Internet Engineering Task Force (IETF)
Request for Comments: 5971
Category: Experimental
ISSN: 2070-1721

H. Schulzrinne
Columbia U.
R. Hancock
RMR

October 2010

GIST: General Internet Signalling Transport

Abstract

This document specifies protocol stacks for the routing and transport of per-flow signalling 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 Signalling Transport (GIST), which provides a common service for diverse signalling applications. GIST does not handle signalling 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 Signalling" (NSIS) framework.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5971.

Table of Contents

1. Introduction

Signalling 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 that is managed by other mechanisms. This specification concentrates mainly on path-coupled signalling, controlling resources on network elements that are located on the path taken by a particular data flow, possibly including but not limited to the flow endpoints. Examples of state management include network 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 signalling is referred to as a signalling application.

GIST assumes other mechanisms are responsible for controlling routing within the network, and GIST is not designed to set up or modify paths itself; therefore, it is complementary to protocols like Resource Reservation Protocol - Traffic Engineering (RSVP-TE) [22] or LDP [23] rather than an alternative. There are almost always more than two participants in a path-coupled signalling session, although there is no need for every node on the path to participate; indeed, support for GIST and any signalling applications imposes a performance cost, and deployment for flow-level signalling is much more likely on edge devices than core routers. GIST path-coupled signalling does not directly support multicast flows, but the current GIST design could be extended to do so, especially in environments where the multicast replication points can be made GIST-capable. GIST can also be extended to cover other types of signalling pattern, not related to any end-to-end flow in the network, in which case the distinction between GIST and end-to-end higher-layer signalling will be drawn differently or not at all.

Every signalling 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 signalling applications, and hence can be developed as a common protocol. The NSIS framework given in [29] 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). Several concepts in the framework are derived from RSVP [14], as are several aspects of the GIST protocol design. The application-specific protocols are referred to as NSIS Signalling Layer Protocols (NSLPs), and are defined in separate documents. The NSIS framework [29] and the accompanying threats document [30] provide important background

information to this specification, including information on how GIST is expected to be used in various network types and what role it is expected to perform.

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 Signalling Transport (GIST). GIST does not handle signalling application state itself; in that crucial respect, it differs from higher layer signalling protocols such as SIP, the Real-time Streaming Protocol (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 signalling messages on behalf of signalling applications in both directions along the flow path. The purpose of GIST is thus to provide the common functionality of node discovery, message routing, and message transport in a way that is simple for multiple signalling applications to re-use.

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 machines in Section 6. Section 7 describes some more advanced protocol features, and security considerations are contained in Section 8. In addition, Appendix B describes an abstract API for the service that GIST provides to signalling applications, and Appendix D provides an example message flow. Parts of the GIST design use packets with IP options to probe the network, that leads to some migration issues in the case of IPv4, and these are discussed in Appendix C.

Because of the layered structure of the NSIS protocol suite, protocol extensions to cover a new signalling requirement could be carried out either within GIST, or within the signalling application layer, or both. General guidelines on how to extend different layers of the protocol suite, and in particular when and how it is appropriate to extend GIST, are contained in a separate document [12]. In this document, Section 9 gives the formal IANA considerations for the registries defined by the GIST specification.

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 RFC 2119 [3].

The terminology used in this specification is defined in this section. The basic entities relevant at the GIST level are shown in Figure 1. In particular, this diagram distinguishes the different address types as being associated with a flow (end-to-end addresses) or signalling (addresses of adjacent signalling peers).

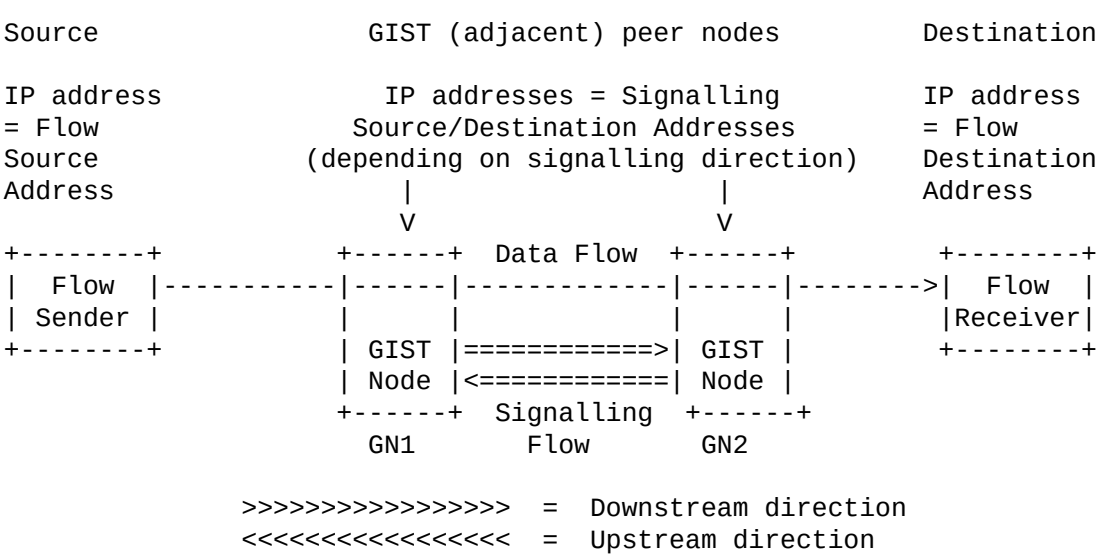


Figure 1: Basic Terminology

[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 exchange of information for which some state information is to be manipulated or monitored. See Section 3.7 for further detailed discussion.

Session Identifier (SID): An identifier for a session; the syntax is a 128-bit value that is opaque to GIST.

[Flow] Sender: The node in the network that is the source of the packets in a flow. A sender could be a host, or a router if, for example, the flow is actually an aggregate.

[Flow] Receiver: The node in the network that 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 supporting the GIST protocol, regardless of what signalling applications it supports.

[Adjacent] Peer: The next node along the signalling path, in the upstream or downstream direction, with which a GIST node explicitly interacts.

Querying Node: The GIST node that initiates the handshake process to discover the adjacent peer.

Responding Node: The GIST node that responds to the handshake, becoming the adjacent peer to the Querying node.

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

Connection Mode (C-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 (MA): A single connection between two explicitly identified GIST adjacent peers, i.e., between a given signalling source and destination address. A messaging association may use a transport protocol; if security protection is required, it may use a network layer security association, or use a transport layer security association internally. A messaging association is bidirectional: signalling messages can be sent over it in either direction, referring to flows of either direction.

[Message] Routing: Message routing describes the process of determining which is the next GIST peer along the signalling path. For signalling along a flow path, the message routing carried out by GIST is built on top of normal IP routing, that is, forwarding packets within the network layer based on their destination IP address. In this document, the term 'routing' generally refers to GIST message routing unless particularly specified.

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

Message Routing Information (MRI): The set of data item values that is used to route a signalling message according to a particular MRM; for example, for routing along a flow path, the MRI includes flow source and destination addresses, and protocol and port numbers. See Section 3.3.

Router Alert Option (RAO): An option that can be included in IPv4 and v6 headers to assist in the packet interception process; see [13] and [17].

Transfer Attributes: A description of the requirements that a signalling 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 [29] for transport of signalling messages are essentially two-fold:

Routing: Determine how to reach the adjacent signalling 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 signalling 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 three-way handshake that probes the network to set up the necessary routing state between adjacent peers, during which signalling applications can also exchange data. 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 functionality into two parts, a minimal capability provided by GIST itself, with the use of well-understood transport protocols for the harder cases. Here, with details discussed later, the minimal capability is restricted to messages that are sized well below the lowest maximum transmission unit (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 [29], 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, with a specialised GIST messaging

layer running over standard transport and security protocols. The basic concept is shown in Figure 2. Note that not every combination of transport and security protocols implied by the figure is actually possible for use in GIST; the actual combinations allowed by this specification are defined in Section 5.7. The figure also shows GIST offering its services to upper layers at an abstract interface, the GIST API, further discussed in Section 4.1.

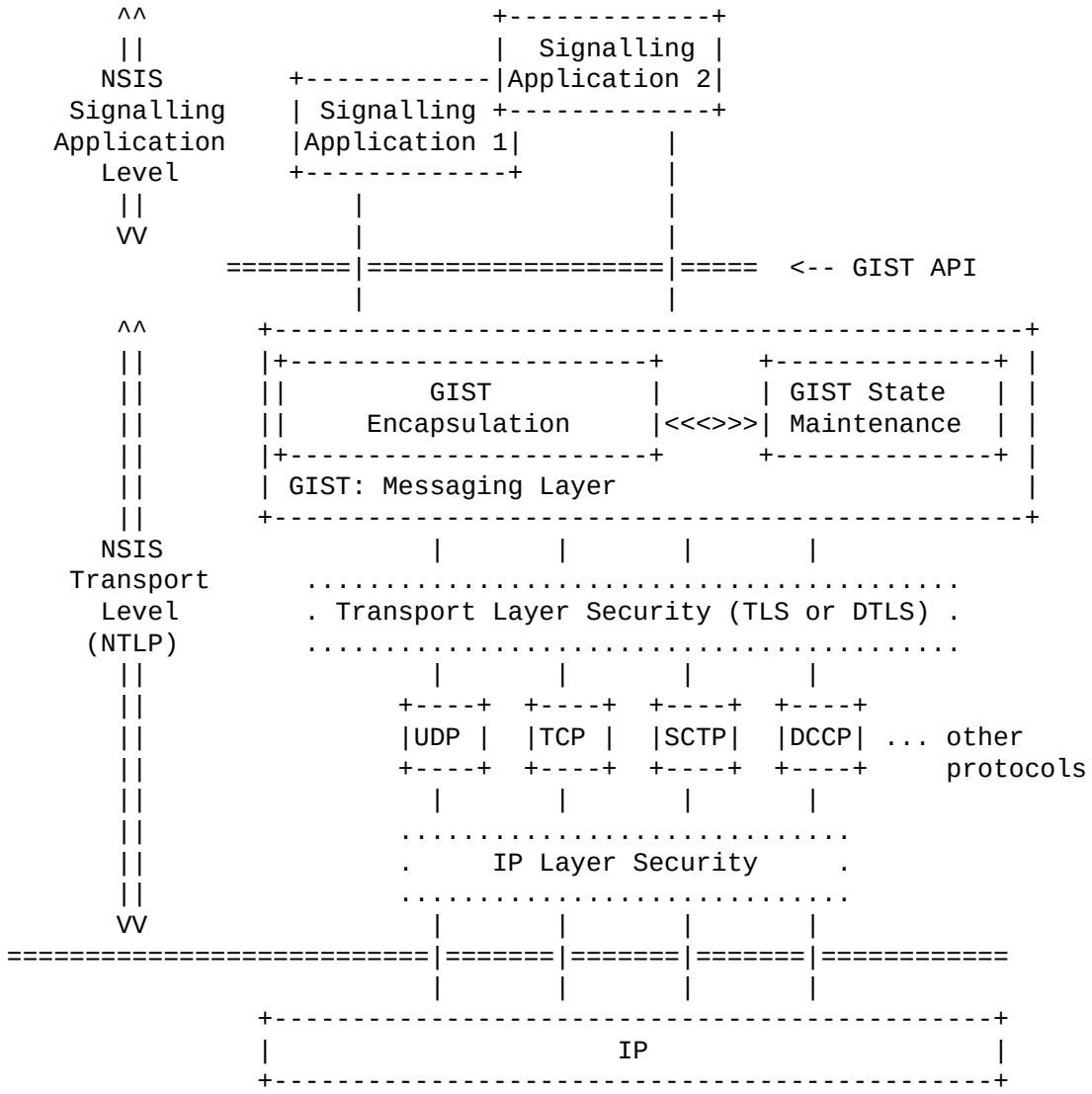


Figure 2: Protocol Stack Architecture for Signalling Transport

3.2. Modes and Messaging Associations

Internally, GIST has two modes of operation:

Datagram mode (D-mode): used for small, infrequent messages with modest delay constraints and no security requirements. A special case of D-mode called Query-mode (Q-mode) is used when no routing state exists.

Connection mode (C-mode): used for all other signalling traffic. In particular, it can support large messages and channel security and provides congestion control for signalling traffic.

C-mode can in principle use any stream or message-oriented transport protocol; this specification defines TCP as the initial choice. It can in principle 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 C-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.

D-mode uses UDP, as a suitable NAT-friendly encapsulation that does not require per-message shared state to be maintained between the peers. Long-term evolution of GIST is assumed to preserve the simplicity of the current D-mode design. Any extension to the security or transport capabilities of D-mode can be viewed as the selection of a different protocol stack under the GIST messaging layer; this is then equivalent to defining another option within the overall C-mode framework. This includes both the case of using existing protocols and the specific development of a message exchange and payload encapsulation to support GIST requirements. Alternatively, if any necessary parameters (e.g., a shared secret for use in integrity or confidentiality protection) can be negotiated out-of-band, then the additional functions can be added directly to D-mode by adding an optional object to the message (see Appendix A.2.1). Note that in such an approach, downgrade attacks as discussed in Section 8.6 would need to be prevented by policy at the destination node.

It is possible to mix these two modes along a path. This allows, for example, the use of D-mode at the edges of the network and C-mode towards the core. Such combinations may make operation more efficient for mobile endpoints, while allowing shared security associations and transport connections to be used for messages for multiple flows and signalling applications. The setup for these

protocols imposes an initialisation cost for the use of C-mode, but in the long term this cost can be shared over all signalling sessions between peers; once the transport layer state exists, retransmission algorithms can operate much more aggressively than would be possible in a pure D-mode design.

It must be understood that the routing and transport functions within GIST are not independent. If the message transfer has requirements that require C-mode, for example, if 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 three-way handshake initially in D-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 signalling application does not make the D-mode/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 C-mode messaging to a particular peer (signalling destination address, protocol and port numbers, internal protocol configuration, and state information) is referred to as a messaging association (MA). MAs are totally internal to GIST (they are not visible to signalling applications). Although GIST may be using an MA to deliver messages about a particular flow, there is no direct correspondence between them: the GIST message routing algorithms consider each message in turn and select an appropriate MA to transport it. There may be any number of MAs between two GIST peers although the usual case is zero or one, 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 signalling 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 signalling 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 the routing behaviour being used. Therefore, the GIST design encapsulates the routing-dependent details as a message routing method (MRM), and allows multiple MRMs to be defined. This specification defines the path-coupled MRM, corresponding to the baseline functionality described above, and a second ("Loose-End") MRM for the NAT Address Reservation case. The detailed specifications are given in Section 5.8.

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

- The format of the information that describes the path that the signalling 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 signalling messages can take, denoted ’upstream’ and ’downstream’.

- A specification of the IP-level encapsulation of the 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. Current MRMs rely on the interception of probe messages in the data plane, but other mechanisms are also possible within the overall GIST design and would be appropriate for other types of signalling pattern.

- 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 or 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 may require translation of fields in the MRI object carried in GIST messages (see Section 7.2.2). The generic MRI format includes a flag that must be given as part of the MRM definition, to indicate if some kind of translation is necessary. Development of a new MRM therefore includes updates to the GIST specification, and may include updates to specifications of NAT behaviour. These updates may be done in separate documents as is the case for NAT traversal for the MRMs of the base GIST specification, as described in Section 7.2.3 and [44].

The MRI is passed explicitly between signalling applications and GIST; therefore, signalling application specifications must define which MRMs they require. Signalling 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 signalling application needs to use a GIST implementation that supports the corresponding MRMs. The GIST processing rules allow nodes not hosting the signalling application to ignore messages for it at the GIST level, so it does not matter if these nodes support the MRM or not.

3.4. GIST Messages

GIST has six message types: Query, Response, Confirm, Data, Error, and MA-Hello. Apart from the invocation of the messaging association protocols used by C-mode, all GIST communication consists of these messages. In addition, all signalling application data is carried as additional payloads in these messages, alongside the GIST information.

The Query, Response, and Confirm messages implement the handshake that GIST uses to set up routing state and messaging associations. The handshake is initiated from the Querying node towards the Responding node. The first message is the Query, which is encapsulated in a specific way depending on the message routing method, in order to probe the network infrastructure so that the correct peer will intercept it and become the Responding node. A Query always triggers a Response in the reverse direction as the second message of the handshake. The content of the Response controls whether a Confirm message is sent: as part of the defence against denial-of-service attacks, the Responding node can delay state installation until a return routability check has been performed, and require the Querying node to complete the handshake with the Confirm message. In addition, if the handshake is being used to set up a new MA, the Response is required to request a Confirm. All of these three messages can optionally carry signalling application data. The handshake is fully described in Section 4.4.1.

The Data message is used purely to encapsulate and deliver signalling application data. Usually, it is sent using pre-established routing state. However, if there are no security or transport requirements and no need for persistent reverse routing state, it can also be sent in the same way as the Query. Finally, Error messages are used to indicate error conditions at the GIST level, and the MA-Hello message can be used as a diagnostic and keepalive for the messaging association protocols.

3.5. GIST Peering Relationships

Peering is the process whereby two GIST nodes create message routing states that point to each other.

A peering relationship can only be created by a GIST handshake. Nodes become peers when one issues a Query and gets a Response from another. Issuing the initial Query is a result of an NSLP request on that node, and the Query itself is formatted according to the rules of the message routing method. For current MRMs, the identity of the Responding node is not known explicitly at the time the Query is sent; instead, the message is examined by nodes along the path until one decides to send a Response, thereby becoming the peer. If the node hosts the NSLP, local GIST and signalling application policy determine whether to peer; the details are given in Section 4.3.2. Nodes not hosting the NSLP forward the Query transparently (Section 4.3.4). Note that the design of the Query message (see Section 5.3.2) is such that nodes have to opt-in specifically to carry out the message interception -- GIST-unaware nodes see the Query as a normal data packet and so forward it transparently.

An existing peering relationship can only be changed by a new GIST handshake; in other words, it can only change when routing state is refreshed. On a refresh, if any of the factors in the original peering process have changed, the peering relationship can also change. As well as network-level rerouting, changes could include modifications to NSIS signalling functions deployed at a node, or alterations to signalling application policy. A change could cause an existing node to drop out of the signalling path, or a new node to become part of it. All these possibilities are handled as rerouting events by GIST; further details of the process are described in Section 7.1.

3.6. Effect on Internet Transparency

GIST relies on routers inside the network to intercept and process packets that would normally be transmitted end-to-end. This processing may be non-transparent: messages may be forwarded with modifications, or not forwarded at all. This interception applies

only to the encapsulation used for the Query messages that probe the network, for example, along a flow path; all other GIST messages are handled only by the nodes to which they are directly addressed, i.e., as normal Internet traffic.

Because this interception potentially breaks Internet transparency for packets that have nothing to do with GIST, the encapsulation used by GIST in this case (called Query-mode or Q-mode) has several features to avoid accidental collisions with other traffic:

- Q-mode messages are always sent as UDP traffic, and to a specific well-known port (270) allocated by IANA.

- All GIST messages sent as UDP have a magic number as the first 32-bit word of the datagram payload.

Even if a node intercepts a packet as potentially a GIST message, unless it passes both these checks it will be ignored at the GIST level and forwarded transparently. Further discussion of the reception process is in Section 4.3.1 and the encapsulation in Section 5.3.

3.7. Signalling Sessions

GIST requires signalling applications to associate each of their messages with a signalling session. Informally, given an application layer exchange of information for which some network control state information is to be manipulated or monitored, the corresponding signalling messages should be associated with the same session. Signalling applications provide the session identifier (SID) whenever they wish to send a message, and GIST reports the SID when a message is received; on messages forwarded at the GIST level, the SID is preserved unchanged. Usually, NSLPs will preserve the SID value along the entire signalling path, but this is not enforced by or even visible to GIST, which only sees the scope of the SID as the single hop between adjacent NSLP peers.

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

- The simplest case is that all signalling 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 signalling applications.

Because of this range of options, GIST does not perform any validation on how signalling applications map between flows and sessions, nor does it perform any direct validation on the properties of the SID itself, such as any enforcement of uniqueness. 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 that are 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 MRI/SID/NSLPID combination.

Strictly speaking, the routing state should not depend on the SID. However, if the routing state is keyed only by (MRI, NSLP), 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. Such an attack would not redirect the traffic but would reroute the signalling. Instead, the routing state is also segregated between different SIDs, which means that the attacking node can only disrupt a signalling session if it can guess the corresponding SID. Normative rules on the selection of SIDs are given in Section 4.1.3.

3.8. Signalling Applications and NSLPIDs

The functionality for signalling applications is supported by NSIS Signalling Layer Protocols (NSLPs). Each NSLP is identified by a 16-bit NSLP identifier (NSLPID), assigned by IANA (Section 9). A single signalling application, such as resource reservation, may define a family of NSLPs to implement its functionality, for example, to carry out signalling operations at different levels in a hierarchy (cf. [21]). 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 signalling application level; the NSLPID is the only information visible to GIST about the signalling application being used.

3.9. GIST Security Services

GIST has two distinct security goals:

- to protect GIST state from corruption, and to protect the nodes on which it runs from resource exhaustion attacks; and
- to provide secure transport for NSLP messages to the signalling applications.

The protocol mechanisms to achieve the first goal are mainly internal to GIST. They include a cookie exchange and return routability check to protect the handshake that sets up routing state, and a random SID is also used to prevent off-path session hijacking by SID guessing. Further details are given in Section 4.1.3 and Section 4.4.1, and the overall security aspects are discussed in Section 8.

A second level of protection is provided by the use of a channel security protocol in messaging associations (i.e., within C-mode). This mechanism serves two purposes: to protect against on-path attacks on GIST and to provide a secure channel for NSLP messages. For the mechanism to be effective, it must be able to provide the following functions:

- mutual authentication of the GIST peer nodes;
- ability to verify the authenticated identity against a database of nodes authorised to take part in GIST signalling;
- confidentiality and integrity protection for NSLP data, and provision of the authenticated identities used to the signalling application.

The authorised peer database is described in more detail in Section 4.4.2, including the types of entries that it can contain and the authorisation checking algorithm that is used. The only channel security protocol defined by this specification is a basic use of TLS, and Section 5.7.3 defines the TLS-specific aspects of how these functions (for example, authentication and identity comparison) are integrated with the rest of GIST operation. At a high level, there are several alternative protocols with similar functionality, and the handshake (Section 4.4.1) provides a mechanism within GIST to select between them. However, they differ in their identity schemes and authentication methods and dependencies on infrastructure support for the authentication process, and any GIST extension to incorporate them would need to define the details of the corresponding interactions with GIST operation.

3.10. Example of Operation

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

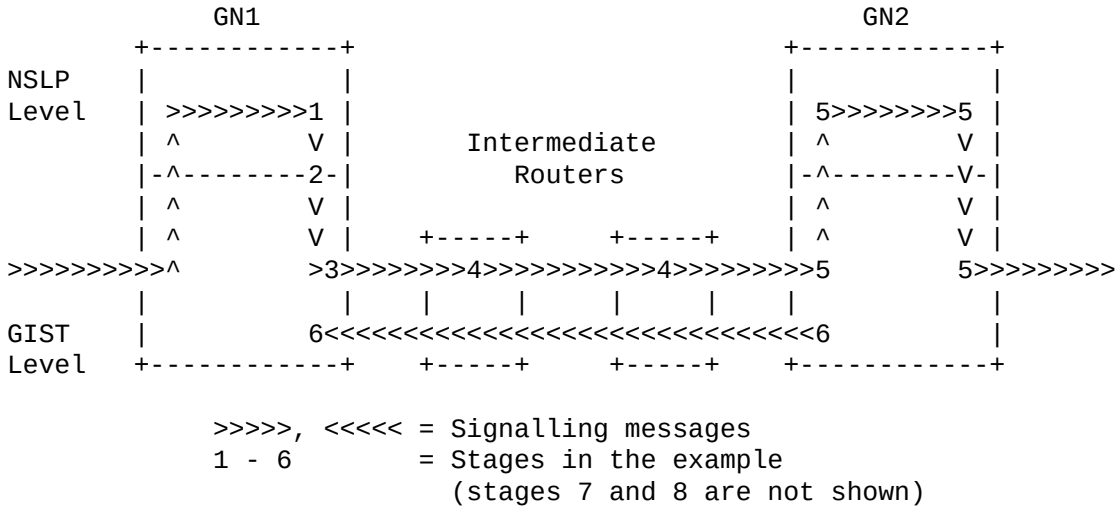


Figure 3: Example of Operation

Consider the case of an RSVP-like signalling application that makes receiver-based resource reservations for a single unicast flow. In general, signalling can take place along the entire end-to-end path (between flow source and destination), but the role of GIST is only to transfer signalling messages over a single segment of the path, between neighbouring resource-capable nodes. Basic GIST operation is the same, whether it involves the endpoints or only interior nodes: in either case, GIST is triggered by a request from a local signalling application. The example here describes how GIST transfers messages between two adjacent peers some distance along the path, GN1 and GN2 (see Figure 3). We take up the story at the point where a message is being processed above the GIST layer by the signalling application in GN1.

1. The signalling 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 signalling 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 (in this case, requesting no reliable transmission or channel security protection). GIST determines that this particular message does not require fragmentation and that it has no knowledge of the next peer for this flow and signalling 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-internal policy interactions between GIST and the signalling application.

3. GN1 therefore constructs a GIST Query carrying the NSLP payload, and additional payloads at the GIST level which will be used to initiate a messaging association. The Query is encapsulated in a UDP datagram and injected into the network. At the IP level, the destination address is the flow receiver, and an IP Router Alert Option (RAO) is also included.

4. The Query 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 that 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 after initial interception.

5. The message is intercepted at GN2. The GIST layer identifies the message as relevant to a local signalling application, and passes the NSLP payload and flow description upwards to it. This signalling 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 signalling application continues to process the message as in GN1 (compare step 1), passing the message back down to GIST so that it is sent further downstream, and this will eventually result in the message reaching the flow receiver. GIST itself operates hop-by-hop, and the signalling application joins these hops together to manage the end-to-end signalling operations.

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

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

6.B - No Association: GN2 sends the Response in D-mode directly to GN1, identifying itself and agreeing to the messaging association setup. The protocol exchanges needed to complete this will proceed in parallel with the following stages.

In each case, the result is that GN1 and GN2 are now in a peering relationship for the flow.

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 signalling application in GN2 passes this payload to the GIST level, along with the flow definition and transfer attributes; in this case, it could request reliable transmission and use of a secure channel for integrity protection. (Other combinations of attributes are possible.)

8. The GIST layer in GN2 identifies the upstream peer for this flow and NSLP as GN1, and determines that it has an MA with the appropriate properties. The message is queued on the MA for transmission; this may incur 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 signalling 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 signalling 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 D-mode operation, where no messaging associations are necessary. Finally, Section 4.4 describes how routing state and messaging associations are created and managed.

4.1. GIST Service Interface

This section describes the interaction between GIST and signalling applications in terms of an abstract service interface, including a definition of the attributes of the message transfer that GIST can offer. The service interface presented here is non-normative and does not constrain actual implementations of any interface between GIST and signalling applications; the interface is provided to aid understanding of how GIST can be used. However, requirements on SID selection and internal GIST behaviour to support message transfer semantics (such as in-order delivery) are stated normatively here.

The same service interface is presented at every GIST node; however, applications may invoke it differently at different nodes, depending for example on local policy. In addition, the service interface is defined independently of any specific transport protocol, or even the distinction between D-mode and C-mode. The initial version of this specification defines how to support the service interface using a C-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 signalling 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 (the MRI), and the application also provides the session identifier (SID); see Section 4.1.3. Additional message transfer attributes control the specific transport and security properties that the signalling application desires.

The distinction between GIST D- and C-mode is not visible at the service interface. In addition, the functionality to handle fragmentation and reassembly, bundling together of small messages for efficiency, and congestion control are not visible at the service interface; GIST will take whatever action is necessary based on the properties of the messages and local node state.

A signalling application is free to choose the rate at which it processes inbound messages; an implementation MAY allow the application to block accepting messages from GIST. In these circumstances, GIST MAY discard unreliably delivered messages, but for reliable messages MUST propagate flow-control condition back to the sender. Therefore, applications must be aware that they may in turn be blocked from sending outbound messages themselves.

4.1.2. Message Transfer Attributes

Message transfer attributes are used by NSLPs to define minimum required levels of message processing. The attributes available are as follows:

Reliability: This attribute may be 'true' or 'false'. When 'true', the following rules apply:

* messages MUST be delivered to the signalling application in the peer exactly once or not at all;

* for messages with the same SID, the delivery MUST be in order;

* if there is a chance that the message was not delivered (e.g., in the case of a transport layer error), an error MUST be indicated to the local signalling application identifying the routing information for the message in question.

GIST implements reliability by using an appropriate transport protocol within a messaging association, so mechanisms for the detection of message loss depend on the protocol in question; for the current specification, the case of TCP is considered in Section 5.7.2. When 'false', a message may be delivered, once, several times, or not at all, with no error indications in any of these cases.

Security: This attribute defines the set of security properties that the signalling application requires for the message, including the type of protection required, and what authenticated identities should be used for the signalling source and destination. This information maps onto the corresponding properties of the security associations established between the peers in C-mode. Keying material for the security associations is established by the authentication mechanisms within the messaging association protocols themselves; see Section 8.2. The attribute can be specified explicitly by the signalling application, or reported by GIST to the signalling application. The latter can take place

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, such as 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 signalling peer or intermediate nodes. An NSLP may also indicate that upstream path routing state will not be needed for this flow, to inhibit the node requesting its downstream peer to create it; conversely, even if routing state exists, the NSLP may request that it is not used, which will lead to GIST Data messages being sent Q-mode encapsulated instead.

A GIST implementation MAY deliver messages with stronger attribute values than those explicitly requested by the application.

4.1.3. SID Selection

The fact that SIDs index routing state (see Section 4.2.1 below) means that there are requirements for how they are selected. Specifically, signalling applications MUST choose SIDs so that they are cryptographically random, and SHOULD NOT use several SIDs for the same flow, to avoid additional load from routing state maintenance. Guidance on secure randomness generation can be found in [31].

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. Each row in the table corresponds to a unique combination of the following three items:

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 Identifier (SID): The signalling session with which this message should be associated; see Section 3.7.

NSLP Identifier (NSLPID): This is an IANA-assigned identifier associated with the NSLP that is generating messages for this flow; see Section 3.8. The inclusion of this identifier allows the routing state to be different for different NSLPs.

The information associated with a given MRI/SID/NSLPID combination 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 in the table, one each for the upstream and downstream MRI. The routing state includes information about the peer identity (see Section 4.4.3), and a UDP port number for D-mode, or a reference to one or more MAs for C-mode. Entries in the routing state table are created by the GIST handshake, which is described in more detail in Section 4.4.

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

- The signalling application has indicated that no messages will actually be sent in that direction.

- The node is the endpoint of the signalling path, for example, because it is acting as a proxy, or because it has determined that there are no further signalling nodes in that direction.

- The node is using other techniques to route the message. For example, it can send it in Q-mode and rely on the peer to intercept it.

In particular, if the node is a flow endpoint, GIST will refuse to create routing state for the direction beyond the end of the flow (see Section 4.3.3). Each entry in the routing state table has an associated validity timer indicating for how long it can be considered accurate. When this timer expires, the entry MUST be purged if it has not been refreshed. Installation and maintenance of routing state are described in more detail in Section 4.4.

4.2.2. Peer-Peer 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 MAs. Since these are not per-flow, they are stored separately from the routing state, including the following per-MA information:

- a queue of any messages that require the use of an MA, pending transmission while the MA is being established;
- the time since the peer re-stated its desire to keep the MA open (see Section 4.4.5).

In addition, per-MA state, such as TCP port numbers or timer information, 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 GIST Message Processing

This section describes how signalling 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. First, message reception, local processing, and message transmission are described for the case where the node hosts the NSLPID identified in the message. Second, in Section 4.3.4, the case where the message is handled directly in the IP or GIST layer (because there is no matching signalling application on the node) is given. An overview is given in Figure 4. This section concentrates on the GIST-level processing, with full details of IP and transport layer encapsulation in Section 5.3 and Section 5.4.

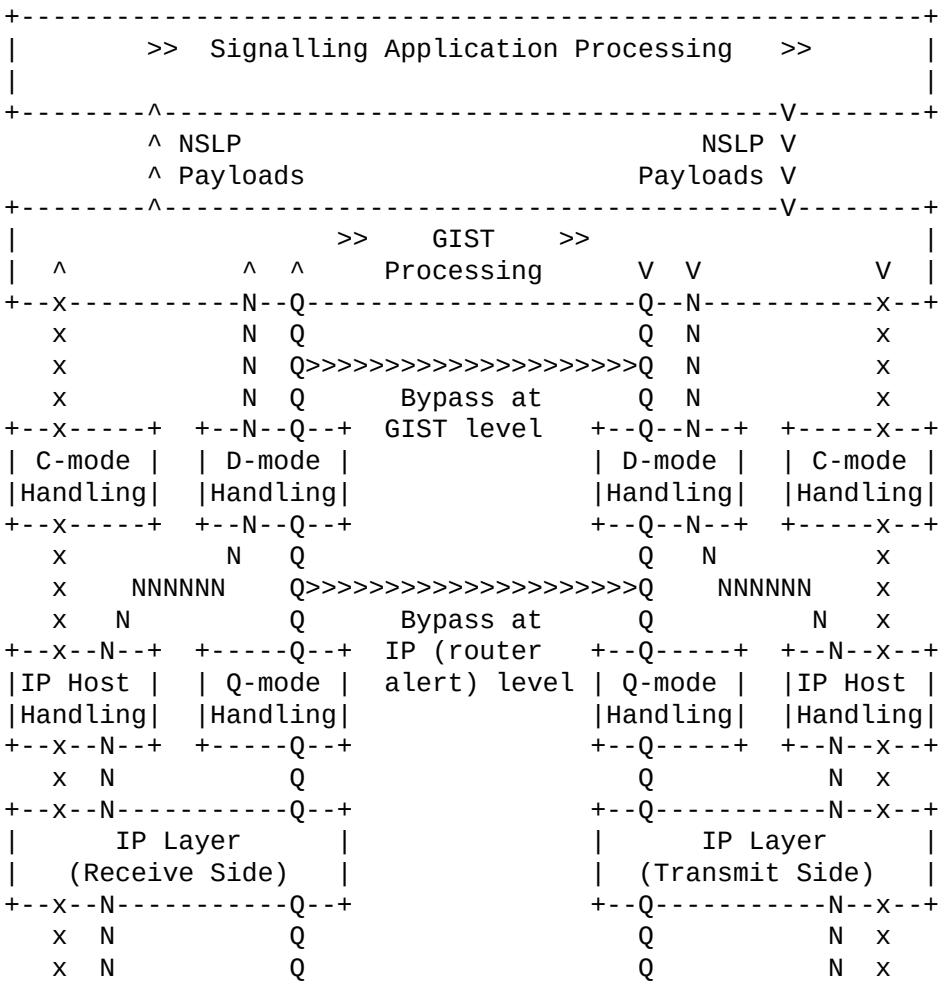


Figure 4: Message Paths through a GIST Node

4.3.1. Message Reception

Messages can be received in C-mode or D-mode.

Reception in C-mode is simple: incoming packets undergo the security and transport treatment associated with the MA, and the MA provides complete messages to the GIST layer for further processing.

Reception in D-mode depends on the message type.

Normal encapsulation: Normal messages arrive UDP-encapsulated and addressed directly to the receiving signalling 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 C-mode case.

Q-mode encapsulation: 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, and MAY include an IP Router Alert Option (RAO) if required by the MRM. Each GIST node can therefore see every such message, but unless the message exactly matches the Q-mode encapsulation rules (Section 5.3.2) it MUST be forwarded transparently at the IP level. If it does match, GIST MUST check the NSLPID in the common header. The case where the NSLPID does not match a local signalling application at all is considered below in Section 4.3.4; otherwise, the message MUST be passed up to the GIST layer for further processing.

Several different RAO values may be used by the NSIS protocol suite. GIST itself does not allocate any RAO values (for either IPv4 or IPv6); an assignment is made for each NSLP using MRMs that use the RAO in the Q-mode encapsulation. The assignment rationale is discussed in a separate document [12]. The RAO value assigned for an NSLPID may be different for IPv4 and IPv6. Note the different significance between the RAO and the NSLPID values: the meaning of a message (which signalling application it refers to, whether it should be processed at a node) is determined only from the NSLPID; the role of the RAO value is simply to allow nodes to pre-filter which IP datagrams are analysed to see if they might be Q-mode GIST messages.

For all assignments associated with NSIS, the RAO-specific processing is the same and is as defined by this specification, here and in Section 4.3.4 and Section 5.3.2.

Immediately after reception, the GIST hop count is checked. Any message with a GIST hop count of zero MUST be rejected with a "Hop Limit Exceeded" error message (Appendix A.4.4.2); note that a correct GIST implementation will never send a message with a GIST hop count of zero. Otherwise, the GIST hop count MUST be decremented by one before the next stage.

4.3.2. Local Processing and Validation

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

This section concentrates on the interaction with the signalling application, in particular, the decision to peer and how data is delivered to the NSLP.

In the case of a Query, there is an interaction with the signalling application to determine which of two courses to follow. The first option (peering) MUST be chosen if the node is the final destination of the Query message.

1. The receiving signalling application wishes to become a signalling 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 Response.

2. The signalling application does not wish to set up state with the Querying node and become its peer. This includes the case where a node wishes to avoid taking part in the signalling for overload protection reasons. 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 forwarded by GIST. Note that if the node that finally processes the Query returns an Error message, this will be sent directly back to the originating node, bypassing any forwarders. For these diagnostics to be meaningful, any GIST node forwarding a Query, or relaying it with modified NSLP payload, MUST NOT modify it except in the GIST hop count; in particular, it MUST NOT modify any other GIST payloads or their order. An implementation MAY choose to achieve this by retaining the original message, rather than reconstructing it from some parsed internal representation.

This interaction with the signalling application, including the generation or update of an NSLP payload, SHOULD take place synchronously as part of the Query 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 GIST message types other than Queries, if the message includes an NSLP payload, this MUST be delivered locally to the signalling 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, which MUST be applied in the sequence given:

1. if the message was explicitly routed (see Section 7.1.5) 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;

2. else, if the message arrived on an association that 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);

3. else, if there is no routing state for this MRI/SID/NSLPID combination, the message MUST either be dropped or be rejected with an error message (see Section 4.4.6 for further details);

4. else, the payload is delivered as normal.

4.3.3. Message Transmission

Signalling applications can generate their messages for transmission, either asynchronously or in reply to an input message delivered by GIST, and GIST can also generate messages autonomously. GIST MUST verify that it is not the direct destination of an outgoing message, and MUST reject such messages with an error indication to the signalling application. When the message is generated by a signalling application, it may be carried in a Query if local policy and the message transfer attributes allow it; otherwise, this may trigger setup of an MA over which the NSLP payload is sent in a Data message.

Signalling applications may specify a value to be used for the GIST hop count; otherwise, GIST selects a value itself. GIST MUST reject messages for which the signalling application has specified a value of zero. Although the GIST hop count is only intended to control message looping at the GIST level, 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-control mechanism for NSLPs that do not define anything more sophisticated. Note that the count will be decremented on forwarding through every GIST-aware node. Initial values for the GIST hop count are an implementation matter; one suitable approach is to use the same algorithm as for IP TTL setting [1].

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 signalling

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 C-mode or D-mode. Reasons for using C-mode are:

- message transfer attributes: for example, the signalling application has specified security attributes that require channel-secured delivery, or reliable delivery.

- message size: a message whose size (including the GIST header, GIST objects and any NSLP payload, and an allowance for the IP and transport layer encapsulation required by D-mode) exceeds a fragmentation-related threshold MUST be sent over C-mode, using a messaging association that supports fragmentation and reassembly internally. The allowance for IP and transport layer encapsulation is 64 bytes. The message size MUST NOT exceed the Path MTU to the next peer, if this is known. If this is not known, the message size MUST NOT exceed the least of the first-hop MTU, and 576 bytes. The same limit applies to IPv4 and IPv6.

- congestion control: D-mode SHOULD NOT be used for signalling where it is possible to set up routing state and use C-mode, unless the network can be engineered to guarantee capacity for D-mode traffic within the rate control limits imposed by GIST (see Section 5.3.3).

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. For the case of reliable delivery, GIST MUST NOT distribute messages for the same session over multiple messaging associations in parallel, but MUST use a single association at any given time. The case of moving over to a new association is covered in Section 4.4.5.

If the use of a messaging association (i.e., C-mode) 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), which will trigger the exchange of additional GIST messages. If no association can be created, this is an error condition, and should be indicated back to the local signalling application.

If a messaging association is not appropriate, the message is sent in D-mode. The processing in this case depends on the message type, local policy, and whether or not routing state exists.

- If the message is not a Query, and local policy does not request the use of Q-mode for this message, and routing state exists, it is sent with the normal D-mode encapsulation directly to the address from the routing state table.

- If the message is a Query, or the message is Data and local policy as given by the message transfer attributes requests the use of Q-mode, then it is sent in Q-mode as defined in Section 5.3.2; the details depend on the message routing method.

- If no routing state exists, GIST can attempt to use Q-mode as in the Query case: either sending a Data message with the Q-mode encapsulation or using the event as a trigger for routing state setup (see Section 4.4). If this is not possible, e.g., because the encapsulation for the MRM is only defined for one message direction, then this is an error condition that is reported back to the local signalling application.

4.3.4. Nodes not Hosting the NSLP

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

1. A message contains an RAO value that is relevant to NSIS, but it does not exactly match the Q-mode encapsulation rules of Section 5.3.2. The message MUST be transparently forwarded at the IP layer. See Section 3.6.

2. A Q-mode encapsulated message contains an RAO value that has been assigned to some NSIS signalling application but that is not used on this specific node, but the IP layer is unable to distinguish whether it needs to be passed to GIST for further processing or whether the packet should be forwarded just like a normal IP datagram.

3. A Q-mode encapsulated message contains an RAO value that has been assigned to an NSIS signalling application that is used on this node, but the signalling application does not process the NSLPID in the message. (This covers the case where a signalling application uses a set of NSLPIDs.)

4. A directly addressed message (in D-mode or C-mode) is delivered to a node for which there is no corresponding signalling application. With the current specification, this should not happen in normal operation. 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).

5. A Q-mode encapsulated message arrives at the end-system that does not handle the signalling application. This is possible in normal operation, and MUST be indicated 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.

6. The node is a GIST-aware NAT. See Section 7.2.

In case (2) and (3), the role of GIST is to forward the message essentially as though it were a normal IP datagram, 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-layer TTL field and GIST hop count are managed correctly to prevent message looping, and this should be done consistently independently of where in the packet processing path the decision is made. The rules are that in cases (2) and (3), the IP-layer TTL MUST be decremented just as if the message was a normal IP forwarded packet. In case (3), the GIST hop count MUST be decremented as in the case of normal input processing, which also applies to cases (4) and (5).

A GIST node processing Q-mode 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 forward the message. These conditions are:

1. The message is so large that it would be fragmented on downstream links, for example, because the downstream MTU is abnormally small (less than 576 bytes). 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 reached zero. 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.

3. The MRI represents a flow definition that 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 signalling 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 of the GIST message is examined; in the third case, the MRI is also examined. The rest of the message MUST NOT be inspected in any case. Similar to the case of Section 4.3.2, the GIST payloads MUST NOT be modified or re-ordered; an implementation MAY choose to achieve this by retaining the original message, rather than reconstructing it from some parsed internal representation.

4.4. Routing State and Messaging Association Maintenance

The main responsibility of GIST is to manage the routing state and messaging associations that are used in the message processing described above. Routing state is installed and refreshed by GIST handshake messages. Messaging associations are set up by the normal procedures of the transport and security protocols that comprise them, using peer IP addresses from the routing state. Once a messaging association has been created, its refresh and expiration can be managed independently from the routing state.

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 a suitable association already exists, 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. Routing State and Messaging Association Creation

The message sequence for GIST state setup between peers is shown in Figure 5 and described in detail below. The figure informally summarises the contents of each message, including optional elements in square brackets. An example is given in Appendix D.

The first message in any routing state maintenance operation is a Query, sent from the Querying node and intercepted at the responding node. This message has addressing and other identifiers appropriate for the flow and signalling application that state maintenance is being done for, addressing information about the node that generated the Query itself, and 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 later messages is to protect against certain denial-of-service attacks and to correlate the events in the message sequence (see Section 8.5 for further details).

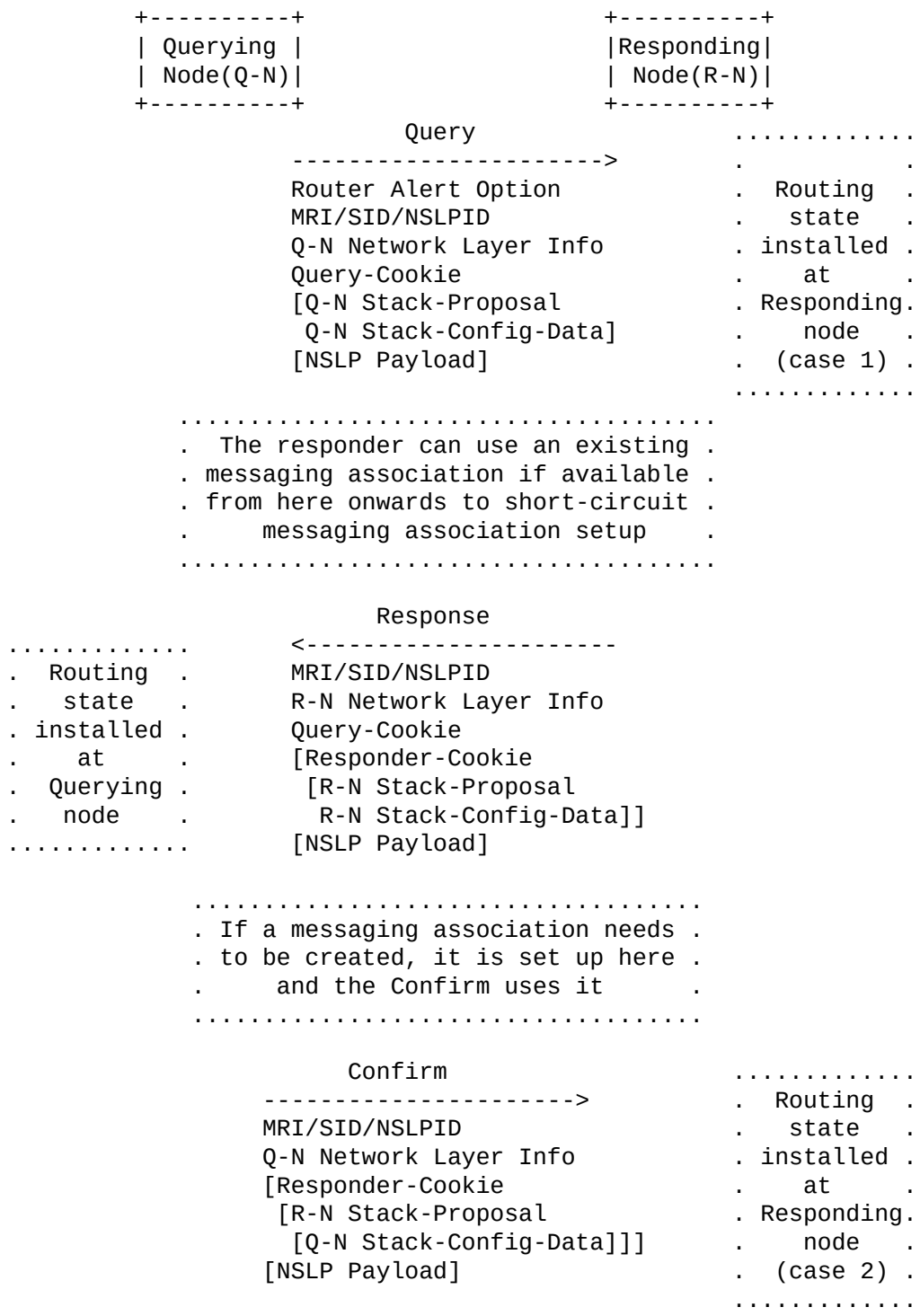


Figure 5: Message Sequence at State Setup

Provided that the signalling application has indicated that message routing state should be set up (see Section 4.3.2), reception of a Query MUST elicit a Response. This is a normally encapsulated D-mode message with additional GIST payloads. It contains network layer information about the Responding node, echoes the Query-Cookie, and MAY contain an NSLP payload, possibly a reply 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. Any setup MUST take place immediately after the specific 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. If the messaging association includes a channel security protocol, each GIST node MUST verify the authenticated identity of the peer against its authorised peer database, and if there is no match the messaging association MUST be torn down. The database and authorisation check are described in more detail in Section 4.4.2 below. Note that the verification can depend on what the MA is to be used for (e.g., for which MRI or session), so this step may not be possible immediately after authentication has completed but some time later.

Finally, after any necessary messaging association setup has completed, a Confirm MUST be sent if the Response requested it. Once the Confirm has been sent, the Querying node assumes that routing state has been installed at the responder, and can send normal Data messages for the flow in question; recovery from a lost Confirm is discussed in Section 5.3.3. If a messaging association is being used, the 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 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 (see Section 8.6). This first Confirm on a new

association MUST also contain a Stack-Configuration-Data object carrying an MA-Hold-Time value, which supersedes the value given in the original Query. The association can be used in the upstream direction for the MRI and NSLPID carried in the Confirm, after the Confirm has been received.

The Querying node MUST install the responder address, derived from the R-Node Network Layer info, as routing state information after verifying the Query-Cookie in the Response. The Responding node MAY install the querying address as peer state information at two points in time:

Case 1: after the receipt of the initial Query, or

Case 2: after a Confirm containing the Responder-Cookie.

The Responding node SHOULD derive the peer address from the Q-Node Network Layer Info if this was decoded successfully. Otherwise, it MAY be derived from the IP source address of the message if the common header flags this as being the signalling source address. 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 Confirm must contain sufficient information to allow it to be processed in the same way as 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. GIST Peer Authorisation

When two GIST nodes authenticate using a messaging association, both ends have to decide whether to accept the creation of the MA and whether to trust the information sent over it. This can be seen as an authorisation decision:

- Authorised peers are trusted to install correct routing state about themselves and not, for example, to claim that they are on-path for a flow when they are not.

- Authorised peers are trusted to obey transport- and application-level flow control rules, and not to attempt to create overload situations.

- Authorised peers are trusted not to send erroneous or malicious error messages, for example, asserting that routing state has been lost when it has not.

This specification models the decision as verification by the authorising node of the peer's identity against a local list of peers, the authorised peer database (APD). The APD is an abstract construct, similar to the security policy database of IPsec [36]. Implementations MAY provide the associated functionality in any way they choose. This section defines only the requirements for APD administration and the consequences of successfully validating a peer's identity against it.

The APD consists of a list of entries. Each entry includes an identity, the namespace from which the identity comes (e.g., DNS domains), the scope within which the entry is applicable, and whether authorisation is allowed or denied. The following are example scopes:

Peer Address Ownership: The scope is the IP address at which the peer for this MRI should be; the APD entry denotes the identity as the owner of address. If the authorising node can determine this address from local information (such as its own routing tables), matching this entry shows that the peer is the correct on-path node and so should be authorised. The determination is simple if the peer is one IP hop downstream, since the IP address can be derived from the router's forwarding tables. If the peer is more than one hop away or is upstream, the determination is harder but may still be possible in some circumstances. The authorising node may be able to determine a (small) set of possible peer addresses, and accept that any of these could be the correct peer.

End-System Subnet: The scope is an address range within which the MRI source or destination lies; the APD entry denotes the identity as potentially being on-path between the authorising node and that address range. There may be different source and destination scopes, to account for asymmetric routing.

The same identity may appear in multiple entries, and the order of entries in the APD is significant. When a messaging association is authenticated and associated with an MRI, the authorising node scans the APD to find the first entry where the identity matches that presented by the peer, and where the scope information matches the circumstances for which the MA is being set up. The identity matching process itself depends on the messaging association protocol that carries out the authentication, and details for TLS are given in Section 5.7.3. Whenever the full set of possible peers for a specific scope is known, deny entries SHOULD be added for the wildcard identity to reject signalling associations from unknown nodes. The ability of the authorising node to reject inappropriate MAs depends directly on the granularity of the APD and the precision of the scope matching process.

If authorisation is allowed, the MA can be used as normal; otherwise, it MUST be torn down without further GIST exchanges, and any routing state associated with the MA MUST also be deleted. An error condition MAY be logged locally. When an APD entry is modified or deleted, the node MUST re-validate existing MAs and the routing state table against the revised contents of the APD. This may result in MAs being torn down or routing state entries being deleted. These changes SHOULD be indicated to local signalling applications via the NetworkNotification API call (Appendix B.4).

This specification does not define how the APD is populated. As a minimum, an implementation MUST provide an administrative interface through which entries can be added, modified, or deleted. More sophisticated mechanisms are possible in some scenarios. For example, the fact that a node is legitimately associated with a specific IP address could be established by direct embedding of the IP address as a particular identity type in a certificate, or by a mapping that address to another identifier type via an additional database lookup (such as relating IP addresses in in-addr.arpa to domain names). An enterprise network operator could generate a list of all the identities of its border nodes as authorised to be on the signalling path to external destinations, and this could be distributed to all hosts inside the network. Regardless of the technique, it MUST be ensured that the source data justify the authorisation decisions listed at the start of this section, and that the security of the chain of operations on which the APD entry depends cannot be compromised.

4.4.3. Messaging Association Multiplexing

It is a design goal of GIST that, as far as possible, a single messaging association should be used for multiple flows and sessions between two peers, rather than setting up a new MA for each. This re-use of existing MAs is referred to as messaging association multiplexing. Multiplexing ensures that the MA cost scales only with the number of peers, and avoids the latency of new MA setup where possible.

However, multiplexing requires the identification of an existing MA that matches the same routing state and desired properties that would be the result of a normal handshake in D-mode, and this identification must be done as reliably and securely as continuing with a normal D-mode handshake. 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 thus with different addresses).

MA multiplexing is controlled by the Network Layer Information (NLI) object, which is carried in Query, Response, and Confirm messages. The NLI object includes (among other elements):

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 across the set of all potential peers, and SHOULD be stable at least until the next node restart. 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. For routers, the Router-ID [2], which is one of the router's IP addresses, MAY be used as one possible value for the Peer-Identity. In scenarios with nested NATs, the Router-ID alone may not satisfy the uniqueness requirements, in which case it MAY be extended with additional tokens, either chosen randomly or administratively coordinated.

Interface-Address: This is an IP address through which the signalling 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.

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 first set up. There may be more than one MA for a given NLI object, for example, with different security or transport properties.

MA multiplexing is achieved by matching these two elements from the NLI provided in a new GIST message with one associated with an existing MA. The message can be either a Query or Response, although the former is more likely:

- If there is a perfect match to an existing association, that association SHOULD be re-used, provided it meets the criteria on security and transport properties given at the end of Section 5.7.1. This is indicated by sending the remaining messages in the handshake over that association. This will lead to multiplexing on an association to the wrong node if signalling nodes have colliding Peer-Identities and one is reachable at the same Interface-Address as another. This could be caused by an on-path attacker; on-path attacks are discussed further in Section 8.7. When multiplexing is done, and the original MA authorisation was MRI-dependent, the verification steps of Section 4.4.2 MUST be repeated for the new flow.

- In all other cases, the handshake MUST be executed in D-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. These should be rare events, so the expense of a new association setup is acceptable. Another possibility is one node using another node's Peer-Identity, for example, as some kind of attack. Because the Peer-Identity is used only for this multiplexing process, the only consequence this has is to require a new association setup, and this is considered in Section 8.4.

  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. Both elements of the NLI object match: 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.

4.4.4. Routing State Maintenance

Each item of routing state expires after a lifetime that is negotiated during the Query/Response/Confirm handshake. The Network Layer Information (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. A default timer value of 30 seconds is RECOMMENDED. Nodes that can exploit alternative, more powerful, route change detection methods such as those described in Section 7.1.2 MAY choose to use much longer times. Nodes MAY use shorter times to provide more rapid change detection. If the number of active routing state items corresponds to a rate of Queries that will stress the rate limits applied to D-mode traffic (Section 5.3.3), nodes MUST increase the timer for new items and on the refresh of existing ones. A suitable value is

```
2 * (number of routing states) / (rate limit in packets/second)
```

which leaves a factor of two headroom for new routing state creation and Query retransmissions.

The Querying node MUST ensure that a Query is received before this timer expires, if it believes that the signalling session 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 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 signalling session is known to be inactive, either because upstream state has expired or because the signalling 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 at the IP layer.

This specification defines precisely only the time at which routing state expires; it does not define when refresh handshakes should be initiated. Implementations MUST select timer settings that take at least the following into account:

- the transmission latency between source and destination;
- the need for retransmissions of Query messages;
- the need to avoid network synchronisation of control traffic (cf. [42]).

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

4.4.5. Messaging Association Maintenance

Unneeded MAs are torn down by GIST, using the teardown mechanisms of the underlying transport or security protocols if available, for example, by simply closing a TCP connection. The teardown can be initiated by either end. Whether an MA 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 that might generate messages to use it.
- whether the peer still wants the MA to remain in place. During MA setup, as part of the Stack-Configuration-Data, each node advertises its own MA-Hold-Time, i.e., the time for which it will

retain an MA that is not carrying signalling traffic. A node MUST NOT tear down an MA if it has received traffic from its peer over that period. A peer that has generated no traffic but still wants the MA retained can use a special null message (MA-Hello) to indicate the fact. A default value for MA-Hold-Time of 30 seconds is RECOMMENDED. Nodes MAY use shorter times to achieve more rapid peer failure detection, but need to take into account the load on the network created by the MA-Hello messages. Nodes MAY use longer times, but need to take into account the cost of retaining idle MAs for extended periods. Nodes MAY take signalling application behaviour (e.g., NSLP refresh times) into account in choosing an appropriate value.

Because the Responding node can choose not to create state until a Confirm, an abbreviated Stack-Configuration-Data object containing just this information from the initial Query MUST be repeated by the Querying node in the first Confirm sent on a new MA. If the object is missing in the Confirm, an "Object Type Error" message (Appendix A.4.4.9) with subcode 2 ("Missing Object") MUST be returned.

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, signalling applications cannot distinguish this from the case where the MA 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 to be used for given session and peer while messages on a previous association could still be outstanding. GIST MAY use an MA-Hello request/reply exchange on an existing association to verify that messages sent on it have reached the peer. GIST MAY use the same technique to test the liveness of the underlying MA protocols themselves at arbitrary times.

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

- the transmission latency between source and destination;
- the need for retransmissions within the messaging association protocols;
- the need to avoid network synchronisation of control traffic (cf. [42]).

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

4.4.6. Routing State Failures

A GIST node can receive a message from a GIST peer that can only be correctly processed in the context of some routing state, but where no corresponding routing state exists. Cases where this can arise include:

- Where the message is random traffic from an attacker, or backscatter (replies to such traffic).
- Where routing state has been correctly installed but the peer has since lost it, for example, because of aggressive timeout settings at the peer or because the node has crashed and restarted.
- Where the routing state was not correctly installed in the first place, but the sending node does not know this. This can happen if the Confirm message of the handshake is lost.

It is important for GIST to recover from such situations promptly where they represent genuine errors (node restarts, or lost messages that would not otherwise be retransmitted). Note that only Response, Confirm, Data, and Error messages ever require routing state to exist, and these are considered in turn:

Response: A Response can be received at a node that never sent (or has forgotten) the corresponding Query. If the node wants routing state to exist, it will initiate it itself; a diagnostic error would not allow the sender of the Response to take any corrective action, and the diagnostic could itself be a form of backscatter. Therefore, an error message MUST NOT be generated, but the condition MAY be logged locally.

Confirm: For a Responding node that implements delayed state installation, this is normal behaviour, and routing state will be created provided the Confirm is validated. Otherwise, this is a case of a non-existent or forgotten Response, and the node may not have sufficient information in the Confirm to create the correct state. The requirement is to notify the Querying node so that it can recover the routing state.

Data: This arises when a node receives Data where routing state is required, but either it does not exist at all or it has not been finalised (no Confirm message). To avoid Data being black-holed, a notification must be sent to the peer.

Error: Some error messages can only be interpreted in the context of routing state. However, the only error messages that require a reply within the protocol are routing state error messages themselves. Therefore, this case should be treated the same as a Response: an error message MUST NOT be generated, but the condition MAY be logged locally.

For the case of Confirm or Data messages, if the state is required but does not exist, the node MUST reject the incoming message with a "No Routing State" error message (Appendix A.4.4.5). There are then three cases at the receiver of the error message:

No routing state: The condition MAY be logged but a reply MUST NOT be sent (see above).

Querying node: The node MUST restart the GIST handshake from the beginning, with a new Query.

Responding node: The node MUST delete its own routing state and SHOULD report an error condition to the local signalling application.

The rules at the Querying or Responding node make GIST open to disruption by randomly injected error messages, similar to blind reset attacks on TCP (cf. [46]), although because routing state matching includes the SID this is mainly limited to on-path attackers. If a GIST node detects a significant rate of such attacks, it MAY adopt a policy of using secured messaging associations to communicate for the affected MRIs, and only accepting "No Routing State" error messages over such associations.

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 in ABNF [11]; a more detailed description of the header and each object is given in Section 5.2 and bit formats in Appendix A. Note that the NAT traversal mechanism for GIST involves the insertion of an additional NAT-Traversal-Object in Query, Response, and some Data and Error messages; the rules for this are given in Section 7.2.

GIST-Message: The primary messages are either part of the three-way handshake or a simple message carrying NSLP data. Additional types are defined for errors and keeping messaging associations alive.

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

The common header includes a version number, message type and size, and NSLPID. It also carries a hop count to prevent infinite message looping and various control flags, including one (the R-flag) 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" message with the appropriate subcode (Appendix A.4.4.9). Note that unknown objects indicate explicitly how they should be treated and are not covered by the above statement.

Query: A Query MUST be sent in D-mode using the special Q-mode encapsulation. In addition to the common header, it contains certain mandatory control objects, and MAY contain a signalling application payload. A stack proposal and configuration data MUST be included if the message exchange relates to setup of a messaging association, and this is the case even if the Query is intended only for refresh (since a routing change might have taken place in the meantime). The R-flag MUST always be set (R=1) in a Query, since this message always elicits a Response.

```
Query = Common-Header
        [ NAT-Traversal-Object ]
        Message-Routing-Information
        Session-Identifier
        Network-Layer-Information
        Query-Cookie
        [ Stack-Proposal Stack-Configuration-Data ]
        [ NSLP-Data ]
```

Response: A Response MUST be sent in D-mode if no existing messaging association can be re-used. If one is being re-used, the Response MUST be sent in C-mode. It MUST echo the MRI, SID, and Query-Cookie of the Query, and carries its own Network-Layer-Information. If the message exchange relates to setup of a new messaging association, which MUST involve a D-mode Response, a Responder-Cookie MUST be included, as well as the Responder’s 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. Therefore, in particular, a Confirm will always be required if a new MA is being set up. Note that the

direction of this MRI will be inverted compared to that in the Query, that is, an upstream MRI becomes downstream and vice versa (see Section 3.3).

```
Response = Common-Header
           [ NAT-Traversal-Object ]
           Message-Routing-Information
           Session-Identifier
           Network-Layer-Information
           Query-Cookie
           [ Responder-Cookie
             [ Stack-Proposal Stack-Configuration-Data ] ]
           [ NSLP-Data ]
```

Confirm: A Confirm MUST be sent in C-mode if a messaging association is being used for this routing state, and MUST be sent before other messages for this routing state if an association is being set up. If no messaging association is being used, the Confirm MUST be sent in D-mode. The Confirm MUST include the MRI (with inverted direction) and SID, and echo the Responder-Cookie if the Response carried one. In C-mode, the Confirm MUST also echo the Stack-Proposal from the Response (if present) so it can be verified that this has not been tampered with. The first Confirm on a new association MUST also repeat the Stack-Configuration-Data from the original Query in an abbreviated form, just containing the MA-Hold-Time.

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

Data: The Data message is used to transport NSLP data without modifying GIST state. It contains no control objects, but only the MRI and SID associated with the NSLP data being transferred. Network-Layer-Information (NLI) MUST be carried in the D-mode case, but MUST NOT be included otherwise.

```
Data = Common-Header
       [ NAT-Traversal-Object ]
       Message-Routing-Information
       Session-Identifier
       [ Network-Layer-Information ]
       NSLP-Data
```

Error: An Error message reports a problem determined at the GIST level. (Errors generated by signalling applications are reported in NSLP-Data payloads and are not treated specially by GIST.) If the message is being sent in D-mode, the originator of the error message MUST include its own Network-Layer-Information object. All other information related to the error is carried in a GIST-Error-Data object.

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

MA-Hello: This message MUST be sent only in C-mode. It contains the common header, with a NSLPID of zero, and a message identifier, the Hello-ID. It always indicates that a node wishes to keep a messaging association open, and if sent with R=0 and zero Hello-ID this is its only function. A node MAY also invoke a diagnostic request/reply exchange by setting R=1 and providing a non-zero Hello-ID; in this case, the peer MUST send another MA-Hello back along the messaging association echoing the same Hello-ID and with R=0. Use of this diagnostic is entirely at the discretion of the initiating node.

```
MA-Hello = Common-Header
           Hello-ID
```

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. This specification defines GIST version 1.

GIST hop count: A hop count to prevent a message from looping indefinitely.

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

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

Context-free flag: This flag is set (C=1) if the receiver has to be able to process the message without supporting routing state. The C-flag MUST be set for Query messages, and also for Data messages sent in Q-mode. The C-flag is important for NAT traversal processing.

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 IP address of the signalling peer, so replies to this message can be sent safely to this address. S is always set in C-mode. 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 signalling source address.

Response requested: A flag that if set (R=1) indicates that a GIST message should be sent in reply to this message. The appropriate message type for the reply depends on the type of the initial message.

Explicit routing: A flag that if set (E=1) indicates that the message was explicitly routed (see Section 7.1.5).

Note that in D-mode, Section 5.3, there is a 32-bit magic number before the header. However, this is regarded as part of the encapsulation rather than part of the message itself.

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 (D-mode or C-mode).

Message-Routing-Information (MRI): Information sufficient to define how the signalling 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 signalling application. Note that it always includes a flag defining the direction as either 'upstream' or 'downstream' (see Section 3.3). It is provided by the NSLP in the message sender and used by GIST to select the message routing.

Session-Identifier (SID): The GIST session identifier is a 128-bit, cryptographically random identifier chosen by the node that originates the signalling exchange. See Section 3.7.

Network-Layer-Information (NLI): 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 IP hop count between GIST peers to be measured and reported, and a validity time (RS-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 4.4.4. The peer-identity and interface-address are used for matching existing associations, as discussed in Section 4.4.3.

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 signalling message, whether in D-mode or C-mode. If this object is carried in a message with the source addressing mode flag S=1, the interface-address MUST match the source address used in the IP encapsulation, to assist in legacy NAT detection (Section 7.2.1). If this object is carried in a Query or Confirm, the interface-address MUST specifically be set to an address bound to an interface associated with the MRI, to allow its use in route change handling as discussed in Section 7.1. A suitable choice is the interface that is carrying the outbound flow. 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 multihomed host. However, some of these interface addresses may not be usable by the peer. A node MUST follow a 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 (upstream/downstream as determined from the MRI as described above) and encapsulation.

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

* On receiving a downstream message in D-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 sent 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 IP-TTL value reported to signalling applications is the one stored with the routing state for that flow, after it has been updated if necessary 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 = protocol-count 1*protocol-layer
                ;; padded on the right with 0 to 32-bit boundary

protocol-count = %x01-FF
                ;; number of the following <protocol-layer>,
                ;; represented as one byte.  This doesn't include
                ;; padding.

protocol-layer = %x01-FF
```

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

Stack-Configuration-Data (SCD): 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 4.4.5 for more discussion. The MA-protocol-options fields give the configuration of the protocols (e.g., TCP, TLS) 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 Query and MUST be echoed in a Response; a Responder-Cookie MAY be sent in a Response, and if present MUST be echoed in the following Confirm. Cookies are variable-length bit strings, chosen by the cookie generator. See Section 8.5 for further details on requirements and mechanisms for cookie generation.

Hello-ID: The Hello-ID is a 32-bit quantity that is used to correlate messages in an MA-Hello request/reply exchange. A non-zero value MUST be used in a request (messages sent with R=1) and the same value must be returned in the reply (which has R=0). The value zero MUST be used for all other messages; if a message is received with R=1 and Hello-ID=0, an "Object Value Error" message (Appendix A.4.4.10) with subcode 1 ("Value Not Supported") MUST be returned and the message dropped. Nodes MAY use any algorithm to generate the Hello-ID; a suitable approach is a local sequence number with a random starting point.

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

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

```
GIST-Error-Data = error-class error-code error-subcode
                  common-error-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-error-header carries the Common-Header from the original message, and contents of the Message-Routing-Information (MRI) and Session-Identifier (SID) objects are also included if they were successfully decoded. For some errors, additional information fields can be included, and these fields themselves have a simple TLV format. Finally, an optional free-text comment may be added.

5.3. D-mode Transport

This section describes the various encapsulation options for D-mode messages. Although there are several possibilities, depending on message type, MRM, 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 D-mode messages where the signalling peer is already known from previous signalling. 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 message is carried in a single UDP datagram. UDP checksums MUST be enabled. The UDP payload MUST always begin with a 32-bit magic number with value 0x4e04 bda5 in network byte order; this is followed by the GIST common header and the complete set of payloads. If the magic number is not present, the message MUST be silently dropped. The normal encapsulation is shown in outline in Figure 6.

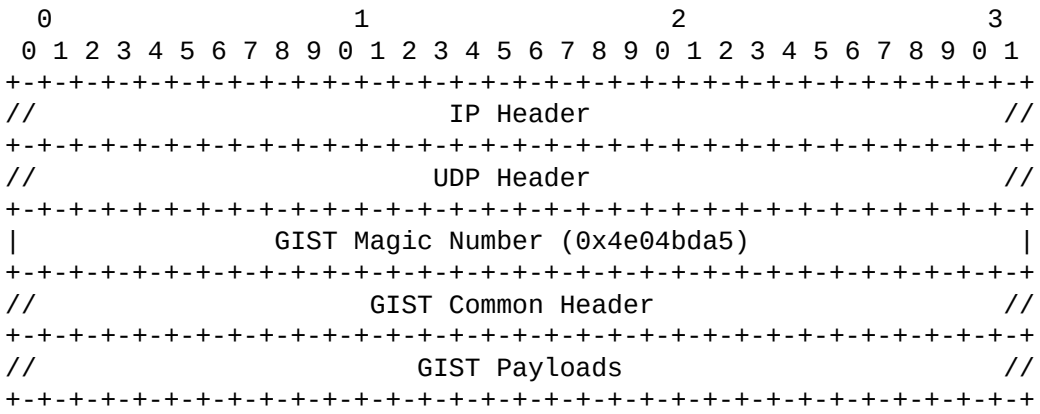


Figure 6: Normal Encapsulation Packet Format

The message is IP addressed directly to the adjacent peer as given by the routing state table. Where the message is a direct reply to a Query and no routing state exists, the destination address is derived from the input message using the same rules as in Section 4.4.1. 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 with source and destination port numbers swapped for messages in the opposite direction. Messages with the normal encapsulation MUST be sent with source addressing mode flag S=1 unless the message is a reply to a message that is known to have passed through a NAT, and the receiver MUST check the IP source address with the interface-address given in the NLI as part of legacy NAT detection. Both these aspects of message processing are discussed further in Section 7.2.1.

5.3.2. Q-mode Encapsulation

Q-mode encapsulation MUST be used for messages where no routing state is available or where the routing state is being refreshed, in particular, for Query messages. Q-mode can also be used when requested by local policy. Q-mode encapsulation is similar to normal encapsulation, with changes in IP address selection, rules about IP options, and a defined method for selecting UDP ports.

It is an essential property of the Q-mode encapsulation that it is possible for a GIST node to intercept these messages efficiently even when they are not directly addressed to it and, conversely, that it is possible for a non-GIST node to ignore these messages without overloading the slow path packet processing. This document specifies that interception is done based on RAOs.

5.3.2.1. Encapsulation and Interception in IPv4

In general, the IP addresses are derived from information in the MRI; the exact rules depend on the MRM. For the case of messages with source addressing mode flag S=1, the receiver MUST check the IP source address against the interface-address given in the NLI as part of legacy NAT detection; see Section 7.2.1.

Current MRMs define the use of a Router Alert Option [13] to assist the peer in intercepting the message depending on the NSLPID. If the MRM defines the use of RAO, the sender MUST include it unless it has been specifically configured not to (see below). A node MAY make the initial interception decision based purely on IP-Protocol number transport header analysis. Implementations MAY provide an option to disable the setting of RAO on Q-mode packets on a per-destination prefix basis; however, the option MUST be disabled by default and MUST only be enabled when it has been separately verified that the next GIST node along the path to the destination is capable of intercepting packets without RAO. The purpose of this option is to allow operation across networks that do not properly support RAO; further details are discussed in Appendix C.

It is likely that fragmented datagrams will not be correctly intercepted in the network, since the checks that a datagram is a Q-mode packet depend on data beyond the IP header. Therefore, the sender MUST set the Don't Fragment (DF) bit in the IPv4 header. Note that ICMP "packet too large" messages will be sent to the source address of the original IP datagram, and since all MRM definitions recommend S=1 for at least some retransmissions, ICMP errors related to fragmentation will be seen at the Querying node.

The upper layer protocol, identified by the IP-Protocol field in the IP header, MUST be UDP.

5.3.2.2. Encapsulation and Interception in IPv6

As for IPv4, the IP addresses are derived from information in the MRI; the exact rules depend on the MRM. For the case of messages with source addressing mode flag S=1, the receiver MUST check the IP source address with the interface-address given in the NLI as part of legacy NAT detection; see Section 7.2.1.

For all current MRMs, the IP header is given a Router Alert Option [8] to assist the peer in intercepting the message depending on the NSLPID. If the MRM defines the use of RAO, the sender MUST include it without exception. It is RECOMMENDED that a node bases its initial interception decision purely on the presence of a hop-by-hop option header containing the RAO, which will be at the start of the header chain.

The upper layer protocol MUST be UDP without intervening encapsulation layers. Following any hop-by-hop option header, the IP header MUST NOT include any extension headers other than routing or destination options [5], and for the last extension header MUST have a next-header field of UDP.

5.3.2.3. Upper Layer Encapsulation and Overall Interception Requirements

For both IP versions, the above rules require that the upper layer protocol identified by the IP header MUST be UDP. Other packets MUST NOT be identified as GIST Q-mode packets; this includes IP-in-IP tunnelled packets, other tunnelled packets (tunnel mode AH/ESP), or packets that have undergone some additional transport layer processing (transport mode AH/ESP). If IP output processing at the originating node or an intermediate router causes such additional encapsulations to be added to a GIST Q-mode packet, this packet will not be identified as GIST until the encapsulation is terminated. If the node wishes to signal for data over the network region where the

encapsulation applies, it MUST generate additional signalling with an MRI matching the encapsulated traffic, and the outbound GIST Q-mode messages for it MUST bypass the encapsulation processing.

Therefore, the final stage of the interception process and the final part of encapsulation is at the UDP level. 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 the sender MUST use the destination UDP port allocated for GIST by IANA (see Section 9). Note that for some MRMs, GIST nodes anywhere along the path can generate GIST packets with source addresses that spoof the source address of the data flow. Therefore, destinations cannot distinguish these packets from genuine end-to-end data purely on address analysis. Instead, it must be possible to distinguish such GIST packets by port analysis; furthermore, the mechanism to do so must remain valid even if the destination is GIST-unaware. GIST solves this problem by using a fixed destination UDP port from the "well known" space for the Q-mode encapsulation. This port should never be allocated on a GIST-unaware host, and therefore Q-mode encapsulated messages should always be rejected with an ICMP error. The usage of this destination port by other applications will result in reduced performance due to increased delay and packet drop rates due to their interception by GIST nodes.

A GIST node will need to be capable to filter out all IP/UDP packets that have a UDP destination port number equal to the one registered for GIST Q-mode encapsulation. These packets SHOULD then be further verified to be GIST packets by checking the magic number (see Section 5.3.1). The packets that meet both port and magic number requirements are further processed as GIST Q-mode packets. Any filtered packets that fail this GIST magic number check SHOULD be forwarded towards the IP packet’s destination as a normal IP datagram. To protect against denial-of-service attacks, a GIST node SHOULD have a rate limiter preventing more packets (filtered as potential Q-mode packets) from being processed than the system can safely handle. Any excess packets SHOULD be discarded.

5.3.2.4. IP Option Processing

For both IPv4 and IPv6, for Q-mode packets with IP options allowed by the above requirements, IP options processing is intended to be carried out independently of GIST processing. Note that for the options allowed by the above rules, the option semantics are independent of the payload: UDP payload modifications are not prevented by the options and do not affect the option content, and conversely the presence of the options does not affect the UDP payload.

On packets originated by GIST, IP options MAY be added according to node-local policies on outgoing IP data. On packets forwarded by GIST without NSLP processing, IP options MUST be processed as for a normally forwarded IP packet. On packets locally delivered to the NSLP, the IP options MAY be passed to the NSLP and equivalent options used on subsequently generated outgoing Q-mode packets. In this case, routing related options SHOULD be processed identically as they would be for a normally forwarded IP packet.

5.3.3. Retransmission and Rate Control

D-mode uses UDP, and hence has no automatic reliability or congestion control capabilities. Signalling applications requiring reliability should be serviced using C-mode, which should also carry the bulk of signalling 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 signalling or state setup requests from the signalling applications.

Query messages that do not receive Responses MAY be retransmitted; retransmissions MUST use a binary exponential backoff. The initial timer value is T1, which the backoff process can increase up to a maximum value of T2 seconds. The default value for T1 is 500 ms. T1 is an estimate of the round-trip time between the Querying and Responding nodes. Nodes MAY use smaller values of T1 if it is known that the Query should be answered within the local network. T1 MAY be chosen larger, and this is RECOMMENDED if it is known in advance (such as on high-latency access links) that the round-trip time is larger. The default value of T2 is 64*T1. Note that Queries may go unanswered either because of message loss (in either direction) or because there is no reachable GIST peer. Therefore, implementations MAY trade off reliability (large T2) against promptness of error feedback to applications (small T2). If the NSLP has indicated a timeout on the validity of this payload (see Appendix B.1), T2 MUST be chosen so that the process terminates within this timeout. Retransmitted Queries MUST use different Query-Cookie values. If the Query carries NSLP data, it may be delivered multiple times to the signalling application. These rules apply equally to the message that first creates routing state, and those that refresh it. 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. Although a node SHOULD terminate its retransmission process when any reply is received, it MUST continue to process further replies as normal.

This algorithm is sufficient to handle lost Queries and Responses. The case of a lost Confirm is more subtle. The Responding node MAY run a retransmission timer to resend the Response until a Confirm is received; the timer MUST use the same backoff mechanism and parameters as for Responses. 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. This approach is only appropriate if the Responding node is prepared to store per-flow state after receiving a single (Query) message, which includes the case where the node has queued NSLP data. If the Responding node has delayed state installation, the error condition will only be detected when a Data message arrives. This is handled as a routing state error (see Section 4.4.6) that causes the Querying node to restart the handshake.

The basic rate-control requirements for D-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. Rate-control applies only to locally generated D-mode messages, not to messages that are being forwarded. When the rate limiter is in effect, D-mode messages MUST be queued until transmission is re-enabled, or they MAY be dropped with an error condition indicated back to local signalling applications. In either case, the effect of this will be to reduce the rate at which new transactions can be initiated by signalling applications, thereby reducing the load on the network.

The rate-limiting mechanism is implementation-defined, but it is RECOMMENDED that a token bucket limiter as described in [33] be used. The token bucket MUST be sized to ensure that a node cannot saturate the network with D-mode traffic, for example, when re-probing the network for multiple flows after a route change. A suitable approach is to restrict the token bucket parameters so that the mean output rate is a small fraction of the node's lowest-speed interface. It is RECOMMENDED that this fraction is no more than 5%. Note that according to the rules of Section 4.3.3, in general, D-mode SHOULD only be used for Queries and Responses rather than normal signalling traffic unless capacity for normal signalling traffic can be engineered.

5.4. C-mode Transport

It is a requirement of the NTLP defined in [29] that it should be able to support bundling of small messages, fragmentation of large messages, and message boundary delineation. TCP provides both bundling and fragmentation, but not message boundaries. However, the

length information in the GIST common header allows the message boundary to be discovered during parsing. The bundling together of small messages either can be done within the transport protocol or can be carried out by GIST 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). 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, for example, because it is congestion controlled.

The second type of bundling is always appropriate. For GIST, the first type MUST NOT be used for trigger messages (i.e., messages that update GIST or signalling application state), but may be appropriate for refresh messages (i.e., messages that just extend timers). These distinctions are known only to the signalling applications, but MAY be indicated (as an implementation issue) by setting the priority transfer attribute (Section 4.1.2).

It can be seen that all of these transport protocol options can be supported by the basic GIST message format already presented. 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. This specification defines only the use of TCP, but other possibilities could be included without additional work on message formatting.

5.5. Message Type/Encapsulation Relationships

GIST has four primary message types (Query, Response, Confirm, and Data) and three possible encapsulation methods (normal D-mode, Q-mode, and C-mode). The combinations of message type and encapsulation that are allowed for message transmission 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 that can only be sent in Q-mode or D-mode arrives in C-mode or vice versa, 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, except that the

decapsulation process may provide additional information, such as translated addresses or IP hop count to be used in the subsequent message processing.

Message	Normal D-mode	Query D-mode (Q-mode)	C-mode
Query	Never	Always, with C-flag=1	Never
Response	Unless a messaging association is being re-used	Never	If a messaging association is being re-used
Confirm	Only if no 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
Data	If routing state exists for the flow but no messaging association	If the MRI can be used to derive the Q-mode encapsulation, and either no routing state exists or local policy requires Q-mode; MUST have C-flag=1	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 reaction to incoming messages. Error messages MUST NOT be generated in reaction to incoming Error messages. The routing and encapsulation of the Error message are 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 D-mode, the error MUST be sent in D-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 Error message reports information about the originator 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 Error message has the value zero. If for any reason the message cannot be sent (for example, because it is too large to send in D-mode, or because the MA over which the original message arrived has since been closed), an error SHOULD be logged locally. The receiver of the Error message can infer the NSLPID for the message that caused the error from the Common Header that is embedded in the Error Object.

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 Query and Response messages, using Stack-Proposal and Stack-Configuration-Data (SCD) objects. If a new messaging association is required, it is then set up, followed by a Confirm. Messaging association multiplexing is achieved by short-circuiting this exchange by sending the Response or Confirm messages on an existing association (Section 4.4.3); whether to do this is a matter of local policy. The end result of this process is a messaging association that 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 SCD 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). A Stack-Proposal is generally accompanied by an SCD object that carries an MA-protocol-options field for any protocol listed in the Stack-Proposal that needs it. An MA-protocol-options field may apply globally, to all instances of the protocol in the Stack-Proposal, or it can be tagged as applying to a specific instance. The latter approach can for example be used to carry different port numbers for TCP depending on whether it is to be used with or without TLS. An message flow that shows several of the features of Stack-Proposal and Stack-Configuration-Data formats can be found in Appendix D.

An MA-protocol-options field may also be flagged as not usable; for example, a NAT that 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 SCD are both present but not consistent, for example, if they refer to different protocols, or an MA-protocol-options field refers to a non-existent profile, an "Object Value Error" message (Appendix A.4.4.10) with subcode 5 ("Stack-Proposal - Stack-Configuration-Data Mismatch") MUST be returned and the message dropped.

A node generating an SCD 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. This MAY require the creation of listening endpoints for the transport and security protocols in question, or a node MAY keep a pool of such endpoints open for extended periods.

However, the received object contents MUST be retained only for the duration of the Query/Response exchange and to allow any necessary association setup to complete. They may become invalid because of expired bindings at intermediate NATs, or because the advertising node is using agile ports. Once the setup is complete, or if it is not necessary or fails for some reason, the object contents MUST be discarded. A default time of 30 seconds to keep the contents is RECOMMENDED.

A Query requesting messaging association setup always contains a Stack-Proposal and SCD object. The Stack-Proposal MUST only include protocol configurations that are suitable for the transfer attributes of the messages for which the Querying node wishes to use the messaging association. For example, it should not simply include all configurations that the Querying node is capable of supporting.

The Response always contains a Stack-Proposal and SCD object, unless multiplexing (where the Responder decides to use an existing association) occurs. For such a Response, the security protocols listed in the Stack-Proposal MUST NOT depend on the Query. A node MAY make different proposals depending on the combination of interface and NSLPID. If multiplexing does occur, which is indicated by sending the Response over an existing messaging association, the following rules apply:

- The re-used messaging association MUST NOT have weaker security properties than all of the options that would have been offered in the full Response that would have been sent without re-use.

- The re-used messaging association MUST have equivalent or better transport and security characteristics as at least one of the protocol configurations that was offered in the Query.

Once the messaging association is set up, the Querying node repeats the responder's Stack-Proposal over it in the Confirm. The Responding node MUST verify that this has not been changed as part of bidding-down attack prevention, as well as verifying the Responder-Cookie (Section 8.5). If either check fails, the Responding node MUST NOT create the message routing state (or MUST delete it if it already exists) and SHOULD log an error condition locally. If this is the first message on a new MA, the MA MUST be torn down. 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. 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 additional options data (the MA-protocol-options Length field is zero).
- in a Response: 2-byte port number at which the connection will be accepted, followed by 2 pad bytes.

The connection is opened in the forwards direction, from the Querying node towards the responder. The Querying node MAY use any source address and source port. The destination information MUST be derived from information in the Response: the address from the interface-address from the Network-Layer-Information object and the port from the SCD object as described above.

Associations using Forwards-TCP can carry messages with the transfer attribute Reliable=True. If an error occurs on the TCP connection such as a reset, as can be detected for example by a socket exception condition, GIST MUST report this to NSLPs as discussed in Section 4.1.2.

5.7.3. Protocol Definition: Transport Layer Security

This MA-Protocol-ID denotes a basic use of transport layer channel security, initially in conjunction with TCP. Support for this protocol in conjunction with TCP is REQUIRED; associations using it can carry messages with transfer attributes requesting confidentiality or integrity protection. The specific TLS version will be negotiated within the TLS layer itself, but implementations MUST NOT negotiate to protocol versions prior to TLS1.0 [15] and MUST use the highest protocol version supported by both peers. Implementation of TLS1.2 [10] is RECOMMENDED. 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. They MAY negotiate any mutually acceptable ciphersuite that provides authentication, integrity, and confidentiality.

The default mode of TLS authentication, which applies in particular to the above ciphersuites, uses a client/server X.509 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, such as the use of pre-shared keys [32]. 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 TLS protocol definition. The configuration information for the transport protocol over which TLS is running (e.g., TCP port number) is provided by the MA-protocol-options for that protocol.

5.7.3.1. Identity Checking in TLS

After TLS authentication, a node MUST check the identity presented by the peer in order to avoid man-in-the-middle attacks, and verify that the peer is authorised to take part in signalling at the GIST layer. The authorisation check is carried out by comparing the presented identity with each Authorised Peer Database (APD) entry in turn, as discussed in Section 4.4.2. This section defines the identity comparison algorithm for a single APD entry.

For TLS authentication with X.509 certificates, an identity from the DNS namespace MUST be checked against each subjectAltName extension of type dNSName present in the certificate. If no such extension is present, then the identity MUST be compared to the (most specific) Common Name in the Subject field of the certificate. When matching DNS names against dNSName or Common Name fields, matching is case-insensitive. Also, a "*" wildcard character MAY be used as the left-most name component in the certificate or identity in the APD. For example, *.example.com in the APD would match certificates for a.example.com, foo.example.com, *.example.com, etc., but would not match example.com. Similarly, a certificate for *.example.com would be valid for APD identities of a.example.com, foo.example.com, *.example.com, etc., but not example.com.

Additionally, a node MUST verify the binding between the identity of the peer to which it connects and the public key presented by that peer. Nodes SHOULD implement the algorithm in Section 6 of [8] for general certificate validation, but MAY supplement that algorithm

with other validation methods that achieve equivalent levels of verification (such as comparing the server certificate against a local store of already-verified certificates and identity bindings).

For TLS authentication with pre-shared keys, the identity in the psk_identity_hint (for the server identity, i.e., the Responding node) or psk_identity (for the client identity, i.e., the Querying node) MUST be compared to the identities in the APD.

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 Q-mode encapsulation rules. These are given in the following subsections for the MRMs currently defined. A GIST implementation on a node MUST support whatever MRMs are required by the NSLPs on that node; GIST implementations SHOULD provide support for both the MRMs defined here, in order to minimise deployment barriers for new signalling applications that need them.

5.8.1. The Path-Coupled MRM

5.8.1.1. Message Routing Information

For the path-coupled MRM, the message routing information (MRI) is conceptually the Flow Identifier as in the NSIS framework [29]. 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 specified if possible (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, IPsec Security Parameters Index (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 or 128 (for IPv4 and IPv6, respectively) MAY be used to implement address wildcarding, allowing the MRI to refer to traffic to or from a wider address range. An additional flag defines the message direction relative to the MRI (upstream vs. downstream).

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 Q-mode Encapsulation

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

- The destination IP 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; therefore, the source addressing mode flag in the common header S=0. This provides the best likelihood that the message will be correctly routed through any region performing per-packet policy-based forwarding or load balancing that takes the source address into account. However, there may be circumstances where the use of the signalling source address (S=1) is preferable, such as:

  * In order to receive ICMP error messages about the signalling message, such as unreachable port or address. If these are delivered to the flow source rather than the signalling source, it will be very difficult for the querying node to detect that it is the last GIST node on the path. Another case is where there is an abnormally low MTU along the path, in which case the querying node needs to see the ICMP error (recall that Q-mode packets are sent with DF set).

  * 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 (see Section 7.2.1).

  Because of these considerations, use of the signalling 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 signalling source address for some retransmissions or as a matter of static configuration,

for example, if a NAT is known to be in the path out of a certain interface. The S-flag in the common header tells the message receiver which option was used.

A Router Alert Option is also included in the IP header. The option value depends on the NSLP being signalled for. In addition, it is essential that the Query mimics the actual data flow as closely as possible, since this is the basis of how the signalling 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.

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, it MUST be rejected with an appropriate error message. Such checks apply only to messages with the Q-mode encapsulation, since only those messages are required to track the flow path. The main checks are that the IP version used in the encapsulation should match that of the MRI and 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 Q-mode Encapsulation

In some deployment scenarios, it is desirable to set up routing state in the upstream direction (i.e., from flow receiver towards the sender). This could be used to support firewall signalling to control traffic from an uncooperative sender, or signalling in general where the flow sender was not NSIS-capable. This capability 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 IP routing. However, in particular cases, the upstream peer can be discovered with a high degree of confidence, for example:

- The upstream GIST peer is one 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 Q-mode 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 with some additional IP TTL checks. No special functionality is needed for this.

It is possible for routing state at a given node, for a specific MRI and NSLPID, to be created by both an upstream Query exchange (initiated by the node itself) and a downstream Query exchange (where the node is the responder). If the SIDs are different, these items of routing state MUST be considered as independent; if the SIDs match, the routing state installed by the downstream exchange MUST take precedence, provided that the downstream Query passed ingress filtering checks. The rationale for this is that the downstream Query is in general a more reliable way to install state, since it directly probes the IP routing infrastructure along the flow path, whereas use of the upstream Query depends on the correctness of the Querying node's understanding of the topology.

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 IP routing MAY use an alternative destination address (e.g., the address of its default router).

- The source address SHOULD be the signalling node address, so in the common header S=1.

- A Router Alert Option is included as in the downstream case.

- The Diffserv codepoint and (for IPv6) flow label MAY be set to match the values from the MRI as in the downstream case, and the UDP port selection is also the same.

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

The sending GIST implementation SHOULD attempt to send the Query via 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 that have reached a node at which they can no longer be trusted. In particular, a node SHOULD reject a message that has been propagated more than one IP hop, with an "Invalid IP layer TTL" error message (Appendix A.4.4.11). This can be determined by examining the received IP layer TTL, similar to the generalised IP TTL security mechanism described in [41].

Alternatively, receipt of an upstream Query at the flow source MAY be used to trigger setup of GIST state in the downstream direction. These restrictions may be relaxed in a future version.

5.8.2. The Loose-End MRM

The Loose-End 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 as in [34].

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 that 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 Q-mode 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 from the MRI (S=0). However, the use of the signalling source address (S=1) is allowed as in the case of the path-coupled MRM.

A Router Alert Option is included in the IP header. The option value depends on the NSLP being signalled for. There are no special requirements on the setting of the Diffserv codepoint, IP layer TTL, or (for IPv6) the flow label. Nor are any special validation checks applied.

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. A more detailed description of the GIST protocol operation in state machine syntax can be found in [45].

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

1. There is a top-level state machine that represents the node itself (Node-SM). It is responsible for the processing of events that 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-MRI state machines to manage the GIST handshake and routing state maintenance procedures.

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

3. For each flow and signalling direction where the node has accepted the creation of routing state by a peer, there is an instance of a Responding-Node state machine (Responding-SM). This machine is responsible for managing the status of the routing state for that flow. Depending on policy, it MAY be responsible for transmission or retransmission of Response messages, or this MAY be handled by the Node-SM, and a Responding-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 an 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 Querying-SM and

Responding-SM for a single flow and signalling direction do not interact. That is, the Responding-SM that accepts the creation of routing state for a flow on one interface has no direct interaction with the Querying-SM that 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 Query has been received.
rx_Response	A Response has been received.
rx_Confirm	A Confirm has been received.
rx_Data	A Data message has been received.
rx_Message	rx_Query||rx_Response||rx_Confirm||rx_Data.
rx_MA-Hello	An MA-Hello message has been received.
tg_NSLPData	A signalling 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.
tg_MAIdle	GIST decides that it is no longer necessary to keep an MA open for itself.
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.

Incoming Events

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; no other events are possible. 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 signalling
    application policy relating to this peer
    // note that this interaction delivers any NSLP-Data to
    // the NSLP as a side effect
if (the signalling application indicates that routing state should
    be created) then
  if (routing state can be created without a 3-way handshake) then
    create Responding-SM and transfer control to it
  else
    send Response with R=1
else
  propagate the Query with any updated NSLP payload provided

Rule 2 (rx_Response):
// a routing state error
discard message

Rule 3 (rx_Confirm):
if (routing state can be created before receiving a Confirm) then
  // we should already have Responding-SM for it,
  // which would handle this message
  discard message
  send "No Routing State" error message
else
  create Responding-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 4 (er_NoRSM):
discard the message
```

```
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 Querying-SM and pass message to it
```

6.2. Query Node Processing

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

- Awaiting Response
- Established
- Awaiting Refresh

The Querying-SM is created by the Node-SM machine as a result of a request to send a message for a flow in a signalling 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 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 (which may be the result of a lost Confirm) MUST be treated the same way. 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 to ensure that a Query is generated before the peer's routing state expires (see Section 4.4.4).

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 NSLP 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 MUST 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 and MUST be ignored.

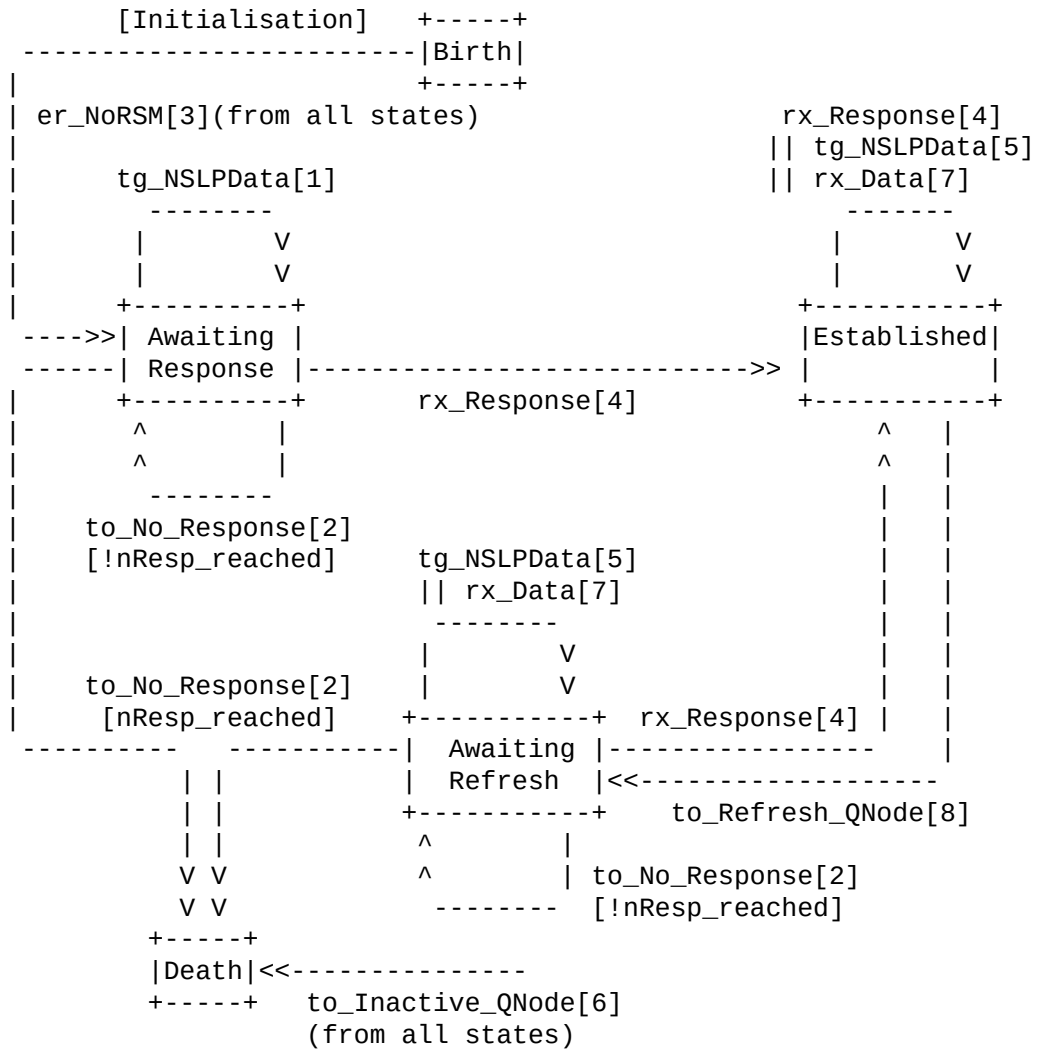


Figure 7: Query Node State Machine

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
  start No_Response timer with new value

Rule 3:
// Assume the Confirm was lost in transit or the peer has reset;
// restart the handshake
send Query
(re)start No_Response timer

Rule 4:
if a new MA-SM is needed create one
if the R-flag was set send a Confirm
send any stored Data messages
stop No_Response timer
start Refresh_QNode timer
start Inactive_QNode timer if it was not running
if there was piggybacked NSLP-Data
  pass it to the NSLP
  restart Inactive_QNode timer

Rule 5:
send Data message
restart Inactive_QNode timer

Rule 6:
   Terminate

Rule 7:
pass any data to the NSLP
restart Inactive_QNode timer

Rule 8:
send Query
start No_Response timer
stop Refresh_QNode timer
```

6.3. Responder Node Processing

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

- Awaiting Confirm
- Established
- Awaiting Refresh

The policy governing the handling of Query messages and the creation of the Responding-SM has three cases:

1. No Confirm is required for a Query, and the state machine can be created immediately.
2. A Confirm is required for a Query, but the state machine can still be created immediately. A timer is used to retransmit Response messages and the Responding-SM is destroyed if no valid Confirm is received.
3. A Confirm is required for a Query, and the state machine can only be created when it is received; the initial Query will have been handled by the Node-level machine.

In case 2, the Responding-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 Responding-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 principle, an implementation MAY change its policy on handling a Query message at any time; however, the state machine descriptions here cover only the case where the policy is fixed while waiting for a Confirm message.

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 Responding-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 reply 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. If this timer fires, the routing state machine is deleted, regardless of whether a No_Confirm timer is running.

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.

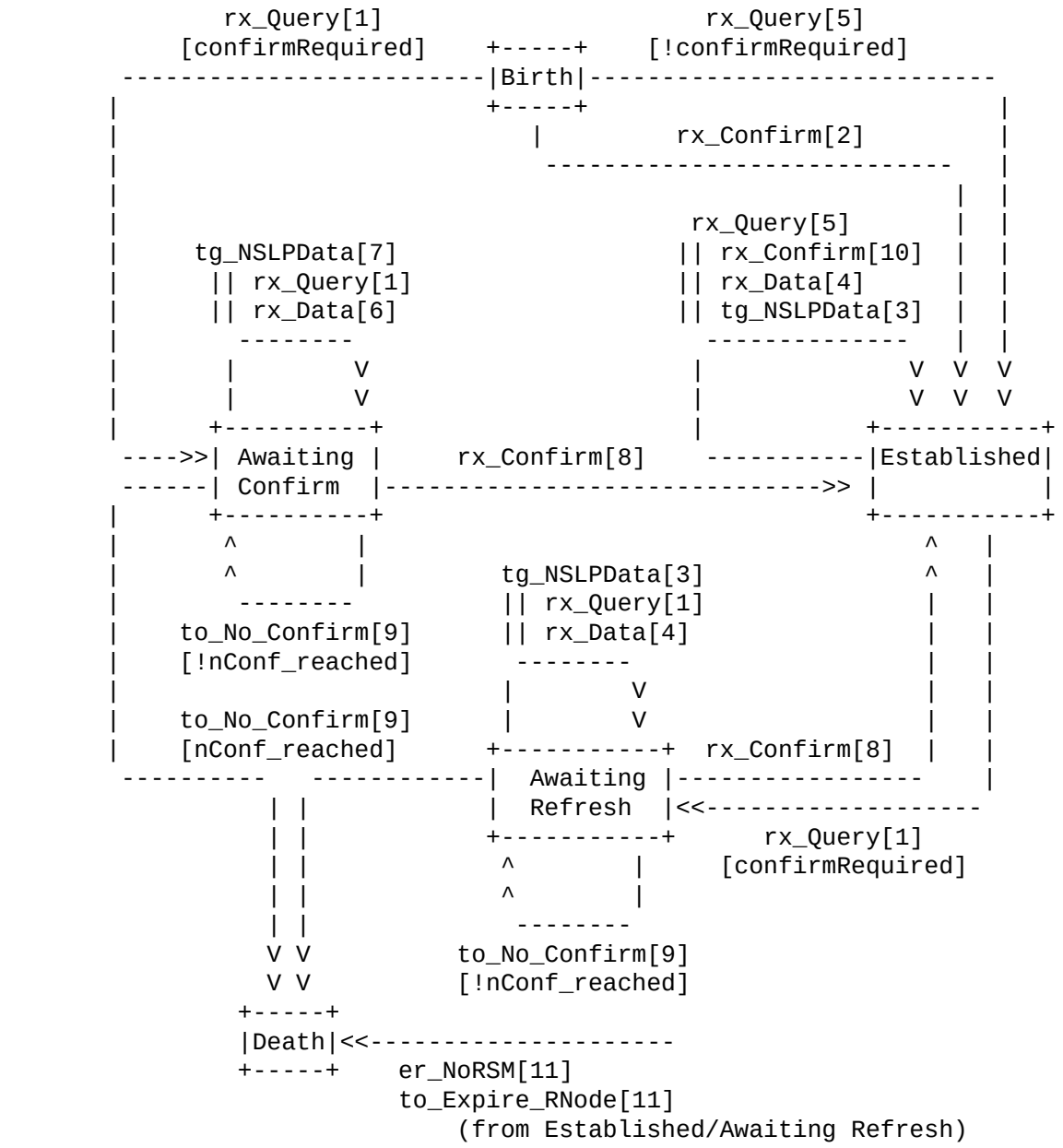


Figure 8: Responder Node State Machine

The processing rules are as follows:

```
Rule 1:
// a Confirm is required
send Response with R=1
(re)start No_Confirm timer with the initial timer value

Rule 2:
pass any NSLP-Data object to the NSLP
start Expire_RNode timer

Rule 3:  send the Data message

Rule 4:  pass data to NSLP

Rule 5:
// no Confirm is required
send Response with R=0
start Expire_RNode timer

Rule 6:
drop incoming data
send "No Routing State" error message

Rule 7:  store Data message

Rule 8:
pass any NSLP-Data object 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
  start No_Response timer

Rule 10:
// can happen e.g., a retransmitted Response causes a duplicate Confirm
silently ignore

Rule 11:  destroy self
```

6.4. Messaging Association Processing

Messaging associations (MAs) are modelled for use within GIST with a simple three-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 and that the node wishes it to remain open. In this state, the node operates a timer (SendHello) to ensure that messages are regularly sent to the peer, to ensure that the peer does not tear down the MA. The node transitions from Connected to Idle (indicating that it no longer needs the association) as a matter of local policy; one way to manage the policy is to use an activity timer but this is not specified explicitly by the state machine (see also Section 4.4.5).

In the Idle state, the node no longer requires the messaging association but the peer still requires it and is indicating this by sending periodic MA-Hello messages. A different timer (NoHello) operates to purge the MA when these messages stop arriving. If real data is transferred over the MA, the state machine transitions back to Connected.

At any time in the Connected or Idle states, a node MAY test the connectivity to its peer and the liveness of the GIST instance at that peer by sending an MA-Hello request with R=1. Failure to receive a reply with a matching Hello-ID within a timeout MAY be taken as a reason to trigger er_MAFailure. Initiation of such a test and the timeout setting are left to the discretion of the implementation. Note that er_MAFailure may also be signalled by indications from the underlying messaging association protocols. If a messaging association fails, this MUST be indicated back to the routing state machines that use it, and these MAY generate indications to signalling applications. In particular, if the messaging association was being used to deliver messages reliably, this MUST be reported as a NetworkNotification error (Appendix B.4).

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 signalling applications, there is some interaction between the two because

security-related information becomes available during the open process, and this may be indicated to signalling applications if they have requested it.

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

SendHello: Indicates that an MA-Hello 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 MA-Hello 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.

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

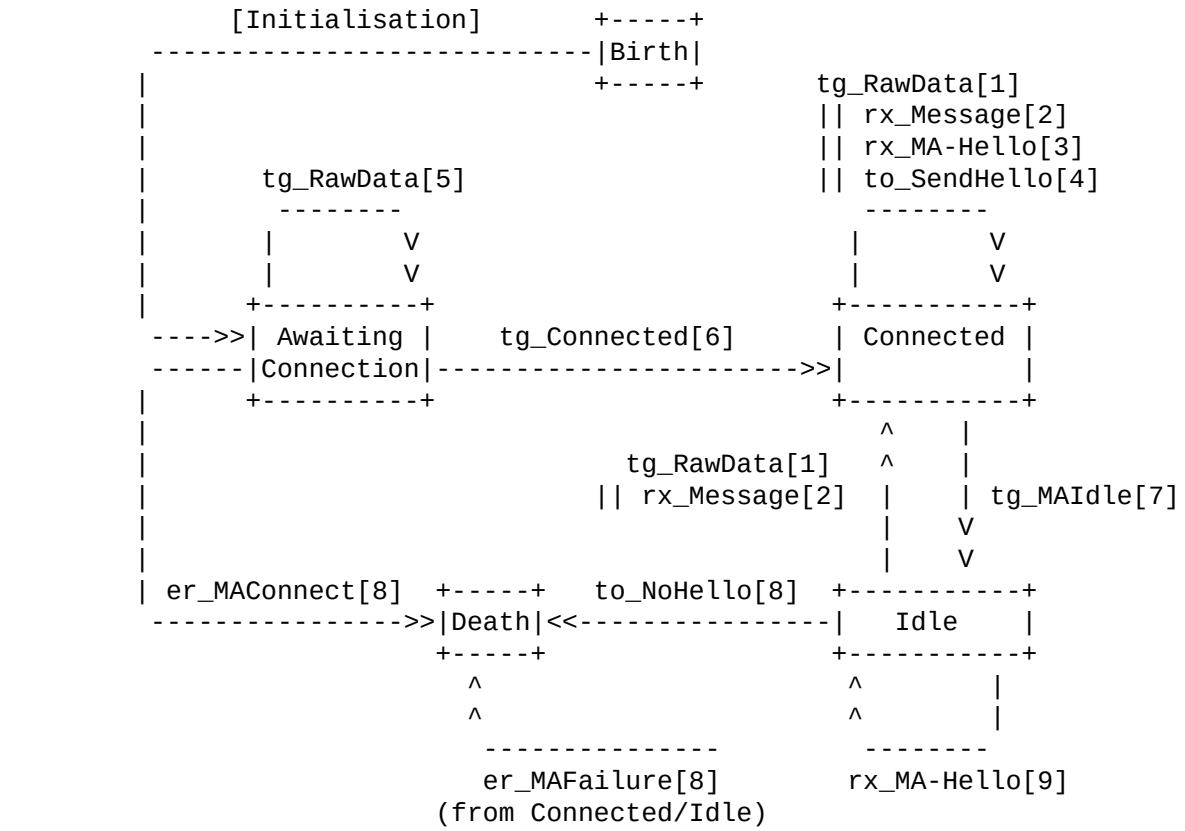


Figure 9: Messaging Association State Machine

The processing rules are as follows:

```
Rule 1:
pass message to transport layer
if the NoHello timer was running, stop it
(re)start SendHello

Rule 2:
pass message to Node-SM, or R-SM (for a Confirm),
   or Q-SM (for a Response)
if the NoHello timer was running, stop it

Rule 3:
if reply requested
  send MA-Hello
  restart SendHello timer

Rule 4:
send MA-Hello message
restart SendHello timer

Rule 5:
   queue message for later transmission

Rule 6:
pass outstanding queued messages to transport layer
stop any timers controlling connection establishment
start SendHello timer

Rule 7:
stop SendHello timer
start NoHello timer

Rule 8:
report failure to routing state machines and signalling applications
destroy self

Rule 9:
if reply requested
  send MA-Hello
restart NoHello timer
```

7. Additional Protocol Features

7.1. Route Changes and Local Repair

7.1.1. Introduction

When IP layer rerouting takes place in the network, GIST and signalling application state need to be updated for all flows whose paths have changed. The updates to signalling application state depend mainly on the signalling application: for example, if the path characteristics have changed, simply moving state from the old to the new path is not sufficient. Therefore, GIST cannot complete the path update processing by itself. Its responsibilities are to detect the route change, update its local routing state consistently, and inform interested signalling applications at affected nodes.

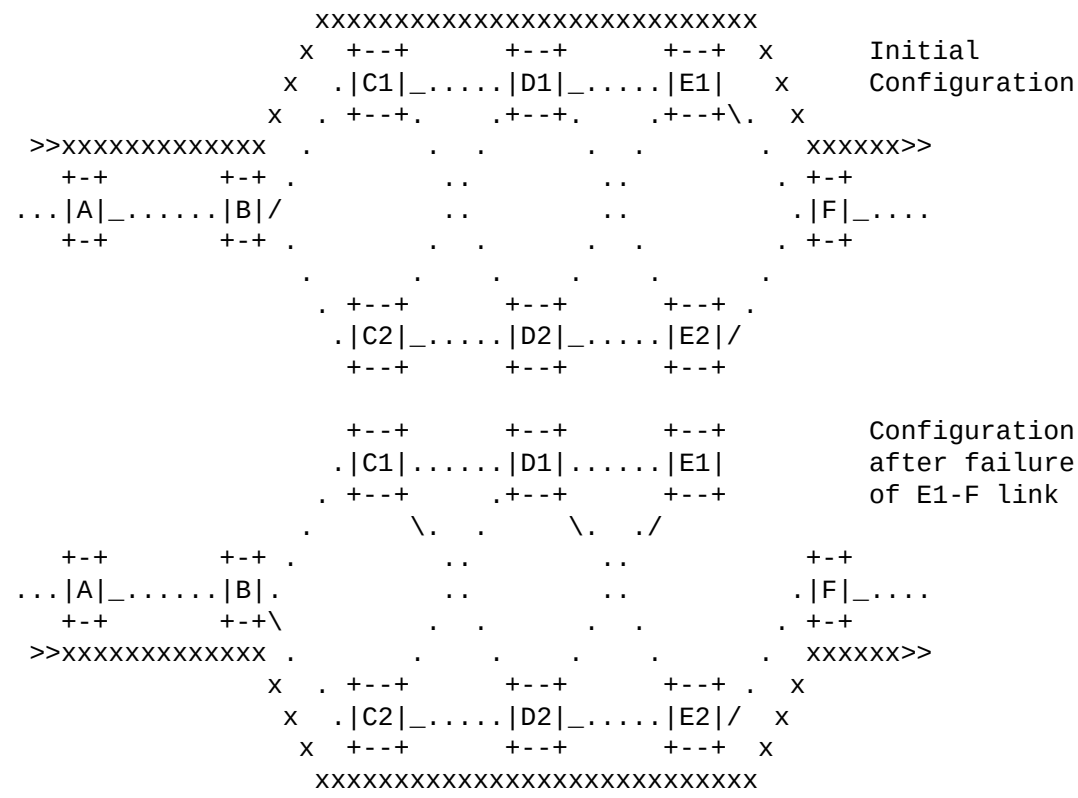


Figure 10: A Rerouting Event

Route change management is complicated by the distributed nature of the problem. Consider the rerouting event shown in Figure 10. 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.

The NSIS framework [29] makes the assumption that signalling applications are soft-state based and operate end to end. In this case, 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 signalling load, the duration of inconsistent state may be very long indeed. Therefore, GIST includes logic to exchange prompt notifications with signalling 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 signalling 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 in 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 local 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 IP 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 IP route computation and only works within a single area for OSPF [16] 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 IP layer 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 signalling application on a node may detect a change in behaviour of the flow, such as IP layer TTL change, arrival on a different interface, or loss of the flow altogether. The signalling application on the node is allowed to convey this information to the local GIST instance (Appendix B.6).

GIST Probing: According to the specification, each GIST node MUST periodically repeat the discovery (Query/Response) operation. Values for the probe frequency are discussed in Section 4.4.4. 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 between probes. The Querying node will discover the route change by a modification in the Network-Layer-Information in the Response. The Responding node can detect a change in the upstream peer similarly; further, if the Responding node can store the interface on which Queries arrive, it can detect if this interface changes even when the peer does not.

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 Response	Discovers changed NLI in Query

7.1.3. GIST Behaviour Supporting Rerouting

The basic GIST behaviour necessary to support rerouting can be modelled using a three-level classification of the validity of each item of current routing state. (In addition to current routing state, NSIS can maintain past routing state, described in Section 7.1.4 below.) This classification applies separately to the Querying and Responding nodes 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 state 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 IP 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 IP 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 signalling 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 signalling 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 nodes. 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 IP 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 signalling 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. Load Splitting and Route Flapping

The Q-mode encapsulation rules of Section 5.8 try to ensure that the Query messages discovering the path mimic the flow as accurately as possible. However, in environments where there is load balancing over multiple routes, and this is based on header fields differing between flow and Q-mode packets or done on a round-robin basis, the path discovered by the Query may vary from one handshake to the next even though the underlying network is stable. This will appear to GIST as a route flap; route flapping can also be caused by problems in the basic network connectivity or routing protocol operation. For example, a mobile node might be switching back and forth between two links, or might appear to have disappeared even though it is still attached to the network via a different route.

This specification does not define mechanisms for GIST to manage multiple parallel routes or an unstable route; instead, GIST MAY expose this to the NSLP, which can then manage it according to signalling application requirements. The algorithms already described always maintain the concept of the current route, i.e., the latest peer discovered for a particular flow. Instead, GIST allows the use of prior signalling paths for some period while the signalling applications still need them. Since NSLP peers are a single GIST hop apart, the necessary information to represent a path can be just an entry in the node's routing state table for that flow (more generally, anything that uniquely identifies the peer, such as the NLI, could be used). Rather than requiring GIST to maintain multiple generations of this information, it is provided to the signalling application in the same node in an opaque form for each message that is received from the peer. The signalling application can store it if necessary and provide it back to the GIST layer in case it needs to be used. Because this is a reference to information about the source of a prior signalling message, it is denoted 'SII-Handle' (for Source Identification Information) in the abstract API of Appendix B.

Note that GIST if possible SHOULD use the same SII-Handle for multiple sessions to the same peer, since this then allows signalling applications to aggregate some signalling, such as summary refreshes or bulk teardowns. Messages sent using the SII-Handle MUST bypass the routing state tables at the sender, and this MUST be indicated by setting the E-flag in the common header (Appendix A.1). Messages other than Data messages MUST NOT be sent in this way. At the receiver, GIST MUST NOT validate the MRI/SID/NSLPID against local

routing state and instead indicates the mode of reception to signalling applications through the API (Appendix B.2). Signalling 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.

GIST notifies signalling applications about route modifications as two types of event, additions and deletions. An addition is notified as a change of the current routing state according to the Bad/Tentative/Good classification above, while deletion is expressed as a statement that an SII-Handle no longer lies on the path. Both can be reported through the NetworkNotification API call (Appendix B.4). A minimal implementation MAY notify a route change as a single (add, delete) operation; however, a more sophisticated implementation MAY delay the delete notification, for example, if it knows that the old route continues to be used in parallel or that the true route is flapping between the two. It is then a matter of signalling application design whether to tear down state on the old path, leave it unchanged, or modify it in some signalling application specific way to reflect the fact that multiple paths are operating in parallel.

7.1.5. Signalling Application Operation

Signalling applications can use these functions as provided by GIST to carry out rapid local repair following rerouting events. The signalling application instances carry out the multi-hop aspects of the procedure, including crossover node detection, and tear-down/reinstallation of signalling 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 10 as an example.

1. The signalling 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 signalling application at E1 MAY begin local repair immediately, or MAY propagate a notification upstream to D1 that rerouting has occurred.

2. The signalling application at node D1 is notified of the route change, either by signalling 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 IP routing protocol, GIST will change the upstream/downstream routing state to Tentative/Bad automatically, and this will cause the signalling 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 signalling 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 signalling 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. For example, it could be an area border router and the route change is only intra-area. In this case, the signalling application and GIST will see that the upstream state is Good and can begin the local repair directly.

After a route deletion, a signalling application may wish to remove state at another node that 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 signalling load. Instead, signalling applications MAY use the SII-Handle described above to route explicit teardown messages.

7.2. NAT Traversal

GIST messages, for example, for the path-coupled MRM, must carry 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 signalling payloads are not translated consistently, the signalling 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.

There is a dual problem of whether the GIST peers on 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 signalling packet payloads. Existing generic NAT traversal techniques such as Session Traversal Utilities for NAT (STUN) [26] or Traversal Using Relays around NAT (TURN) [27] can operate only on the two addresses visible in the IP header. It is therefore intrinsically difficult to use these techniques to discover a consistent translation of the three or four interdependent addresses for the flow and signalling source and destination.

For legacy NATs and MRMs that carry addressing information, the base GIST specification is therefore limited to detecting the situation and triggering the appropriate error conditions to terminate the signalling path. (MRMs that do not contain addressing information could traverse such NATs safely, with some modifications to the GIST processing rules. Such modifications could be described in the documents defining such MRMs.) Legacy NAT handling is covered in Section 7.2.1 below. A more general solution can be constructed using GIST-awareness in the NATs themselves; this solution is outlined in Section 7.2.2 with processing rules in Section 7.2.3.

In all cases, GIST interaction with the NAT is determined by the way the NAT handles the Query/Response messages in the initial GIST handshake; these messages are UDP datagrams. Best current practice for NAT treatment of UDP traffic is defined in [38], and the legacy NAT handling defined in this specification is fully consistent with that document. The GIST-aware NAT traversal technique is equivalent to requiring an Application Layer Gateway in the NAT for a specific class of UDP transactions -- namely, those where the destination UDP port for the initial message is the GIST port (see Section 9).

7.2.1. Legacy NAT Handling

Legacy NAT detection during the GIST handshake depends on analysis of the IP header and S-flag in the GIST common header, and the NLI object included in the handshake messages. The message sequence proceeds differently depending on whether the Querying node is on the internal or external side of the NAT.

For the case of the Querying node on the internal side of the NAT, if the S-flag is not set in the Query (S=0), a legacy NAT cannot be detected. The receiver will generate a normal Response to the interface-address given in the NLI in the Query, but the interface-

address will not be routable and the Response will not be delivered. If retransmitted Queries keep S=0, this behaviour will persist until the Querying node times out. The signalling path will thus terminate at this point, not traversing the NAT.

The situation changes once S=1 in a Query; note the Q-mode encapsulation rules recommend that S=1 is used at least for some retransmissions (see Section 5.8). If S=1, the receiver MUST check the source address in the IP header against the interface-address in the NLI. A legacy NAT has been found if these addresses do not match. For MRMs that contain addressing information that needs translation, legacy NAT traversal is not possible. The receiver MUST return an "Object Type Error" message (Appendix A.4.4.9) with subcode 4 ("Untranslated Object") indicating the MRI as the object in question. The error message MUST be addressed to the source address from the IP header of the incoming message. The Responding node SHOULD use the destination IP address of the original datagram as the source address for IP header of the Response; this makes it more likely that the NAT will accept the incoming message, since it looks like a normal UDP/IP request/reply exchange. If this message is able to traverse back through the NAT, the Querying node will terminate the handshake immediately; otherwise, this reduces to the previous case of a lost Response and the Querying node will give up on reaching its retransmission limit.

When the Querying node is on the external side of the NAT, the Query will only traverse the NAT if some static configuration has been carried out on the NAT to forward GIST Q-mode traffic to a node on the internal network. Regardless of the S-flag in the Query, the Responding node cannot directly detect the presence of the NAT. It MUST send a normal Response with S=1 to an address derived from the Querying node's NLI that will traverse the NAT as normal UDP traffic. The Querying node MUST check the source address in the IP header with the NLI in the Response, and when it finds a mismatch it MUST terminate the handshake.

Note that in either of the error cases (internal or external Querying node), an alternative to terminating the handshake could be to invoke some legacy NAT traversal procedure. This specification does not define any such procedure, although one possible approach is described in [43]. Any such traversal procedure MUST be incorporated into GIST using the existing GIST extensibility capabilities. Note also that this detection process only functions with the handshake exchange; it cannot operate on simple Data messages, whether they are Q-mode or normally encapsulated. Nodes SHOULD NOT send Data messages outside a messaging association if they cannot ensure that they are operating in an environment free of legacy NATs.

7.2.2. GIST-Aware NAT Traversal

The most robust solution to the NAT traversal 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 the MRI, and not other higher layer identifiers. Provided this is done consistently with the data flow header translation, signalling messages can be valid each side of the boundary, without requiring the NAT to be signalling application aware. Note, however, that if the NAT does not understand the MRI, and the N-flag in the MRI is clear (see Appendix A.3.1), it should reject the message with an "Object Type Error" message (Appendix A.4.4.9) with subcode 4 ("Untranslated Object").

The basic concept is that GIST-aware NATs modify any signalling messages that have to be able to be interpreted without routing state being available; these messages are identified by the context-free flag C=1 in the common header, and include the Query in the GIST handshake. In addition, NATs have to modify the remaining handshake messages that set up routing state. When routing state is set up, GIST records how subsequent messages related to that routing state should be translated; if no routing state is being used for a message, GIST directly uses the modifications made by the NAT to translate it.

This specification defines an additional NAT traversal object that a NAT inserts into all Q-mode encapsulated messages with the context-free flag C=1, and which GIST echoes back in any replies, i.e., Response or Error messages. NATs apply GIST-specific processing only to Q-mode encapsulated messages with C=1, or D-mode messages carrying the NAT traversal object. All other GIST messages, either those in C-mode or those in D-mode with no NAT-Traversal object, should be treated as normal data traffic by the NAT, i.e., with IP and transport layer header translation but no GIST-specific processing. Note that the distinction between Q-mode and D-mode encapsulation may not be observable to the NAT, which is why the setting of the C-flag or presence of the NAT traversal object is used as interception criteria. The NAT decisions are based purely on the value of the C-flag and the presence of the NAT traversal object, not on the message type.

The NAT-Traversal object (Appendix A.3.9), carries the translation between the MRIs that are appropriate for the internal and external sides of the NAT. 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 assist the NAT in back-translating D-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 signalling application peers on each side of the NAT are no longer directly compatible. In particular, they use different MRI values to refer to the same flow. However, messages after the Query/Response (the initial Confirm and subsequent Data messages) need to use a common MRI, since the NAT does not rewrite these, and this is chosen to be the MRI of the Querying node. It is the responsibility of the Responding node to translate between the two MRIs on inbound and outbound messages, which is why the unmodified MRI is propagated in the NAT-Traversal object.

7.2.3. Message Processing Rules

This specification normatively defines the behaviour of a GIST node receiving a message containing a NAT-Traversal object. However, it does not define normative behaviour for a NAT translating GIST messages, since much of this will depend on NAT implementation and policy about allocating bindings. In addition, it is not necessary for a GIST implementation itself. Therefore, those aspects of the following description are informative; full details of NAT behaviour for handling GIST messages can be found in [44].

A possible set of operations for a NAT to process a message with C=1 is as follows. Note that for a Data message, only a subset of the operations is applicable.

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 signalling 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 signalling bindings.

5. Add a NAT-Traversal object, listing the objects that 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 Q-mode 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, and this is also true for any other error message generated as a reply. 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, because the message is either the beginning of a handshake for a new flow or a refresh for existing state. In both cases, the GIST node MUST create the Response 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 D-mode messages containing such echoed NAT-Traversal objects. The NAT processing is a subset of the processing for the C=1 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 signalling traffic.

4. Forward the message.

A GIST node receiving such a message (Response or Error) 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 given GIST-specific processing by the NAT. Thus, a Responding node that has delayed state installation until receiving the Confirm only has available the untranslated MRI describing the flow, and the untranslated NLI as peer routing state. This would prevent the correct interpretation of the signalling 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 that 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 signalling is needed for the tunnel itself, this has to be initiated as a separate signalling session by one of the tunnel endpoints -- that is, the tunnel counts as a new flow. Because the relationship between signalling for the microflow and signalling for the tunnel as a whole will depend on the signalling application in question, it is a signalling application responsibility to be aware of the fact that tunnelling is taking place and to carry out additional signalling if necessary; in other words, at least one tunnel endpoint must be signalling application aware.

In some cases, it is the tunnel exit point (i.e., the node where tunnelled data and downstream signalling packets leave the tunnel) that will wish to carry out the tunnel signalling, 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 signalling data from the tunnel entry point; this functionality is the equivalent to the RSVP SESSION_ASSOC object of [18]. In the NSIS protocol suite, these bindings are managed by the signalling 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 [35].) In mixed environments, GIST MUST use the same IP version for Q-mode encapsulated messages as given by the MRI of the flow for which it is signalling, and SHOULD do so for other signalling also (see Section 5.2.2). Messages with mismatching versions MUST be rejected with an "MRI Validation Failure" error message (Appendix A.4.4.12) with subcode 1 ("IP Version Mismatch"). The IP version used in D-mode is closely tied to the IP version used by the data flow, so it is intrinsically impossible for an IPv4-only or IPv6-only GIST node to support signalling for flows using the other IP version. Hosts that are dual stack for applications and routers that are dual stack for forwarding need GIST implementations that 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 [7].) 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 internal and external 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 that 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 [7]. The same analysis applies to NAT-PT, although this technique is no longer proposed as a general purpose transition mechanism [40].

Tunnelling: (Applicable to 6to4 [19].) 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. For GIST, the treatment should be similar to any other IP tunnelling mechanism, as described in Section 7.3. In particular, the end-to-end flow signalling will pass transparently through the tunnel, and signalling 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, [20] is based on using an anycast address as the destination tunnel endpoint. GIST MAY use anycast destination addresses in the Q-mode encapsulation of D-mode messages if necessary, but MUST NOT use them in the Network-Layer-Information addressing field; unicast addresses MUST be used instead. Note that the addresses from the IP header are not used by GIST in matching requests and replies, so there is no requirement to use anycast source addresses.

8. Security Considerations

The security requirement for GIST is to protect the signalling plane against identified security threats. For the signalling problem as a whole, these threats have been outlined in [30]; the NSIS framework [29] assigns a subset of the responsibilities to the NTLP. The main issues to be handled can be summarised as:

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

Routing State Integrity Protection: It is important that signalling messages are delivered to the correct nodes, and nowhere else. Here, 'correct' is defined as 'the appropriate nodes for the signalling given the Message-Routing-Information'. In the case where the MRI is based on the flow identification for path-coupled signalling, '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 to ensure that signalling and data routing are consistent with each other. GIST uses internal state to decide how to route signalling 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 signalling operations (e.g., allocating resources). This is assumed to be done in each signalling 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 signalling 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 that are unknown ahead of time, although the content visible even at the GIST level gives significant opportunities for traffic analysis. Signalling 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 signalling 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 are possible using the protocols associated with the messaging association being secured. TLS incorporates this functionality directly. GIST nodes rely on the messaging association protocol 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 signalling 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 signalling application dependent, for example, whether successful authentication could be made dependent on presenting credentials related to a particular signalling role (e.g., signalling 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, signalling messages may be misdirected.

In the case where the MRM 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 IP routing mechanisms to ensure that the signalling peers are the appropriate ones for any given flow. An overview of security issues and techniques in this context is provided in [37].

In one direction, peer identification is installed and refreshed only on receiving a Response (compare Figure 5). This MUST echo the cookie from a previous Query, which will have been sent along the flow path with the Q-mode encapsulation, 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 Query containing addressing information for the signalling 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, four strategies are used:

Filtering: The receiving node MAY reject signalling messages that claim to be for flows with flow source addresses that 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 D-mode signalling packets. If they are not, it is likely that at least one of the signalling or flow packets is being spoofed.

Return routability checking: The receiving node MAY refuse to install upstream state until it has completed a Confirm handshake with the peer. This echoes the Responder-Cookie of the Response, and discourages nodes from using forged source addresses. This also plays a role in denial-of-service prevention; see below.

Authorisation: A stronger approach is to carry out a peer authorisation check (see Section 4.4.2) as part of messaging association setup. The ideal situation is that the receiving node can determine the correct upstream node address from routing table analysis or knowledge of local topology constraints, and then verify from the authorised peer database (APD) that the peer has this IP address. This is only technically feasible in a limited set of deployment environments. The APD can also be used to list the subsets of nodes that are feasible peers for particular source or destination subnets, or to blacklist nodes that have previously originated attacks or exist in untrustworthy networks, which provide weaker levels of authorisation checking.

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 GIST 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 signalling messages for which state has already been legitimately established.

8.4. Denial-of-Service Prevention and Overload Protection

GIST is designed so that in general each Query only generates at most one Response that is at most only slightly larger than the Query, 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 signalling 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 query an authentication or authorisation infrastructure, 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 Query, but MAY wait for a Confirm, 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 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 [39] and other modern protocols.

There is a potential overload condition if a node is flooded with Query or Confirm messages. One option is for the node to bypass these messages altogether as described in Section 4.3.2, effectively falling back to being a non-NSIS node. If this is not possible, a node MAY still choose to limit the rate at which it processes Query messages and discard the excess, although it SHOULD first adapt its policy to one of sending Responses statelessly if it is not already doing so. A conformant GIST node will automatically decrease the load by retransmitting Queries with an exponential backoff. A non-conformant node (launching a DoS attack) can generate uncorrelated Queries at an arbitrary rate, which makes it hard to apply rate-limiting without also affecting genuine handshake attempts. However,

if Confirm messages are requested, the cookie binds the message to a Querying node address that has been validated by a return routability check and rate-limits can be applied per source.

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 implementations of 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. It is possible for an attacking node to execute these protocols legally to set up large numbers of associations that were never used, and Responding node implementations MAY use rate-limiting or other techniques to control the load in such cases.

Signalling 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). Signalling 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. Signalling applications can also manage overload by invoking flow control, as described in Section 4.1.1.

8.5. Requirements on Cookie Mechanisms

The requirements on the Query-Cookie can be summarised as follows:

Liveness: The cookie must be live; that is, it must change from one handshake to the next. This prevents replay attacks.

Unpredictability: The cookie must not be guessable, e.g., from a sequence or timestamp. This prevents direct forgery after capturing a set of earlier 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 that already exists. This allows to discard responses that have been randomly generated by an adversary, or to discard responses to queries that were generated with forged source addresses or an incorrect address in the included NLI object.

Uniqueness: Each handshake must have a unique cookie since the cookie is used to match responses within a handshake, e.g., when multiple messaging associations are multiplexed over the same transport connection.

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

Liveness: The cookie must be live as above, to prevent replay attacks.

Creation simplicity: The cookie must be lightweight to generate in order 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 for which it was generated.

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 Peer-Identity and Interface-Address given in the NLI of the Query, and the MRI/NSLPID for the messaging.

It can also include the interface on which the Query was received for use later in route change detection (Section 7.1.2). Since a Q-mode encapsulated message is the one that will best follow the data path, subsequent changes in this arrival interface indicate route changes between the peers.

A suitable implementation for the Q-Cookie is a cryptographically strong random number that is unique for this routing state machine handshake. A node MUST implement this or an equivalently strong mechanism. Guidance on random number generation can be found in [31].

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

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

A node MUST 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. In the case of GIST-aware NAT traversal with delayed state installation, it is necessary to carry additional data in the cookie; appropriate constructions are described in [44].

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 that 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" message (Appendix A.4.4.10) with subcode 4 ("Invalid Cookie") to the sender and SHOULD log an error condition locally, as well as dropping the message. However, sending the error in general makes a node a source of backscatter. Therefore, this MUST 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 [25]):

1. The Response does not depend on the Stack-Proposal in the Query (see Section 5.7.1). Therefore, tampering with the Query 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. 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, if the Querying node does not apply a security policy to the messaging association protocols to be created that ensures at least this minimal level of protection is met, it remains open to the threat that a downgrade has occurred. Applying such a policy ensures capability discovery process will result in the setup of a messaging association with the correct security properties 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, vulnerabilities from particular limited modes of TLS usage, and implementation-related weaknesses.

An on-path attacker who can intercept the initial Query can do most things it wants to the subsequent signalling. 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 signalling 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 signalling. Therefore, the additional threat induced by the signalling weakness seems tolerable.

At the NSLP level, there is a concern about transitivity of trust of correctness of routing along the signalling 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 by depending on the security protection provided by GIST. 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 signalling messages with payloads (e.g., authorisation tokens) that are valuable to nodes beyond the adjacent hop, it is up to the NSLP to ensure that the appropriate chain of trust exists. This could be achieved using higher layer security protection such as Cryptographic Message Syntax (CMS) [28].

There is a further residual attack by a node that is not on the path of the Query, 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 peer. In principle, this attack could be prevented by including an additional cryptographic object in the Response that ties the Response to the initial Query and the routing state and can be verified by the Querying node.

GIST depends on TLS for peer node authentication, and subsequent channel security. The analysis in [30] indicates the threats that arise when the peer node authentication is incomplete -- specifically, when unilateral authentication is performed (one node authenticates the other, but not vice versa). In this specification, mutual authentication can be supported either by certificate exchange or the use of pre-shared keys (see Section 5.7.3); if some other TLS authentication mechanism is negotiated, its properties would have to be analysed to determine acceptability for use with GIST. If mutual authentication is performed, the requirements for NTLP security are met.

However, in the case of certificate exchange, this specification allows the possibility that only a server certificate is provided, which means that the Querying node authenticates the Responding node but not vice versa. Accepting such unilateral authentication allows for partial security in environments where client certificates are not widespread, and is better than no security at all; however, it does expose the Responding node to certain threats described in Section 3.1 of [30]. For example, the Responding node cannot verify whether there is a man-in-the-middle between it and the Querying node, which could be manipulating the signalling messages, and it cannot verify the identity of the Querying node if it requests authorisation of resources. Note that in the case of host-network signalling, the Responding node could be either the host or the first

hop router, depending on the signalling direction. Because of these vulnerabilities, modes or deployments of TLS which do not provide mutual authentication can be considered as at best transitional stages rather than providing a robust security solution.

Certain security aspects of GIST operation depend on signalling application behaviour: a poorly implemented or compromised NSLP could degrade GIST security. However, the degradation would only affect GIST handling of the NSLP's own signalling traffic or overall resource usage at the node where the weakness occurred, and implementation weakness or compromise could have just as great an effect within the NSLP itself. GIST depends on NSLPs to choose SIDs appropriately (Section 4.1.3). If NSLPs choose non-random SIDs, this makes off-path attacks based on SID guessing easier to carry out. NSLPs can also leak information in structured SIDs, but they could leak similar information in the NSLP payload data anyway.

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. Note that the guidelines on the technical criteria to be followed in evaluating requests for new codepoint assignments are covered normatively in a separate document that considers the NSIS protocol suite in a unified way. That document discusses the general issue of NSIS extensibility, as well as the technical criteria for particular registries; see [12] for further details.

The registry definitions that follow leave large blocks of codes marked "Reserved". This is to allow a future revision of this specification or another Experimental document to modify the relative space given to different allocation policies, without having to change the initial rules retrospectively if they turn out to have been inappropriate, e.g., if the space for one particular policy is exhausted too quickly.

The allocation policies used in this section follow the guidance given in [4]. In addition, for a number of the GIST registries, this specification also defines private/experimental ranges as discussed in [9]. Note that the only environment in which these codepoints can validly be used is a closed one in which the experimenter knows all the experiments in progress.

This specification allocates the following codepoints in existing registries:

Well-known UDP port 270 as the destination port for Q-mode encapsulated GIST messages (Section 5.3).

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

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

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

Every other NSLPID that uses an MRM that requires RAO usage MUST be associated with a specific RAO value; multiple NSLPIDs MAY be associated with the same RAO value. RAO value assignments require a specification of the processing associated with messages that carry the value. NSLP specifications MUST normatively depend on this document for the processing, specifically Sections 4.3.1, 4.3.4 and 5.3.2. The NSLPID is a 16-bit integer, and the registration procedure is IESG Aproval. Further values are as follows:

1-32703: Unassigned

32704-32767: Private/Experimental Use

32768-65536: Reserved

GIST Message Type: The GIST common header (Appendix A.1) contains a 7-bit message type field. The following values are allocated by this specification:

MType	Message
0	Query
1	Response
2	Confirm
3	Data
4	Error
5	MA-Hello

Registration procedures are as follows:

0-31: IETF Review

32-55: Expert Review

Further values are as follows:

6-55: Unassigned

56-63: Private/Experimental Use

64-127: 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
10	Hello ID

Registration procedures are as follows:

0-1023: IETF Review

1024-1999: Specification Required

Further values are as follows:

11-1999: Unassigned

2000-2047: Private/Experimental Use

2048-4095: Reserved

When a new object type is allocated according to one of the procedures, the specification MUST provide the object format and define the setting of the extensibility bits (A/B; see Appendix A.2.1).

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

MRM-ID	Message Routing Method
0	Path-Coupled MRM
1	Loose-End MRM

Registration procedures are as follows:

0-63: IETF Review

64-119: Specification Required

Further values are as follows:

2-119: Unassigned

120-127: Private/Experimental Use

128-255: Reserved

When a new MRM is allocated according to one of the registration procedures, the specification MUST provide the information described in Section 3.3.

MA-Protocol-IDs: Each protocol that can be used in a messaging association is identified by a 1-byte MA-Protocol-ID (Section 5.7). 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. 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	Protocol
0	Reserved
1	TCP opened in the forwards direction
2	TLS initiated in the forwards direction

Registration procedures are as follows:

0-63: IETF Review

64-119: Expert Review

Further values are as follows:

3-119: Unassigned

120-127: Private/Experimental Use

128-255: Reserved

When a new MA-Protocol-ID is allocated according to one of the registration procedures, a specification document will be required. This MUST define the format for the MA-protocol-options field (if any) in the Stack-Configuration-Data object that is needed to define its configuration. If a protocol is to be used for reliable message transfer, it MUST be described how delivery errors are to be detected by GIST. Extensions to include new channel security protocols MUST include a description of how to integrate the functionality described in Section 3.9 with the rest of GIST operation. If the new MA-Protocol-ID can be used in conjunction with existing ones (for example, a new transport protocol that could be used with Transport Layer Security), the specification MUST define the interaction between the two.

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 0-5 (code 1), 0-5 (code 9), 0-5 (code 10), and 0-2 (code 12). Additional codes and subcodes are allocated on a first-come, first-served basis. When a new code/subcode combination is allocated, the following information MUST be provided:

Error case: textual name of error

Error class: from the categories given in Appendix A.4.3

Error code: allocated by IANA, if a new code is required

Error subcode: subcode point, also allocated by IANA

Additional information: what Additional Information fields are mandatory to include in the error message, from Appendix A.4.2

Additional Information Types: An Error Object (Appendix A.4.1) may contain Additional Information fields. Each possible field type is identified by a 16-bit AI-Type. AI-Types 1-4 are defined in Appendix A.4.2; additional AI-Types are allocated on a first-come, first-served basis.

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, Vitor Bernado, Roland Bless, Bob Braden, Marcus Brunner, Benoit Campedel, Yoshiko Chong, Luis Cordeiro, Elwyn Davies, Michel Diaz, Christian Dickmann, Pasi Eronen, Alan Ford, Xiaoming Fu, Bo Gao, Ruediger Geib, Eleanor Hepworth, Thomas Herzog, Cheng Hong, Teemu Huovila, Jia Jia, Cornelia Kappler, Georgios Karagiannis, Ruud Klaver, Max Laier, Chris Lang, Lauri Liuhto, John Loughney, Allison Mankin, Jukka Manner, Pete McCann, Andrew McDonald, Mac McTiffin, Glenn Morrow, Dave Oran, Andreas Pashalidis, Henning Peters, Tom Phelan, Akbar Rahman, Takako Sanda, Charles Shen, Melinda Shore, Martin Stiemerling, Martijn Swanink, Mike Thomas, Hannes Tschofenig, Sven van den Bosch, Nuutti Varis, 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, RFC 3588. 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 and the name matching algorithm for TLS. 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. Magnus Westerlund carried out a detailed AD review that identified a number of issues and led to significant clarifications, which was followed by an even more detailed IESG review, with comments from Jari Arkko, Ross Callon, Brian Carpenter, Lisa Dusseault, Lars Eggert, Ted Hardie, Sam Hartman, Russ Housley, Cullen

Jennings, and Tim Polk, and a very detailed analysis by Adrian Farrel from the Routing Area directorate; Suresh Krishnan carried out a detailed review for the Gen-ART.

11. References

11.1. Normative References

[1] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[2] Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

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

[4] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[5] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[6] 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.

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

[8] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[9] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

[10] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[11] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[12] Manner, J., Bless, R., Loughney, J., and E. Davies, "Using and Extending the NSIS Protocol Family", RFC 5978, October 2010.

11.2. Informative References

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

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

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

[16] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.

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

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

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

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

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

[22] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[23] Jamoussi, B., Andersson, L., Callon, R., Dantu, R., Wu, L., Doolan, P., Worster, T., Feldman, N., Fredette, A., Girish, M., Gray, E., Heinanen, J., Kilty, T., and A. Malis, "Constraint-Based LSP Setup using LDP", RFC 3212, January 2002.

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

[25] Arkko, J., Torvinen, V., Camarillo, G., Niemi, A., and T. Haukka, "Security Mechanism Agreement for the Session Initiation Protocol (SIP)", RFC 3329, January 2003.

[26] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[27] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[28] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

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

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

[31] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

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

[33] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[34] Stiemerling, M., Tschofenig, H., Aoun, C., and E. Davies, "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", Work in Progress, April 2010.

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

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

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

[38] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[39] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[40] Aoun, C. and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", RFC 4966, July 2007.

[41] Gill, V., Heasley, J., Meyer, D., Savola, P., and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)", RFC 5082, October 2007.

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

[43] Pashalidis, A. and H. Tschofenig, "GIST Legacy NAT Traversal", Work in Progress, July 2007.

[44] Pashalidis, A. and H. Tschofenig, "GIST NAT Traversal", Work in Progress, July 2007.

[45] Tsenov, T., Tschofenig, H., Fu, X., Aoun, C., and E. Davies, "GIST State Machine", Work in Progress, April 2010.

[46] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", Work in Progress, May 2010.

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. The whole of this appendix is normative.

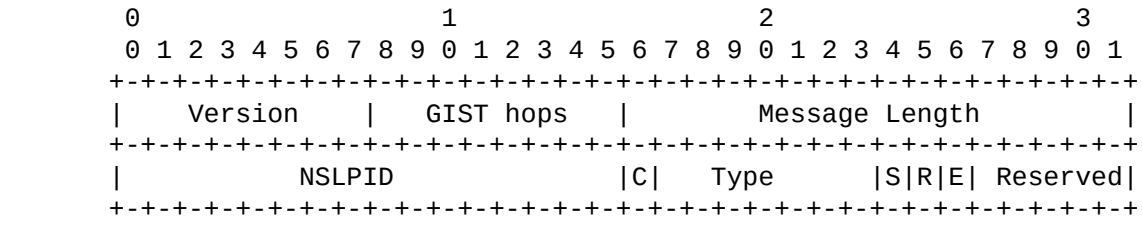
Each GIST message consists of a header and a sequence of objects. The GIST header has a specific format, described in more detail in Appendix A.1 below. An NSLP message is one object within a GIST message. Note that GIST itself provides the NSLP message length information and signalling application identification. 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. Any part of the object used for padding or defined as reserved (marked 'Reserved' or 'Rsv' or, in the case of individual bits, 'r' in the diagrams below) MUST be set to 0 on transmission and MUST be ignored on reception.

The objects are encoded using big endian (network byte order).

A.1. The GIST Common Header

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



Version (8 bits): The GIST protocol version number. This specification defines version number 1.

GIST hops (8 bits): A hop count for the number of GIST-aware nodes this message can still be processed by (including the destination).

Message Length (16 bits): The total number of 32-bit words in the message after the common header itself.

NSLPID (16 bits): IANA-assigned identifier of the signalling application to which the message refers.

C-flag: C=1 if the message has to be able to be interpreted in the absence of routing state (Section 5.2.1).

Type (7 bits): The GIST message type (Query, Response, etc.).

S-flag: S=1 if the IP source address is the same as the signalling source address, S=0 if it is different.

R-flag: R=1 if a reply to this message is explicitly requested.

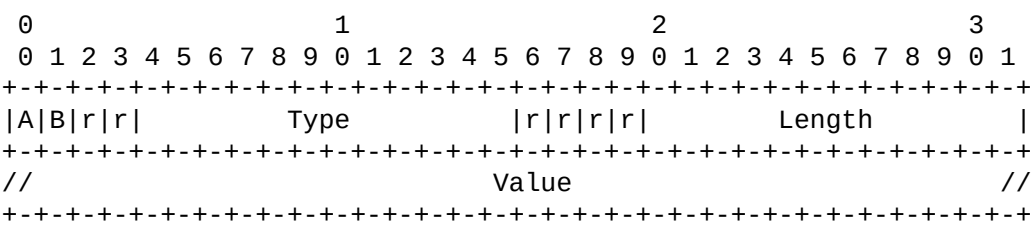
E-flag: E=1 if the message was explicitly routed (Section 7.1.5).

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, since these always elicit a Response, and never in Confirm, Data, or Error messages. It MAY be set in an MA-Hello; if set, another MA-Hello MUST be sent in reply. It MAY be set in a Response, but MUST be set if the Response contains a Responder-Cookie; if set, a Confirm MUST be sent in reply. The E-flag MUST NOT be set unless the message type is a Data message.

Parsing failures may be caused by unknown Version or Type values; inconsistent setting of the C-flag, R-flag, or E-flag; 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.



A/B flags: The bits marked 'A' and 'B' are extensibility flags, which are defined in Appendix A.2.1 below; the remaining bits marked 'r' are reserved.

Type (12 bits): An IANA-assigned identifier for the type of object.

Length (12 bits): 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 (variable): Value is (therefore) a whole number of 32-bit words. If there is any padding required, the length and location are be defined by the object-specific format information; objects that contain variable-length (e.g., string) types may need to include additional length subfields to do so.

A.2.1. Object Extensibility

The leading 2 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 objects 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. If a message is received containing an object with AB=11, it MUST be rejected with an "Object Type Error" message (Appendix A.4.4.9) with subcode 5 ("Invalid Extensibility Flags").

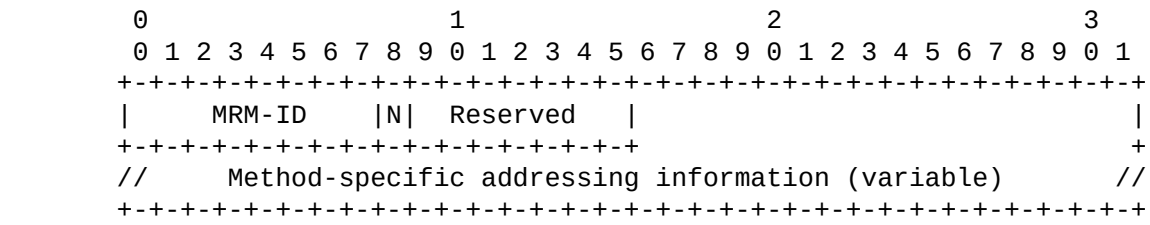
These extensibility rules define only the processing within the GIST layer. There is no requirement on GIST implementations to support an extensible service interface to signalling applications, so unrecognised objects with AB=01 or AB=10 do not need to be indicated to NSLPs.

A.3. GIST TLV Objects

A.3.1. Message-Routing-Information (MRI)

Type: Message-Routing-Information

Length: Variable (depends on MRM)



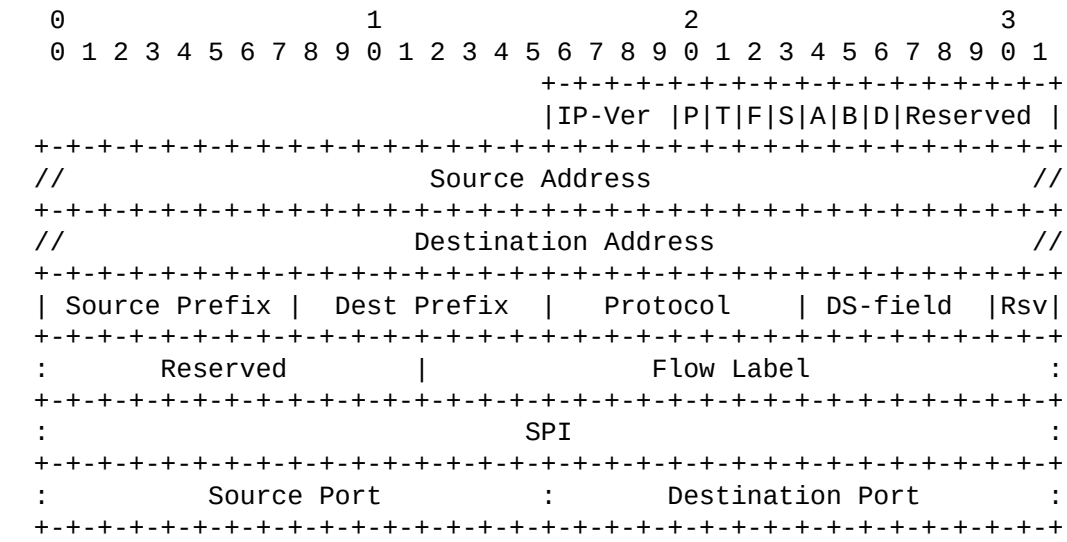
MRM-ID (8 bits): An IANA-assigned identifier for the message routing method.

N-flag: If set (N=1), this means that NATs do not need to translate this MRM; if clear (N=0), it means that the method-specific information contains network or transport layer information that a NAT must process.

The remainder of the object contains method-specific addressing information, which is described below.

A.3.1.1. Path-Coupled MRM

In the case of basic path-coupled routing, the addressing information takes the following format. The N-flag has a value of 0 for this MRM.



IP-Ver (4 bits): The IP version number, 4 or 6.

Source/Destination address (variable): The source and destination addresses are always present and of the same type; their length depends on the value in the IP-Ver field.

Source/Dest Prefix (each 8 bits): The length of the mask to be applied to the source and destination addresses for address wildcarding. In the normal case where the MRI refers only to traffic between specific host addresses, the Source/Dest Prefix values would both be 32 or 128 for IPv4 and IPv6, respectively.

P-flag: P=1 means that the Protocol field is significant.

Protocol (8 bits): The IP protocol number. This MUST be ignored if P=0. 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.

T-flag: T=1 means that the Diffserv field (DS-field) is significant.

DS-field (6 bits): The Diffserv field. See [6] and [24].

F-flag: F=1 means that flow label is present and is significant. F MUST NOT be set if IP-Ver is not 6.

Flow Label (20 bits): The flow label; only present if F=1. If F=0, the entire 32-bit word containing the Flow Label is absent.

S-flag: S=1 means that the SPI field is present and is significant. The S-flag MUST be 0 if the P-flag is 0.

SPI field (32 bits): The SPI field; see [36]. If S=0, the entire 32-bit word containing the SPI is absent.

A/B flags: These can only be set if P=1. If either is set, the port fields are also present. The A flag indicates the presence of a source port, the B flag that of a destination port. If P=0, the A/B flags MUST both be zero and the word containing the port numbers is absent.

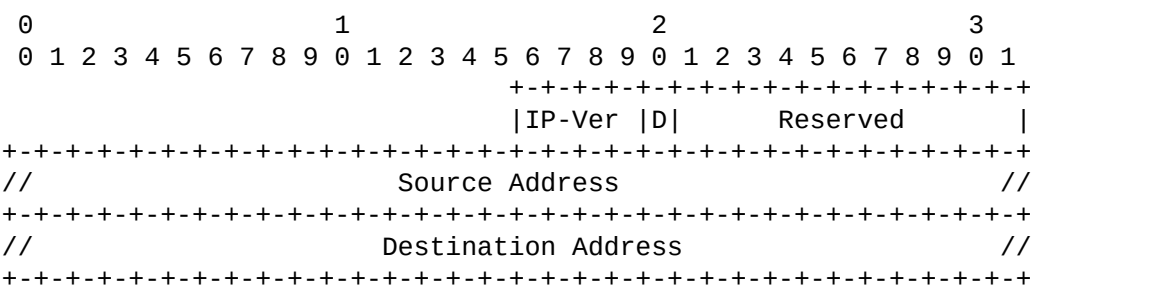
Source/Destination Port (each 16 bits): If either of A (source), B (destination) is set, the word containing the port numbers is included in the object. However, the contents of each field is only significant if the corresponding flag is set; otherwise, the contents of the field is regarded as padding, 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.

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

The MRI format defines a number of constraints on the allowed combinations of flags and fields in the object. If these constraints are violated, this constitutes a parse error, and an "Object Value Error" message (Appendix A.4.4.10) with subcode 2 ("Invalid Flag-Field Combination") MUST be returned.

A.3.1.2. Loose-End MRM

In the case of the loose-end MRM, the addressing information takes the following format. The N-flag has a value of 0 for this MRM.



IP-Ver (4 bits): The IP version number, 4 or 6.

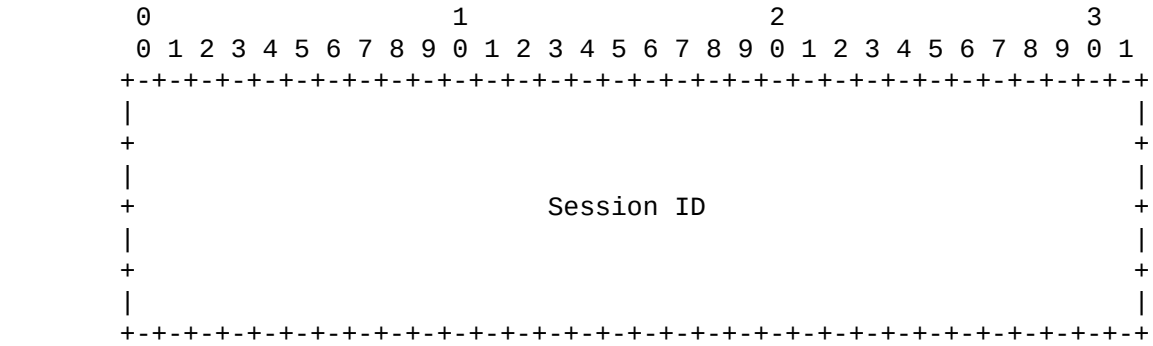
Source/Destination address (variable): The source and destination addresses are always present and of the same type; their length depends on the value in the IP-Ver field.

D-flag: The Direction flag has the following meaning: the value 0 means 'towards the edge of the network', and the value 1 means 'from the edge of the network'. Note that for Q-mode messages, the only valid value is D=0 (see Section 5.8.2).

A.3.2. Session Identifier

Type: Session-Identifier

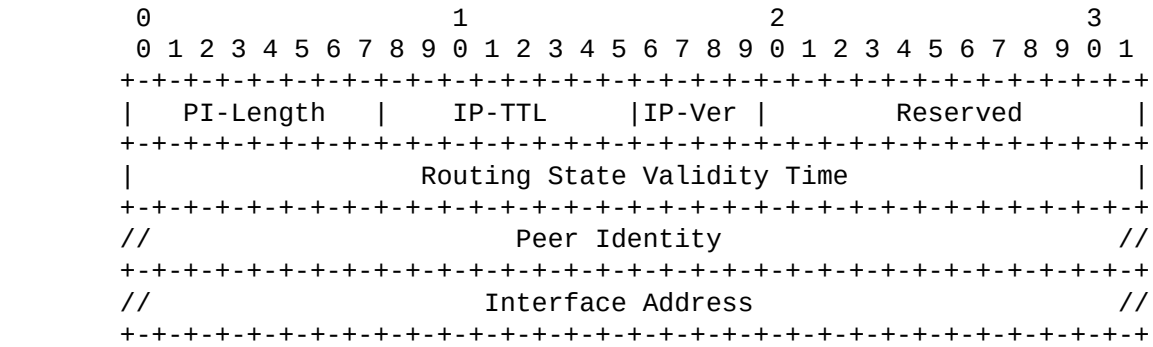
Length: Fixed (4 32-bit words)



A.3.3. Network-Layer-Information (NLI)

Type: Network-Layer-Information

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



PI-Length (8 bits): The byte length of the Peer Identity field.

Peer Identity (variable): The Peer Identity field. Note that the Peer-Identity field itself is padded to a whole number of words.

IP-TTL (8 bits): Initial or reported IP layer TTL.

IP-Ver (4 bits): The IP version for the Interface Address field.

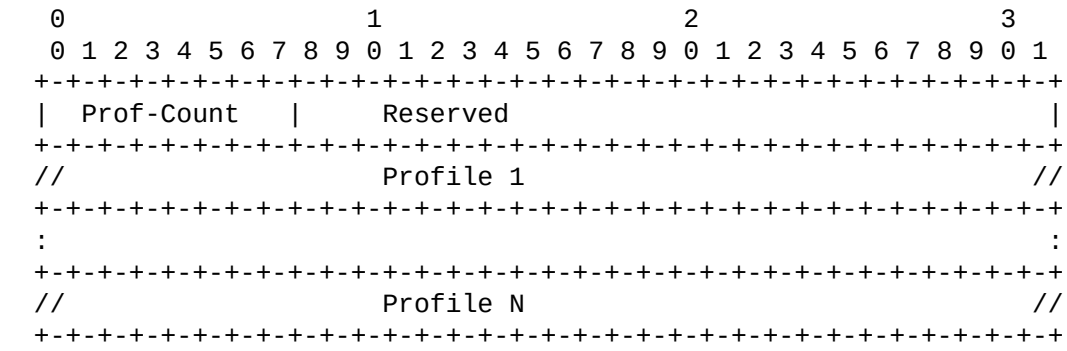
Interface Address (variable): The IP address allocated to the interface, matching the IP-Ver field.

Routing State Validity Time (32 bits): The time for which the routing state for this flow can be considered correct without a refresh. Given in milliseconds. The value 0 (zero) is reserved and MUST NOT be used.

A.3.4. Stack-Proposal

Type: Stack-Proposal

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



Prof-Count (8 bits): The number of profiles listed. 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. MUST be > 0.
- This is followed by a sequence of 1-byte MA-Protocol-IDs as described in Section 5.7.

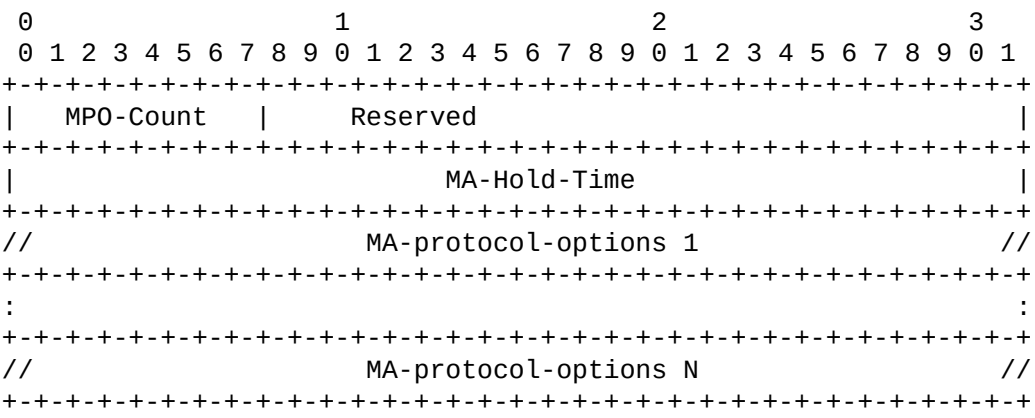
o The profile is padded to a word boundary with 0, 1, 2, or 3 zero bytes. These bytes MUST be ignored at the receiver.

If there are no profiles (Prof-Count=0), then an "Object Value Error" message (Appendix A.4.4.10) with subcode 1 ("Value Not Supported") MUST be returned; if a particular profile is empty (the leading byte of the profile is zero), then subcode 3 ("Empty List") MUST be used. In both cases, the message MUST be dropped.

A.3.5. Stack-Configuration-Data

Type: Stack-Configuration-Data

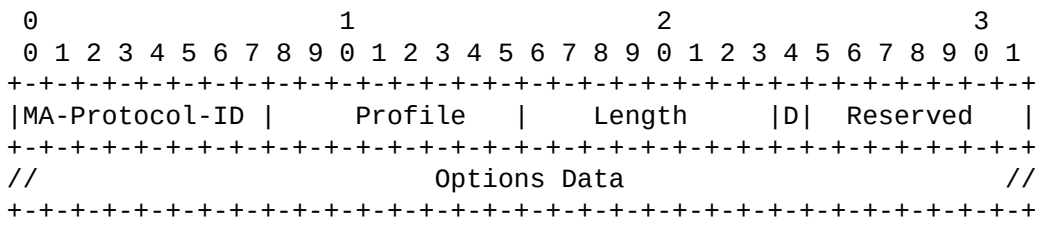
Length: Variable (depends on number of protocols and size of each MA-protocol-options field)



MPO-Count (8 bits): The number of MA-protocol-options fields present (these contain their own length information). The MPO-Count MAY be zero, but this will only be the case if none of the MA-protocols referred to in the Stack-Proposal require option data.

MA-Hold-Time (32 bits): The time for which the messaging association will be held open without traffic or a hello message. Note that this value is given in milliseconds, so the default time of 30 seconds (Section 4.4.5) corresponds to a value of 30000. The value 0 (zero) is reserved and MUST NOT be used.

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



MA-Protocol-ID (8 bits): Protocol identifier as described in Section 5.7.

Profile (8 bits): 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 (8 bits): The byte length of MA-protocol-options field that follows. This will be zero-padded up to the next word boundary.

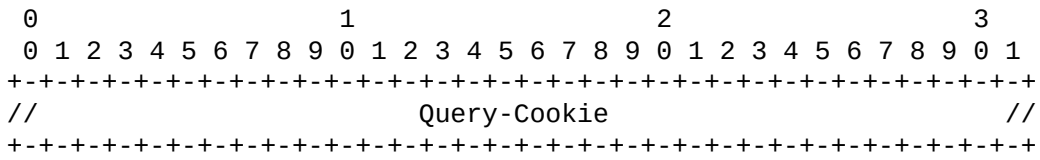
D-flag: If set (D=1), this protocol MUST NOT be used for a messaging association.

Options Data (variable): Any options data for this protocol. Note that the format of the options data might differ depending on whether the field is in a Query or Response.

A.3.6. Query-Cookie

Type: Query-Cookie

Length: Variable (selected by Querying node)

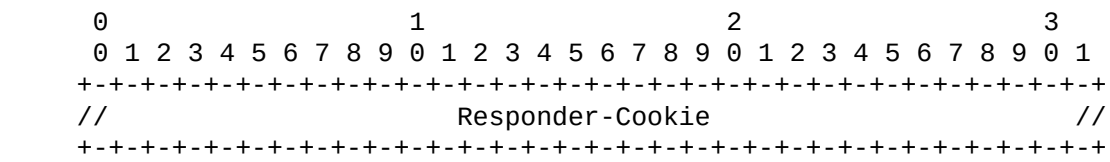


The content is defined by the implementation. See Section 8.5 for further discussion.

A.3.7. Responder-Cookie

Type: Responder-Cookie

Length: Variable (selected by Responding node)

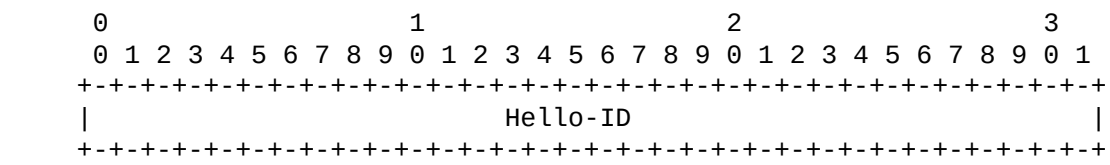


The content is defined by the implementation. See Section 8.5 for further discussion.

A.3.8. Hello-ID

Type: Hello-ID

Length: Fixed (1 32-bit word)



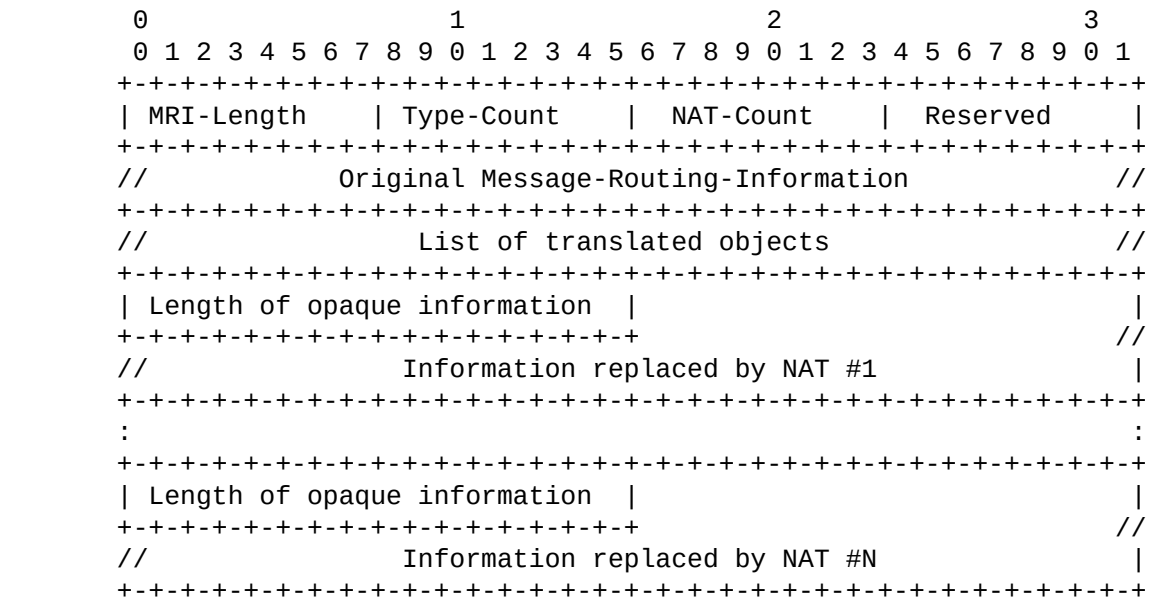
The content is defined by the implementation. See Section 5.2.2 for further discussion.

A.3.9. 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 7.2.2.



MRI-Length (8 bits): The length of the included MRI payload in 32-bit words.

Original Message-Routing-Information (variable): The MRI data from when the message was first sent, not including the object header.

Type-Count (8 bits): The number of objects in the 'List of translated objects' field.

List of translated objects (variable): This field lists the types of objects that were translated by every NAT through which the message has passed. Each element in the list is a 16-bit field containing the first 16 bits of the object TLV header, including the AB extensibility flags, 2 reserved bits, and 12-bit object type. The list is initialised by the first NAT on the path; subsequent NATs may delete elements in the list. Padded with 2 null bytes if necessary.

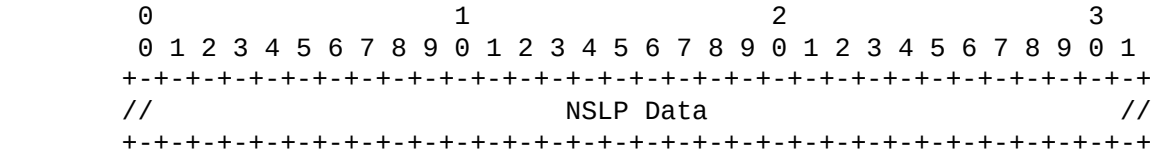
NAT-Count (8 bits): The number of NATs traversed by the message, and the number of opaque payloads at the end of the object. The length fields for each opaque payload are byte counts, not including the 2 bytes of the length field itself. Note that each opaque information field is zero-padded to the next 32-bit word boundary if necessary.

A.3.10. NSLP-Data

Type: NSLP-Data

Length: Variable (depends on NSLP)

This object is used to deliver data between NSLPs. GIST regards the data as a number of complete 32-bit words, as given by the length field in the TLV; any padding to a word boundary must be carried out within the NSLP itself.

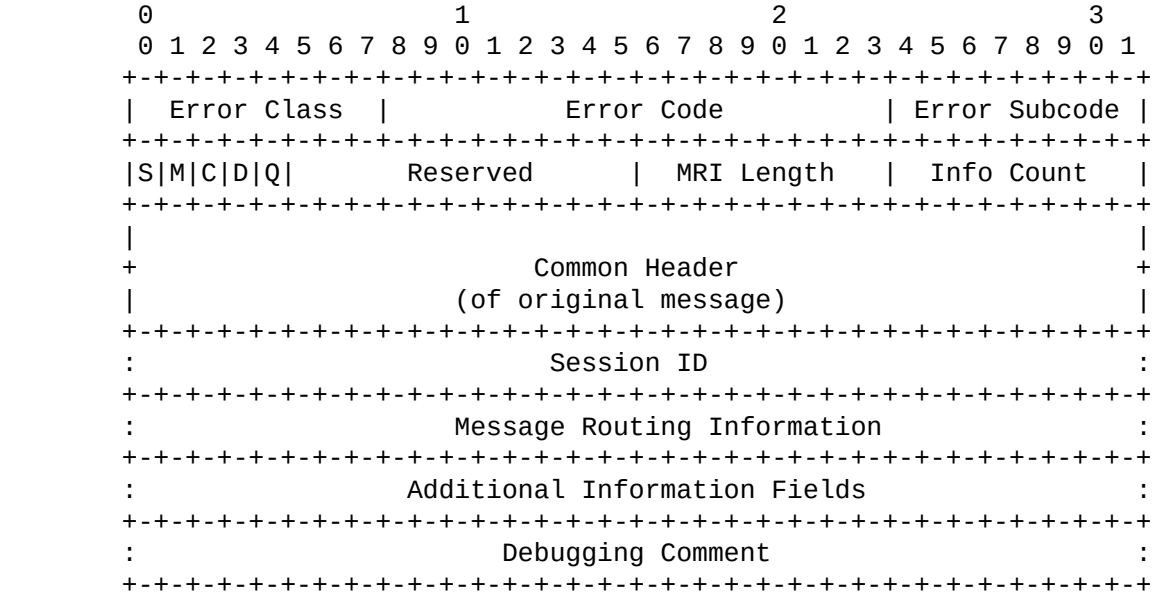


A.4. Errors

A.4.1. Error Object

Type: Error

Length: Variable (depends on error)



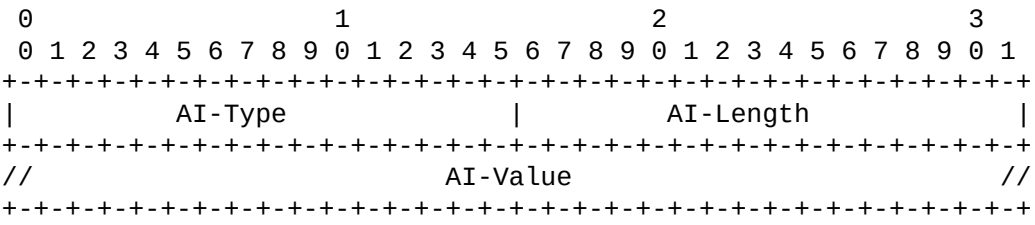
The flags are:
S - S=1 means the Session ID object is present.
M - M=1 means MRI object is present.
C - C=1 means a debug Comment is present after header.
D - D=1 means the original message was received in D-mode.
Q - Q=1 means the original message was received Q-mode encapsulated (can't be set if D=0).

A GIST Error Object contains an 8-bit error-class (see Appendix A.4.3), a 16-bit error-code, an 8-bit error-subcode, and as much information about the message that triggered the error as is available. This information MUST include the common header of the original message and MUST 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 MRI Length field gives the length of the MRI object in 32-bit words.

The Info Count field contains the number of Additional Information fields in the object, and the possible formats for these fields are given in Appendix A.4.2. The precise set of fields to include depends on the error code/subcode. For every error description in the error catalogue Appendix A.4.4, the line "Additional Info:" states what fields MUST be included; further fields beyond these MAY be included by the sender, and the fields may be included in any order. 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.

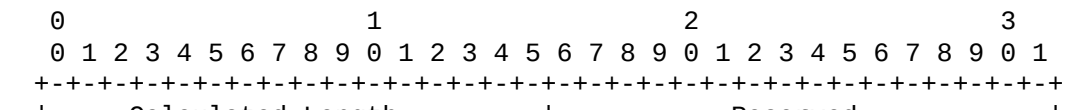
A.4.2. Additional Information Fields (AI)

The Common Error Header may be followed by some Additional Information fields. Each Additional Information field has a simple TLV format as follows:



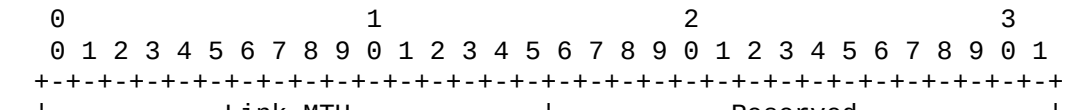
The AI-Type is a 16-bit IANA-assigned value. The AI-Length gives the number of 32-bit words in AI-Value; if an AI-Value is not present, AI-Length=0. The AI-Types and AI-Lengths and AI-Value formats of the currently defined Additional Information fields are shown below.

Message Length Info:



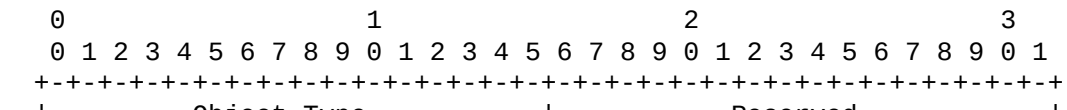
AI-Type: 1
AI-Length: 1
Calculated Length (16 bits): the length of the original message calculated by adding up all the objects in the message. Measured in 32-bit words.

MTU Info:



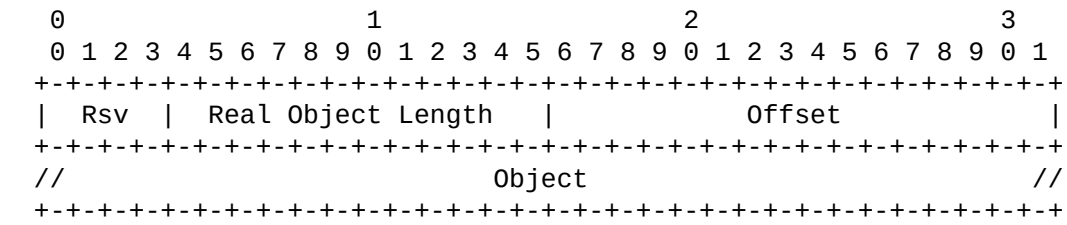
AI-Type: 2
AI-Length: 1
Link MTU (16 bits): the IP MTU for a link along which a message could not be sent. Measured in bytes.

Object Type Info:



AI-Type: 3
AI-Length: 1
Object type (16 bits): This provides information about the type of object that caused the error.

Object Value Info:



AI-Type: 4
AI-Length: variable (depends on object length)

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

Real Object Length (12 bits): 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. Measured in 32-bit words.

Offset (16 bits): The byte in the object at which the GIST node found the error. The first byte in the object has offset=0.

Object (variable): The invalid TLV object (including the TLV header).

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): reply data that should not be thought of as changing the condition of the protocol state machine.

1 (Success): reply data that 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 that 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 that 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 MUST be carried in the Error Object.

A.4.4.1. Common Header Parse Error

Class:	Protocol-Error
Code:	1
Additional Info:	For subcode 3 only, Message Length Info carries the calculated message length.

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 MUST NOT be 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 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.

4: Invalid E-flag: The E-flag is set in the header, but this is not a Data message.

5: Invalid C-flag: The C-flag was set on something other than a Query message or Q-mode Data message, or was clear on a Query message.

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 count of zero, or a GIST node tries to forward a message after its GIST hop count has been decremented to zero on reception. This message indicates either a routing loop or too small an initial hop count 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 that uses an incorrect encapsulation method (e.g., a Query arrives over an MA, or the Confirm for a handshake that sets up a messaging association arrives in D-mode).

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 that 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 Querying-SM or Responding-SM. This can occur on receiving a Data or Confirm 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 that it does not support.

A.4.4.7. Endpoint Found

```
Class:              Permanent-Failure
Code:               7
Additional Info:    None
```

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

A.4.4.8. Message Too Large

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

This message is sent if a router receives a message that it can't forward because it exceeds the IP 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 indicates the object type at fault in the additional info field. 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 that 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 receives a message containing an object that it does not support, and the extensibility flags AB=00.

2: Missing Object: This subcode is used if a GIST node receives a message that 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 Type: 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 that has not been translated by an intervening NAT.

5: Invalid Extensibility Flags: This subcode is used if an object is received with the extensibility flags AB=11.

A.4.4.10. Object Value Error

Class: Protocol-Error
Code: 10
Additional Info: 1 or 2 Object Value Info fields as given below

This message is sent if a node receives a message containing an object that cannot be properly parsed. The error message contains a single Object Value Info object, except for subcode 5 as 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 MRI (Appendix A.3.1.1), 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 contains only null bytes (a length of 0 is handled as "Value Not Supported").

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

5: Stack-Proposal - Stack-Configuration-Data 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 MUST be included in separate Object Value Info fields in that order.

A.4.4.11. Invalid IP-Layer TTL

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

This error indicates that a message was received with an IP-layer TTL outside an acceptable range, for example, that an upstream Query was received with an IP layer 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 that 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 refers to an IP version that is not implemented on the interface used, or is different from the IP version of the Query encapsulation (see Section 7.4).

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

Appendix B. API between GIST and Signalling Applications

This appendix provides an abstract API between GIST and signalling applications. It should not constrain implementers, 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 that 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 signalling application to GIST. It is used whenever the signalling 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, GIST-Hop-Count )
```

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 signalling 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 (such as 'integrity' or 'confidentiality') and can also be used to specify what authenticated signalling source and destination identities should be used to send the message. The possibilities can be learned by the signalling application from prior MessageStatus or RecvMessage notifications. If an NSLP-Message-Handle is provided, GIST will inform the signalling 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 signalling 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 layer TTL that should be used when sending this message (may be overridden by GIST for particular messages).

GIST-Hop-Count: The value for the hop count when sending the message.

B.2. RecvMessage

This primitive is passed from GIST to a signalling 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, GIST-Hop-Count,
              Inbound-Interface )
```

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 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 signalling 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 signalling 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 Response or forwarded Query respectively.

This mechanism could be extended to enable the signalling 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 signalling source address is one of the flow endpoints (i.e., whether this is the first or last GIST hop).

* Whether the signalling 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 layer TTL this message was received with (if available).

IP-Distance: The number of IP hops from the peer signalling node that sent this message along the path, or 0 if this information is not available.

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

Inbound-Interface: Attributes of the interface on which the message was received, such as whether it lies on the internal or external side of a NAT. These attributes have only local significance and are defined by the implementation.

B.3. MessageStatus

This primitive is passed from GIST to a signalling application. It is used to notify the signalling application that a message that it requested to be sent could not be dispatched, or to inform the signalling application about the transfer attributes that have been selected for the message (specifically, security attributes). The signalling 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 signalling 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 signalling application. It indicates that a network event of possible interest to the signalling 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. Five 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, has detected that existing routing state may no longer be valid, or has re-established existing 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.

Route Deletion: GIST has determined that an old route is now definitely invalid, e.g., that flows are definitely not using it (see Section 7.1.4). The SII-Handle of the peer is also reported.

Node Authorisation Change: The authorisation status of a peer has changed, meaning that routing state is no longer valid or that a signalling peer is no longer reachable; see Section 4.4.2.

Communication Failure: Communication with the peer has failed; messages may have been lost.

B.5. SetStateLifetime

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

SetStateLifetime (NSLPID, MRI, SID, 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).

SID: The session ID that the signalling application will be using with this routing state. Can be wildcarded.

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

B.6. InvalidateRoutingState

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

InvalidateRoutingState (NSLPID, MRI, Status, NSLP-Data,
NSLP-Data-Size, Urgent)

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

NSLP-Data, NSLP-Data-Size: (optional) A payload provided by the NSLP to be used the next GIST handshake. This can be used as part of a conditional peering process (see Section 4.3.2). The payload will be transmitted without security protection.

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

Appendix C. Deployment Issues with Router Alert Options

The GIST peer discovery handshake (Section 4.4.1) depends on the interception of Q-mode encapsulated IP packets (Section 4.3.1 and Section 5.3.2) by routers. There are two fundamental requirements on the process:

1. Packets relevant to GIST must be intercepted.

2. Packets not relevant to GIST must be forwarded transparently.

This specification defines the GIST behaviour to ensure that both requirements are met for a GIST-capable node. However, GIST packets will also encounter non-GIST nodes, for which requirement (2) still applies. If non-GIST nodes block Q-mode packets, GIST will not function. It is always possible for middleboxes to block specific traffic types; by using a normal UDP encapsulation for Q-mode traffic, GIST allows NATs at least to pass these messages (Section 7.2.1), and firewalls can be configured with standard policies. However, where the Q-mode encapsulation uses a Router Alert Option (RAO) at the IP level this can lead to additional problems. The situation is different for IPv4 and IPv6.

The IPv4 RAO is defined by [13], which defines the RAO format with a 2-byte value field; however, only one value (zero) is defined and there is no IANA registry for further allocations. It states that unknown values should be ignored (i.e., the packets forwarded as normal IP traffic); however, it has also been reported that some existing implementations simply ignore the RAO value completely (i.e. process any packet with an RAO as though the option value was zero). Therefore, the use of non-zero RAO values cannot be relied on to make GIST traffic transparent to existing implementations. (Note that it may still be valuable to be able to allocate non-zero RAO values for IPv4: this makes the interception process more efficient for nodes that do examine the value field, and makes no difference to nodes that *incorrectly* ignore it. Whether or not non-zero RAO values are used does not change the GIST protocol operation, but needs to be decided when new NSLPs are registered.)

The second stage of the analysis is therefore what happens when a non-GIST node that implements RAO handling sees a Q-mode packet. The RAO specification simply states "Routers that recognize this option shall examine packets carrying it more closely (check the IP Protocol

field, for example) to determine whether or not further processing is necessary". There are two possible basic behaviours for GIST traffic:

1. The "closer examination" of the packet is sufficiently intelligent to realise that the node does not need to process it and should forward it. This could either be by virtue of the fact that the node has not been configured to match IP-Protocol=UDP for RAO packets at all or that even if UDP traffic is intercepted the port numbers do not match anything locally configured.

2. The "closer examination" of the packet identifies it as UDP, and delivers it to the UDP stack on the node. In this case, it can no longer be guaranteed to be processed appropriately. Most likely, it will simply be dropped or rejected with an ICMP error (because there is no GIST process on the destination port to which to deliver it).

Analysis of open-source operating system source code shows the first type of behaviour, and this has also been seen in direct GIST experiments with commercial routers, including the case when they process other uses of the RAO (i.e., RSVP). However, it has also been reported that other RAO implementations will exhibit the second type of behaviour. The consequence of this would be that Q-mode packets are blocked in the network and GIST could not be used. Note that although this is caused by some subtle details in the RAO processing rules, the end result is the same as if the packet was simply blocked for other reasons (for example, many IPv4 firewalls drop packets with options by default).

The GIST specification allows two main options for circumventing nodes that block Q-mode traffic in IPv4. Whether to use these options is a matter of implementation and configuration choice.

- A GIST node can be configured to send Q-mode packets without the RAO at all. This should avoid the above problems, but should only be done if it is known that nodes on the path to the receiver are able to intercept such packets. (See Section 5.3.2.1.)

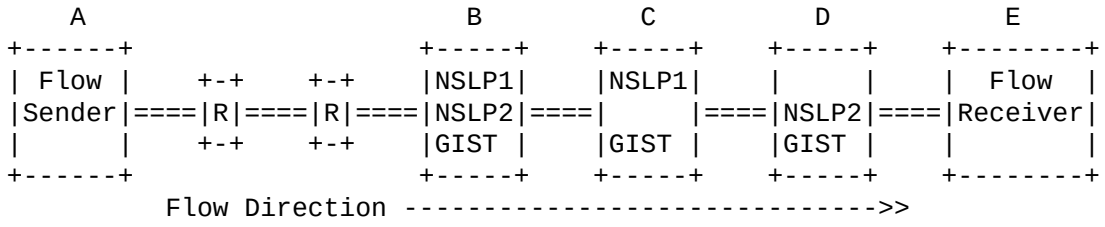
- If a GIST node can identify exactly where the packets are being blocked (e.g., from ICMP messages), or can discover some point on the path beyond the blockage (e.g., by use of traceroute or by routing table analysis), it can send the Q-mode messages to that point using IP-in-IP tunelling without any RAO. This bypasses the input side processing on the blocking node, but picks up normal GIST behaviour beyond it.

If in the light of deployment experience the problem of blocked Q-mode traffic turns out to be widespread and these techniques turn out to be insufficient, a further possibility is to define an alternative Q-mode encapsulation that does not use UDP. This would require a specification change. Such an option would be restricted to network-internal use, since operation through NATs and firewalls would be much harder with it.

The situation with IPv6 is rather different, since in that case the use of non-zero RAO values is well established in the specification ([17]) and an IANA registry exists. The main problem is that several implementations are still immature: for example, some treat any RAO-marked packet as though it was for local processing without further analysis. Since this prevents any RAO usage at all (including the existing standardised ones) in such a network, it seems reasonable to assume that such implementations will be fixed as part of the general deployment of IPv6.

Appendix D. Example Routing State Table and Handshake

Figure 11 shows a signalling scenario for a single flow being managed by two signalling 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	NSLPID	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 11: A Signalling Scenario

The upstream state is just the same address for each application. For the downstream direction, NSLP1 only requires D-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 12 shows a GIST handshake setting up a messaging association for B-D signalling, with the exchange of Stack Proposals and MA-protocol-options in each direction. The Querying node selects TLS/TCP as the stack configuration and sets up the messaging association over which it sends the Confirm.

```
    -------------------------- Query ---------------------------->
    IP(Src=IP#A; Dst=IP#E; RAO for NSLP2); UDP(Src=6789; Dst=GIST)
    D-mode magic number (0x4e04 bda5)
    GIST(Header(Type=Query; NSLPID=NSLP2; C=1; 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(HoldTime=300; #MPO=2;
           TCP(Applicable: all; Data: null)
           SCTP(Applicable: all; Data: null)))

    <---------------------- Response ----------------------------
    IP(Src=IP#D; Dst=IP#B); UDP(Src=GIST; Dst=6789)
    D-mode magic number (0x4e04 bda5)
    GIST(Header(Type=Response; NSLPID=NSLP2; C=0; 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(HoldTime=200; #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------------------------>

    ------------------------ Confirm ---------------------------->
    [Sent within messaging association]
    GIST(Header(Type=Confirm; NSLPID=NSLP2; C=0; 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)
         StackConfigurationData(HoldTime=300))
```

Figure 12: GIST Handshake Message Sequence

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
Roke Manor Research
Old Salisbury Lane
Romsey, Hampshire SO51 0ZN
UK

EMail: robert.hancock@roke.co.uk
URI: http://www.roke.co.uk