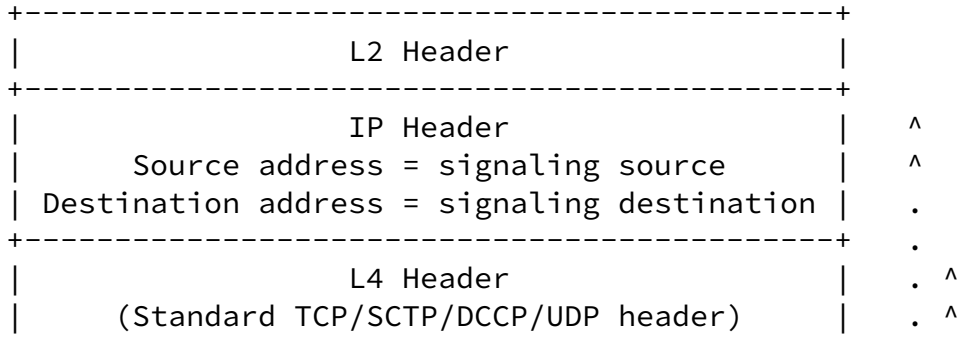
2. Messages awaiting transmission are gathered together while the node is not allowed to send them (e.g. because it is congestion controlled).

The second type of bundling is always appropriate. For GIST, the first type SHOULD NOT be used for 'trigger' (i.e. state-changing) messages, but may be appropriate for refresh messages. These distinctions are known only to the signaling applications, but MAY be indicated (as an implementation issue) by setting the priority transfer attribute.

It can be seen that all of these transport protocol options can be supported by the basic GIST message format already presented. GIST messages requiring fragmentation must be carried using a reliable transport protocol, TCP or SCTP. This specification defines only the use of TCP, but other possibilities could be included without additional work on message formatting.

5.4.2. Encapsulation Format

The GIST message, consisting of common header and TLVs, is carried directly in the transport protocol (possibly incorporating transport layer security protection). Further messages can be carried in a continuous stream (for TCP), or up to the next transport layer message boundary (for SCTP/DCCP/UDP). This situation is shown in Figure 5.



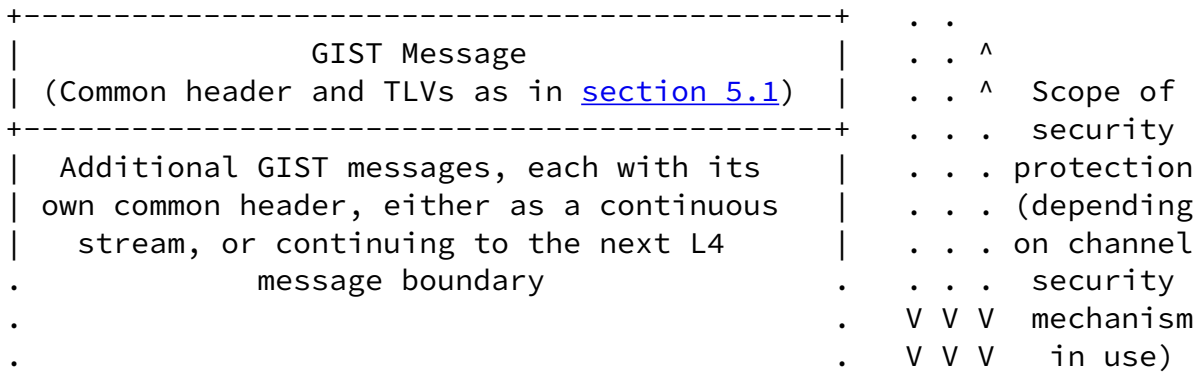


Figure 5: Connection Mode Encapsulation

5.5. Message Type/Encapsulation Relationships

GIST has four primary message types (Query/Response/Confirm/Data) and three possible encapsulation methods (D-Mode Normal/D-Mode Query/ C-Mode). For information, the possible combinations of message type and encapsulation are given in the table below. In some cases there are several possible choices, depending on the existence of routing state or messaging associations. The rules governing GIST policy, including whether or not to create such state to handle a message, are described normatively in the other sections of this specification. If a message arrives with an invalid encapsulation (e.g. a Query arrives over a messaging association), this MUST be rejected with an "Incorrect Encapsulation" error message (Appendix A.4.4.3). However, it should be noted that the processing of the message at the receiver is not otherwise affected by the encapsulation method used, with the exception that the decapsulation process may provide additional information (e.g. translated addresses

or IP hop count) which is used in the subsequent message processing.

Message	D-Mode Normal	D-Mode Query	C-Mode
GIST-Query	Never	Always	Never
GIST-Response	Unless a messaging association is being	Never	If a messaging association is being

	re-used			re-used
GIST-Confirm	Unless a messaging association has been set up or is being re-used	Never		If a messaging association has been set up or is being re-used
GIST-Data	If routing state exists for the flow but no messaging association	If no routing state exists and the MRI can be used to derive the query encapsulation		If a messaging association exists

5.6. Error Message Processing

Special rules apply to the encapsulation and transmission of error messages.

GIST only generates error messages in response to incoming messages. (Error messages MUST NOT be generated in response to incoming error messages.) The routing and encapsulation of the error message is derived from that of the message that caused the error; in particular, local routing state is not consulted. Routing state and messaging association state MUST NOT be created to handle the error, and error messages MUST NOT be retransmitted explicitly by GIST, although they are subject to the same rate control as other messages.

- If the incoming message was received in datagram mode, the error MUST be sent in datagram mode using the 'normal' encapsulation, using the addressing information from the NLI object in the incoming message. If the NLI could not be determined, the error MUST be sent to the IP source of the incoming message if the S flag was set in it. The NLI object in the GIST-Error message

reports information about the generator of the error.

- If the incoming message was received over a messaging association, the error MUST be sent back over the same messaging association.

The NSLPID in the common header of the GIST-Error is the null value (as for GIST-MA-Hello). If for any reason the error message cannot be sent (for example, because an error message is too large to send in datagram mode), an error should be logged locally.

5.7. Messaging Association Setup

5.7.1. Overview

A key attribute of GIST is that it is flexible in its ability to use existing transport and security protocols. Different transport protocols may have performance attributes appropriate to different environments; different security protocols may fit appropriately with different authentication infrastructures. Even given an initial default mandatory protocol set for GIST, the need to support new protocols in the future cannot be ruled out, and secure feature negotiation cannot be added to an existing protocol in a backwards-compatible way. Therefore, some sort of capability discovery is required.

Capability discovery is carried out in GIST-Query/Response messages, using Stack-Proposal and Stack-Configuration-Data objects. If a new messaging association is required it is then set up, followed by a GIST-Confirm. Messaging association re-use is achieved by short-circuiting this exchange by sending the GIST-Response or GIST-Confirm messages on an existing association (Section 4.4.2); whether to do this is a matter of local policy. The end result of this process is a messaging association which is a stack of protocols. If multiple associations exist, it is a matter of local policy how to distribute messages over them, subject to respecting the transfer attributes requested for each message.

Every possible protocol for a messaging association has the following attributes:

- MA-Protocol-ID, a 1-byte IANA assigned value (see Section 9).
- A specification of the (non-negotiable) policies about how the protocol should be used (for example, in which direction a connection should be opened).
- [Depending on the specific protocol:] Formats for an MA-protocol-options field to carry the protocol addressing and other

configuration information in the Stack-Configuration-Data object. The format may differ depending on whether the field is present in the Query or Response. Some protocols do not require the definition of such additional data, in which case no corresponding MA-protocol-options field will occur in the SCD object.

A Stack-Proposal object is simply a list of profiles; each profile is a sequence of MA-Protocol-IDs. A profile lists the protocols in 'top to bottom' order (e.g. TLS over TCP, or TCP over IPsec, etc.) A Stack-Proposal is generally accompanied by a Stack-Configuration-Data object which carries an MA-protocol-options field for any protocol listed in the Stack-Proposal which needs it. An MA-protocol-options field may apply globally (to all instances of the protocol in the Stack-Proposal) or be tagged as applying to a specific instance; for example, this can be used to carry different port numbers for TCP depending on whether it is to be used with or without TLS. An MA-protocol-options field may also be flagged as 'not usable'; for example, a NAT which could not handle SCTP would set this in an MA-protocol-options field about SCTP. A protocol flagged this way MUST NOT be used for a messaging association. If the Stack-Proposal and Stack-Configuration-Data are both present but not consistent (e.g. they refer to different protocols, or an MA-protocol-options field refers to a non-existent profile), an "Object Value Error" error message (Appendix A.4.4.10) with subcode 5 ("SP-SCD Mismatch") MUST be returned and the message dropped.

A node generating a Stack-Configuration-Data object MUST honour the implied protocol configurations for the period during which a messaging association might be set up; in particular, it MUST be immediately prepared to accept incoming datagrams or connections at the protocol/port combinations advertised. However, the object contents MUST be retained only for the duration of the Query/Response exchange and any following association setup, and afterwards discarded. (They may become invalid because of expired bindings at intermediate NATs, or because the advertising node is using agile ports.)

A GIST-Query requesting association setup always contains a Stack-Proposal and Stack-Configuration-Data object, and unless re-use occurs, the GIST-Response does so also. For a GIST-Response, the Stack-Proposal MUST NOT depend on the GIST-Query. A node MAY make different proposals depending on the combination of interface and NSLPID. Once the messaging association is set up, the querying node repeats the responder's Stack-Proposal over it in the GIST-Confirm. The responding node MUST verify this to ensure that no bidding-down attack has occurred; see Section 8.6 for further discussion.

5.7.2. Protocol Definition: Forwards-TCP

This MA-Protocol-ID denotes a basic use of TCP between peers. Support for this protocol is REQUIRED; associations using it can carry messages with the transfer attribute Reliable=True. The connection is opened in the forwards direction, from the querying node, towards the responder at a previously advertised port. If this protocol is offered, MA-protocol-options data MUST also be carried in the SCD object. The MA-protocol-options field formats are:

- in a Query: no information apart from the field header.
- in a Response: 2 byte port number at which the connection will be accepted, followed by 2 pad bytes.

5.7.3. Protocol Definition: Transport Layer Security

This MA-Protocol-ID denotes a basic use of transport layer channel security. Support for this protocol is mandatory; associations using it can carry messages with the transfer attribute Secure=True. For use with TCP, implementation of TLS1.0 [6] is REQUIRED and implementation of TLS1.1 [8] is RECOMMENDED. (If an unreliable transport such as DCCP or UDP is defined for GIST in the future, this MA-Protocol-ID would be implemented for it using DTLS [35].) GIST nodes supporting TLS1.0 or TLS1.1 MUST be able to negotiate the TLS ciphersuite TLS_RSA_WITH_3DES_EDE_CBC_SHA and SHOULD be able to negotiate the TLS ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA.

The default mode of TLS authentication (which applies in particular to the above ciphersuites) uses a client/server certificate exchange. The Querying node acts as a TLS client, and the Responding node acts as a TLS server. Where one of the above ciphersuites is negotiated, the GIST node acting as a server MUST provide a certificate, and MUST request one from the GIST node acting as a TLS client. This allows either server-only or mutual authentication, depending on the certificates available to the client and the policy applied at the server.

GIST nodes MAY negotiate other TLS ciphersuites. In some cases, the negotiation of alternative ciphersuites is used to trigger

alternative authentication procedures (for example, the use of pre-shared keys, [24]). The use of other authentication procedures may require additional specification work to define how they can be used as part of TLS within the GIST framework, and may or may not require the definition of additional MA-Protocol-IDs.

No MA-protocol-options field is required for this use of TLS.

5.7.4. Additional Protocol Options

Further protocols or configurations could be defined in the future for additional performance or flexibility. Examples are:

- SCTP or DCCP as alternatives to TCP, with essentially the same configuration.
- SigComp [17] for message compression.
- IPsec [30], ssh [31], or HIP/IPsec [32] for channel security.
- Alternative modes of TCP operation, for example where it is set up from the responder to the querying node.

5.8. Specific Message Routing Methods

Each message routing method (see Section 3.3) requires the definition of the format of the message routing information (MRI) and Query-encapsulation rules. These are given in the following subsections for the various possible message routing methods.

5.8.1. The Path-Coupled MRM

5.8.1.1. Message Routing Information

For the path-coupled MRM, this is essentially the Flow Identifier as in [22]. Minimally, this could just be the flow destination address; however, to account for policy based forwarding and other issues a more complete set of header fields should be used (see Section 4.3.4 and Section 7.2 for further discussion).

```
MRI = network-layer-version
```

```
source-address prefix-length
destination-address prefix-length
IP-protocol
diffserv-codepoint
[ flow-label ]
[ ipsec-SPI / L4-ports]
```

Additional control information defines whether the flow-label, SPI and port information are present, and whether the IP-protocol and diffserv-codepoint fields should be interpreted as significant. The source and destination addresses MUST be real node addresses, but prefix lengths other than 32/128 (for IPv4/6) MAY be used to implement address 'wildcarding', allowing the MRI to refer to traffic to or from a wider address range. The MRI format allows a potentially very large number of different flag and field

combinations. A GIST implementation that cannot interpret the MRI in a message MUST return an "Object Value Error" message (Appendix A.4.4.10) with subcodes 1 ("Value Not Supported") or 2 ("Invalid Flag-Field Combination") and drop the message.

5.8.1.2. Downstream Query Encapsulation

Where the signaling message is travelling in the same ('downstream') direction as the flow defined by the MRI, the IP addressing for Query messages is as follows. Support for this encapsulation is REQUIRED.

- The destination address MUST be the flow destination address as given in the MRI of the message payload.
- By default, the source address is the flow source address, again from the MRI. This provides the best likelihood that the message will be correctly routed through any region performing per-packet policy-based forwarding or load balancing which takes the source address into account. However, there may be circumstances where the use of the signaling source address is preferable, such as:
  - In order to receive ICMP error messages about the Query message (such as unreachable port or address). If these are delivered to the flow source rather than the signaling source, it will be very difficult for the querying node to detect that it is the last GIST node on the path.

* In order to receive GIST error messages where the error message sender could not interpret the NLI in the original message.

* In order to attempt to run GIST through an unmodified NAT, which will only process and translate IP addresses in the IP header.

Because of these considerations, use of the signaling source address is allowed as an option, with use based on local policy. A node SHOULD use the flow source address for initial Query messages, but SHOULD transition to the signaling source address for some retransmissions or as a matter of static configuration (e.g. if a NAT is known to be in the path out of a certain interface). A flag in the common header tells the message receiver which option was used.

It is vital that the Query message mimics the actual data flow as closely as possible, since this is the basis of how the signaling message is attached to the data path. To this end, GIST SHOULD set the DiffServ codepoint and (for IPv6) flow label to match the values in the MRI.

Any message sent in datagram mode SHOULD be below a conservative estimate of the path MTU, for which this specification takes the value 512 bytes as a default. It is possible that fragmented datagrams including an RAO will not be correctly handled in the network, so the sender SHOULD set the DF (do not fragment) bit in the IPv4 header in order to detect that a message has encountered a link with an unusually low MTU. In this case, it MUST use the signaling source address for the IP source address in order to receive the ICMP error.

A GIST implementation SHOULD apply validation checks to the MRI, to reject Query messages that are being injected by nodes with no legitimate interest in the flow being signalled for. In general, if the GIST node can detect that no flow could arrive over the same interface as the Query message, it MUST be rejected with an appropriate error message. (Such checks apply only to messages with the query encapsulation, since only those messages are required to track the flow path.) The main checks are that the IP version should match the version(s) used on that interface, and that the full range

of source addresses (the source-address masked with its prefix-length) would pass ingress filtering checks. For these cases, the error message is "MRI Validation Failure" (Appendix A.4.4.12) with subcodes 1 or 2 ("IP Version Mismatch" or "Ingress Filter Failure") respectively.

5.8.1.3. Upstream Query Encapsulation

In some deployment scenarios it is desirable and logically possible to set up routing state in the upstream direction (from flow receiver towards the sender). This could be used to support firewall signaling to control traffic from an 'un-cooperative' sender, or signaling in general where the flow sender was not NSIS-capable. This is incorporated into GIST by defining an encapsulation and processing rules for sending Query messages upstream.

In general, it is not possible to determine the hop-by-hop route upstream because of asymmetric routing. However, in particular cases, the upstream peer can be discovered with a high degree of confidence, for example:

- The upstream GIST peer is 1 IP hop away, and can be reached by tracing back through the interface on which the flow arrives.
- The upstream peer is a border router of a single-homed (stub) network.

This section defines an upstream Query encapsulation and validation checks for when it can be used. The functionality to generate

upstream Queries is OPTIONAL, but if received they MUST be processed in the normal way (no special functionality is needed for this). It is possible for routing state (for a given MRI and NSLPID) to be installed by both upstream and downstream Query exchanges. If the SIDs are different, these items of routing state MUST be considered as independent; if they match, that installed by the downstream exchange MUST take precedence.

The details of the encapsulation are as follows:

- The destination address SHOULD be the flow source address as given in the MRI of the message payload. An implementation with more

detailed knowledge of local routing MAY use an alternative destination address (e.g. the address of its default router).

- The source address SHOULD be the signaling node address.

- The DiffServ codepoint and (for IPv6) flow label MAY be set to match the values from the MRI, as in the downstream case. The same considerations about message size and fragmentation also apply as in the downstream case, and RAO setting and UDP port selection are also the same.

- The IP-TTL of the message MUST be set to 255.

The sending GIST implementation SHOULD attempt to send the Query message out of the same interface and to the same link layer neighbour from which the data packets of the flow are arriving.

The receiving GIST node MAY apply validation checks to the message and MRI, to reject Query messages which have reached a node at which they can no longer be trusted. In particular, a node SHOULD reject a message which has been propagated more than one IP hop, with an "Invalid IP TTL" error message (Appendix A.4.4.11). This can be determined by examining the received IP TTL, similar to the generalised IP TTL security mechanism described in [21]. Alternatively, receipt of an upstream Query at the flow source MAY be used to trigger setup of NTLP state in the downstream direction. These restrictions may be relaxed in a future version.

5.8.2. The Loose-End MRM

This MRM is used to discover GIST nodes with particular properties in the direction of a given address, for example to discover a NAT along the upstream data path (e.g. as in [27].

5.8.2.1. Message Routing Information

For the loose-end MRM, only a simplified version of the Flow Identifier is needed.

```
MRI = network-layer-version
      source-address
      destination-address
```

The source address is the address of the node initiating the discovery process, for example the node that will be the data receiver in the NAT discovery case. The destination address is the address of a node which is expected to be the 'other side' of the node to be discovered. Additional control information defines the direction of the message relative to this flow as in the path-coupled case.

5.8.2.2. Downstream Query Encapsulation

Only one encapsulation is defined for the loose-end MRM; by convention, this is referred to as the downstream encapsulation, and is defined as follows:

- The IP destination address MUST be the destination address as given in the MRI of the message payload.

- By default, the IP source address is the source address, again from the MRI. However, the use of the signaling source address is allowed as in the case of the path-coupled MRM.

There are no special requirements on the setting of the DiffServ codepoint, IP TTL, or (for IPv6) the flow label. Nor are any special validation checks applied.

Restrictions on message size and setting of the DF (do not fragment) bit apply as in the case of the path-coupled MRM.

6. Formal Protocol Specification

This section provides a more formal specification of the operation of GIST processing, in terms of rules for transitions between states of a set of communicating state machines within a node. The following description captures only the basic protocol specification; additional mechanisms can be used by an implementation to accelerate route change processing, and these are captured in Section 7.1.

Conceptually, GIST processing at a node may be seen in terms of 4 types of cooperating state machine:

1. There is a top-level state machine which represents the node itself (Node-SM). It is responsible for the processing of events which cannot be directed towards a more specific state machine, for example, inbound messages for which no routing state currently exists. This machine exists permanently, and is responsible for creating 'per-flow' state machines to manage the GIST handshake and routing state maintenance procedures.

2. For each flow and signaling direction where the node is responsible for the creation of routing state, there is an instance of a Query-Node state machine (Query-SM). This machine sends Query and Confirm messages and waits for Responses, according to the requirements from local API commands or timer processing (e.g. message repetition or routing state refresh).

3. For each flow and signaling direction where the node has accepted the creation of routing state by a peer, there is an instance of a Responding-Node state machine (Response-SM). This machine is responsible for managing the status of the routing state for that flow. Depending on policy, it MAY be responsible for [re]transmission of Response messages, or this MAY be handled by the Node-SM, and a Response-SM is not even created for a flow until a properly formatted Confirm has been accepted.

4. Messaging associations have their own lifecycle, represented by MA-SM, from when they are first created (in an 'incomplete' state, listening for an inbound connection or waiting for outbound connections to complete), to when they are active and available for use.

Apart from the fact that the various machines can be created and destroyed by each other, there is almost no interaction between them. The machines for different flows do not interact; the Query-SM and Response-SM for a single flow and signaling direction do not interact. That is, the Response-SM which accepts the creation of routing state for a flow on one interface has no direct interaction

with the Query-SM which sets up routing state on the next interface along the path. This interaction is mediated instead through the NSLP.

The state machine descriptions use the terminology rx_MMMM, tg_TTTT and er_EEEE for incoming messages, API/lower layer triggers and error conditions respectively. The possible events of these types are given in the table below. In addition, timeout events denoted to_TTTT may also occur; the various timers are listed independently for each type of state machine in the following subsections.

Name	Meaning
rx_Query	A GIST Query message has been received.
rx_Response	A GIST Response message has been received.
rx_Confirm	A GIST Confirm message has been received.
rx_Data	A GIST Data message has been received.
rx_Message	rx_Query||rx_Response||rx_Confirm||rx_Data.
rx_Hello	A GIST MA-Hello message has been received.
tg_NSLPData	A signaling application has requested data transfer (via API SendMessage).
tg_Connected	The protocol stack for a messaging association has completed connecting.
tg_RawData	GIST wishes to transfer data over a particular messaging association.
er_NoRSM	A "No Routing State" error was received.
er_MAConnect	A messaging association protocol failed to complete a connection.
er_MAFailure	A messaging association failed.

6.1. Node Processing

The Node level state machine is responsible for processing events for which no more appropriate messaging association state or routing state exists. Its structure is trivial: there is a single state

('Idle'); all events cause a transition back to Idle. Some events cause the creation of other state machines. The only events that are processed by this state machine are incoming GIST messages (Query/Response/Confirm/Data) and API requests to send data; all other events are impossible. In addition to this event processing, the Node level machine is responsible for managing listening endpoints for messaging associations (although these relate to Responding node operation, they cannot be handled by the Responder state machine since they are not created per flow). The processing rules for each event are as follows:

Rule 1 (rx_Query):

```
use the GIST service interface to determine the signaling application
    policy relating to this peer
if (the signaling application indicates that routing state should
    be created) then
  if (routing state can be created without a 3-way handshake) then
    create R-SM and transfer control to it
  else
    send Response
else
  propagate the Query with any updated NSLP payload provided
```

Rule 2 (rx_Response):

```
// should already have a Q-SM to handle this
discard message
send "No Routing State" error message
```

Rule 3 (rx_Confirm):

```
if (routing state can be created before receiving a Confirm) then
  // should already have R-SM for it which would handle this message
  discard message
```

```
  send "No Routing State" error message
else
  create R-SM and pass message to it

Rule 4 (rx_Data):

if (node policy will only process Data messages with matching
    routing state) then
  send "No Routing State" error message
else
  pass directly to NSLP
```



```
Rule 5 (tg_NSLPData):

if Q-mode encapsulation is not possible for this MRI
  reject message with an error
else
  if (local policy & transfer attributes say routing
      state is not needed) then
    send message statelessly
  else
    create Q-SM and pass message to it
```

6.2. Query Node Processing

The Querying-Node state machine (Q-SM) has three states:

- Awaiting Response
- Established
- Awaiting Refresh

The Q-SM is created by the N-SM machine as a result of a request to send a message for a flow in a signaling direction where the appropriate state does not exist. The Query is generated immediately and the No_Response timer is started. The NSLP data MAY be carried in the Query if local policy and the transfer attributes allow it, otherwise it MUST be queued locally pending MA establishment. Then the machine transitions to the Awaiting Response state, in which

timeout-based retransmissions are handled. Data messages (rx_Data events) should not occur in this state; if they do, this may indicate a lost Response and a node MAY also retransmit a Query for this reason.

Once a Response has been successfully received and routing state created, the machine transitions to Established, during which NSLP data can be sent and received normally. Further Responses received in this state MUST be treated the same way (this may be the result of a lost Confirm). The Awaiting Refresh state can be considered as a substate of Established, where a new Query has been generated to refresh the routing state (as in Awaiting Response) but NSLP data can be handled normally.

The timers relevant to this state machine are as follows:

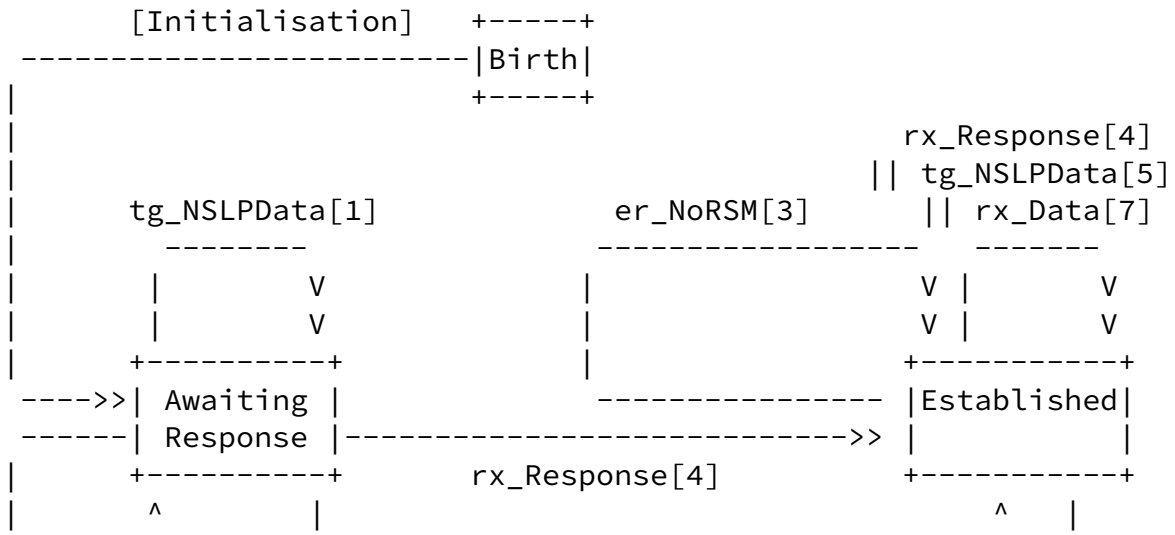
Refresh_QNode: Indicates when the routing state stored by this state machine must be refreshed. It is reset whenever a Response is received indicating that the routing state is still valid. Implementations MUST set the period of this timer based on the

value in the RS-validity-time field of a Response message to ensure that a Query is generated before the peer's routing state expires.

No_Response: Indicates that a Response has not been received in answer to a Query. This is started whenever a Query is sent and stopped when a Response is received.

Inactive_QNode: Indicates that no traffic is currently being handled by this state machine. This is reset whenever the state machine handles NSLP data (in either direction). When it expires, the state machine MAY be deleted. The period of the timer can be set at any time via the API (SetStateLifetime), and if the period is reset in this way the timer itself SHOULD be restarted.

The main events (including all those that cause state transitions) are shown in the figure below, tagged with the number of the processing rule that is used to handle the event. These rules are listed after the diagram. All events not shown or described in the text above are assumed to be impossible in a correct implementation.


[Initialisation]
Birth
rx_Response[4]
tg_NSLPData[5]
tg_NSLPData[1]
er_NoRSM[3]
rx_Data[7]
Awaiting
Response
Established
rx_Response[4]

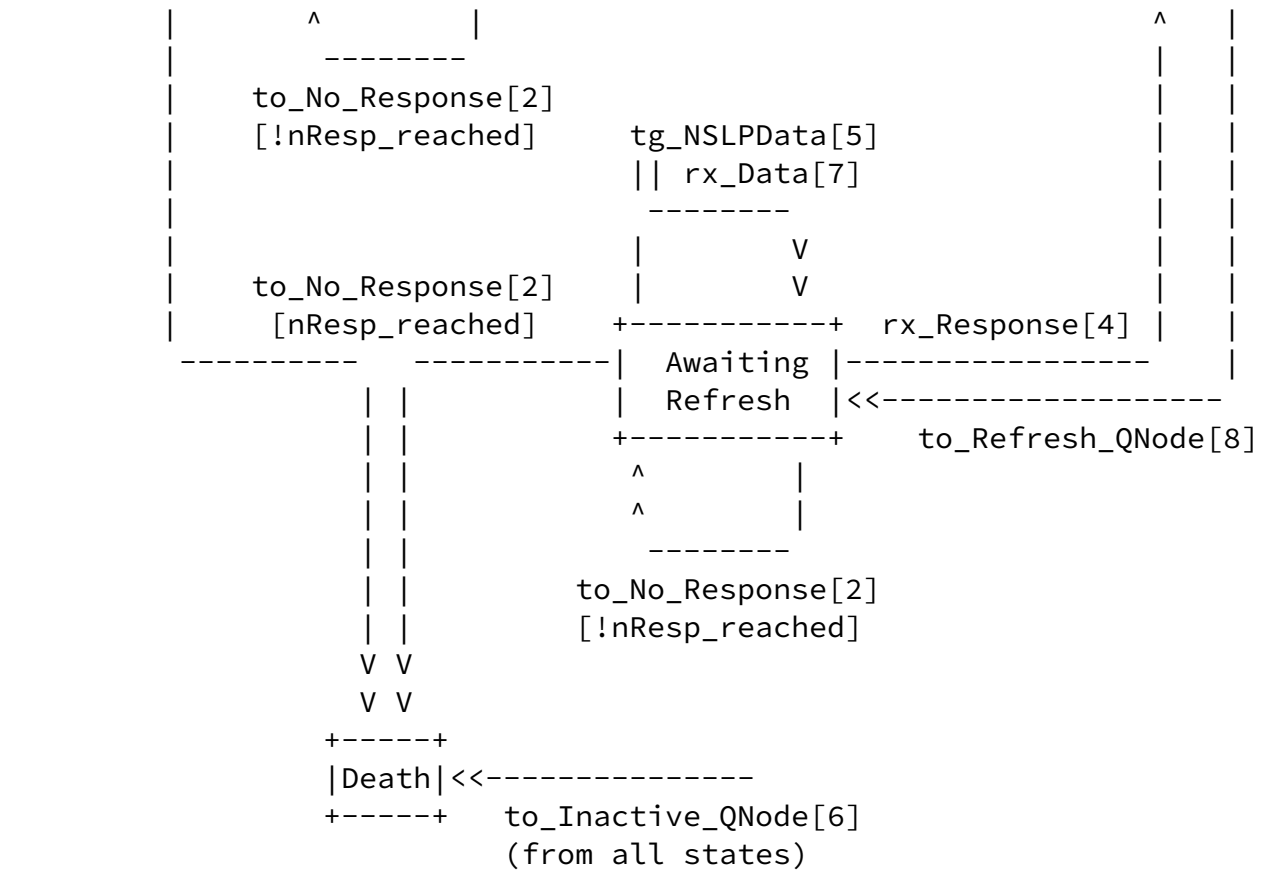


Figure 6: Query Node State Machine

The processing rules are as follows:

Rule 1: Store the message for later transmission

Rule 2:


```
if number of Queries sent has reached the threshold
  // nQuery_isMax is true
  indicate No Response error to NSLP
  destroy self
else
  send Query message
  start No_Response timer with new value
```

```
Rule 3:

// Assume the Confirm was lost in transit so resend it
// for the last Response we received
send Confirm message
restart Refresh_QNode and Inactive_QNode timers

Rule 4:

if a new MA-SM is needed create one
if the R flag was set send a Confirm message
pass any NSLP data to the NSLP
send any stored Data messages
stop No_Response timer
start Refresh_QNode and Inactive_QNode timers

Rule 5:

send Data message
restart Inactive_QNode timer

Rule 6: Terminate

Rule 7:

pass any data to the NSLP
(re)start Inactive_QNode timer

Rule 8:

send Query message
start No_Response timer
stop Refresh_QNode timer
```

6.3. Responder Node Processing

The Responding-Node state machine (R-SM) has three states:

- Awaiting Confirm

- Established

- Awaiting Refresh

The policy governing the creation of the R-SM has 3 cases (ignoring the case of pure stateless operation where a Response may be generated or the message propagated forwards, but no routing state is created at the GIST level):

1. It is created on receiving a Query, no Confirm is requested.

2. It is created on receiving a Query, but a Confirm is requested. A timer is used to retransmit Response messages and the R-SM is destroyed if no valid Confirm is received.

3. It cannot be created until a valid Confirm is received (the initial Query will have been handled by the Node level machine).

In case 2 the R-SM is created in the Awaiting Confirm state, and remains there until a Confirm is received, at which point it transitions to Established. In cases 1 and 3 the R-SM is created directly in the Established state. Note that if the machine is created on receiving a Query, some of the message processing will already have been performed in the Node state machine. In the Established state the NSLP can send and receive data normally, and any additional rx_Confirm events MUST be silently ignored. The Awaiting Refresh state can be considered a substate of Established, where a Query has been received to begin the routing state refresh.

In the Awaiting Refresh state the R-SM behaves as in the Awaiting Confirm state, except that the NSLP can still send and receive data. In particular, in both states there is timer-based retransmission of Response messages until a Confirm is received; additional rx_Query events in these states MUST also generate a response and restart the no_Confirm timer.

The timers relevant to the operation of this state machine are as follows:

Expire_RNode: Indicates when the routing state stored by this state machine needs to be expired. It is reset whenever a Query or Confirm (depending on local policy) is received indicating that the routing state is still valid. Note that state cannot be refreshed from the R-Node.

No_Confirm: Indicates that a Confirm has not been received in answer to a Response. This is started/reset whenever a Response is sent and stopped when a Confirm is received.

The detailed state transitions and processing rules are described below as in the Query node case.

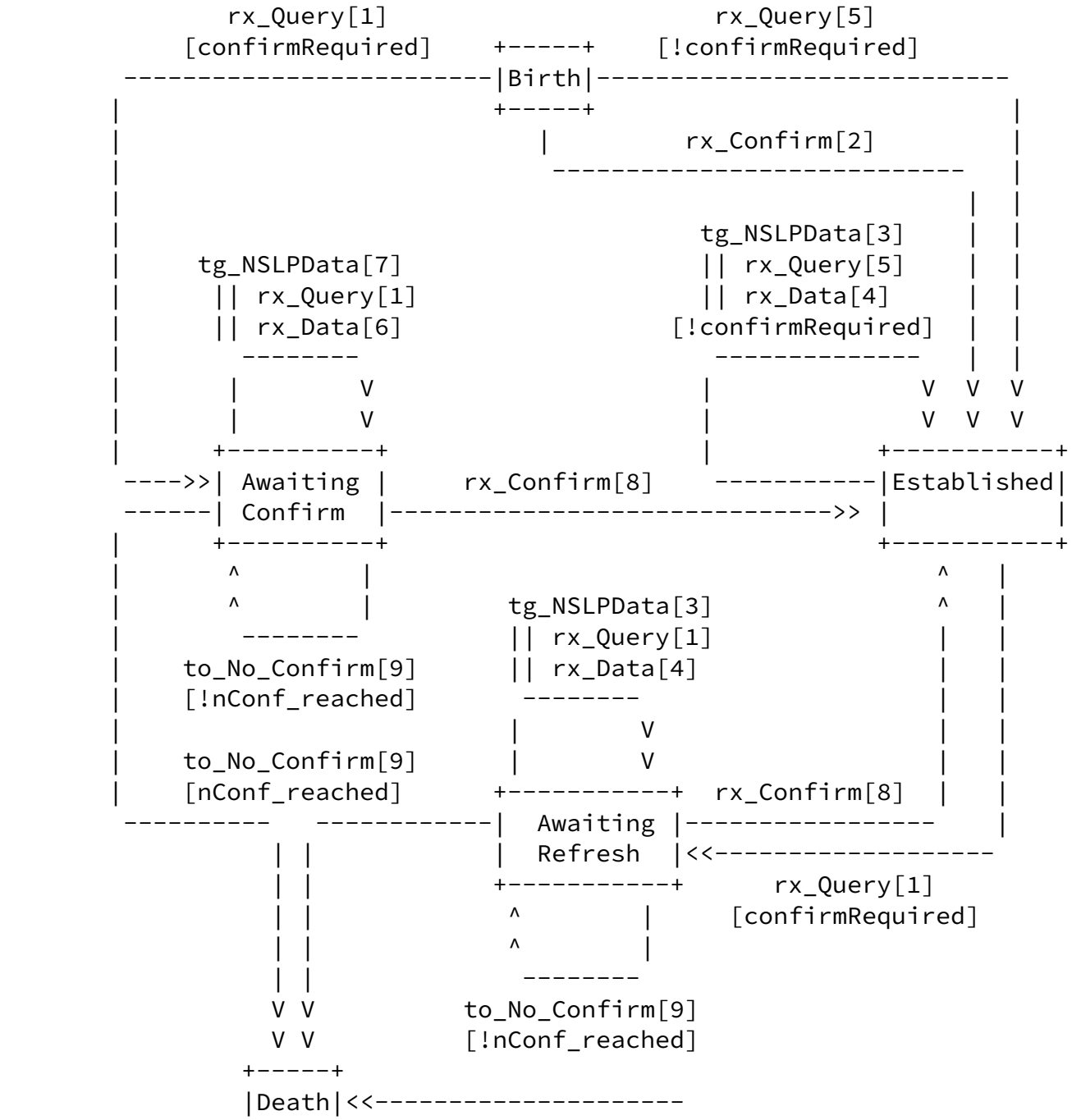


Figure 7: Responder Node State Machine

The processing rules are as follows:

Rule 1:

```
// a Confirm message is required
send Response message
(re)start No_Confirm timer
```

Rule 2:

```
pass any piggybacked data to the NSLP
if a new MA-SM would be needed for this peer
  create one in listening state
start Expire_RNode timer
```

Rule 3: send the Data message

Rule 4: pass data to NSLP

Rule 5:

```
// no Confirm message is required
send Response message
start Expire_RNode timer
```

Rule 6: send "No Routing State" error message

Rule 7: store Data message

Rule 8:

```
pass any piggybacked data to the NSLP
send any stored Data messages
stop No_Confirm timer
start Expire_RNode timer
```

```
Rule 9:

if number of Responses sent has reached threshold
  // nResp_isMax is true
  destroy self
else
  send Response message
  start No_Response timer
```



```
Rule 10: destroy self
```

6.4. Messaging Association Processing

Messaging associations are modelled for use within GIST with a simple 3-state process. The Awaiting Connection state indicates that the MA is waiting for the connection process(es) for every protocol in the messaging association to complete; this might involve creating listening endpoints or attempting active connects. Timers may also be necessary to detect connection failure (e.g. no incoming connection within a certain period), but these are not modelled explicitly. The Connected state indicates that the MA is open and ready to use. In addition there is an Idle state in which the local node no longer requires the messaging association but the remote node still wants it to be kept open.

Clearly, many internal details of the messaging association protocols are hidden in this model, especially where the messaging association uses multiple protocol layers. Note also that although the existence of messaging associations is not directly visible to signaling applications, there is some interaction between the two because security-related information becomes available during the open process, and this may be indicated to signaling applications if they have requested it.

The timers relevant to the operation of this state machine are as follows:

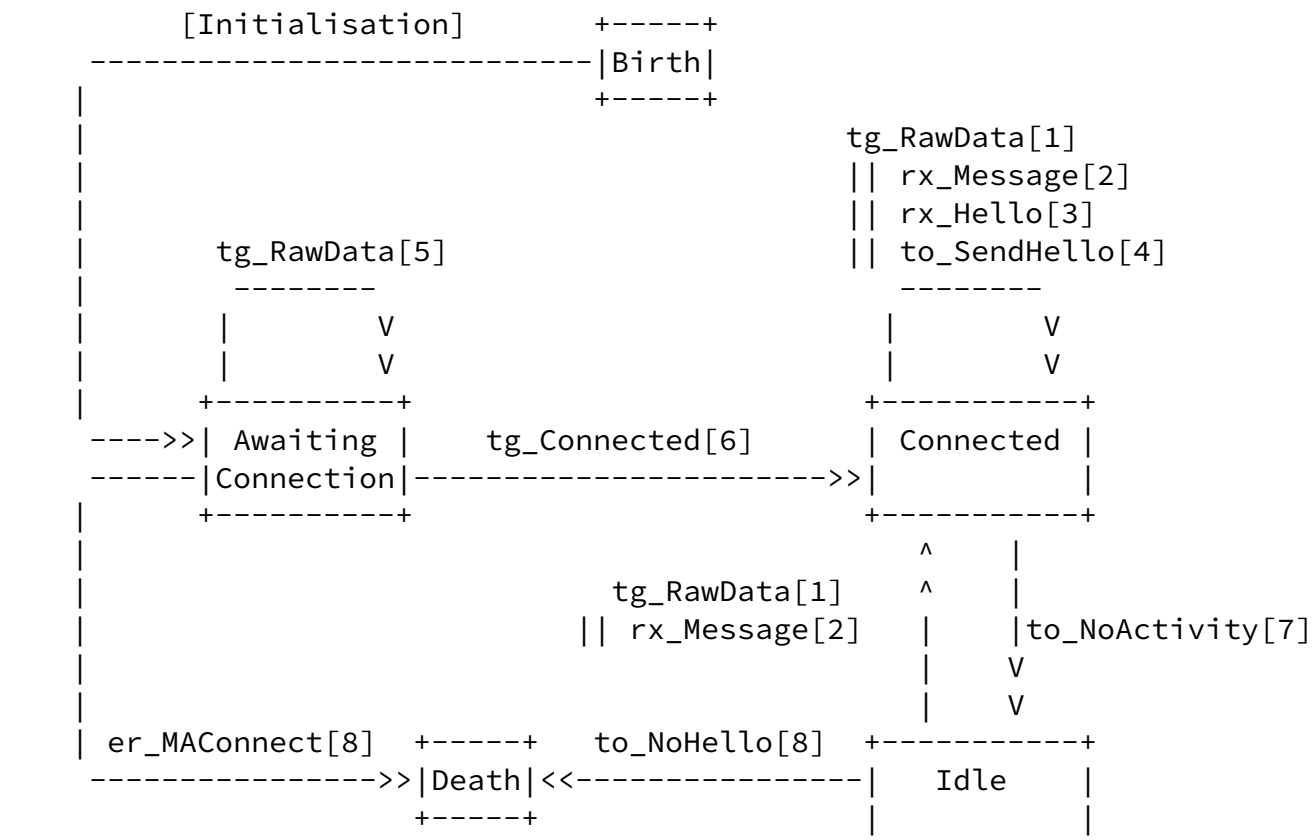
SendHello: Indicates that an MAHello message should be sent to the

remote node. The period of this timer is determined by the MA-Hold-Time sent by the remote node during the Query/Response/Confirm exchange.

NoHello: Indicates that no MAHello has been received from the remote node for a period of time. The period of this timer is sent to the remote node as the MA-Hold-Time during the Query/Response exchange.

NoActivity: Indicates that the link has been inactive for a period of time. The period of this timer is implementation specific but is likely to be related to the period of the NoHello timer.

The detailed state transitions and processing rules are described below as in the Query node case.



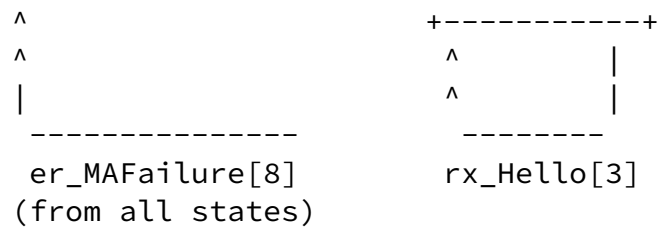


Figure 8: Messaging Association State Machine

The processing rules are as follows:

Rule 1:

pass message to transport layer
(re)start NoActivity timer
(re)start SendHello

Rule 2:

pass message to N-SM
(re)start NoActivity timer

Rule 3:

if reply requested
  send MA-Hello
restart NoHello timer

Rule 4:

send MA-Hello message
restart NoHello timer

Rule 5: queue message for later transmission

Rule 6:

```
pass outstanding queued messages to transport layer
stop any timers controlling connection establishment
start NoActivity timer
start SendHello timer

Rule 7:

stop NoActivity timer
stop sendHello timer
start NoHello timer

Rule 8: destroy self
```


7. Advanced Protocol Features

7.1. Route Changes and Local Repair

7.1.1. Introduction

When re-routing takes place in the network, GIST and signaling application state need to be updated for all flows whose paths have

changed. The updates to signaling application state are usually signaling application dependent: for example, if the path characteristics have actually changed, simply moving state from the old to the new path is not sufficient. Therefore, GIST cannot carry out the complete path update processing. Its responsibilities are to detect the route change, update its local routing state consistently, and inform interested signaling applications at affected nodes.

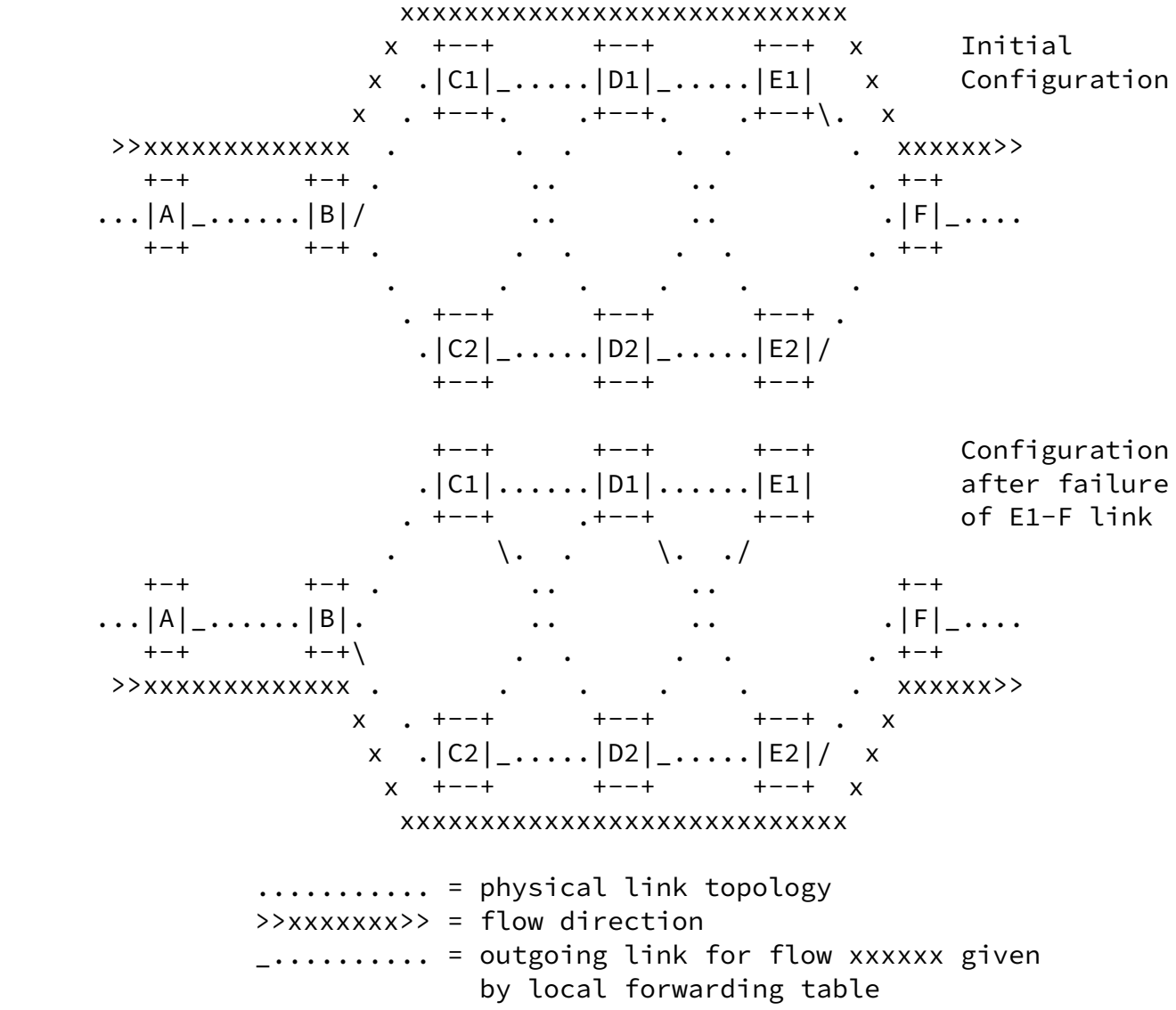


Figure 9: A Re-Routing Event

Route change management is complicated by the distributed nature of

the problem. Consider the re-routing event shown in Figure 9. An external observer can tell that the main responsibility for controlling the updates will probably lie with nodes B and F; however, E1 is best placed to detect the event quickly at the GIST level, and C1 and D1 could also attempt to initiate the repair.

On the assumption that signaling applications are soft-state based and operate end to end, and because GIST also periodically updates its picture of routing state, route changes will eventually be repaired automatically. The specification as already given includes this functionality. However, especially if upper layer refresh times are extended to reduce signaling load, the duration of inconsistent state may be very long indeed. Therefore, GIST includes logic to exchange prompt notifications with signaling applications, to allow local repair if possible. The additional mechanisms to achieve this are described in the following subsections. To a large extent, these additions can be seen as implementation issues; the protocol messages and their significance are not changed, but there are extra interactions through the API between GIST and signaling applications, and additional triggers for transitions between the various GIST states.

7.1.2. Route Change Detection Mechanisms

There are two aspects to detecting a route change at a single node:

- Detecting that the outgoing path, in the direction of the Query, has (or may have) changed.

- Detecting that the incoming path, in the direction of the Response, has (or may have) changed (in which case the node may no longer be on the path at all).

At a single node, these processes are largely independent, although clearly a change in one direction at a node corresponds to a change the opposite direction at its peer. Note that there are two possible forms for a route change: the interface through which a flow leaves or enters a node may change, and the adjacent peer may change. In general, a route change can include one or the other or both (or indeed neither, although such changes are very hard to detect).

The route change detection mechanisms available to a node depend on the MRM in use and the role the node played in setting up the routing state in the first place (i.e. as Querying or Responding node). The following discussion is specific to the case of the path-coupled MRM using downstream Queries only; other scenarios may require other methods. However, the repair logic described in the subsequent

subsections is intended to be universal.

There are five mechanisms for a node to detect that a route change has occurred, which are listed below. They apply differently depending on whether the change is in the Query or Response direction, and these differences are summarised in the following table.

Local Trigger: In trigger mode, GIST finds out from the local forwarding table that the next hop has changed. This only works if the routing change is local, not if it happens a few routing hops away (including the case that it happens at a GIST-unaware node).

Extended Trigger: Here, GIST checks a link-state topology database to discover that the path has changed. This makes certain assumptions on consistency of route computation and only works within a single area for OSPF and similar link-state protocols. Where available, this offers the most accurate and rapid indication of route changes, but requires more access to the routing internals than a typical operating system may provide.

GIST C-mode Monitoring: GIST may find that C-mode packets are arriving (from either peer) with a different TTL or on a different interface. This provides no direct information about the new flow path, but indicates that routing has changed and that rediscovery may be required.

Data Plane Monitoring: The signaling application on a node may detect a change in behaviour of the flow, such as TTL change, arrival on a different interface, or loss of the flow altogether. The signaling application on the node is allowed to notify this information locally to GIST (Appendix B.6).

GIST Probing: According to the specification, each GIST node MUST periodically repeat the discovery (GIST-Query/GIST-Response) operation. The querying node will discover the route change by a modification in the Network-Layer-Information in the GIST-Response. The period can be negotiated independently for each GIST hop, so nodes that have access to the other techniques listed above MAY use long periods for the probing operation.

Method	Query direction	Response direction
Local Trigger	Discovers new interface (and peer if local)	Not applicable
Extended Trigger	Discovers new interface and may determine new peer	May determine that route from peer will have changed
C-Mode Monitoring	Provides hint that change has occurred	Provides hint that change has occurred
Data Plane Monitoring	Not applicable	NSLP informs GIST that a change may have occurred
Probing	Discovers changed NLI in GIST-Response	Discovers changed NLI in GIST-Query

7.1.3. GIST Behaviour Supporting Re-Routing

The GIST behaviour necessary to support re-routing can be modelled using a 3-level classification of the validity of each item of routing state. (This classification applies separately to the Querying and Responding node for each pair of GIST peers.) The levels are:

Bad: The routing state is either missing altogether, or not safe to use to send data.

Tentative: The routing state may have changed, but it is still usable for sending NSLP data pending verification.

Good: The routing state has been established and no events affecting it have since been detected.

These classifications are not identical to the states described in Section 6, but there are dependencies between them. Specifically,

routing state is considered Bad until the machine first enters the Established state, at which point it becomes Good. Thereafter, the status may be invalidated for any of the reasons discussed above; it is an implementation issue to decide which techniques to implement in any given node, and how to reclassify routing state (as Bad or Tentative) for each. The status returns to Good, either when the state machine re-enters the Established state, or if GIST can determine from direct examination of the routing or forwarding tables that the peer has not changed. When the status returns to Good, GIST

MUST if necessary update its routing state table so that the relationships between MRI/SID/NSLPID tuples and messaging associations are up to date.

When classification of the routing state for the downstream direction changes to Bad/Tentative because of local routing indications, GIST MAY automatically change the classification in the upstream direction to Tentative unless local routing indicates that this is not necessary. This SHOULD NOT be done in the case where the initial change was indicated by the signaling application. This mechanism accounts for the fact that a routing change may affect several nodes, and so can be an indication that upstream routing may also have changed. In any case, whenever GIST updates the routing status, it informs the signaling application with the NetworkNotification API (Appendix B.4), unless the change was caused via the API in the first place.

The GIST behaviour for state repair is different for the Querying and Responding node. At the Responding node, there is no additional behaviour, since the Responding node cannot initiate protocol transitions autonomously, it can only react to the Querying node. The Querying node has three options, depending on how the transition from 'Good' was initially caused:

1. To inspect the routing/forwarding table and verifying that the next peer has not changed. This technique MUST NOT be used if the transition was caused by a signaling application, but SHOULD be used otherwise if available.

2. To move to the 'Awaiting Refresh' state. This technique MUST NOT be used if the current status is 'Bad', since data is being incorrectly delivered.

3. To move to the 'Awaiting Response' state. This technique may be used at any time, but has the effect of freezing NSLP communication while GIST state is being repaired.

The second and third techniques trigger the execution of a GIST handshake to carry out the repair. It may be desirable to delay the start of the handshake process, either to wait for the network to stabilise, to avoid flooding the network with Query traffic for a large number of affected flows, or to wait for confirmation that the node is still on the path from the upstream peer. One approach is to delay the handshake until there is NSLP data to be transmitted. Implementation of such delays is a matter of local policy; however, GIST MUST begin the handshake immediately if the status change was caused by an InvalidateRoutingState API call marked as 'Urgent', and SHOULD begin it if the upstream routing state is still known to be

Good.

7.1.4. Signaling Application Operation

Signaling applications can use these functions as provided by GIST to carry out rapid local repair following re-routing events. The signaling application instances carry out the multi-hop aspects of the procedure, including crossover node detection, and tear-down/ reinstallation of signaling application state; they also trigger GIST to carry out the local routing state maintenance operations over each individual hop. The local repair procedures depend heavily on the fact that stateful NSLP nodes are a single GIST hop apart; this is enforced by the details of the GIST peer-discovery process.

The following outline description of a possible set of NSLP actions takes the scenario of Figure 9 as an example.

1. The signaling application at node E1 is notified by GIST of route changes affecting the downstream and upstream directions. The downstream status was updated to Bad because of a trigger from the local forwarding tables, and the upstream status changed automatically to Tentative as a consequence. The signaling application at E1 MAY begin local repair immediately, or MAY propagate a notification upstream to D1 that re-routing has occurred.

2. The signaling application at node D1 is notified of the route change, either by signaling application notifications or from the GIST level (e.g. by a trigger from a link-state topology database). If the information propagates faster within the routing protocol, GIST will change the upstream/downstream routing state to Tentative/Bad automatically, and this will cause the signaling application to propagate the notification further upstream.

3. This process continues until the notification reaches node A. Here, there is no downstream routing change, so GIST only learns of the update via the signaling application trigger. Since the upstream status is still Good, it therefore begins the repair handshake immediately.

4. The handshake initiated by node A causes its downstream routing state to be confirmed as Good and unchanged there; it also confirms the (Tentative) upstream routing state at B as Good. This is enough to identify B as the crossover router, and the signaling application and GIST can begin the local repair process.

An alternative way to reach step (4) is that node B is able to determine autonomously that there is no likelihood of an upstream route change (e.g. it is an area border router and the route change is only intra-area). In this case, the signaling application and GIST will see that the upstream state is Good and can begin the local repair directly.

After a route change, a signaling application may wish to remove state at another node which is no longer on the path. However, since it is no longer on the path, in principle GIST can no longer send messages to it. (In general, provided this state is soft, it will time out anyway; however, the timeouts involved may have been set to be very long to reduce signaling load.) The requirement to remove state in a specific peer node is identified in [28].

This requirement can be met provided that GIST is able to use the old path to the signaling application peer for some period while the signaling application still needs it. Since NSLP peers are a single

GIST hop apart, the necessary information is just the old entry in the node's routing state table for that flow. Rather than requiring the GIST level to maintain multiple generations of this information, it can just be provided to the signaling application in the same node (in an opaque form), which can store it if necessary and provide it back to the GIST layer in case it needs to be used. This information is denoted as 'SII-Handle' in the abstract API of Appendix B. Messages sent this way MUST bypass the GIST routing state tables at the sender, and this is indicated by setting the E flag in the common header (Appendix A.1); at the receiver, GIST MUST NOT validate the MRI/SID/NSLPID against local routing state and instead indicates the mode of reception to signaling applications through the API (Appendix B.2). Signaling applications should validate the source and effect of the message themselves, and if appropriate should in particular indicate to GIST (see Appendix B.5) that routing state is no longer required for this flow. This is necessary to prevent GIST in nodes on the old path initiating routing state refresh and thus causing state conflicts at the crossover router.

7.2. NAT Traversal

GIST messages must carry packet addressing and higher layer information as payload data in order to define the flow signalled for. (This applies to all GIST messages, regardless of how they are encapsulated or which direction they are travelling in.) At an addressing boundary the data flow packets will have their headers translated; if the signaling payloads are not translated consistently, the signaling messages will refer to incorrect (and probably meaningless) flows after passing through the boundary. In addition, GIST handshake messages carry additional addressing

information about the GIST nodes themselves, and this must also be processed appropriately when traversing a NAT.

The simplest solution to this problem is to require that a NAT is GIST-aware, and to allow it to modify messages based on the contents of the MRI. (This makes the assumption that NATs only rewrite the header fields included in this payload, and not other higher layer identifiers.) Provided this is done consistently with the data flow header translation, signaling messages will be valid each side of the boundary, without requiring the NAT to be signaling application aware. (Note, however, that if the NAT does not understand the MRI,

it should reject the message with an appropriate error.)

This specification defines an additional object that a NAT can insert into Query-encapsulated messages and which is echoed back in any responses to those messages. The new object, the NAT-Traversal object (Appendix A.3.8), carries the translation between the 'public' and 'private' MRI. It also carries a list of which other objects in the message have been translated. This should always include the NLI, and the Stack-Configuration-Data (if present); if GIST is extended with further objects that carry addressing data, this list allows a message receiver to know if the new objects were supported by the NAT. Finally, the NAT-Traversal object MAY be used to carry data to be used in back-translating datagram mode responses; this could be the original NLI or SCD, or opaque equivalents in the case of topology hiding.

A consequence of this approach is that the routing state tables at the signaling application peers (each side of the NAT) are no longer directly compatible. In particular, the values for Message-Routing-Information are different, which is why the unmodified MRI is propagated in the NAT-Traversal payload to allow subsequent C-mode messages to be interpreted correctly.

This specification does not define normative behaviour for a NAT translating GIST messages, since much of this will depend on NAT policy about allocating bindings; the description is purely informative. However, it does define the behaviour of a GIST node receiving a message containing a NAT-Traversal object.

A possible set of operations for a NAT to process a Query-encapsulated message is as follows:

1. Verify that bindings for any data flow are actually in place.

2. Create a new Message-Routing-Information object with fields modified according to the data flow bindings.

3. Create bindings for subsequent C-mode signaling (based on the information in the Network-Layer-Information and Stack-Configuration-Data objects).

4. Create new Network-Layer-Information and if necessary Stack-Configuration-Data objects with fields to force D-mode response messages through the NAT, and to allow C-mode exchanges using the C-mode signaling bindings.

5. Add a NAT-Traversal payload, listing the objects which have been modified and including the unmodified MRI and any other data needed to interpret the response. (If a NAT-Traversal object is already present, in the case of a sequence of NATs, the list of modified objects may be updated and further opaque data added, but the MRI contained in it is left unchanged.)

6. Encapsulate the message according to the normal rules of this specification for the Query-encapsulation. If the S-flag was set in the original message, the same IP source address selection policy should be applied to the forwarded message.

7. Forward the message with these new payloads.

A GIST node receiving such a message MUST verify that all mandatory objects containing addressing have been translated correctly, or else reject the message with an 'Object Type Error' message (Appendix A.4.4.9) with subcode 4 ('Untranslated Object'). The error message MUST include the NAT-Traversal object as the first TLV after the common header (this is true for any other error message generated as a response). Otherwise, the message is processed essentially as normal. If no state needs to be updated for the message, the NAT-Traversal object can be effectively ignored. The other possibility is that a Response must be returned, either because the message is the beginning of a handshake for a new flow, or it is a refresh for existing state. In both cases, the GIST node MUST create the Response message in the normal way using the 'local' form of the MRI, and its own NLI and (if necessary) SCD. It MUST also include the NAT-Traversal object as the first object in the Response after the common header.

A NAT will intercept datagram mode messages with the normal encapsulation containing such echoed NAT-Traversal objects. (All other GIST messages, either in connection mode, or datagram mode messages with no NAT-Traversal object, should be treated as 'normal' data traffic by the NAT, i.e. with IP and transport layer translation but no GIST-specific processing.) The NAT processing is a subset of the processing for the Query-encapsulated case:

1. Verify the existence of bindings for the data flow.

2. Leave the Message-Routing-Information object unchanged.

3. Modify the NLI and SCD objects for the Responding node if necessary, and create or update any bindings for C-mode signaling traffic.

4. Forward the message.

A GIST node receiving such a message MUST use the MRI from the NAT-Traversal object as the key to index its internal routing state; it MAY also store the 'translated' MRI for additional (e.g. diagnostic) information, but this is not used in the GIST processing. The remainder of GIST processing is unchanged.

Note that Confirm messages are not translated. Thus, a Responding node has available only the untranslated MRI describing the flow, and the untranslated NLI as peer routing state. This would prevent the correct interpretation of the signaling messages; also, subsequent Query (refresh) messages would always be seen as route changes because of the NLI change. Therefore, a Responding node that wishes to delay state installation until receiving a Confirm must somehow reconstruct the translations when the Confirm arrives. How to do this is an implementation issue; one approach is to carry the translated objects as part of the Responder cookie which is echoed in the Confirm. (Indeed, for one of the cookie constructions in Section 8.5 this is automatic.)

7.3. Interaction with IP Tunnelling

The interaction between GIST and IP tunnelling is very simple. An IP packet carrying a GIST message is treated exactly the same as any other packet with the same source and destination addresses: in other words, it is given the tunnel encapsulation and forwarded with the other data packets.

Tunnelled packets will not be identifiable as GIST messages until they leave the tunnel, since any router alert option and the standard GIST protocol encapsulation (e.g. port numbers) will be hidden within the standard tunnel encapsulation. If signaling is needed for the tunnel itself, this has to be initiated as a separate signaling session by one of the tunnel endpoints - that is, the tunnel counts as a new flow. Because the relationship between signaling for the 'microflow' and signaling for the tunnel as a whole will depend on the signaling application in question, it is a signaling application responsibility to be aware of the fact that tunnelling is taking

place and to carry out additional signaling if necessary; in other

words, at least one tunnel endpoint must be signaling application aware.

In some cases, it is the tunnel exit point (i.e. the node where tunnelled data and downstream signaling packets leave the tunnel) that will wish to carry out the tunnel signaling, but this node will not have knowledge or control of how the tunnel entry point is carrying out the data flow encapsulation. The information about how the inner MRI/SID relate to the tunnel MRI/SID needs to be carried in the signaling data from the tunnel entry point (this functionality is the equivalent to the RSVP SESSION_ASSOC object of [10]). In the NSIS protocol suite, these bindings are managed by the signaling applications, either implicitly (e.g. by SID re-use) or explicitly (by carrying objects that bind the inner and outer SIDs as part of the NSLP payload).

7.4. IPv4-IPv6 Transition and Interworking

GIST itself is essentially IP version neutral: version dependencies are isolated in the formats of the Message-Routing-Information, Network-Layer-Information and Stack-Configuration-Data objects, and GIST also depends on the version independence of the protocols that support messaging associations. In mixed environments, GIST operation will be influenced by the IP transition mechanisms in use. This section provides a high level overview of how GIST is affected, considering only the currently predominant mechanisms.

Dual Stack: (As described in [29].) In mixed environments, GIST MUST use the same IP version for Query-encapsulated messages as the flow it is signaling for, and SHOULD do so for other signaling also (see Section 5.2.2). The IP version used in datagram mode is closely tied to the IP version used by the data flow, so it is intrinsically impossible for a IPv4-only or IPv6-only GIST node to support signaling for flows using the other IP version. Hosts which are dual stack for applications and routers which are dual stack for forwarding need GIST implementations which can support both IP versions. Applications with a choice of IP versions might select a version based on which could be supported in the network by GIST, which could be established by invoking parallel discovery procedures.

Packet Translation: (Applicable to SIIT [5] and NAT-PT [11].) Some transition mechanisms allow IPv4 and IPv6 nodes to communicate by placing packet translators between them. From the GIST perspective, this should be treated essentially the same way as any other NAT operation (e.g. between 'public' and 'private' addresses) as described in Section 7.2. The translating node needs to be GIST-aware; it will have to translate the addressing

payloads between IPv4 and IPv6 formats for flows which cross between the two. The translation rules for the fields in the MRI payload (including e.g. DiffServ-codepoint and flow-label) are as defined in [5].

Tunnelling: (Applicable to 6to4 [13].) Many transition mechanisms handle the problem of how an end to end IPv6 (or IPv4) flow can be carried over intermediate IPv4 (or IPv6) regions by tunnelling; the methods tend to focus on minimising the tunnel administration overhead.

From the GIST perspective, the treatment should be as similar as possible to any other IP tunnelling mechanism, as described in Section 7.3. In particular, the end to end flow signaling will pass transparently through the tunnel, and signaling for the tunnel itself will have to be managed by the tunnel endpoints. However, additional considerations may arise because of special features of the tunnel management procedures. In particular, [14] is based on using an anycast address as the destination tunnel endpoint. GIST MAY use anycast destination addresses in the Query-encapsulation of D-mode messages if necessary, but MUST NOT use them in the Network-Layer-Information addressing field; normal unicast addresses MUST be used instead. Note that the addresses from the IP header are not used by GIST in matching requests and responses, so there is no requirement to use anycast source addresses.

8. Security Considerations

The security requirement for GIST is to protect the signaling plane against identified security threats. For the signaling problem as a whole, these threats have been outlined in [23]; the NSIS framework [22] assigns a subset of the responsibilities to the NTLP. The main issues to be handled can be summarised as:

Message Protection: Signaling message content can be protected against eavesdropping, modification, injection and replay while in transit. This applies both to GIST payloads, and GIST should also provide such protection as a service to signaling applications between adjacent peers.

Routing State Integrity Protection: It is important that signaling messages are delivered to the correct nodes, and nowhere else. Here, 'correct' is defined as 'the appropriate nodes for the signaling given the Message-Routing-Information'. In the case where the MRI is based on the Flow Identification for path-coupled signaling, 'appropriate' means 'the same nodes that the infrastructure will route data flow packets through'. (GIST has no role in deciding whether the data flow itself is being routed correctly; all it can do is ensure the signaling is routed consistently with it.) GIST uses internal state to decide how to route signaling messages, and this state needs to be protected against corruption.

Prevention of Denial of Service Attacks: GIST nodes and the network have finite resources (state storage, processing power, bandwidth). The protocol tries to minimise exhaustion attacks against these resources and not allow GIST nodes to be used to launch attacks on other network elements.

The main additional issue is handling authorisation for executing signaling operations (e.g. allocating resources). This is assumed to be done in each signaling application.

In many cases, GIST relies on the security mechanisms available in messaging associations to handle these issues, rather than introducing new security measures. Obviously, this requires the interaction of these mechanisms with the rest of the GIST protocol to be understood and verified, and some aspects of this are discussed in Section 5.7.

8.1. Message Confidentiality and Integrity

GIST can use messaging association functionality, specifically in this version TLS (Section 5.7.3), to ensure message confidentiality

and integrity. Implementation of this functionality is REQUIRED but its use for any given flow or signaling application is OPTIONAL. In some cases, confidentiality of GIST information itself is not likely to be a prime concern, in particular since messages are often sent to parties which are unknown ahead of time, although the content visible even at the GIST level gives significant opportunities for traffic analysis. Signaling applications may have their own mechanism for securing content as necessary; however, they may find it convenient to rely on protection provided by messaging associations, since it runs unbroken between signaling application peers.

8.2. Peer Node Authentication

Cryptographic protection (of confidentiality or integrity) requires a security association with session keys. These can be established by an authentication and key exchange protocol based on shared secrets, public key techniques or a combination of both. Authentication and key agreement is possible using the protocols associated with the messaging association being secured (TLS incorporates this

functionality directly; IKE, IKEv2 or KINK could provide it for IPsec). GIST nodes rely on these protocols to authenticate the identity of the next hop, and GIST has no authentication capability of its own.

With routing state discovery, there are few effective ways to know what is the legitimate next or previous hop as opposed to an impostor. In other words, cryptographic authentication here only provides assurance that a node is 'who' it is (i.e. the legitimate owner of identity in some namespace), not 'what' it is (i.e. a node which is genuinely on the flow path and therefore can carry out signaling for a particular flow). Authentication provides only limited protection, in that a known peer is unlikely to lie about its role. Additional methods of protection against this type of attack are considered in Section 8.3 below.

It is an implementation issue whether peer node authentication should be made signaling application dependent; for example, whether successful authentication could be made dependent on presenting credentials related to a particular signaling role (e.g. "signaling for QoS"). The abstract API of Appendix B leaves open such policy and authentication interactions between GIST and the NSLP it is serving. However, it does allow applications to inspect the authenticated identity of the peer to which a message will be sent before transmission.

8.3. Routing State Integrity

Internal state in a node (see Section 4.2) is used to route messages.

If this state is corrupted, signaling messages may be misdirected.

In the case where the message routing method is path-coupled, the messages need to be routed identically to the data flow described by the MRI, and the routing state table is the GIST view of how these flows are being routed through the network in the immediate neighbourhood of the node. Routes are only weakly secured (e.g. there is no cryptographic binding of a flow to a route), and there is no authoritative information about flow routes other than the current state of the network itself. Therefore, consistency between GIST and network routing state has to be ensured by directly interacting with the routing mechanisms to ensure that the signaling peers are the

appropriate ones for any given flow. An overview of security issues and techniques in this context is provided in [33].

In one direction, peer identification is installed and refreshed only on receiving a GIST-Response message (compare Figure 4). This MUST echo the cookie from a previous GIST-Query message, which will have been sent along the flow path (in datagram mode, i.e. end-to-end addressed). Hence, only the true next peer or an on-path attacker will be able to generate such a message, provided freshness of the cookie can be checked at the querying node.

In the other direction, peer identification MAY be installed directly on receiving a GIST-Query message containing addressing information for the signaling source. However, any node in the network could generate such a message (indeed, many nodes in the network could be the genuine upstream peer for a given flow). To protect against this, three strategies are used:

Filtering: the receiving node MAY reject signaling messages which claim to be for flows with flow source addresses which could be ruled out by ingress filtering. An extension of this technique would be for the receiving node to monitor the data plane and to check explicitly that the flow packets are arriving over the same interface and if possible from the same link layer neighbour as the datagram mode signaling packets. (If they are not, it is likely that at least one of the signaling or flow packets is being spoofed.) Signaling applications SHOULD only install state on the route taken by the data itself.

Authentication (weak or strong): the receiving node MAY refuse to install upstream state until it has completed a GIST-Confirm handshake with the peer. This echoes the Response cookie of the GIST-Response, and discourages nodes from using forged source addresses. This also plays a role in denial of service prevention, see below. A stronger approach is to require full peer authentication within the messaging association, the

reasoning being that an authenticated peer can be trusted not to pretend that it is on path when it is not.

SID segregation: The routing state lookup for a given MRI and NSLPID MUST also take the SID into account. A malicious node can only

overwrite existing routing state if it can guess the corresponding SID; it can insert state with random SID values, but generally this will not be used to route messages for which state has already been legitimately established.

8.4. Denial of Service Prevention

GIST is designed so that in general each Query message only generates at most one Response, so that a GIST node cannot become the source of a denial of service amplification attack. (There is a special case of retransmitted Response messages, see Section 5.3.3.)

However, GIST can still be subjected to denial-of-service attacks where an attacker using forged source addresses forces a node to establish state without return routability, causing a problem similar to TCP SYN flood attacks. Furthermore, an adversary might use modified or replayed unprotected signaling messages as part of such an attack. There are two types of state attacks and one computational resource attack. In the first state attack, an attacker floods a node with messages that the node has to store until it can determine the next hop. If the destination address is chosen so that there is no GIST-capable next hop, the node would accumulate messages for several seconds until the discovery retransmission attempt times out. The second type of state-based attack causes GIST state to be established by bogus messages. A related computational/ network-resource attack uses unverified messages to cause a node to make AAA queries or attempt to cryptographically verify a digital signature.

We use a combination of two defences against these attacks:

1. The responding node need not establish a session or discover its next hop on receiving the GIST-Query message, but MAY wait for a GIST-Confirm message, possibly on a secure channel. If the channel exists, the additional delay is one one-way delay and the total is no more than the minimal theoretically possible delay of a three-way handshake, i.e., 1.5 node-to-node round-trip times. The delay gets significantly larger if a new connection needs to be established first.

2. The Response to the Query message contains a cookie, which is repeated in the Confirm. State is only established for messages that contain a valid cookie. The setup delay is also 1.5 round-

trip times. (This mechanism is similar to that in SCTP [12] and other modern protocols.)

Once a node has decided to establish routing state, there may still be transport and security state to be established between peers. This state setup is also vulnerable to denial of service attacks. GIST relies on the lower layer protocols that make up messaging associations to mitigate such attacks. In the current specification, the querying node is always the one wishing to establish a messaging association, so it is the responding node that needs to be protected.

Signaling applications can use the services provided by GIST to defend against certain (e.g. flooding) denial of service attacks. In particular, they can elect to process only messages from peers that have passed a return routability check or been authenticated at the messaging association level (see Appendix B.2). Signaling applications that accept messages under other circumstances (in particular, before routing state has been fully established at the GIST level) need to take this into account when designing their denial of service prevention mechanisms, for example by not creating local state as a result of processing such messages.

8.5. Requirements on Cookie Mechanisms

The requirements on the Query cookie can be summarised as follows:

Liveness: The cookie must be live (must change from one handshake to the next). To prevent replay attacks.

Unpredictability: The cookie must not be guessable (e.g. not from a sequence or timestamp). To prevent direct forgery based on seeing a history of captured messages.

Easily validated: It must be efficient for the Q-Node to validate that a particular cookie matches an in-progress handshake, for a routing state machine which already exists. To discard spoofed responses, or responses to spoofed queries.

Uniqueness: The cookie must be unique to a given handshake (since it is actually used to match the Response to a handshake anyway, e.g. during messaging association re-use).

Likewise, the requirements on the Responder cookie can be summarised as follows:

Liveness: The cookie must be live (must change from one handshake to the next). To prevent replay attacks.

Creation simplicity: The cookie must be lightweight to generate. To avoid resource exhaustion at the responding node.

Validation simplicity: It must be simple for the R-node to validate that an R-cookie was generated by itself (and no-one else), without storing state about the handshake it was generated for.

Binding: The cookie must be bound to the routing state that will be installed. To prevent use with different routing state e.g. in a modified Confirm. The routing state here includes:

The NLI of the Query

The MRI/NSLPID for the messaging

The interface on which the Query was received

A suitable implementation for the Q-Cookie is a cryptographically strong random number which is unique for this routing state machine handshake. A node SHOULD implement this or an equivalently strong mechanism.

A suitable implementation for the R-Cookie is as follows:

```
R-Cookie = liveness data + hash (locally known secret,
                                 Q-Node NLI, MRI, NSLPID,
                                 reception interface,
                                 liveness data)
```

A node SHOULD implement this or an equivalently strong mechanism. There are several alternatives for the liveness data. One is to use a timestamp like SCTP. Another is to give the local secret a (rapid) rollover, with the liveness data as the generation number of the secret, like IKEv2. In both cases, the liveness data has to be carried outside the hash, to allow the hash to be verified at the Responder. Another approach is to replace the hash with encryption under a locally known secret, in which case the liveness data does not need to be carried in the clear. Any symmetric cipher immune to known plaintext attacks can be used.

To support the validation simplicity requirement, the Responder can check the liveness data to filter out some blind (flooding) attacks before beginning any cryptographic cookie verification. To support this usage, the liveness data must be carried in the clear and not be easily guessable; this rules out the timestamp approach, and suggests

the use of sequence of secrets with the liveness data identifying the position in the sequence. The secret strength and rollover frequency must be high enough that the secret cannot be brute-forced during its lifetime. Note that any node can use a Query to discover the current liveness data, so it remains hard to defend against sophisticated attacks which disguise such 'probes' within a flood of Queries from forged source addresses. Therefore, it remains important to use an efficient hashing mechanism or equivalent.

If a node receives a message for which cookie validation fails, it MAY return an "Object Value Error" error message (Appendix A.4.4.10) with subcode 4 ("Invalid Cookie") to the sender, as well as dropping the message. However, doing so in general makes a node a source of backscatter. Therefore, this SHOULD only be enabled selectively, e.g. during initial deployment or debugging.

8.6. Security Protocol Selection Policy

This specification defines a single mandatory-to-implement security protocol (TLS, Section 5.7.3). However, it is possible to define additional security protocols in the future, for example to allow re-use with other types of credentials, or migrate towards protocols with stronger security properties. In addition, use of any security protocol for a messaging association is optional. Security protocol selection is carried out as part of the GIST handshake mechanism (Section 4.4.1).

The selection process may be vulnerable to downgrade attacks, where a man in the middle modifies the capabilities offered in the Query or Response to mislead the peers into accepting a lower level of protection than is achievable. There is a two part defence against such attacks (the following is based the same concepts as [18]):

1. The Response does not depend on the Stack-Proposal in the Query (see Section 5.7.1). Therefore, tampering with the Query message

has no effect on the resulting messaging association configuration.

2. The Responding node's Stack-Proposal is echoed in the Confirm. The Responding node checks this to validate that the proposal it made in the Response is the same as the one received by the Querying node. (Note that as a consequence of the previous point, the Responding node does not have to remember the proposal explicitly, since it is a static function of local policy.)

The validity of the second part depends on the strength of the security protection provided for the Confirm message. If the Querying node is prepared to create messaging associations with null

security properties (e.g. TCP only), the defence is ineffective, since the man in the middle can re-insert the original Responder's Stack-Proposal, and the Responding node will assume that the minimal protection is a consequence of Querying node limitations. However, if the messaging association provides at least integrity protection that cannot be broken in real-time, the Confirm cannot be modified in this way. Therefore, provided that the Querying node applies a security policy on the messaging association protocols it will create that ensures at least this minimal level of protection is met, it can be assured that the capability discovery process will result in the setup of a messaging association with the correct security properties as appropriate for the two peers involved.

8.7. Residual Threats

Taking the above security mechanisms into account, the main residual threats against NSIS are three types of on-path attack.

An on-path attacker who can intercept the initial Query can do most things it wants to the subsequent signaling. It is very hard to protect against this at the GIST level; the only defence is to use strong messaging association security to see whether the Responding node is authorised to take part in NSLP signaling exchanges. To some extent, this behaviour is logically indistinguishable from correct operation, so it is easy to see why defence is difficult. Note that an on-path attacker of this sort can do anything to the traffic as well as the signaling. Therefore, the additional threat induced by the signaling weakness seems tolerable.

At the NSLP level, there is a concern about transitivity of trust of correctness of routing along the signaling chain. The NSLP at the querying node can have good assurance that it is communicating with an on-path peer (or a node delegated by the on-path node). However, it has no assurance that the node beyond the responder is also on-path, or that the MRI (in particular) is not being modified by the responder to refer to a different flow. Therefore, if it sends signaling messages with payloads (e.g. authorisation tokens) which are "valuable" to nodes beyond the adjacent hop, it is up to the NSLP to ensure that the appropriate chain of trust exists, which must in general use messaging association (strong) security.

There is a further residual attack by a node which is not on the path of the flow, but is on the path of the Response, or is able to use a Response from one handshake to interfere with another. The attacker modifies the Response to cause the Querying node to form an adjacency with it rather than the true downstream node. In principle, this attack could be prevented by including an additional cryptographic object in the Response message which ties the Response to the initial

Query and the routing state and can be verified by the Querying node.

9. IANA Considerations

This section defines the registries and initial codepoint assignments for GIST. It also defines the procedural requirements to be followed by IANA in allocating new codepoints. Guidelines on the technical criteria to be followed in evaluating requests for new codepoint assignments are given for the overall NSIS protocol suite in a separate NSIS extensibility document [36].

This specification allocates the following codepoints in existing registries:

- Well-known UDP port XXX as the destination port for Query-encapsulated GIST messages (Section 5.3).

This specification creates the following registries with the structures as defined below:

NSLP Identifiers: Each signaling application requires the assignment of one of more NSLPIDs. The following NSLPID is allocated by this specification:

NSLPID	Application
0	Used for GIST messages not related to any signaling application.

Every other NSLPID MUST be associated with a specific RAO value (multiple NSLPIDs MAY be associated with the same value). The NSLPID is a 16 bit integer, and allocation policies for further values are as follows:

1-32703: IESG Approval

32704-32767: Private/Experimental Use

32768-65536: Reserved

GIST Message Type: The GIST common header (Appendix A.1) contains a 1 byte message type field. The following values are allocated by this specification:

MType	Message
0	Query
1	Response

2	Confirm
3	Data
4	Error
5	MAHello

Allocation policies for further values are as follows:

6-63: Standards Action

64-119: Expert Review

120-127: Private/Experimental Use

128-255: Reserved

Object Types: There is a 12-bit field in the object header (Appendix A.2). The following values for object type are defined by this specification:

OType	Object Type
0	Message Routing Information
1	Session ID
2	Network Layer Information
3	Stack Proposal
4	Stack Configuration Data
5	Query Cookie
6	Responder Cookie
7	NAT Traversal
8	NSLP Data
9	Error

Allocation policies for further values are as follows:

10-1023: Standards Action

1024-1999: Specification Required

2000-2047: Private/Experimental Use

2048-4095: Reserved

When a new object type is defined, the extensibility bits (A/B, see Appendix A.2.1) must also be defined.

Message Routing Methods: GIST allows multiple message routing methods (see Section 3.3). The message routing method is indicated in the leading byte of the MRI object (Appendix A.3.1). This specification defines the following values:

MRM	Message Routing Method
0	Path Coupled MRM

1	Loose End MRM

Allocation policies for further values are as follows:

2-63: Standards Action

64-119: Expert Review

120-127: Private/Experimental Use

128-255: Reserved

When a new MRM is defined, the information described in Section 3.3 must be provided.

MA-Protocol-IDs: Each upper layer protocol that can be used in a messaging association is identified by a 1-byte MA-Protocol-ID (Section 5.7). This is used as a tag in the Stack-Proposal and Stack-Configuration-Data objects (Appendix A.3.4 and Appendix A.3.5). The following values are defined by this specification:

MA-Protocol-ID	Higher Layer Protocol
1	TCP opened in the forwards direction
2	TLS initiated in the forwards direction

Allocation policies for further values are as follows:

3-63: Standards Action

64-119: Expert Review

120-127: Private/Experimental Use

128-255: Reserved

Allocating a new MA-Protocol-ID requires defining the format for the MA-protocol-options field (if any) in the Stack-Configuration-

Data object that is needed to define its configuration. Note that the MA-Protocol-ID is not an IP Protocol number (indeed, some of the messaging association protocols - such as TLS - do not have an IP Protocol number).

Error Codes/Subcodes: There is a 2 byte error code and 1 byte subcode in the Value field of the Error object (Appendix A.4.1). Error codes 1-12 are defined in Appendix A.4.4 together with subcodes 0-3 for code 1, 0-4 for code 9, 0-5 for code 10, and 0-2 for code 12. Additional codes and subcodes are allocated on a first-come, first served basis. When a new error code/subcode combination is allocated, the Error Class and the format of any associated error-specific information must also be defined.

10. Acknowledgements

This document is based on the discussions within the IETF NSIS working group. It has been informed by prior work and formal and informal inputs from: Cedric Aoun, Attila Bader, Roland Bless, Bob Braden, Marcus Brunner, Benoit Campedel, Luis Cordeiro, Elwyn Davies, Christian Dickmann, Pasi Eronen, Alan Ford, Xiaoming Fu, Ruediger Geib, Eleanor Hepworth, Thomas Herzog, Cheng Hong, Jia Jia, Cornelia Kappler, Georgios Karagiannis, Chris Lang, John Loughney, Allison Mankin, Jukka Manner, Pete McCann, Andrew McDonald, Glenn Morrow, Dave Oran, Andreas Pashalidis, Henning Peters, Tom Phelan, Takako Sanda, Charles Shen, Melinda Shore, Martin Stiemerling, Martijn Swanink, Mike Thomas, Hannes Tschofenig, Sven van den Bosch, Michael Welzl, Lars Westberg, and Mayi Zoumaro-djayoon. Parts of the TLS usage description (Section 5.7.3) were derived from the Diameter base protocol specification, RFC3588. In addition, Hannes Tschofenig provided a detailed set of review comments on the security section, and Andrew McDonald provided the formal description for the initial packet formats. Chris Lang's implementation work provided objective feedback on the clarity and feasibility of the specification, and he also provided the state machine description and the initial error catalogue and formats.

11. References

11.1. Normative References

[1] Katz, D., "IP Router Alert Option", RFC 2113, February 1997.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[4] Partridge, C. and A. Jackson, "IPv6 Router Alert Option", RFC 2711, October 1999.

[5] Nordmark, E., "Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, February 2000.

[6] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[7] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[8] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.1", draft-ietf-tls-rfc2246-bis-13 (work in progress), June 2005.

11.2. Informative References

[9] Braden, B., Zhang, L., Berson, S., Herzog, S., and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[10] Terzis, A., Krawczyk, J., Wroclawski, J., and L. Zhang, "RSVP Operation Over IP Tunnels", RFC 2746, January 2000.

[11] Tsirtsis, G. and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)", RFC 2766, February 2000.

[12] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[13] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001.

[14] Huitema, C., "An Anycast Prefix for 6to4 Relay Routers", RFC 3068, June 2001.

[15] Baker, F., Iturralde, C., Le Faucheur, F., and B. Davie, "Aggregation of RSVP for IPv4 and IPv6 Reservations", RFC 3175, September 2001.

[16] Grossman, D., "New Terminology and Clarifications for Diffserv", RFC 3260, April 2002.

[17] Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., and J. Rosenberg, "Signaling Compression (SigComp)", RFC 3320, January 2003.

[18] Arkko, J., Torvinen, V., Camarillo, G., Niemi, A., and T.

Haukka, "Security Mechanism Agreement for the Session Initiation Protocol (SIP)", RFC 3329, January 2003.

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

[20] Rosenberg, J., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-08 (work in progress), September 2005.

[21] Gill, V., Heasley, J., and D. Meyer, "The Generalized TTL Security Mechanism (GTSM)", RFC 3682, February 2004.

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

[23] Tschofenig, H. and D. Kroeselberg, "Security Threats for Next Steps in Signaling (NSIS)", RFC 4081, June 2005.

[24] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[25] Kohler, E., "Datagram Congestion Control Protocol (DCCP)", draft-ietf-dccp-spec-13 (work in progress), December 2005.

[26] Conta, A., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", draft-ietf-ipngwg-icmp-v3-07 (work in progress), July 2005.

[27] Stiemerling, M., "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", draft-ietf-nsis-nslp-natfw-09 (work in progress),

February 2006.

[28] Manner, J., "NSLP for Quality-of-Service signalling", draft-ietf-nsis-qos-nslp-09 (work in progress), February 2006.

[29] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.

[30] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[31] Ylonen, T. and C. Lonvick, "SSH Protocol Architecture", draft-ietf-secsh-architecture-22 (work in progress), March 2005.

[32] Moskowitz, R., "Host Identity Protocol", draft-ietf-hip-base-04 (work in progress), October 2005.

[33] Nikander, P., Arkko, J., Aura, T., Montenegro, G., and E. Nordmark, "Mobile IP Version 6 Route Optimization Security Design Background", RFC 4225, December 2005.

[34] Floyd, S. and V. Jacobson, "The Synchronisation of Periodic Routing Messages", SIGCOMM Symposium on Communications Architectures and Protocols pp. 33--44, September 1993.

[35] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", draft-rescorla-dtls-05 (work in progress), June 2005.

[36] Loughney, J., "NSIS Extensibility Model", draft-loughney-nsis-ext-01 (work in progress), July 2005.

Appendix A. Bit-Level Formats and Error Messages

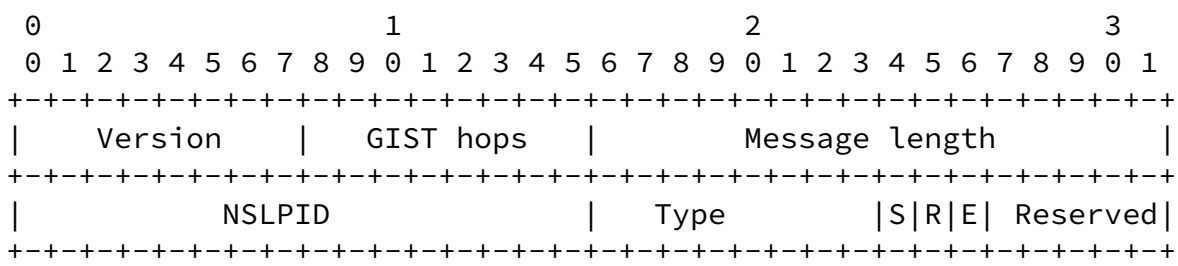
This appendix provides formats for the various component parts of the GIST messages defined abstractly in Section 5.2.

Each GIST message consists of a header and a sequence of objects. The GIST header has a specific format, described in more detail in Appendix A.1 below. An NSLP message is one object within a GIST message. Note that GIST itself provides the message length information and signaling application identification. General object formatting guidelines are provided in Appendix A.2 below, followed in Appendix A.3 by the format for each object. Finally, Appendix A.4 provides the formats used for error reporting.

In the following object diagrams, '//' is used to indicate a variable sized field and ':' is used to indicate a field that is optionally present.

A.1. The GIST Common Header

This header precedes all GIST messages. It has a fixed format, as shown below.



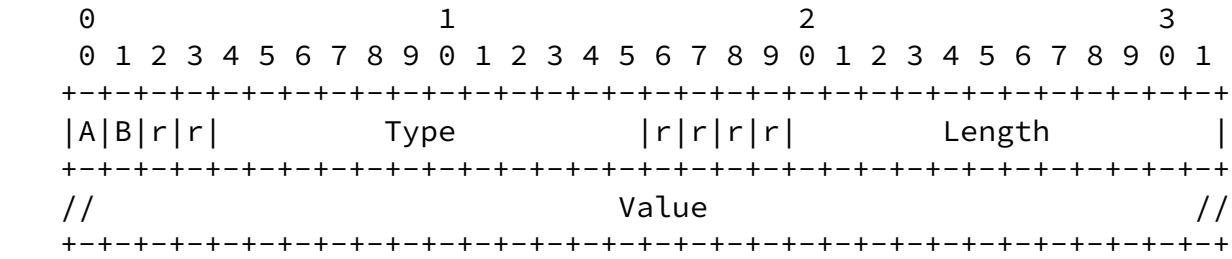

```
Message length = the total number of words in the message after
                 the common header itself
Type           = the GIST message type (Query, Response, etc.)
S flag         = S=1 if the IP source address is the same as the
                 signaling source address, S=0 if it is different
R flag         = R=1 if a reply to this message is explicitly
                 requested
E flag         = E=1 if the message was explicitly routed
Section 7.1.4
```

The rules governing the use of the R-flag depend on the GIST message type. It MUST always be set (R=1) in Query messages (these always elicit a Response), and never in Confirm, Data or Error messages. It is optional in a MA-Hello; if set, another MA-Hello is sent in reply. It is optional in a Response (but MUST be set if the Response contains a Responder cookie); if set, a Confirm is sent in reply.

Parsing failures may be caused by unknown Version or Type values, inconsistent R flag setting, or a Message Length inconsistent with the set of objects carried. In all cases the receiver MUST if possible return a "Common Header Parse Error" message (Appendix A.4.4.1) with the appropriate subcode, and not process the message further.

A.2. General Object Format

Each object begins with a fixed header giving the object Type and object Length. This is followed by the object Value, which is a whole number of 32-bit words long.



The individual components are as follows:

- The bits marked 'A' and 'B' are extensibility flags which are defined below; the remaining bits marked 'r' are reserved.

- Length has the units of 32 bit words, and measures the length of Value. If there is no Value, Length=0. If the Length is not consistent with the contents of the object, an "Object Value Error" message (Appendix A.4.4.10) with subcode 0 "Incorrect Length" MUST be returned and the message dropped.

- Value is (therefore) a whole number of 32 bit words. If there is any padding required, the length and location must be defined by the object-specific format information; objects which contain variable length (e.g. string) types may need to include additional length subfields to do so.

Any part of the object used for padding or defined as reserved (marked 'Reserved' or 'Rsv' in the diagrams below) MUST be set to 0 on transmission and MUST be ignored on reception.

A.2.1. Object Extensibility

The leading two bits of the TLV header are used to signal the desired treatment for objects whose Type field is unknown at the receiver.

The following three categories of object have been identified, and are described here.

AB=00 ("Mandatory"): If the object is not understood, the entire message containing it MUST be rejected with an "Object Type Error" message (Appendix A.4.4.9) with subcode 1 ("Unrecognised Object").

AB=01 ("Ignore"): If the object is not understood, it MUST be deleted and the rest of the message processed as usual.

AB=10 ("Forward"): If the object is not understood, it MUST be retained unchanged in any message forwarded as a result of message processing, but not stored locally.

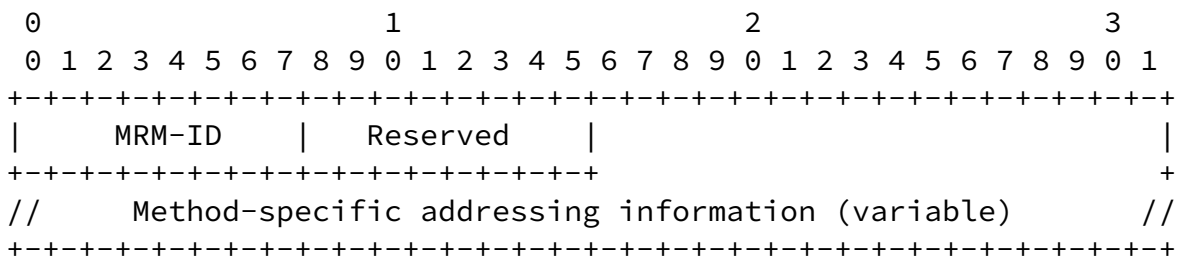
The combination AB=11 is reserved.

A.3. GIST TLV Objects

A.3.1. Message-Routing-Information

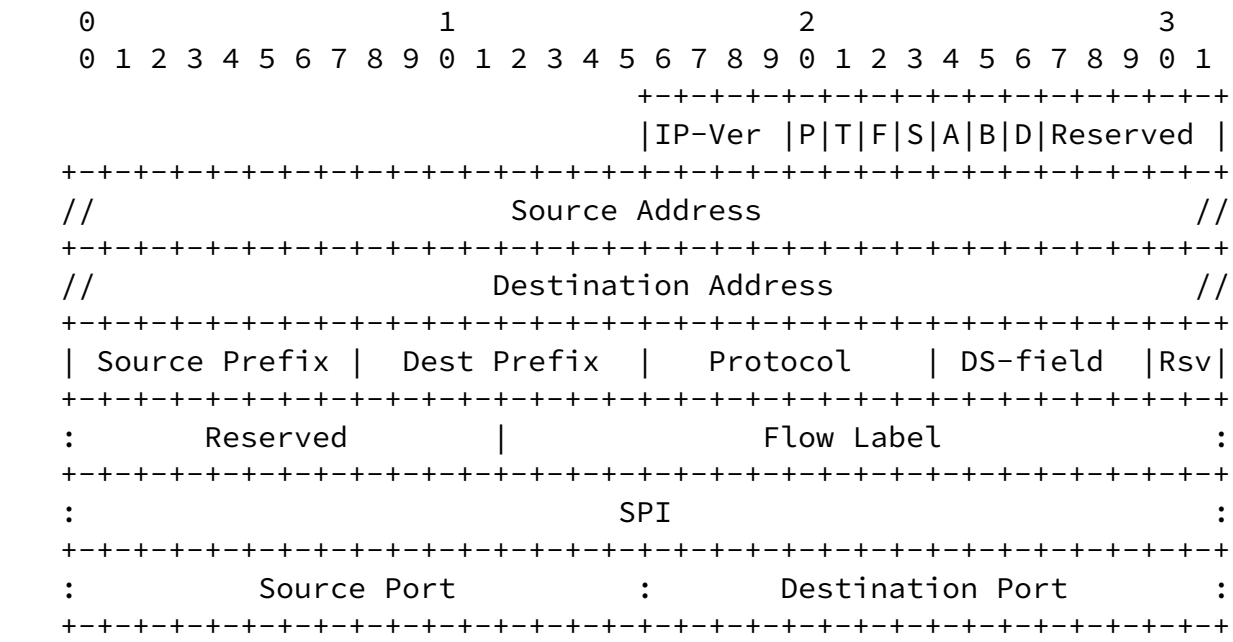
Type: Message-Routing-Information

Length: Variable (depends on message routing method)



A.3.1.1. Path-Coupled MRM

In the case of basic path-coupled routing, the addressing information takes the following format:



The flags are:
P - P=1 means that IP Protocol should be interpreted
T - T=1 means that DS-Field should be interpreted; see [3] and [16]
F - F=1 means that flow Label is present and should be interpreted
S - S=1 means that SPI is present and should be interpreted; see [30]
A/B - Source/Destination Port (see below)
D - Direction of message relative to flow

The source and destination addresses are always present and of the same type; their length depends on the value in the IP-Ver field. In the normal case where the MRI refers only to traffic between specific host addresses, the Source/Dest Prefix values would both be 32/128 for IPv4/6 respectively.

In the case of IPv6, the Protocol field refers to the true upper layer protocol carried by the packets, i.e. excluding any IP option headers. This is therefore not necessarily the same as the Next Header value from the base IPv6 header.

F may only be set if IP-Ver is 6. If F is not set, the entire 32 bit word for the Flow Label is absent.

The S/A/B flags can only be set if P is set. The SPI field is only present if the S flag is set.

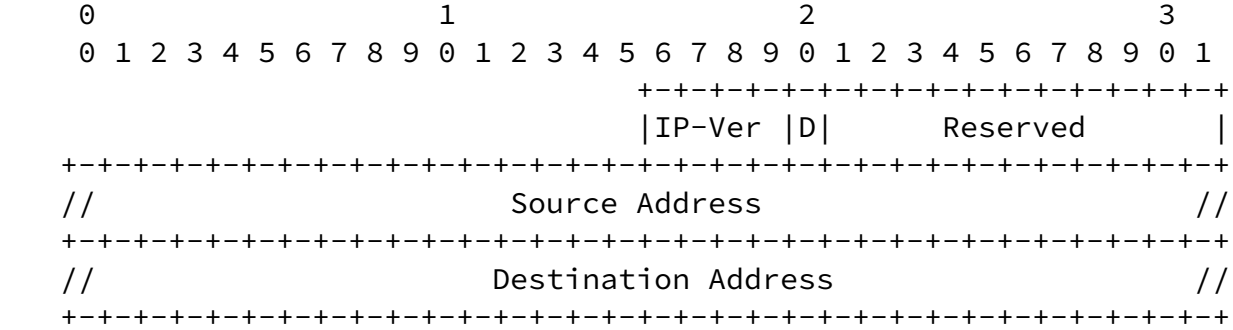
If either of A, B is set (value=1), the word containing the port numbers is included in the object. However, the contents of each field is only significant if the corresponding flag is set; otherwise, the contents of the field is regarded as padding, and the MRI refers to all ports (i.e. acts as a wildcard). If the flag is

set and Port=0x0000, the MRI will apply to a specific port, whose value is not yet known. If neither of A or B is set, the word is absent.

The Direction flag has the following meaning: the value 0 means 'in the same direction as the flow' (or "downstream"), and the value 1 means 'in the opposite direction to the flow' (or "upstream").

A.3.1.2. Loose-End MRM

In the case of the loose-end message routing method, the addressing information takes the following format:



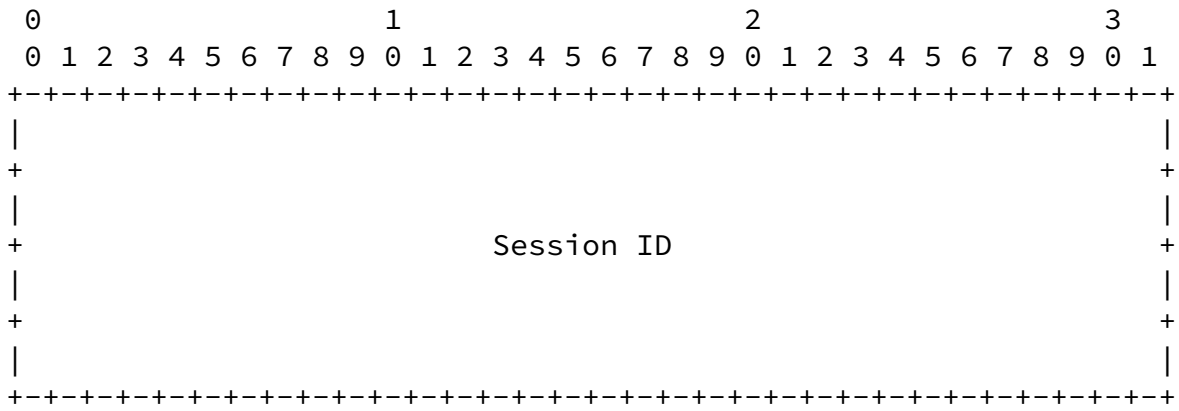
The only flag defined is:
D - Direction (always 0 for "downstream")

The source and destination addresses are always present and of the same type; their length depends on the value in the IP-Ver field.

A.3.2. Session Identification

Type: Session-Identification

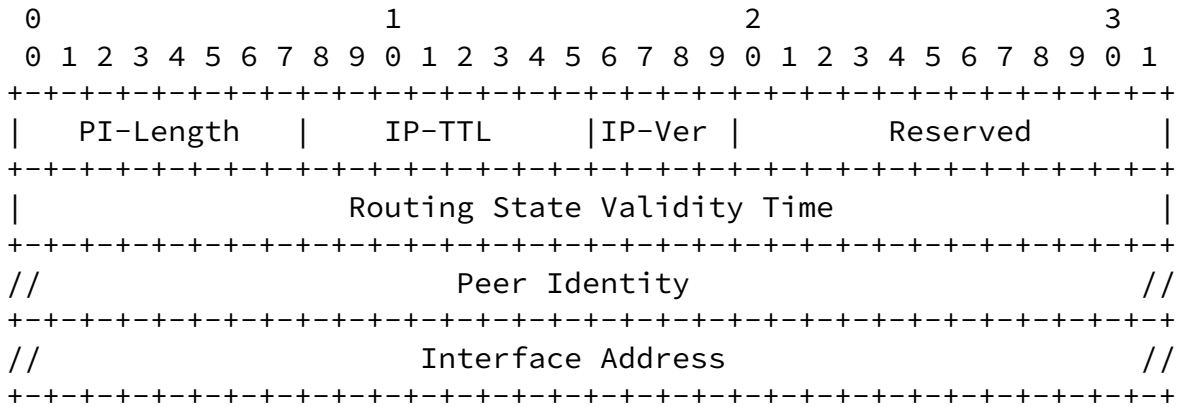
Length: Fixed (4 32-bit words)



A.3.3. Network-Layer-Information

Type: Network-Layer-Information

Length: Variable (depends on length of Peer-Identity and IP version)

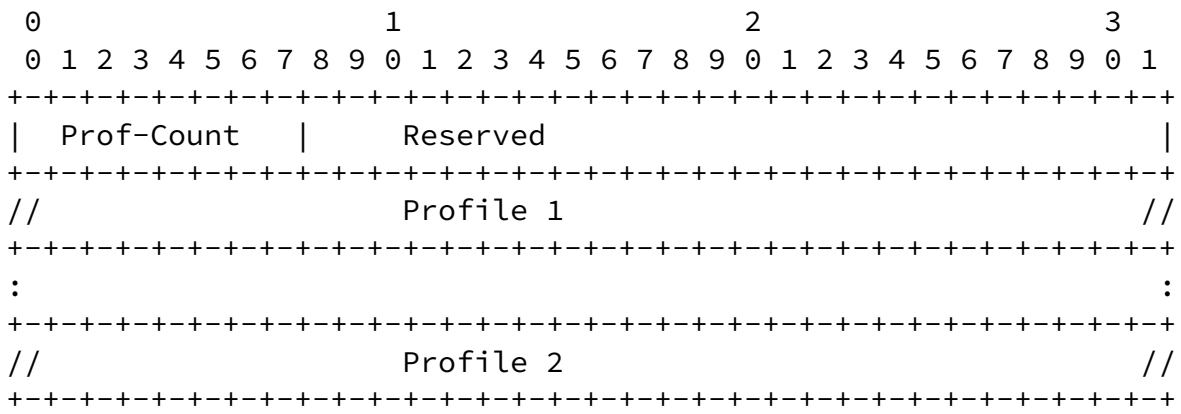


Routing State Validity Time = the time for which the routing state
for this flow can be considered correct without a
refresh. Given in milliseconds.
PI-Length = the byte length of the Peer-Identity field
(note that the Peer-Identity field itself is padded
to a whole number of words)
IP-TTL = initial or reported IP-TTL
IP-Ver = the IP version for the Interface-Address field

A.3.4. Stack Proposal

Type: Stack-Proposal

Length: Variable (depends on number of profiles and size of each profile)



Prof-Count = The number of profiles in the proposal. MUST be > 0.

Each profile is itself a sequence of protocol layers, and the profile is formatted as a list as follows:

- The first byte is a count of the number of layers in the profile.
- This is followed by a sequence of 1-byte MA-Protocol-IDs as described in Section 5.7.
- The profile is padded to a word boundary with 0, 1, 2 or 3 zero

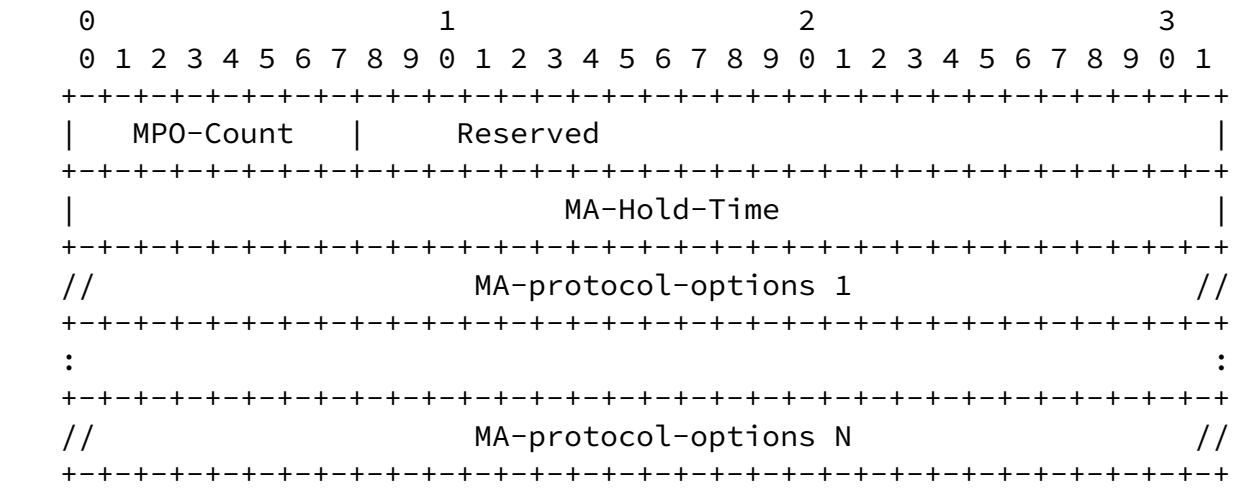
bytes.

If there are no profiles (i.e. all bytes are null), then a "Object Value Error" message (Appendix A.4.4.10) with subcode 3 ("Empty List") MUST be returned and the message dropped.

A.3.5. Stack-Configuration-Data

Type: Stack-Configuration-Data

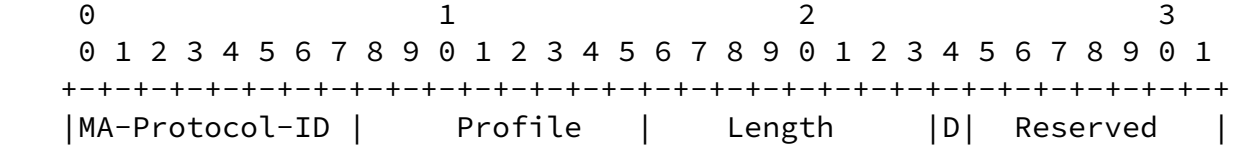
Length: Variable (depends on number of protocols and size of each MA-protocol-options field)

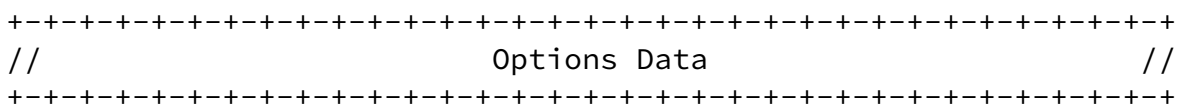


MA-Hold-Time = the time for which the messaging association will be held open without traffic or a hello message. Given in milliseconds.

MPO-Count = the number of MA-protocol-options fields present (these contain their own length information)

The MA-protocol-options fields are formatted as follows:





MA-Protocol-ID = Protocol identifier as described in Section 5.7

.

Profile = Tag indicating which profile from the accompanying Stack-Proposal object this applies to. Profiles are numbered from 1 upwards; the special value 0 indicates 'applies to all profiles'.

Length = the byte length of MA-protocol-options field that follows. This will be zero-padded up to the next word boundary.

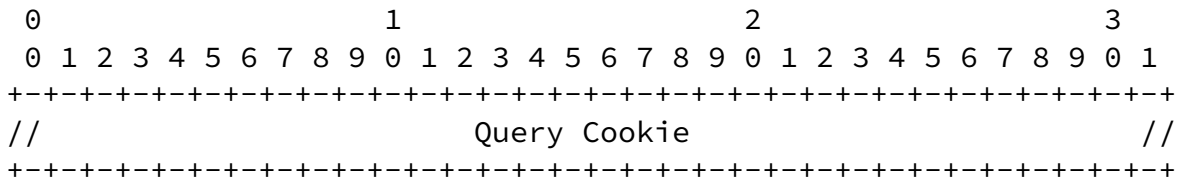
D flag = If set (D=1), this protocol MUST not be used for a messaging association.

Note that the format of the options data may differ depending on whether the field is in a GIST-Query or GIST-Response.

A.3.6. Query Cookie

Type: Query-Cookie

Length: Variable (selected by querying node)

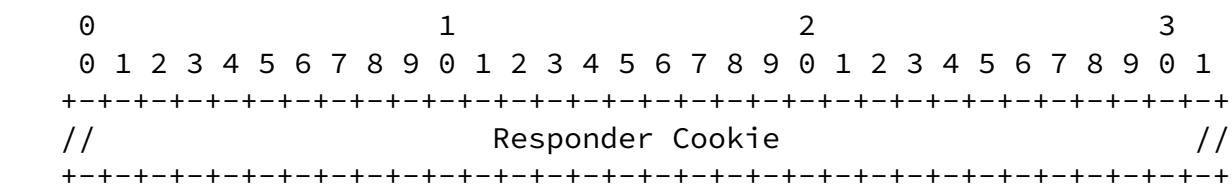


The contents are implementation defined. See Section 8.5 for further discussion.

A.3.7. Responder Cookie

Type: Responder-Cookie

Length: Variable (selected by responding node)



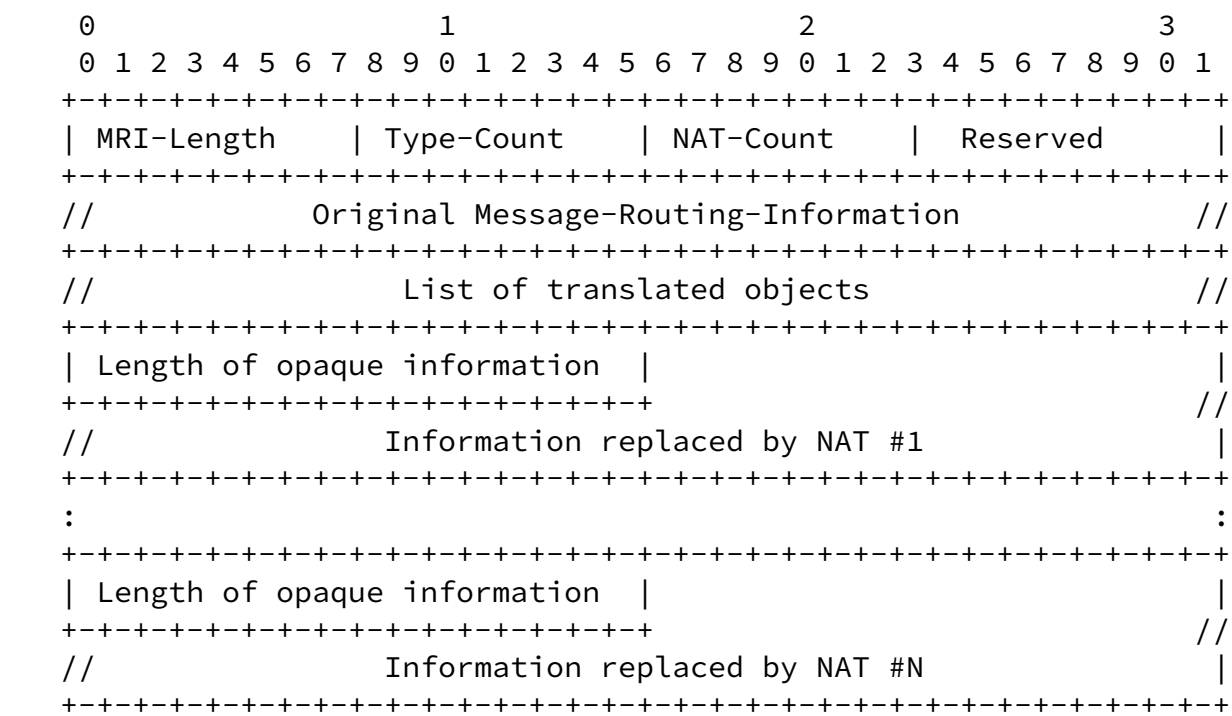
The contents are implementation defined. See Section 8.5 for further discussion.

A.3.8. NAT Traversal

Type: NAT-Traversal

Length: Variable (depends on length of contained fields)

This object is used to support the NAT traversal mechanisms described in Section 7.2.



MRI-Length = the word length of the included MRI payload
Type-Count = the number of GIST payloads translated by the NAT; the Type numbers are included as a list (padded with 2 null bytes if necessary)
NAT-Count = the number of NATs traversed by the message, and the number of opaque payloads at the end of the object

The length fields in the body of the message are byte counts, not including the 2 bytes of the length field itself. Note that each

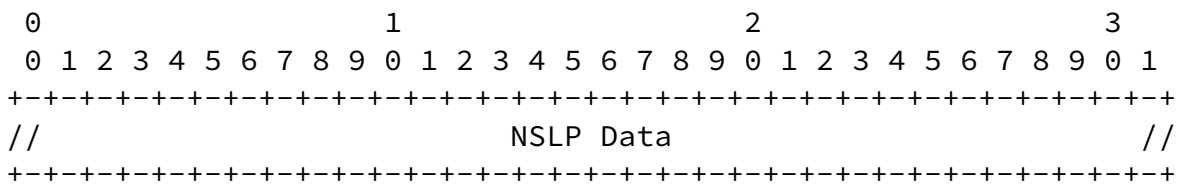
opaque information field is zero-padded to the next 32-bit word

boundary if necessary.

A.3.9. NSLP Data

Type: NSLP-Data

Length: Variable (depends on NSLP)

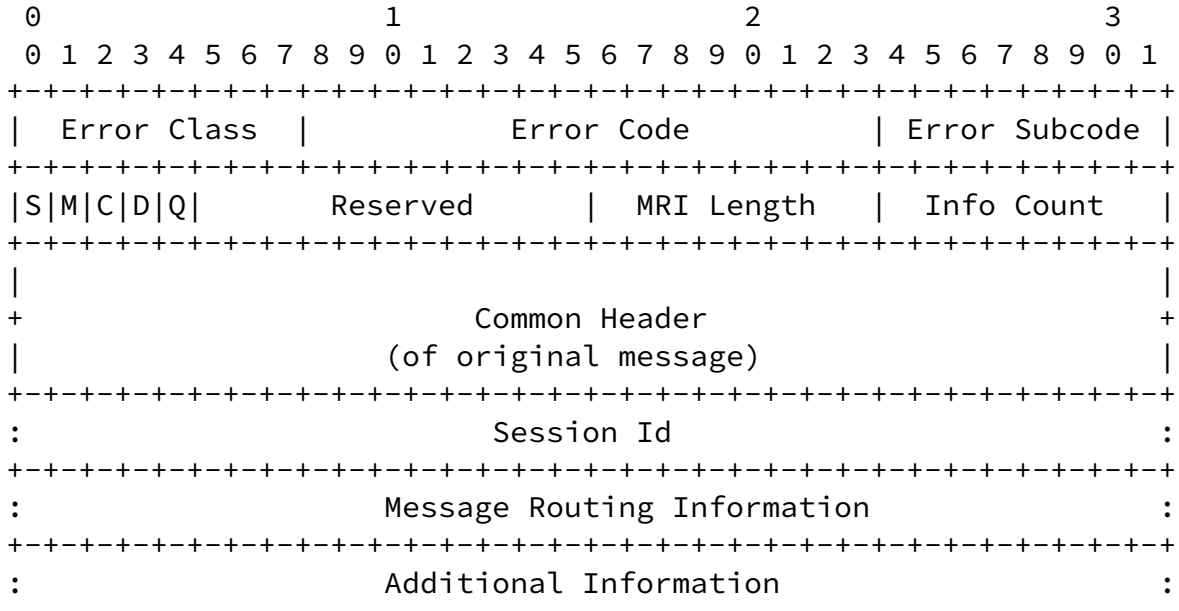


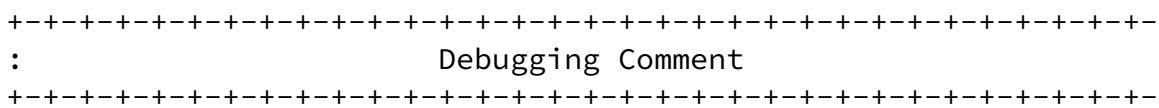
A.4. Errors

A.4.1. Error Object

Type: Error

Length: Variable (depends on error)





The flags are:
S - S=1 means the Session ID object is present
M - M=1 means MRI object is present
C - C=1 means a debug Comment is present after header.
D - D=1 means the original message was received in D-Mode
Q - Q=1 means the original message was received Q-Mode encapsulated (can't be set if D=0).

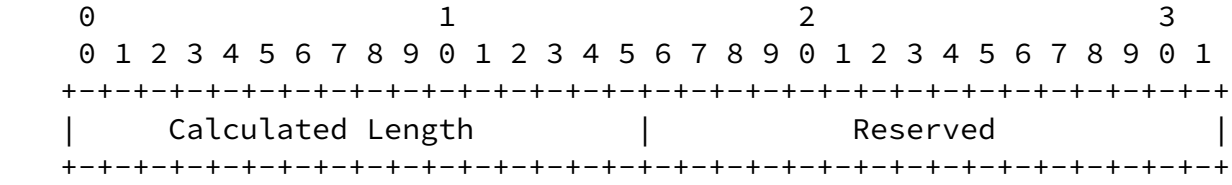
A GIST Error object contains an error-class (see Appendix A.4.3), an error-code, an error-subcode, and as much information about the message which triggered the error as is available. This information MUST include the Common header of the original message and SHOULD also include the Session Id and MRI objects if these could be decoded correctly. These objects are included in their entirety, except for their TLV Headers.

The Info Count field contains the number of Additional Information fields in the object. This count is usually 0 or 1, but may be more for certain messages; the precise set of fields to include is defined with the error code/subcode. The field formats are given in Appendix A.4.2 and their use for the different errors is given in the error catalogue Appendix A.4.4. The Debugging Comment is a null-terminated UTF-8 string, padded if necessary to a whole number of 32-bit words with more null characters.

A.4.2. Additional Information Fields

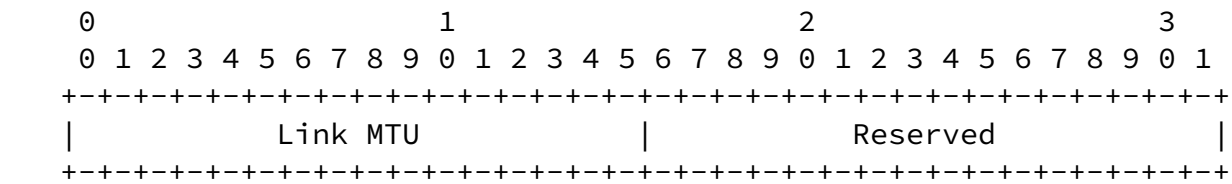
The Common Error Header may be followed by some Additional Information objects. The possible formats of these objects are shown below.

Message Length Info:



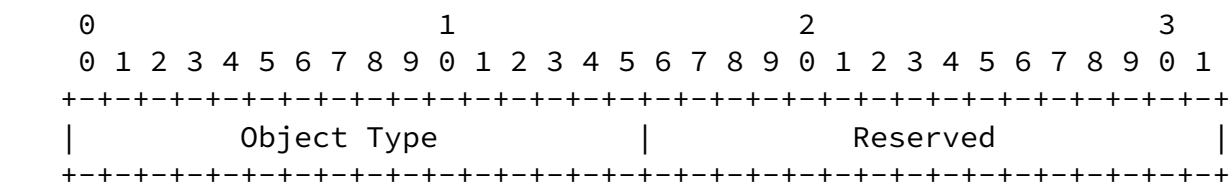
Calculated Length = the length of the original message calculated by adding up all the objects in the message.

MTU Info:



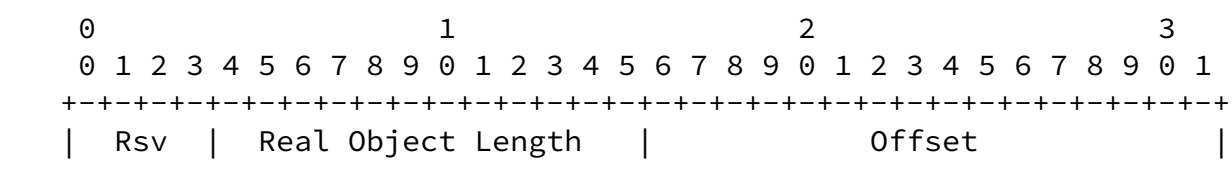
This object provides information about the MTU for a link along which a message could not be sent.

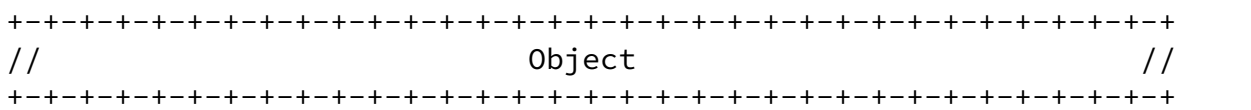
Object Type Info:



This object provides information about the type of object which caused the error.

Object Value Info:





Real Object Length: Since the length in the original TLV header may be inaccurate, this field provides the actual length of the object (including the TLV Header) included in the error message.

Offset: The byte in the object at which the GIST node found the error.

Object: The invalid TLV object (including the TLV Header)

This object carries information about a TLV object which was found to be invalid in the original message. An error message may contain more than one Object Value Info object.

A.4.3. Error Classes

The first byte of the error object, "Error Class", indicates the severity level. The currently defined severity levels are:

0 (Informational): response data which should not be thought of as changing the condition of the protocol state machine.

1 (Success): response data which indicates that the message being responded to has been processed successfully in some sense.

2 (Protocol-Error): the message has been rejected because of a protocol error (e.g. an error in message format).

3 (Transient-Failure): the message has been rejected because of a particular local node status which may be transient (i.e. it may be worthwhile to retry after some delay).

4 (Permanent-Failure): the message has been rejected because of local node status which will not change without additional out of band (e.g. management) operations.

Additional error class values are reserved.

The allocation of error classes to particular errors is not precise; the above descriptions are deliberately informal. Actual error processing should take into account the specific error in question; the error class may be useful supporting information (e.g. in network debugging).

A.4.4. Error Catalogue

This section lists all the possible GIST errors, including when they are raised and what additional information fields should be carried in the error object.

A.4.4.1. Common Header Parse Error

Class:	Protocol-Error
Code:	1
Additional Info:	Depends on subcode

This message is sent if a GIST node receives a message where the common header cannot be parsed correctly, or where an error in the overall message format is detected. Note that in this case the original MRI and Session ID are not included in the Error Object. This error code is split into subcodes as follows:

0: Unknown Version: The GIST version is unknown. The (highest) supported version supported by the node can be inferred from the Common Header of the GIST-Error message itself.

1: Unknown Type: The GIST message type is unknown.

2: Invalid R-flag: The R flag in the header is inconsistent with the message type.

3: Incorrect Message Length: The overall message length is not consistent with the set of objects carried. An Additional Info field of Message Length Info carries the calculated message length.

A.4.4.2. Hop Limit Exceeded

Class: Permanent-Failure
Code: 2
Additional Info: None

This message is sent if a GIST node receives a message with a GIST Hop Limit of zero, or a GIST node decrements a packet's GIST Hop Limit to zero. This message indicates either a routing loop or too small an initial Hop Limit value.

A.4.4.3. Incorrect Encapsulation

Class: Protocol-Error
Code: 3
Additional Info: None

This message is sent if a GIST node receives a message which uses an incorrect encapsulation method (e.g. a Query arrives over an MA).

A.4.4.4. Incorrectly Delivered Message

Class: Protocol-Error
Code: 4
Additional Info: None

This message is sent if a GIST node receives a message over an MA which is not associated with the MRI/NSLPID/SID combination in the message.

A.4.4.5. No Routing State

Class: Protocol-Error
Code: 5
Additional Info: None

This message is sent if a node receives a message for which routing state should exist, but has not yet been created (and thus there is

no appropriate Q/R-SM). This can occur either on receiving a Response to an unknown Query, or on receiving a Data message at a node whose policy requires routing state to exist before such messages can be accepted. See also Section 6.1 and Section 6.3.

A.4.4.6. Unknown NSLPID

```
Class:              Permanent-Failure
Code:               6
Additional Info:    None
```

This message is sent if a router receives a directly addressed message for an NSLP which it does not support.

A.4.4.7. Endpoint Found

```
Class:              Informational
Code:               7
Additional Info:    None
```

This message is sent if a GIST node at a flow endpoint receives a Query message for an NSLP which it does not support.

A.4.4.8. Message Too Large

```
Class:              Permanent-Failure
Code:               8
Additional Info:    MTU Info
```

A router receives a message which it can't forward because it exceeds the MTU on the next or subsequent hops.

A.4.4.9. Object Type Error

```
Class:              Protocol-Error
Code:               9
Additional Info:    Object Type Info
```

This message is sent if a GIST node receives a message containing a TLV object with an invalid type. The message includes the type of the object at fault. This error code is split into subcodes as follows:

0: Duplicate Object: This subcode is used if a GIST node receives a message containing multiple instances of an object which may only appear once in a message (in the current specification this applies to all objects).

1: Unrecognised Object: This subcode is used if a GIST node receive a message containing an object which it does not support, and the extensibility flags AB=00.

2: Missing Object: This subcode is used if a GIST node receives a message which is missing one or more mandatory objects. This message is also sent if a Stack-Proposal is sent without a matching Stack-Configuration-Data object when one was necessary, or vice versa.

3: Invalid Object: This subcode is used if the object type is known, but it is not valid for this particular GIST message type.

4: Untranslated Object: This subcode is used if the object type is known and is mandatory to interpret, but it contains addressing data which has not been translated by an intervening NAT.

A.4.4.10. Object Value Error

Class:	Protocol-Error
Code:	10
Additional Info:	Object Value Info

This message is sent if a router receives a packet containing an object which cannot be properly parsed. The message contains a single Object Value Info object, unless otherwise stated below. This error code is split into subcodes as follows:

0: Incorrect Length: The overall length does not match the object length calculated from the object contents.

1: Value Not Supported: The value of a field is not supported by the GIST node.

2: Invalid Flag-Field Combination: An object contains an invalid combination of flags and/or fields. At the moment this only relates to the Path-Coupled MRM object, but in future there may be more.

3: Empty List: At the moment this only relates to Stack-Proposals. The error message is sent if a stack proposal with a length > 0 (a length of 0 is handled as "Value Not Supported") contains only null bytes.

4: Invalid Cookie: The message contains a cookie which could not be verified by the node.

5: SP-SCD Mismatch: This subcode is used if a GIST node receives a message in which the data in the Stack-Proposal object is inconsistent with the information in the Stack Configuration Data object. In this case, both the Stack-Proposal object and Stack-Configuration-Data object are included in the message, in separate Object Value Info fields.

A.4.4.11. Invalid IP TTL

Class:	Permanent-Failure
Code:	11
Additional Info:	None

This error indicates that a message was received with an IP-TTL outside an acceptable range; for example, that an upstream Query was received with an IP-TTL of less than 254 (i.e. more than one IP hop from the sender). The actual IP distance can be derived from the IP-TTL information in the NLI object carried in the same message.

A.4.4.12. MRI Validation Failure

Class:	Permanent-Failure
Code:	12
Additional Info:	Object Value Info

This error indicates that a message was received with an MRI that could not be accepted, e.g. because of too much wildcarding or failing some validation check (cf. Section 5.8.1.2). The Object Value Info includes the MRI so the error originator can indicate the part of the MRI which caused the problem. The error code is divided into subcodes as follows:

0: MRI Too Wild: The MRI contained too much wildcarding (e.g. too short a destination address prefix) to be forwarded correctly down a single path.

1: IP Version Mismatch: The MRI in a path-coupled Query message uses an IP version which is not implemented on the interface used.

2: Ingress Filter Failure: The MRI in a path-coupled Query message describes a flow which would not pass ingress filtering on the interface used.

Appendix B. API between GIST and Signaling Applications

This appendix provides an abstract API between GIST and signaling applications. It should not constrain implementors, but rather help clarify the interface between the different layers of the NSIS protocol suite. In addition, although some of the data types carry the information from GIST information elements, this does not imply that the format of that data as sent over the API has to be the same.

Conceptually the API has similarities to the sockets API, particularly that for unconnected UDP sockets. An extension for an API like that for UDP connected sockets could be considered. In this case, for example, the only information needed in a SendMessage primitive would be NSLP-Data, NSLP-Data-Size, and NSLP-Message-Handle (which can be null). Other information which was persistent for a group of messages could be configured once for the socket. Such extensions may make a concrete implementation more efficient but do not change the API semantics, and so are not considered further here.

B.1. SendMessage

This primitive is passed from a signaling application to GIST. It is used whenever the signaling application wants to initiate sending a message.

```
SendMessage ( NSLP-Data, NSLP-Data-Size, NSLP-Message-Handle,
              NSLPID, Session-ID, MRI,
              SII-Handle, Transfer-Attributes, Timeout, IP-TTL, GHC )
```

The following arguments are mandatory.

NSLP-Data: The NSLP message itself.

NSLP-Data-Size: The length of NSLP-Data.

NSLP-Message-Handle: A handle for this message, that can be used by GIST as a reference in subsequent MessageStatus notifications (Appendix B.3). Notifications could be about error conditions or about the security attributes that will be used for the message. A NULL handle may be supplied if the NSLP is not interested in such notifications.

NSLPID: An identifier indicating which NSLP this is.

Session-ID: The NSIS session identifier. Note that it is assumed that the signaling application provides this to GIST rather than GIST providing a value itself.

MRI: Message routing information for use by GIST in determining the correct next GIST hop for this message. The MRI implies the message routing method to be used and the message direction.

The following arguments are optional.

SII-Handle: A handle, previously supplied by GIST, to a data structure that should be used to route the message explicitly to a particular GIST next hop.

Transfer-Attributes: Attributes defining how the message should be handled (see Section 4.1.2). The following attributes can be considered:

Reliability: Values 'unreliable' or 'reliable'.

Security: This attribute allows the NSLP to specify what level of security protection is requested for the message (selected from 'integrity' and 'confidentiality'), and can also be used to specify what authenticated signaling source and destination identities should be used to send the message. The possibilities can be learned by the signaling application from prior MessageStatus or RecvMessage notifications. If an NSLP-Message-Handle is provided, GIST will inform the signaling application of what values it has actually chosen for this attribute via a MessageStatus callback. This might take place either synchronously (where GIST is selecting from available messaging associations), or asynchronously (when a new

messaging association needs to be created).

Local Processing: This attribute contains hints from the signaling application about what local policy should be applied to the message; in particular, its transmission priority relative to other messages, or whether GIST should attempt to set up or maintain forward routing state.

Timeout: Length of time GIST should attempt to send this message before indicating an error.

IP-TTL: The value of the IP TTL that should be used when sending this message (may be overridden by GIST for particular messages).

GHC: The value for the GIST hop count when sending the message.

B.2. RecvMessage

This primitive is passed from GIST to a signaling application. It is used whenever GIST receives a message from the network, including the

case of 'null' messages (zero length NSLP payload), typically initial Query messages. For Queries, the results of invoking this primitive are used by GIST to check whether message routing state should be created (see the discussion of the 'Routing-State-Check' argument below).

```
RecvMessage ( NSLP-Data, NSLP-Data-Size, NSLPID, Session-ID, MRI,
              Routing-State-Check, SII-Handle, Transfer-Attributes,
              IP-TTL, IP-Distance, GHC )
```

NSLP-Data: The NSLP message itself (may be empty).

NSLP-Data-Size: The length of NSLP-Data (may be zero).

NSLPID: An identifier indicating which NSLP this is message is for.

Session-ID: The NSIS session identifier.

MRI: Message routing information that was used by GIST in forwarding this message. Implicitly defines the message routing method that was used and the direction of the message relative to the MRI.

Routing-State-Check: This boolean is True if GIST is checking with the signaling application to see if routing state should be created with the peer or the message should be forwarded further (see Section 4.3.2). If True, the signaling application should return the following values via the RecvMessage call:

A boolean indicating whether to set up the state.

Optionally, an NSLP-Payload to carry in the generated GIST-Response or forwarded Query message respectively.

This mechanism could be extended to enable the signaling application to indicate to GIST whether state installation should be immediate or deferred (see Section 5.3.3 and Section 6.3 for further discussion).

SII-Handle: A handle to a data structure, identifying a peer address and interface. Can be used to identify route changes and for explicit routing to a particular GIST next hop.

Transfer-Attributes: The reliability and security attributes that were associated with the reception of this particular message. As well as the attributes associated with SendMessage, GIST may indicate the level of verification of the addresses in the MRI. Three attributes can be indicated:

* Whether the signaling source address is one of the flow endpoints (i.e. whether this is the first or last GIST hop);

* Whether the signaling source address has been validated by a return routability check.

* Whether the message was explicitly routed (and so has not been validated by GIST as delivered consistently with local routing state).

IP-TTL: The value of the IP TTL this message was received with (if available).

IP-Distance: The number of IP hops from the peer signaling node which

sent this message along the path, or 0 if this information is not available.

GHC: The value of the GIST hop count the message was received with, after being decremented in the GIST receive-side processing.

B.3. MessageStatus

This primitive is passed from GIST to a signaling application. It is used to notify the signaling application that a message that it requested to be sent could not be dispatched, or to inform the signaling application about the transfer attributes that have been selected for the message (specifically, security attributes). The signaling application can respond to this message with a return code to abort the sending of the message if the attributes are not acceptable.

MessageStatus (NSLP-Message-Handle, Transfer-Attributes, Error-Type)

NSLP-Message-Handle: A handle for the message provided by the signaling application in SendMessage.

Transfer-Attributes: The reliability and security attributes that will be used to transmit this particular message.

Error-Type: Indicates the type of error that occurred. For example, 'no next node found'.

B.4. NetworkNotification

This primitive is passed from GIST to a signaling application. It indicates that a network event of possible interest to the signaling application occurred.

NetworkNotification (NSLPID, MRI, Network-Notification-Type)

NSLPID: An identifier indicating which NSLP this is message is for.

MRI: Provides the message routing information to which the network notification applies.

Network-Notification-Type: Indicates the type of event that caused the notification and associated additional data. Two events have been identified:

Last Node: GIST has detected that this is the last NSLP-aware node in the path. See Section 4.3.4.

Routing Status Change: GIST has installed new routing state, or has detected that the routing state may no longer be valid, or has re-established the routing state. See Section 7.1.3. The new status is reported; if the status is Good, the SII-Handle of the peer is also reported, as for RecvMessage.

B.5. SetStateLifetime

This primitive is passed from a signaling application to GIST. It indicates the duration for which the signaling application would like GIST to retain its routing state. It can also give a hint that the signaling application is no longer interested in the state.

SetStateLifetime (NSLPID, MRI, State-Lifetime)

NSLPID: Provides the NSLPID to which the routing state lifetime applies.

MRI: Provides the message routing information to which the routing state lifetime applies; includes the direction (in the D flag).

State-Lifetime: Indicates the lifetime for which the signaling application wishes GIST to retain its routing state (may be zero, indicating that the signaling application has no further interest in the GIST state).

B.6. InvalidateRoutingState

This primitive is passed from a signaling application to GIST. It indicates that the signaling application has knowledge that the next signaling hop known to GIST may no longer be valid, either because of changes in the network routing or the processing capabilities of signaling application nodes. See Section 7.1.

InvalidateRoutingState (NSLPID, MRI, Status, Urgent)

NSLPID: The NSLP originating the message. May be null (in which case the invalidation applies to all signaling applications).

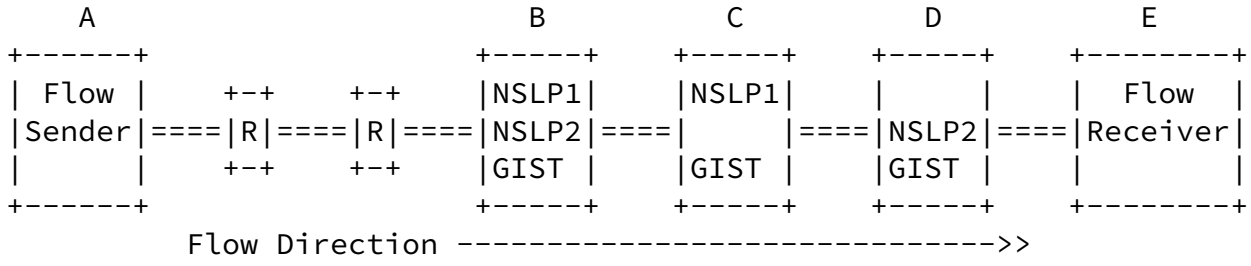
MRI: The flow for which routing state should be invalidated; includes the direction of the change (in the D flag).

Status: The new status that should be assumed for the routing state, one of Bad or Tentative (see Section 7.1.3).

Urgent: A hint as to whether rediscovery should take place immediately, or only with the next signaling message.

Appendix C. Example Routing State Table and Handshake Message Sequence

Figure 10 shows a signaling scenario for a single flow being managed by two signaling applications using the path-coupled message routing method. The flow sender and receiver and one router support both, two other routers support one each. The figure also shows the routing state table at node B.



Message Routing Information	Session ID	NSLP ID	Routing State
MRM = Path Coupled; Flow ID = {IP-A, IP-E, proto/ports}; D=up	0xABCD	NSLP1	IP-A
MRM = Path Coupled; Flow ID = {IP-A, IP-E, proto/ports}; D=down	0xABCD	NSLP1	(null)
MRM = Path Coupled; Flow ID = {IP-A, IP-E, proto/ports}; D=up	0x1234	NSLP2	IP-A
MRM = Path Coupled; Flow ID = {IP-A, IP-E, proto/ports}; D=down	0x1234	NSLP2	Points to B-D MA

Figure 10: A Signaling Scenario

The upstream state is just the same address for each application. For the downstream direction, NSLP1 only requires datagram mode messages and so no explicit routing state towards C is needed. NSLP2 requires a messaging association for its messages towards node D, and node C does not process NSLP2 at all, so the peer state for NSLP2 is a pointer to a messaging association that runs directly from B to D. Note that E is not visible in the state table (except implicitly in the address in the message routing information); routing state is

stored only for adjacent peers. (In addition to the peer identification, IP hop counts are stored for each peer where the state itself if not null; this is not shown in the table.)

Figure 11 shows the message sequence for a GIST handshake that sets

up the messaging association for B-D signaling. It shows the exchange of Stack Proposals and MA-protocol-options data in each direction. Then the Querying node selects TLS/TCP as the stack configuration to use and sets up the messaging association over which it sends the Confirm.

```
 ----------------------GIST-Query --------------------->
 IP(Src=IP#A; Dst=IP#E; RAO for NSLP2); UDP(Src=6789; Dst=GIST)
 GIST(Header(Type=Query; NSLPID=NSLP2; R=1; S=0)
      MRI(MRM=Path-Coupled; Flow=F; Direction=down)
      SessionID(0x1234)
      NLI(Peer='string1'; IA=IP#B)
      QueryCookie(0x139471239471923526)
      StackProposal(#Proposals=3;1=TLS/TCP; 2=TLS/SCTP; 3=TCP)
      StackConfigurationData(#MPO=2;
        TCP(Applicable: all; Data: null)
        SCTP(Applicable: all; Data: null)))

 <---------------------GIST-Response---------------------
 IP(Src=IP#D; Dst=IP#B); UDP(Src=GIST; Dst=6789)
 GIST(Header(Type=Response; NSLPID=NSLP2; R=1; S=1)
      MRI(MRM=Path-Coupled; Flow=F; Direction=up)
      SessionID(0x1234)
      NLI(Peer='stringr2', IA=IP#D)
      QueryCookie(0x139471239471923526)
      ResponderCookie(0xacdefedcdfaeeeded)
      StackProposal(#Proposals=3; 1=TCP; 2=SCTP; 3=TLS/TCP)
      StackConfigurationData(#MPO=3;
        TCP(Applicable: 3; Data: port=6123)
        TCP(Applicable: 1; Data: port=5438)
        SCTP(Applicable: all; Data: port=3333)))

 -------------------------TCP SYN----------------------->
 <----------------------TCP SYN/ACK----------------------
 -------------------------TCP ACK----------------------->
 TCP connect(IP Src=IP#B; IP Dst=IP#D; Src Port=9166; Dst Port=6123)
```

```
 <-----------------------TLS INIT----------------------->

 ----------------------GIST-Confirm---------------------->
 [Sent within messaging association]
 GIST(Header(Type=Confirm; NSLPID=NSLP2; R=0; S=1)
      MRI(MRM=Path-Coupled; Flow=F; Direction=down)
      SessionID(0x1234)
      NLI(Peer='string1'; IA=IP#B)
      ResponderCookie(0xacdefedcdfaeeeded)
      StackProposal(#Proposals=3; 1=TCP; 2=SCTP; 3=TLS/TCP))
```

Figure 11: GIST Handshake Message Sequence

Appendix D. Change History

D.1. Changes In Version -09

1. Added a new Section 3.5 clarifying the relationship between signaling applications and NSLPIDs; modified terminology in the remainder of the document likewise.

2. Added a new Section 8.6 explaining the rationale behind the downgrade attack prevention mechanism.

3. Re-wrote parts of Section 4.3.2, Section 6.1 and Appendix B.2 to clarify the way that GIST is assumed to interact with signaling applications to exercise policy control over whether or not two nodes become signaling peers during a GIST handshake.

4. Generalised an error message Appendix A.4.4.12 to cover additional MRI validation checks in Section 4.3.4 and Section 5.8.1.2.

5. Allowed an optional Stack-Configuration-Data object in Confirm messages to allow messaging association lifetime to be negotiated even in the case of late state installation at the Responding node (see Section 4.4.1 and Section 4.4.3).

6. Removed the option in Section 4.4.2 of allowing a node to treat messaging associations with the same authenticated end points as equivalent.

7. Include additional guidance in Section 4.4.3 to prevent routing state being erroneously refreshed in the case of rerouting events; also included general guidance notes on timer setting.

8. Clarified that the Stack-Proposal lists protocols in top-to-bottom order (see Section 5.7.1).

9. Enhanced the definition of TLS usage in Section 5.7.3 with details on ciphersuite requirements and authentication methods.

10. Tidied up terminology and discussion of how protocol options data is carried in the SCD; renamed higher-layer-addressing to MA-protocol-options.

D.2. Changes In Version -08

1. Changed the protocol name from GIMPS to GIST (everywhere).

2. Inserted RFC2119 language (MUST etc.) in the appropriate places.

3. Added references to the actions to be taken in various error conditions, including the error messages to be send (throughout).

4. Added legacy NAT traversal to the list of excluded functions in Section 1.1.

5. Included some text at the end of Section 3.3 analysing the case of a GIST node which does not support a particular MRM.

6. Added a flag to mark when messages have been explicitly routed, so they can bypass validation against current routing state (see Section 4.3.1, TBD).

7. Re-wrote the discussion in Section 4.3.4 to cover all cases of nodes not hosting an NSLP (including end systems), in particular the validations that can be performed at intermediate GIST nodes (this replaces the old section 7.2).

8. Clarified the rules about R and S flag setting in the common

header and D flag in the MRI (Section 5).

9. Included discussion of how a node with a choice of interfaces or IP versions should select one to use in the NLI (Section 5.2.2).

10. Modified the description of messaging association protocol selections (Section 5.7 and elsewhere) to clarify that this is essentially capability discovery rather than an open ended protocol negotiation.

11. Modified the description of how higher layer addressing information is carried (Section 5.7.1 and Appendix A.3.5) to allow the data to be tagged against a specific profile if necessary, or omitted if the protocol does not need it.

12. Added a higher layer protocol definition for TLS in Section 5.7.3.

13. Simplified and restructured the state machine presentation in Section 6, in particular using a single list for the events and eliminating the transition tables. Also modified the operation of the Responder machine to handle retransmitted Query messages correctly.

14. Re-wrote the route change handling text in Section 7.1 to clarify the relative responsibilities of GIST and NSLPs and

their interaction through the API. Notifications are now assumed to be a signaling application responsibility, and GIST behaviour is defined in terms of handling changes in a 3-state model of the correctness of the routing state for each direction.

15. Updated the NAT traversal description in Section 7.2, including normative text about how GIST nodes should handle messages containing NAT-Traversal objects.

16. Likewise, clarified that the responsibility for session/flow binding in the case of tunnelling is handled by NSLPs (Section 7.3).

17. Formalised the IANA considerations (Section 9).

18. Extended the routing state example (Appendix C) to include a message sequence for association setup.

19. Re-arranged the sequence of sections, including placing this change history at the end.

D.3. Changes In Version -07

1. The open issues section has finally been removed in favour of the authoritative list of open issues in an online issue tracker at h ttp://nsis.srmr.co.uk/cgi-bin/roundup.cgi/nsis-ntlp-issues/index.

2. Clarified terminology on peering and adjacencies that there may be NSIS nodes between GIMPS peers that do some message processing, but that are not explicitly visible in the peer state tables.

3. Added a description of the loose-end MRM (Section 5.8.2 and Appendix A.3.1.2).

4. Added a description of an upstream Query encapsulation for the path-coupled MRM, Section 5.8.1.3, including rationale for and restrictions on its use.

5. The formal description of the protocol in Section 6 has been significantly updated and extended in terms of detail.

6. Modified the description of the interaction between NSLPs and GIMPS for handling inbound messages for which no routing state exists, to allow the NSLP to indicate whether state setup should proceed and to provide NSLP payloads for the Response or forwarded message (Section 3.6, Section 4.3.2 and Appendix B).

7. Included new text, Section 5.6, on the processing and encapsulation of error messages. Also added formats and an error message catalogue in Appendix A.4, including a modified format for the overall GIMPS-Error message and the GIMPS-Error-Data object.

8. Removed the old section 5.3.3 on NSLPID/RAO setting on the assumption that this will be covered in the extensibility

document.

9. Included a number of other minor corrections and clarifications.

D.4. Changes In Version -06

Version -06 does not introduce any major structural changes to the protocol definition, although it does clarify a number of details and resolve some outstanding open issues. The primary changes are as follows:

1. Added a new high level Section 3.3 which gathers together the various aspects of the message routing method concept.

2. Added a new high level Section 3.4 which explains the concept and significance of the session identifier. Also clarified that the routing state always depends on the session identifier.

3. Added notes about the level of address validation performed by GIMPS in Section 4.1.2 and extensions to the API in Appendix B.

4. Split the old Node-Addressing object into a Network-Layer-Information object and Stack-Configuration-Data object. The former refers to basic information about a node, and the latter carries information about messaging association configuration. Redefined the content of the various handshake messages accordingly in Section 4.4.1 and Section 5.1.

5. Re-wrote Section 4.4.3 to clarify the rules on refresh and purge of routing state and messaging associations. Also, moved the routing state lifetime into the Network-Layer-Information object and added a messaging association lifetime to the Stack-Configuration-Data object (Section 5.2).

6. Added specific message types for errors and MA-Refresh in Section 5.1. The error object is now GIMPS-specific (Appendix A.4.1).

7. Moved the Flow-Identifier information about the message routing method from the general description of the object to the path-

coupled MRM section (Section 5.8.1.1), and made a number of

clarifications to the bit format (Appendix A.3.1.1).

8. Removed text about assumptions on the version numbering of NSLPs, and restricted the scope of the description of TLV object formats and extensibility flags to GIMPS rather than the whole of NSIS (Appendix A).

9. Added a new Section 5.5 explaining the possible relationships between message types and encapsulation formats.

10. Added a new Section 6 in outline form, to capture the formal specification of the protocol operation.

11. Added new security sections on cookie requirements (Section 8.5) and residual threats (Section 8.7).

D.5. Changes In Version -05

Version -05 reformulates the specification, to describe routing state maintenance in terms of exchanging explicitly identified Query/ Response/Confirm messages, leaving the upstream/downstream distinction as a specific detail of how Query messages are encapsulated. This necessitated widespread changes in the specification text, especially Section 4.2.1, Section 4.4, Section 5.1 and Section 5.3 (although the actual message sequences are unchanged). A number of other issues, especially in the area of message encapsulation, have also been closed. The main changes are the following:

1. Added a reference to an individual draft on the Loose End MRM as a concrete example of an alternative message routing method.

2. Added further text (particularly in Section 2) on what GIMPS means by the concept of 'session'.

3. Firmed up the selection of UDP as the encapsulation choice for datagram mode, removing the open issue on this topic.

4. Defined the interaction between GIMPS and signaling applications for communicating about the cryptographic security properties of how a message will be sent or has been received (see Section 4.1.2 and Appendix B).

5. Closed the issue on whether Query messages should use the signaling or flow source address in the IP header; both options are allowed by local policy and a flag in the common header indicates which was used. (See Section 5.8.1.2.)

6. Added the necessary information elements to allow the IP hop count between adjacent GIMPS peers to be measures and reported. (See Section 5.2.2 and Appendix A.3.3.)

7. The old open-issue text on selection of IP router alert option values has been moved into the main specification to capture the technical considerations that should be used in assigning such values (in old section 5.3.3).

8. Resolved the open issue on lost Confirm messages by allowing a choice of timer-based retransmission of the Response, or an error message from the responding node which causes the retransmission of the Confirm (see Section 5.3.3).

9. Closed the open issue on support for message scoping (this is now assumed to be a NSLP function).

10. Moved the authoritative text for most of the remaining open issues to an online issue tracker.

D.6. Changes In Version -04

Version -04 includes mainly clarifications of detail and extensions in particular technical areas, in part to support ongoing implementation work. The main details are as follows:

1. Substantially updated Section 4, in particular clarifying the rules on what messages are sent when and with what payloads during routing and messaging association setup, and also adding some further text on message transfer attributes.

2. The description of messaging association protocol setup including the related object formats has been centralised in a new Section 5.7, removing the old Section 6.6 and also closing old open issues 8.5 and 8.6.

3. Made a number of detailed changes in the message format definitions (Appendix A), as well as incorporating initial rules for encoding message extensibility information. Also included explicit formats for a general purpose Error object, and the objects used to discover supported messaging association protocols. Updated the corresponding open issues section (old section 9.3) with a new item on NSLP versioning.

4. Updated the GIMPS API (Appendix B), including more precision on message transfer attributes, making the NSLP hint about storing

reverse path state a return value rather than a separate primitive, and adding a new primitive to allow signaling

applications to invalidate GIMPS routing state. Also, added a new parameter to SendMessage to allow signaling applications to 'bypass' a message statelessly, preserving the source of an input message.

5. Added an outline for the future content of an IANA considerations section (Section 9). Currently, this is restricted to identifying the registries and allocations required, without defining the allocation policies and other considerations involved.

6. Shortened the background design discussion in Section 3.

7. Made some clarifications in the terminology section relating to how the use of C-mode does and does not mandate the use of transport or security protection.

8. The ABNF for message formats in Section 5.1 has been re-written with a grammar structured around message purpose rather than message direction, and additional explanation added to the information element descriptions in Section 5.2.

9. The description of the datagram mode transport in Section 5.3 has been updated. The encapsulation rules (covering IP addressing and UDP port allocation) have been corrected, and a new subsection on message retransmission and rate limiting has been added, superseding the old open issue on the same subject (section 8.10).

10. A new open issue on IP TTL measurement to detect non-GIMPS capable hops has been added (old section 9.5).

D.7. Changes In Version -03

Version -03 includes a number of minor clarifications and extensions compared to version -02, including more details of the GIMPS API and messaging association setup and the node addressing object. The full list of changes is as follows:

1. Added a new section pinning down more formally the interaction between GIMPS and signaling applications (Section 4.1), in particular the message transfer attributes that signaling applications can use to control GIMPS (Section 4.1.2).

2. Added a new open issue identifying where the interaction between the security properties of GIMPS and the security requirements of signaling applications should be identified (old section 9.10).

3. Added some more text in Section 4.2.1 to clarify that GIMPS has the (sole) responsibility for generating the messages that refresh message routing state.

4. Added more clarifying text and table to GHC and IP TTL handling discussion of Section 4.3.4.

5. Split Section 4.4 into subsections for different scenarios, and added more detail on Node-Addressing object content and use to handle the case where association re-use is possible in Section 4.4.2.

6. Added strawman object formats for Node-Addressing and Stack-Proposal objects in Section 5.1 and Appendix A.

7. Added more detail on the bundling possibilities and appropriate configurations for various transport protocols in Section 5.4.1.

8. Included some more details on NAT traversal in Section 7.2, including a new object to carry the untranslated address-bearing payloads, the NAT-Traversal object.

9. Expanded the open issue discussion in old section 9.3 to include an outline set of extensibility flags.

D.8. Changes In Version -02

Version -02 does not represent any radical change in design or structure from version -01; the emphasis has been on adding details in some specific areas and incorporation of comments, including early review comments. The full list of changes is as follows:

1. Added a new Section 1.1 which summarises restrictions on scope and applicability; some corresponding changes in terminology in Section 2.

2. Closed the open issue on including explicit GIMPS state teardown functionality. On balance, it seems that the difficulty of specifying this correctly (especially taking account of the security issues in all scenarios) is not matched by the saving of state enabled.

3. Removed the option of a special class of message transfer for reliable delivery of a single message. This can be implemented (inefficiently) as a degenerate case of C-mode if required.

4. Extended Appendix A with a general discussion of rules for message and object formats across GIMPS and other NSLPs. Some

remaining open issues are noted in old section 9.3 (since removed).

5. Updated the discussion of RAO/NSLPID relationships to take into account the proposed message formats and rules for allocation of NSLP id, and propose considerations for allocation of RAO values.

6. Modified the description of the information used to route messages (first given in Section 4.2.1 but also throughout the document). Previously this was related directly to the flow identification and described as the Flow-Routing-Information. Now, this has been renamed Message-Routing-Information, and identifies a message routing method and any associated addressing.

7. Modified the text in Section 4.3 and elsewhere to impose sanity checks on the Message-Routing-Information carried in C-mode messages, including the case where these messages are part of a GIMPS-Query/Response exchange.

8. Added rules for message forwarding to prevent message looping in a new Section 4.3.4, including rules on IP TTL and GIMPS hop count processing. These take into account the new RAO considerations described above.

9. Added an outline mechanism for messaging association protocol stack setup, with the details in a new Section 6.6 and other changes in Section 4.4 and the various sections on message formats.

10. Removed the open issue on whether storing reverse routing state is mandatory or optional. This is now explicit in the API (under the control of the local NSLP).

11. Added an informative annex describing an abstract API between GIMPS and NSLPs in Appendix B.

D.9. Changes In Version -01

The major change in version -01 is the elimination of 'intermediaries', i.e. imposing the constraint that signaling application peers are also GIMPS peers. This has the consequence that if a signaling application wishes to use two classes of signaling transport for a given flow, maybe reaching different subsets of nodes, it must do so by running different signaling sessions; and it also means that signaling adaptations for passing through NATs which are not signaling application aware must be

carried out in datagram mode. On the other hand, it allows the elimination of significant complexity in the connection mode handling and also various other protocol features (such as general route recording).

The full set of changes is as follows:

1. Added a worked example in Section 3.6.

2. Stated that nodes which do not implement the signaling application should bypass the message (Section 4.3).

3. Decoupled the state handling logic for routing state and messaging association state in Section 4.4. Also, allow messaging associations to be used immediately in both directions once they are opened.

4. Added simple ABNF for the various GIMPS message types in a new

Section 5.1, and more details of the common header and each object in Section 5.2, including bit formats in Appendix A. The common header format means that the encapsulation is now the same for all transport types (Section 5.4.1).

5. Added some further details on datagram mode encapsulation in Section 5.3, including more explanation of why a well known port is needed.

6. Removed the possibility for fragmentation over DCCP (Section 5.4.1), mainly in the interests of simplicity and loss amplification.

7. Removed all the tunnel mode encapsulations (old sections 5.3.3 and 5.3.4).

8. Fully re-wrote the route change handling description (Section 7.1), including some additional detection mechanisms and more clearly distinguishing between upstream and downstream route changes. Included further details on GIMPS/NSLP interactions, including where notifications are delivered and how local repair storms could be avoided. Removed old discussion of propagating notifications through signaling application unaware nodes (since these are now bypassed automatically). Added discussion on how to route messages for local state removal on the old path.

9. Revised discussion of policy-based forwarding (old Section 7.2) to account for actual Flow-Routing-Information definition, and also how wildcarding should be allowed and handled.

10. Removed old route recording section (old Section 6.3).

11. Extended the discussion of NAT handling (Section 7.2) with an extended outline on processing rules at a GIMPS-aware NAT and a pointer to implications for C-mode processing and state management.

12. Clarified the definition of 'correct routing' of signaling messages in Section 8 and GIMPS role in enforcing this. Also, opened the possibility that peer node authentication could be signaling application dependent.

13. Removed old open issues on Connection Mode Encapsulation (section 8.7); added new open issues on Message Routing (old Section 9.3 of version -05, later moved to Section 3.3) and Datagram Mode congestion control.

14. Added this change history.

Authors' Addresses

Henning Schulzrinne
Columbia University

Department of Computer Science
450 Computer Science Building
New York, NY 10027
US

Phone: +1 212 939 7042
Email: hgs+nsis@cs.columbia.edu
URI: http://www.cs.columbia.edu

Robert Hancock
Siemens/Roke Manor Research
Old Salisbury Lane
Romsey, Hampshire SO51 0ZN
UK

Email: robert.hancock@roke.co.uk
URI: http://www.roke.co.uk

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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