IPTEL Working Group
Internet Draft
draft-ietf-iptel-trip-09.txt
August 2001
Expiration Date: February 2002

J. Rosenberg, dynamicsoft
H. Salama, Cisco Systems
M. Squire, WindWire

Telephony Routing over IP (TRIP)

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

This document presents the Telephony Routing over IP (TRIP). TRIP is a policy driven inter-administrative domain protocol for advertising the reachability of telephony destinations between location servers, and for advertising attributes of the routes to those destinations. TRIP's operation is independent of any signaling protocol, hence TRIP can serve as the telephony routing protocol for any signaling protocol.

The Border Gateway Protocol (BGP-4) is used to distribute routing information between administrative domains. TRIP is used to distribute telephony routing information between telephony administrative domains. The similarity between the two protocols is obvious, and hence TRIP is modeled after BGP-4.

Table of Contents

1. Terminology and Definitions

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 [1].

A framework for a Telephony Routing over IP (TRIP) is described in [2]. We assume the reader is familiar with the framework and terminology of [2]. We define and use the following terms in addition to those defined in [2].

Telephony Routing Information Base (TRIB): The database of reachable telephony destinations built and maintained at an LS as a result of its participation in TRIP.

IP Telephony Administrative Domain (ITAD): The set of resources (gateways, location servers, etc.) under the control of a single administrative authority. End users are customers of an ITAD.

Less/More Specific Route: A route X is said to be less specific than a route Y if every destination in Y is also a destination in X, and X and Y are not equal. In this case, Y is also said to be more specific than X.

Aggregation: Aggregation is the process by which multiple routes are combined into a single less specific route that covers the same set of destinations. Aggregation is used to reduce the size of the TRIB synchronized with peer LSs by reducing the number of exported TRIP routes.

Peers: Two LSs that share a logical association (a transport connection). If the LSs are in the same ITAD, they are internal peers. Otherwise, they are external peers. The logical association between two peer LSs is called a peering session.

Telephony Routing Information Protocol (TRIP): The protocol defined in this specification. The function of TRIP is to advertise the reachability of telephony destinations, attributes associated with the destinations, as well as the attributes of the path towards those destinations.

TRIP destination: TRIP can be used to manage routing tables for multiple protocols (SIP, H323, etc.). In TRIP, a destination is the combination of (a) a set of addresses (given by an address family and address prefix), and (b) an application protocol (SIP, H323, etc).

2. Introduction

The gateway location and routing problem has been introduced in [2]. It is considered one of the more difficult problems in IP telephony. The selection of an egress gateway for a telephony call, traversing an IP network towards an ultimate destination in the PSTN, is driven in large part by the policies of the various parties along the path, and by the relationships established between these parties. As such, a global directory of egress gateways in which users look up destination phone numbers is not a feasible solution. Rather, information about the availability of egress gateways is exchanged between providers, and subject to policy, made available locally and then propagated to other providers in other ITADs, thus creating routes towards these egress gateways. This would allow each provider to create its own database of reachable phone numbers and the associated routes - such a database could be very different for each provider depending on policy.

TRIP is an inter-domain (i.e., inter-ITAD) gateway location and routing protocol. The primary function of a TRIP speaker, called a location server (LS), is to exchange information with other LSs. This information includes the reachability of telephony destinations, the routes towards these destinations, and information about gateways towards those telephony destinations residing in the PSTN. The TRIP requirements are set forth in [2].

LSs exchange sufficient routing information to construct a graph of ITAD connectivity so that routing loops may be prevented. In addition, TRIP can be used to exchange attributes necessary to enforce policies and to select routes based on path or gateway characteristics. This specification defines TRIP's transport and synchronization mechanisms, its finite state machine, and the TRIP data. This specification defines the basic attributes of TRIP. The TRIP attribute set is extendible, so additional attributes may be defined in future drafts.

TRIP is modeled after the Border Gateway Protocol 4 (BGP-4) [3] and enhanced with some link state features as in the Open Shortest Path First (OSPF) protocol [4], IS-IS [5], and the Server Cache Synchronization Protocol (SCSP) [6]. TRIP uses BGP's inter-domain transport mechanism, BGP's peer communication, BGP's finite state machine, and similar formats and attributes as BGP. Unlike BGP however, TRIP permits generic intra-domain LS topologies, which simplifies configuration and increases scalability in contrast to BGP's full mesh requirement of internal BGP speakers. TRIP uses an intra-domain flooding mechanism similar to that used in OSPF [4], IS-IS [5], and SCSP [6].

TRIP permits aggregation of routes as they are advertised through the network. TRIP does not define a specific route selection algorithm.

TRIP runs over a reliable transport protocol. This eliminates the need to implement explicit fragmentation, retransmission, acknowledgment, and sequencing. The error notification mechanism used in TRIP assumes that the transport protocol supports a graceful close, i.e., that all outstanding data will be delivered before the connection is closed.

TRIP's operation is independent of any particular telephony signaling protocol. Therefore, TRIP can be used as the routing protocol for any of these protocols, e.g., H.323 [7] and SIP [8].

The LS peering topology is independent of the physical topology of the network. In addition, the boundaries of ITAD are independent of the boundaries of the layer 3 routing autonomous systems. Neither internal nor external TRIP peers need be physically adjacent.

3. Summary of Operation

This section summarizes the operation of TRIP. Details are provided in later sections.

3.1. Peering Session Establishment and Maintenance

Two peer LSs form a transport protocol connection between one another. They exchange messages to open and confirm the connection parameters, and to negotiate the capabilities of each LS as well as the type of information to be advertised over this connection.

KeepAlive messages are sent periodically to ensure adjacent peers are operational. Notification messages are sent in response to errors or special conditions. If a connection encounters an error condition, a Notification message is sent and the connection is closed.

3.2. Database Exchanges

Once the peer connection has been established, the initial data flow is a dump of all routes relevant to the new peer (In case of an external peer, all routes in the LS's Adj-TRIB-Out for that external peer. In case of an internal peer, all routes in the Ext-TRIB and all Adj-TRIBs-In). Note that the different TRIBs are defined in Section 3.5.

Incremental updates are sent as the TRIP routing tables (TRIBs) change. TRIP does not require periodic refresh of the routes. Therefore, an LS must retain the current version of all routing entries.

If a particular ITAD has multiple LSs and is providing transit service for other ITADs, then care must be taken to ensure a consistent view of routing within the ITAD. When synchronized the TRIP routing tables, i.e., the Loc-TRIBs, of all internal peers are identical.

3.3. Internal Versus External Synchronization

As with BGP, TRIP distinguishes between internal and external peers. Within an ITAD, internal TRIP uses link-state mechanisms to flood database updates over an arbitrary topology. Externally, TRIP uses point-to-point peering relationships to exchange database information.

To achieve internal synchronization, internal peer connections are configured between LSs of the same ITAD such that the resulting intra-domain LS topology is connected and sufficiently redundant. This is different from BGP's approach that requires all internal peers to be connected in a full mesh topology, which may result in scaling problems. When an update is received from an internal peer, the routes in the update are checked to determine if they are newer than the version already in the database. Newer routes are then flooded to all other peers in the same domain.

3.4. Advertising TRIP Routes

In TRIP, a route is defined as the combination of (a) a set of destination addresses (given by an address family indicator and an address prefix), and (b) an application protocol (e.g. SIP, H323, etc.). Generally, there are additional attributes associated with each route (for example, the next-hop server).

TRIP routes are advertised between a pair of LSs in UPDATE messages. The destination addresses are included in the ReachableRoutes attribute of the UPDATE, while other attributes describe things like the path or egress gateway.

If an LS chooses to advertise the TRIP route, it may add to or modify the attributes of the route before advertising it to a peer. TRIP provides mechanisms by which an LS can inform its peer that a previously advertised route is no longer available for use. There are three methods by which a given LS can indicate that a route has been withdrawn from service:

- Include the route in the WithdrawnRoutes Attribute in an UPDATE message, thus marking the associated destinations as being no longer available for use.
- Advertise a replacement route with the same set of destinations in the ReachableRoutes Attribute.
- For external peers where flooding is not in use, the LS-to-LS peer connection can be closed, which implicitly removes from service all routes which the pair of LSs had advertised to each other over that peer session. Note that terminating an internal peering session does not necessarily remove the routes advertised by the peer LS as the same routes may have been received from multiple internal peers because of flooding. If an LS determines that the another internal LS is no longer active (from the ITAD Topology attributes of the UPDATE messages from other internal peers), then it MUST remove all routes originated into the LS by that LS and rerun its decision process.

3.5. Telephony Routing Information Bases

A TRIP LS processes three types of routes:

- External routes: An external route is a route received from an external peer LS
- Internal routes: An internal route is a route received from an internal LS in the same ITAD.
- Local routes: A local route is a route locally injected into TRIP, e.g. by configuration or by route redistribution from another routing protocol.

The Telephony Routing Information Base (TRIB) within an LS consists of four distinct parts:

- Adj-TRIBs-In: The Adj-TRIBs-In store routing information that has been learned from inbound UPDATE messages. Their contents represent TRIP routes that are available as an input to the Decision Process. These are the "unprocessed" routes received. The routes from each external peer LS and each internal LS are maintained in this database independently, so that updates from one peer do not affect the routes received from another LS. Note that there is an Adj-TRIBs-In for every LS within the domain, even those with which the LS is not directly peered.
- Ext-TRIB: There is only one Ext-TRIB database per LS. The LS runs the route selection algorithm on all external routes (stored in the Adj-TRIBs-In of the external peers) and local routes (may be stored in an Adj-TRIB-In representing the local LS) and selects the best route for a given destination and stores it in the Ext-TRIB. The use of Ext-TRIB will be explained further in Section 10.3.1
- Loc-TRIB: The Loc-TRIB contains the local TRIP routing information that the LS has selected by applying its local policies to the routing information contained in its Adj-TRIBs-In of internal LSs and the Ext-TRIB.
- Adj-TRIBs-Out: The Adj-TRIBs-Out store the information that the local LS has selected for advertisement to its external peers. The routing information stored in the Adj-TRIBs-Out will be carried in the local LS's UPDATE messages and advertised to its peers.

Figure 1 illustrates the relationship between the three parts of the routing information base.

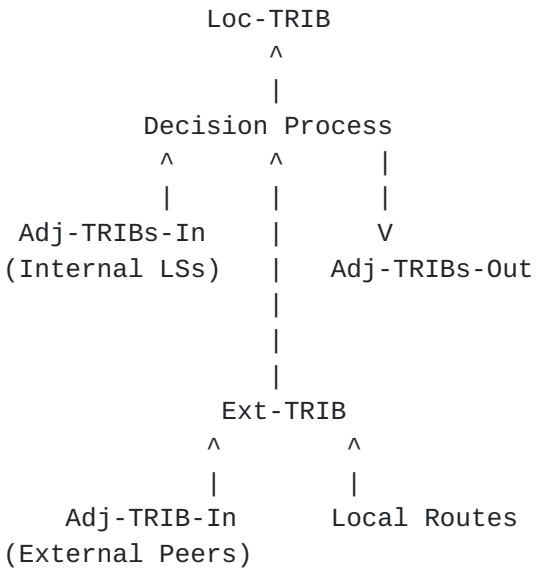

Figure 1: TRIB Relationships

Although the conceptual model distinguishes between Adj-TRIBs-In, Loc-TRIB, and Adj-TRIBs-Out, this neither implies nor requires that an implementation must maintain three separate copies of the routing information. The choice of implementation (for example, 3 copies of the information vs. 1 copy with pointers) is not constrained by the protocol.

3.6. Routes in TRIP

A route in TRIP specifies a range of numbers by being a prefix of those numbers (the exact definition & syntax of route are in 5.1.1). Arbitrary ranges of numbers are not atomically representable by a route in TRIP. A prefix range is the only type of range supported atomicly. An arbitrary range can be accomplished by using multiple prefixes in a ReachableRoutes attribute (see Section 5.1 & 5.2). For example, 222-xxxx thru 999-xxxx could be represented by including the prefixes 222, 223, 224,...,23,24,...,3,4,...,9 in a ReachableRoutes attribute.

3.7. Aggregation

Aggregation is a scaling enhancement used by an LS to reduce the number of routing entries that it has to synchronize with its peers. Aggregation may be peformed by an LS when there is a set of routes {R1, R2, ...} in its TRIB such that there exists a less specific route R where every valid destination in R is also a valid destination in {R1, R2, ...} and vice-versa. Section 5 includes a

description of how to combine each attribute (by type) on the {R1, R2, ...} routes into an attribute for R.

Note that there is no mechanism within TRIP to communicate that a particular address prefix is not used or valid within a particular address family, and thus that these addresses could be skipped during aggregation. LSs may use methods outside of TRIP to learn of invalid prefixes that may be ignored during aggregation.

An LS is not required to perform aggregation, however it is recommended whenever maintaining a smaller TRIB is important. Whether an LS aggregate routes is considered a local policy decision.

Whenever an LS aggregates multiple routes where the NextHopServer is not identical in all aggregated routes, the NextHopServer attribute of the aggregate route must be set to a signalling server in the aggregating LS's domain.

When an LS resets the NextHopServer of any route, and this may be performed because of aggregation or other reasons, it has the effect of adding another signalling server along the signalling path to these desinations. The end result is that the signalling path between two destinations may consist of multiple signalling servers across multiple domains.

4. Message Formats

This section describes message formats used by TRIP. Messages are sent over a reliable transport protocol connection. A message MUST be processed only after it is entirely received. The maximum message size is 4096 octets. All implementations MUST support this maximum message size. The smallest message that MAY be sent consists of a TRIP header without a data portion, or 3 octets.

4.1. Message Header Format

Each message has a fixed-size header. There may or may not be a data portion following the header depending on the message type. The layout of the header fields is shown in Figure 2.

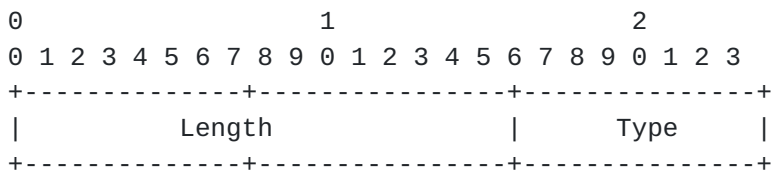

Figure 2: TRIP Header

Length:
This 2-octet unsigned integer indicates the total length of the message, including the header, in octets. Thus, it allows one to locate in the transport-level stream the beginning of the next message. The value of the Length field must always be at least 3 and no greater than 4096, and may be further constrained depending on the message type. No padding of extra data after the message is allowed, so the Length field must have the smallest value possible given the rest of the message.

Type:
This 1-octet unsigned integer indicates the type code of the message. The following type codes are defined:

1 - OPEN
2 - UPDATE
3 - NOTIFICATION
4 - KEEPALIVE

4.2. OPEN Message Format

After a transport protocol connection is established, the first message sent by each side is an OPEN message. If the OPEN message is acceptable, a KEEPALIVE message confirming the OPEN is sent back. Once the OPEN is confirmed, UPDATE, KEEPALIVE, and NOTIFICATION messages may be exchanged.

The minimum length of the OPEN message is 14 octets (including message header). OPEN messages not meeting this minimum requirement are handled as defined in Section 6.2.

In addition to the fixed-size TRIP header, the OPEN message contains the following fields:

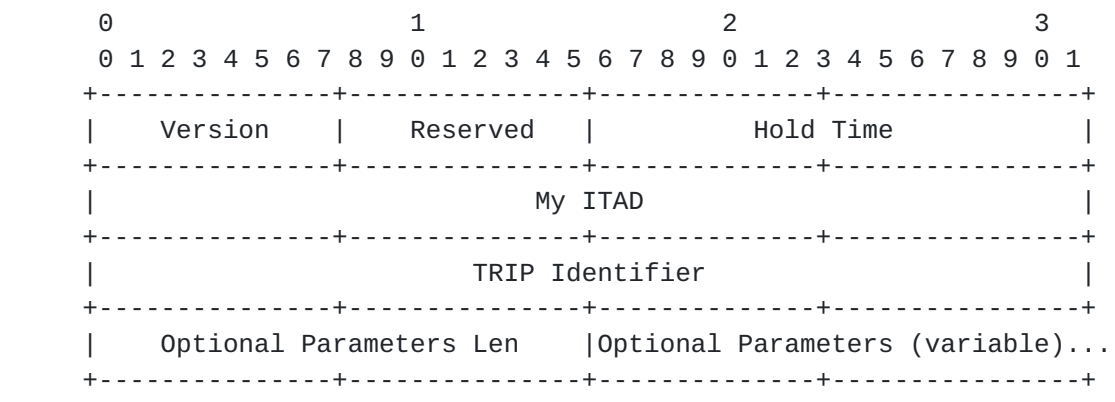

Figure 3: TRIP OPEN Header

Version:
This 1-octet unsigned integer indicates the protocol version of the message. The current TRIP version number is 1.

Hold Time:
This 2-octet unsigned integer indicates the number of seconds that the sender proposes for the value of the Hold Timer. Upon receipt of an OPEN message, an LS MUST calculate the value of the Hold Timer by using the smaller of its configured Hold Time and the Hold Time received in the OPEN message. The Hold Time MUST be either zero or at least three seconds. An implementation MAY reject connections on the basis of the Hold Time. The calculated value indicates the maximum number of seconds that may elapse between the receipt of successive KEEPALIVE and/or UPDATE messages by the sender.

This 4-octet unsigned integer indicates the ITAD number of the sender. The ITAD number must be unique for this domain within this confederation of cooperating LSs.

ITAD numbers are assigned by IANA as specified in Section 13. This document reserves ITAD number 0. ITAD numbers from 1 to 255 are designated for private use.

TRIP Identifier:
This 4-octet unsigned integer indicates the TRIP Identifier of the sender. The TRIP Identifier MUST uniquely identify this LS within its ITAD. A given LS MAY set the value of its TRIP Identifier to an IPv4 address assigned to that LS. The value of the TRIP Identifier is determined on startup and MUST be the same for all peer connections. When comparing two TRIP identifiers, the TRIP Identifier is interpreted as a numerical 4-octet unsigned integer.

Optional Parameters Length:
This 2-octet unsigned integer indicates the total length of the Optional Parameters field in octets. If the value of this field is

zero, no Optional Parameters are present.

Optional Parameters:
This field may contain a list of optional parameters, where each parameter is encoded as a <Parameter Type, Parameter Length, Parameter Value> triplet.

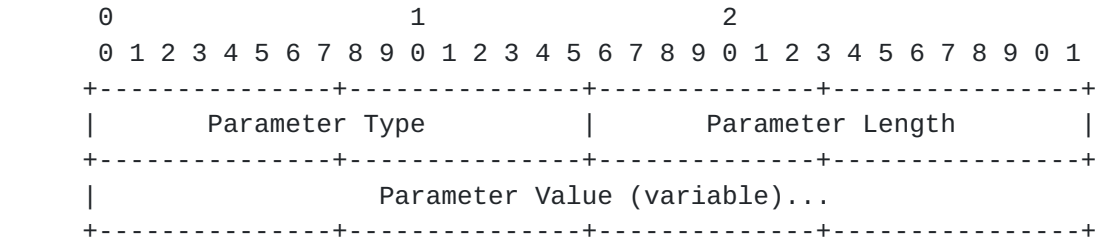

Figure 4: Optional Parameter Encoding

Parameter Type:
This is a 2-octet field that unambiguously identifies individual parameters.

Parameter Length:
This is a 2-octet field that contains the length of the Parameter Value field in octets.

Parameter Value:
This is a variable length field that is interpreted according to the value of the Parameter Type field.

4.2.1. Open Message Optional Parameters

This document defines the following Optional Parameters for the OPEN message.

4.2.1.1. Capability Information

Capability Information uses Optional Parameter type 1. This is an optional parameter used by an LS to convey to its peer the list of capabilities supported by the LS. This permits an LS to learn of the capabilities of its peer LSs. Capability negotiation is defined in Section 8.

The parameter contains one or more triples <Capability Code, Capability Length, Capability Value>, where each triple is encoded as shown below:

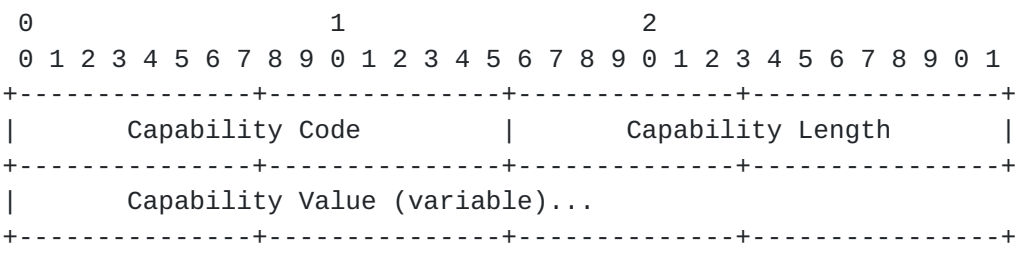

Figure 5: Capability Optional Parameter

Capability Code:
Capability Code is a 2-octet field that unambiguously identifies individual capabilities.

Capability Length:
Capability Length is a 2-octet field that contains the length of the Capability Value field in octets.

Capability Value:
Capability Value is a variable length field that is interpreted according to the value of the Capability Code field.

Any particular capability, as identified by its Capability Code, may appear more than once within the Optional Parameter.

This document reserves Capability Codes 32768-65535 for vendor-specific applications (these are the codes with the first bit of the code value equal to 1). This document reserves value 0. Capability Codes (other than those reserved for vendor specific use) are controlled by IANA. See Section 13 for IANA considerations.

The following Capability Codes are defined by this specification:

Code	Capability
1	Route Types Supported
2	Send Receive Capability

4.2.1.1.1. Route Types Supported

The Route Types Supported Capability Code lists the route types supported in this peering session by the transmitting LS. An LS MUST NOT use route types that are not supported by the peer LS in any particular peering session. If the route types supported by a peer are not satisfactory, an LS SHOULD terminate the peering session. The format for a Route Type is:

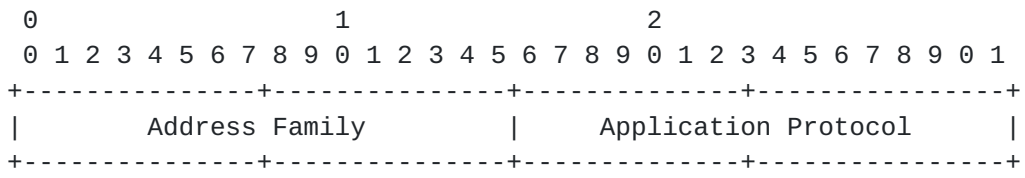

Figure 6: Route Types Supported Capability

The Address Family and Application Protocol are as defined in Section 5.1.1. Address Family gives the address family being routed (within the ReachableRoutes attribute). The application protocol lists the application for which the routes apply. As an example, a route type for TRIP could be <POTS, SIP>, indicating a set of POTS destinations for the SIP protocol.

The Route Types Supported Capability MAY contain multiple route types in the capability. The number of route types within the capability is the maximum number that can fit given the capability length. The Capability Code is 1 and the length is variable.

4.2.1.1.2. Send Receive Capability

This capability specifies the mode in which the LS will operate with this particular peer. The possible modes are: Send Only mode, Receive Only mode, or Send Receive mode. The default mode is Send Receive mode.

In Send Only mode, an LS transmits UPDATE messages to its peer, but the peer MUST NOT transmit UPDATE messages to that LS. If an LS in Send Only mode receives an UPDATE message from its peer, it MUST discard that message, but no further action should be taken.

The UPDATE messages sent by an LS in Send Only mode to its intra-domain peer MUST include the ITAD Topology attribute whenever the topology changes. A useful application of an LS in Send Only mode with an external peer is to enable gateway termination services.

If a service provider terminates calls to a set of gateways it owns, but never initiates calls, it can set its LSs to operate in Send Only mode, since they only ever need to generate UPDATE messages, not receive them.

If an LS in Send Receive mode has a peering session with a peer in Send Only mode, that LS MUST set its route dissemination policy such that it does not send any UPDATE messages to its peer.

In Receive Only mode, the LS acts as a passive TRIP listener. It receives and processes UPDATE messages from its peer, but it MUST NOT transmit any UPDATE messages to its peer. This is useful for management stations that wish to collect topology information for display purposes.

The behavior of an LS in Send Receive mode is the default TRIP operation specified throughout this document.

The Send Receive capability is a 4-octet unsigned numeric value. It can only take one of the following three values:

1 - Send Receive mode
2 - Send only mode
3 - Receive Only mode

A peering session MUST NOT be established between two LSs, both of them in either Send Only mode or in Receive Only mode. If a peer LS detects such a capability mismatch when processing an OPEN message, it MUST respond with a NOTIFICATION message and close the peer session. The error code in the NOTIFICATION message must be set to "Capability Mismatch."

An LS MUST be configured in the same Send Receive mode for all peers.

4.3. UPDATE Message Format

UPDATE messages are used to transfer routing information between LSs. The information in the UPDATE packet can be used to construct a graph describing the relationships between the various ITADs. By applying rules to be discussed, routing information loops and some other anomalies can be prevented.

An UPDATE message is used to both advertise and withdraw routes from service. An UPDATE message may simultaneously advertise and withdraw TRIP routes.

In addition to the TRIP header, the TRIP UPDATE contains a list of routing attributes as shown in Figure 7. There is no padding between routing attributes.

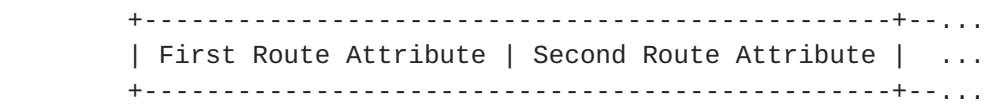

Figure 7: TRIP UPDATE Format

The minimum length of an UPDATE message 11 octets (the TRIP header plus at least the WithdrawnRoutes and ReachableRoutes attributes).

4.3.1. Routing Attributes

A variable length sequence of routing attributes is present in every UPDATE message. Each attribute is a triple <attribute type, attribute length, attribute value> of variable length.

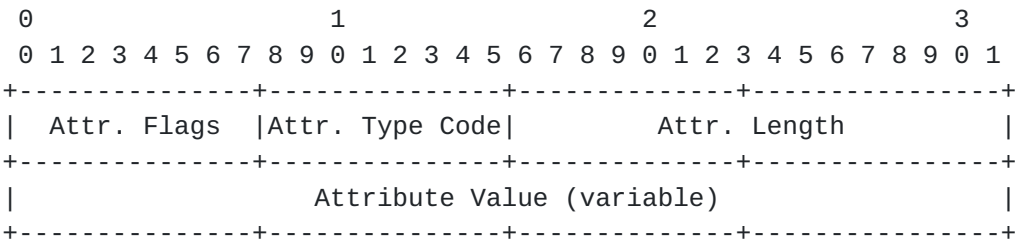

Figure 8: Routing Attribute Format

Attribute Type is a two-octet field that consists of the Attribute Flags octet followed by the Attribute Type Code octet.

The Attribute Type Code defines the type of attribute. The basic TRIP-defined Attribute Type Codes are discussed later in this section. Attributes MUST appear in the UPDATE message in numerical order of the Attribute Type Code. An attribute MUST NOT be included more than once in the same UPDATE message. Attribute Flags are used to control attribute processing when the attribute type is unknown. Attribute Flags are further defined in Section 4.3.2.

This document reserves Attribute Type Codes 224-255 for vendor-specific applications (these are the codes with the first three bits of the code equal to 1). This document reserves value 0. Attribute Type Codes (other than those reserved for vendor specific use) are controlled by IANA. See Section 13 for IANA considerations.

The third and the fourth octets of the route attribute contain the length of the attribute value field in octets.

The remaining octets of the attribute represent the Attribute Value and are interpreted according to the Attribute Flags and the Attribute Type Code. The basic supported attribute types, their values, and their uses are defined in this specification. These are the attributes necessary for proper loop free operation of TRIP, both inter-domain and intra-domain. Additional attributes may be defined in future documents.

4.3.2. Attribute Flags

It is clear that the set of attributes for TRIP will evolve over time. Hence it is essential that mechanisms be provided to handle attributes with unrecognized types. The handling of unrecognized attributes is controlled via the flags field of the attribute. Recognized attributes should be processed according to their specific definition.

The following are the attribute flags defined by this specification:

Bit	Flag
0	Well-Known Flag
1	Transitive Flag
2	Dependent Flag
3	Partial Flag
4	Link-state Encapsulated Flag

The high-order bit (bit 0) of the Attribute Flags octet is the Well-Known Bit. It defines whether the attribute is not well-known (if set to 1) or well-known (if set to 0). Implementations are not required to support not well-known attributes, but MUST support well-known attributes.

The second high-order bit (bit 1) of the Attribute Flags octet is the Transitive bit. It defines whether a not well-known attribute is transitive (if set to 1) or non-transitive (if set to 0). For well-known attributes, the Transitive bit MUST be zero on transmit and MUST be ignored on receipt.

The third high-order bit (bit 2) of the Attribute Flags octet is the Dependent bit. It defines whether a transitive attribute is dependent (if set to 1) or independent (if set to 0). For well-known attributes and for non-transitive attributes, the Dependent bit is irrelevant, and MUST be set to zero on transmit and MUST be ignored on receipt.

The fourth high-order bit (bit 3) of the Attribute Flags octet is the Partial bit. It defines whether the information contained in the not well-known transitive attribute is partial (if set to 1) or complete (if set to 0). For well-known attributes and for non- transitive attributes the Partial bit MUST be set to 0 on transmit and MUST be ignored on receipt.

The fifth high-order bit (bit 4) of the Attribute Flags octet is the Link-state Encapsulation bit. This bit is only applicable to certain attributes (ReachableRoutes and WithdrawnRoutes) and determines the encapsulation of the routes within those attributes. If this bit is set, link-state encapsulation is used within the attribute. Otherwise, standard encapsulation is used within the attribute. The

Link-state Encapsulation technique is described in Section 4.3.2.4. This flag is only valid on the ReachableRoutes and WithdrawnRoutes attributes. It MUST be cleared on transmit and MUST be ignored on receipt for all other attributes.

The other bits of the Attribute Flags octet are unused. They MUST be zeroed on transmit and ignored on receipt.

4.3.2.1. Attribute Flags and Route Selection

Any recognized attribute can be used as input to the route selection process, although the utility of some attributes in route selection is minimal.

4.3.2.2. Attribute Flags and Route Dissemination

TRIP provides for two variations of transitivity due to the fact that intermediate LSs need not modify the NextHopServer when propagating routes. Attributes may be non-transitive, dependent transitive, or independent transitive. An attribute cannot be both dependent transitive and independent transitive.

Unrecognized independent transitive attributes may be propagated by any intermediate LS. Unrecognized dependent transitive attributes MAY only be propagated if the LS is NOT changing the next-hop server. The transitivity variations permit some unrecognized attributes to be carried end-to-end (independent transitive), some to be carried between adjacent next-hop servers (dependent transitive), and other to be restricted to peer LSs (non- transitive).

An LS that passes an unrecognized transitive attribute to a peer MUST set the Partial flag on that attribute. Any LS along a path MAY insert a transitive attribute into a route. If any LS except the originating LS inserts a new independent transitive attribute into a route, then it MUST set the Partial flag on that attribute. If any LS except an LS that modifies the NextHopServer inserts a new dependent transitive attribute into a route, then it MUST set the Partial flag on that attribute. The Partial flag indicates that not every LS along the relevant path has processed and understood the attribute. For independent transitive attributes, the "relevant path" is the path given in the AdvertisementPath attribute. For dependent transitive attributes, the relevant path consists only of those domains thru which this object has passed since the NextHopServer was last modified. The Partial flag in an independent transitive attribute MUST NOT be unset by any other LS along the path. The Partial flag in a dependent transitive attribute MUST be reset

whenever the NextHopServer is changed, but MUST NOT be unset by any LS that is not changing the NextHopServer.

The rules governing the addition of new non-transitive attributes are defined independently for each non-transitive attribute. Any attribute MAY be updated by an LS in the path.

4.3.2.3. Attribute Flags and Route Aggregation

Each attribute defines how it is to be handled during route aggregation.

The rules governing the handling of unknown attributes are guided by the Attribute Flags. Unrecognized transitive attributes are dropped during aggregation. There should be no unrecognized non-transitive attributes during aggregation because non-transitive attributes must be processed by the local LS in order to be propagated.

4.3.2.4. Attribute Flags and Encapsulation

Normally attributes have the simple format as described in Section 4.3.1. If the Link-state Encapsulation Flag is set, then the two additional fields are added to the attribute header as shown in Figure 9.

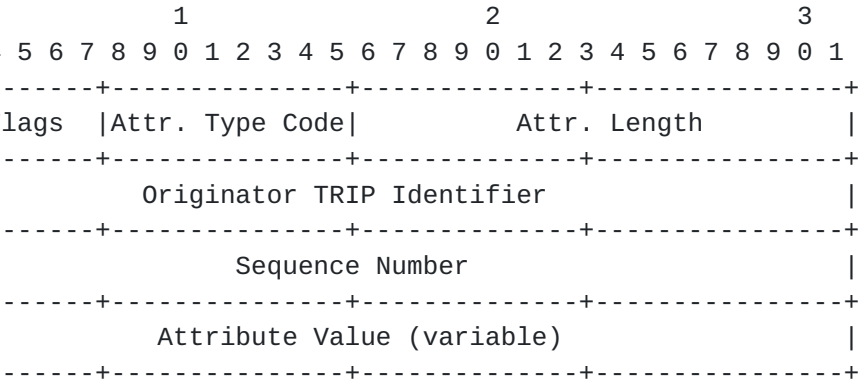

Figure 9: Link State Encapsulation

The Originator TRIP ID and Sequence Number are used to control the flooding of routing updates within a collection of servers. These fields are used to detect duplicate and old routes so that they are not further propagated within the servers. The use of these fields is defined in Section 10.1.

4.3.3. Mandatory Attributes

There are no Mandatory attributes in TRIP. However, there are Conditional Mandatory attributes. A conditional mandatory attribute is an attribute, which MUST be included in an UPDATE message if another attribute is included in that message. For example, if an UPDATE message includes a ReachableRoutes attribute, it MUST include an AdvertisementPath attribute as well.

The three base attributes in TRIP are WithdrawnRoutes, ReachableRoutes, and ITAD Topology. Their presence in an UPDATE message is entirely optional and independent of any other attributes.

4.3.4. TRIP UPDATE Attributes

This section summarizes the attributes that may be carried in an UPDATE message. Attributes MUST appear in the UPDATE message in increasing order of the Attribute Type Code. Additional details are provided in Section 5.

4.3.4.1. WithdrawnRoutes

This attribute lists a set of routes that are being withdrawn from service. The transmitting LS has determined that these routes should no longer be advertised, and is propagating this information to its peers.

4.3.4.2. ReachableRoutes

This attribute lists set of routes that are being added to service. These routes will have the potential to be inserted into the Adj-TRIBs-In of the receiving LS and the route selection process will be applied to them.

4.3.4.3. NextHopServer

This attribute gives the identity of the entity to which messages should be sent along this routed path. It specifies the identity of the next hop server as either a host domain name or an IP address. It MAY optionally specify the UDP/TCP port number for the next hop signaling server. If not specified, then the default port SHOULD be used. The NextHopServer is specific to the set of destinations and application protocol defined in the ReachableRoutes attribute. Note that this is NOT the address to which media (voice, video, etc.)

should be transmitted, it is only for the application protocol as given in the ReachableRoutes attribute.

4.3.4.4. AdvertisementPath

The AdvertisementPath is analogous to the AS_PATH in BGP4 [3]. The attribute records the sequence of domains through which this advertisement has passed. The attribute is used to detect when the routing advertisement is looping. This attribute does NOT reflect the path through which messages following this route would traverse. Since the next-hop need not be modified by each LS, the actual path to the destination might not have to traverse every domain in the AdvertisementPath.

4.3.4.5. RoutedPath

The RoutedPath attribute is analogous to the AdvertisementPath attribute, except that it records the actual path (given by the list of domains) *to* the destinations. Unlike AdvertisementPath, which is modified each time the route is propagated, RoutedPath is only modified when the NextHopServer attribute changes. Thus, it records the subset of the AdvertisementPath over which messages following this particular route would traverse.

4.3.4.6. AtomicAggregate

The AtomicAggregate attribute indicates that a route may actually include domains not listed in the RoutedPath. If an LS, when presented with a set of overlapping routes from a peer LS, selects a less specific route without selecting the more specific route, then the LS MUST include the AtomicAggregate attribute with the route. An LS receiving a route with an AtomicAggregate attribute MUST NOT make the set of destinations more specific when advertising it to other LSs.

4.3.4.7. LocalPreference

The LocalPreference attribute is an intra-domain attribute used to inform other LSs of the local LSs preference for a given route. The preference of a route is calculated at the ingress to a domain and passed as an attribute with that route throughout the domain. Other LSs within the same ITAD use this attribute in their route selection process. This attribute has no significance between domains.

4.3.4.8. MultiExitDisc

There may be more than one LS peering relationship between neighboring domains. The MultiExitDisc attribute is used by an LS to express a preference for one link between the domains over another link between the domains. The use of the MultiExitDisc attribute is controlled by local policy.

4.3.4.9. Communities

The Communities attribute is a not well-known attribute used to facilitate and simplify the control of routing information by grouping destinations into communities.

4.3.4.10. ITAD Topology

The ITAD topology attribute is an intra-domain attribute that is used by LSs to indicate their intra-domain topology to other LSs in the domain.

4.3.4.11. ConvertedRoute

The ConvertedRoute attribute indicates that an intermediate LS has altered the route by changing the route's Application Protocol.

4.4. KEEPALIVE Message Format

TRIP does not use any transport-based keep-alive mechanism to determine if peers are reachable. Instead, KEEPALIVE messages are exchanged between peers often enough as not to cause the Hold Timer to expire. A reasonable maximum time between KEEPALIVE messages would be one third of the Hold Time interval. KEEPALIVE messages MUST NOT be sent more than once every 3 seconds. An implementation SHOULD adjust the rate at which it sends KEEPALIVE messages as a function of the negotiated Hold Time interval.

If the negotiated Hold Time interval is zero, then periodic KEEPALIVE messages MUST NOT be sent.

KEEPALIVE message consists of only message header and has a length of 3 octets.

4.5. NOTIFICATION Message Format

A NOTIFICATION message is sent when an error condition is detected. The TRIP transport connection is closed immediately after sending a NOTIFICATION message

In addition to the fixed-size TRIP header, the NOTIFICATION message contains the following fields:

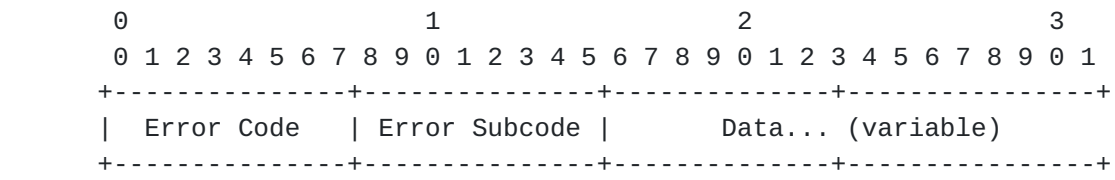

Figure 10: TRIP NOTIFICATION Format

Error Code:
This 1-octet unsigned integer indicates the type of NOTIFICATION. The following Error Codes have been defined:

Error Code	Symbolic Name	Reference
1	Message Header Error	Section 6.1
2	OPEN Message Error	Section 6.2
3	UPDATE Message Error	Section 6.3
4	Hold Timer Expired	Section 6.5
5	Finite State Machine Error	Section 6.6
6	Cease	Section 6.7

Error Subcode:
This 1-octet unsigned integer provides more specific information about the nature of the reported error. Each Error Code may have one or more Error Subcodes associated with it. If no appropriate Error Subcode is defined, then a zero (Unspecific) value is used for the Error Subcode field.

Message Header Error Subcodes:
- 1 - Bad Message Length.
- 2 - Bad Message Type.

OPEN Message Error Subcodes:
- 1 - Unsupported Version Number.
- 2 - Bad Peer ITAD.
- 3 - Bad TRIP Identifier.
- 4 - Unsupported Optional Parameter.
- 5 - Unacceptable Hold Time.
- 6 - Unsupported Capability.
- 7 - Capability Mismatch.

UPDATE Message Error Subcodes:
1 - Malformed Attribute List.
2 - Unrecognized Well-known Attribute.
3 - Missing Well-known Mandatory Attribute.
4 - Attribute Flags Error.
5 - Attribute Length Error.
6 - Invalid Attribute.

Data:
This variable-length field is used to diagnose the reason for the NOTIFICATION. The contents of the Data field depend upon the Error Code and Error Subcode.

Note that the length of the data can be determined from the message length field by the formula:

Data Length = Message Length - 5

The minimum length of the NOTIFICATION message is 5 octets (including message header).

5. TRIP Attributes

This section provides details on the syntax and semantics of each TRIP UPDATE attribute.

5.1. WithdrawnRoutes

Conditional Mandatory: False.
Required Flags: Well-known.
Potential Flags: Link-State Encapsulation (when flooding).
TRIP Type Code: 1

The WithdrawnRoutes attribute MUST be included in every UPDATE message. It specifies a set of routes that are to be removed from service by the receiving LS(s). The set of routes MAY be empty, indicated by a length field of zero.

5.1.1. Syntax of WithdrawnRoutes

The WithdrawnRoutes Attribute encodes a sequence of routes in its value field. The format for individual routes is given in Section 5.1.1.1. The WithdrawnRoutes Attribute lists the individual routes sequentially with no padding as shown in Figure 11. Each route includes a length field so that the individual routes within the

attribute can be delineated.

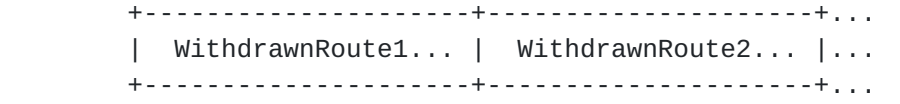

Figure 11: WithdrawnRoutes Format

5.1.1.1. Generic TRIP Route Format

The generic format for a TRIP route is given in Figure 12.

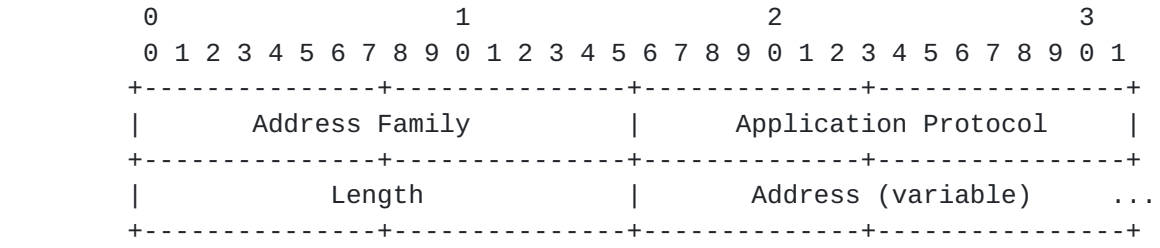

Figure 12: Generic TRIP Route Format

Address Family:
The address family field gives the type of address for the route. Two address families are defined in this Section:

Code	Address Family
1	Decimal Routing Numbers
2	PentaDecimal Routing Numbers
3	E.164 Numbers

This document reserves address family code 0. This document reserves address family codes 32768-65535 for vendor-specific applications (these are the codes with the first bit of the code value equal to 1).Additional address families may be defined in the future. Assignment of address family codes is controlled by IANA. See Section 13 for IANA considerations.

Application Protocol:
The application protocol gives the protocol for which this routing table is maintained. The currently defined application protocols are:

Code	Protocol
1	SIP
2	H.323-H.225.0-Q.931
3	H.323-H.225.0-RAS
4	H.323-H.225.0-Annex-G

This document reserves application protocol code 0. This document reserves application protocol codes 32768-65535 for vendor-specific applications (these are the codes with the first bit of the code value equal to 1). Additional application protocols may be defined in the future. Assignment of application protocol codes is controlled by IANA. See Section 13 for IANA considerations.

Length:
The length of the address field, in bytes.

Address:
This is an address (prefix) of the family type given by Address Family. The octet length of the address is variable and is determined by the length field of the route.

5.1.1.2. Decimal Routing Numbers

The Decimal Routing Numbers address family is a super set of all E.164 numbers, national numbers, local numbers, and private numbers. It can also be used to represent the decimal routing numbers used in conjunction with Number Portability in some countries/regions. A set of telephone numbers is specified by a Decimal Routing Number prefix. Decimal Routing Number prefixes are represented by a string of digits, each digit encoded by its ASCII character representation. This routing object covers all phone numbers starting with this prefix. The syntax for the Decimal Routing Number prefix is:

```
Decimal-routing-number  = *decimal-digit
decimal-digit           = DECIMAL-DIGIT
DECIMAL-DIGIT           = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

This DECIMAL Routing Number prefix is not bound in length. This format is similar to the format for a global telephone number as defined in SIP [8] without visual separators and without the "+" prefix for international numbers. This format facilitates efficient comparison when using TRIP to route SIP or H323, both of which use character based representations of phone numbers. The prefix length is determined from the length field of the route. The type of Decimal Routing Number (private, local, national, or international) can be deduced from the first few digits of the prefix.

5.1.1.3. PentaDecimal Routing Numbers

This address family is used to represent PentaDecimal Routing Numbers used in conjunction with Number Portability in some countries/regions. PentaDecimal Routing Number prefixes are

represented by a string of digits, each digit encoded by its ASCII character representation. This routing object covers all routing numbers starting with this prefix. The syntax for the PentaDecimal Routing Number prefix is:

```
PentaDecimal-routing-number   = *pentadecimal-digit
pentadecimal-routing-digit    = PENTADECIMAL-DIGIT
PENTADECIMAL-DIGIT            = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|
                                "8"|"9"|"A"|"B"|"C"|"D"|"E"
```

Note the difference in alphabets between Decimal Routing Numbers and PentaDecimal Routing Numbers. A PentaDecimal Routing Number prefix is not bound in length.

Note that the address family, which suits the routing numbers of a specific country/region depends on the alphabets used for routing numbers in that country/region. For example, North American routing numbers SHOULD use the Decimal Routing Numbers address family, because their alphabet is limited to the digits "0" through "9". Another example, in most European countries routing numbers use the alphabet "0" through "9" and "A" through "F", and hence these countries SHOULD use the PentaDecimal Routing Numbers address family.

5.1.1.4. E.164 Numbers

The E.164 Numbers address family is dedicated to fully qualified E.164 numbers. A set of telephone numbers is specified by a E.164 prefix. E.164 prefixes are represented by a string of digits, each digit encoded by its ASCII character representation. This routing object covers all phone numbers starting with this prefix. The syntax for the E.164 prefix is:

```
E164-number          = *e164-digit
E164-digit           = E164-DIGIT
E164-DIGIT           = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"
```

This format facilitates efficient comparison when using TRIP to route SIP or H323, both of which use character based representations of phone numbers. The prefix length is determined from the length field of the route.

The E.164 Numbers address family and the Decimal Routing Numbers address family have the same alphabet. The E.164 Numbers address family SHOULD be used whenever possible. The Decimal Routing Numbers address family can be used in case of private numbering plans or applications that do not desire to advertise fully expanded, fully qualified telephone numbers. If Decimal routing Numbers are used to

advertise non-fully qualified prefixes, the prefixes may have to be manipulated (e.g. expanded) at the boundary between ITADs. This adds significant complexity to the egress LS, because, it has to map the prefixes from the format used in its own ITAD to the format used in the peer ITAD.

5.2. ReachableRoutes

Conditional Mandatory: False.
Required Flags: Well-known.
Potential Flags: Link-State Encapsulation (when flooding).
TRIP Type Code: 2

The ReachableRoutes attribute MUST be included in every UPDATE message. It specifies a set of routes that are to be added to service by the receiving LS(s). The set of routes MAY be empty, this is indicated by setting the length field to zero.

5.2.1. Syntax of ReachableRoutes

The ReachableRoutes Attribute has the same syntax as the WithdrawnRoutes Attribute. See Section 5.1.1.

5.2.2. Route Origination and ReachableRoutes

Routes are injected into TRIP by a method outside the scope of this specification. Possible methods include a front-end protocol, an intra-domain routing protocol, or static configuration.

5.2.3. Route Selection and ReachableRoutes

The routes in ReachableRoutes are necessary for route selection.

5.2.4. Aggregation and ReachableRoutes

To aggregate multiple routes, the set of ReachableRoutes to be aggregated MUST combine to form a less specific set.

There is no mechanism within TRIP to communicate that a particular address prefix is not used and thus that these addresses could be skipped during aggregation. LSs MAY use methods outside of TRIP to learn of invalid prefixes that may be ignored during aggregation.

If an LS advertises an aggregated route, it MUST include the AtomicAggregate attribute.

5.2.5. Route Dissemination and ReachableRoutes

The ReachableRoutes attribute is recomputed at each LS except where flooding is being used (e.g., within a domain). It is therefore possible for an LS to change Application Protocol field of a route before advertising that route to an external peer.

If an LS changes the Application Protocol of a route it advertises, it MUST include the ConvertedRoute attribute in the UPDATE message.

5.2.6. Aggregation Specifics for Decimal Routing Numbers, E.164 Numbers, and PentaDecimal Routing Numbers

An LS that has routes to all valid numbers in a specific prefix SHOULD advertise that prefix as the ReachableRoutes, even if there are more specific prefixes that do not actually exist on the PSTN. Generally, it takes 10 Decimal Routing/E.164 prefixes, or 15 PentaDecimal Routing prefixes, of length n to aggregate into a prefix of length n-1. However, if an LS is aware that a prefix is an invalid Decimal Routing/E.164 prefix, or PentaDecimal Routing prefix, then the LS MAY aggregate by skipping this prefix. For example, if the Decimal Routing prefix 19191 is known not to exist, then an LS can aggregate to 1919 without 19191. A prefix representing an invalid set of PSTN destinations is sometimes referred to as a "black-hole." The method by which an LS is aware of black-holes is not within the scope of TRIP, but if an LS has such knowledge, it can use the knowledge when aggregating.

5.3. NextHopServer

Conditional Mandatory: True (if ReachableRoutes and/or WithdrawnRoutes attribute is present).
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 3.

Given a route with application protocol A and destinations D, the NextHopServer indicates the next-hop that messages of protocol A destined for D should be sent. This may or may not represent the ultimate destination of those messages.

5.3.1. NextHopServer Syntax

For generality, the address of the next-hop server may be of various types (domain name, IPv4, IPv6, etc). The NextHopServer attribute includes the ITAD number of next-hop server, a length field , and a next-hop name or address.

The syntax for the NextHopServer is given in Figure 13.

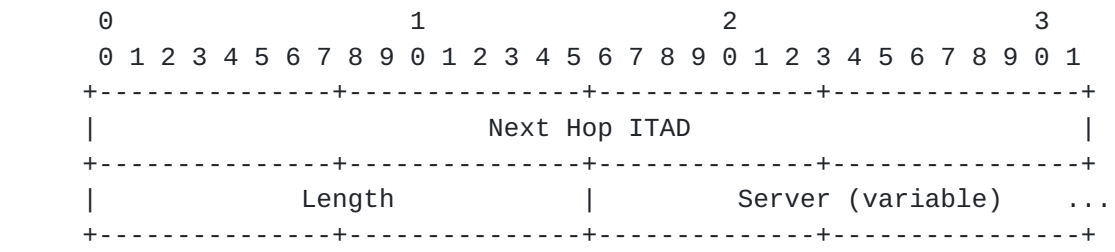

Figure 13: NextHopServer Syntax

The Next-Hop ITAD indicates the domain of the next-hop. Length field gives the number of octets in the Server field, and the Server field contains the name or address of the next-hop server. The server field is represented as a string of ASCII characters. It is defined as follows:

```
Server  = host [":" port ]
host    = <   A legal Internet host domain name
          or an IPv4 address using the textual representation
             defined in Section 2.1 of RFC 1123 [9]
          or an IPv6 address using the textual representation
             defined in Section 2.2 of RFC 2373 [10]. The IPv6
             address MUST be enclosed in "[" and "]"
             characters.>
port    = *DIGIT
```

If the port is empty or not given, the default port is assumed (e.g., port 5060 if the application protocol is SIP).

5.3.2. Route Origination and NextHopServer

When an LS originates a routing object into TRIP, it MUST include a NextHopServer within its domain. The NextHopServer could be an address of the egress gateway or of a signaling proxy.

5.3.3. Route Selection and NextHopServer

LS policy may prefer certain next-hops or next-hop domains over others.

5.3.4. Aggregation and NextHopServer

When aggregating multiple routing objects into a single routing object, an LS MUST insert a new signaling server from within its domain as the new NextHopServer unless all of the routes being aggregated have the same next-hop.

5.3.5. Route Dissemination and NextHopServer

When propagating routing objects to peers, an LS may choose to insert a signaling proxy within its domain as the new next-hop, or it may leave the next-hop unchanged. Inserting a new next-hop will cause the signaling messages to be sent to that address, and will provide finer control over the signaling path. Leaving the next-hop unchanged will yield a more efficient signaling path (fewer hops). It is a local policy decision of the LS to decide whether to propagate or change the NextHopServer.

5.4. AdvertisementPath

Conditional Mandatory: True (if ReachableRoutes and/or WithdrawnRoutes attribute is present).
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 4.

This attribute identifies the ITADs through which routing information carried in an advertisement has passed. The AdvertisementPath attribute is analogous to the AS_PATH attribute in BGP. The attributes differ in that BGP's AS_PATH also reflects the path to the destination. In TRIP, not every domain need modify the next-hop, so the AdvertisementPath may include many more hops than the actual path to the destination. The RoutedPath attribute (Section 5.5) reflects the actual path to the destination.

5.4.1. AdvertisementPath Syntax

AdvertisementPath is a variable length attribute that is composed of a sequence of ITAD path segments. Each ITAD path segment is represented by a type-length-value triple.

The path segment type is a 1-octet long field with the following values defined:

Value	Segment Type
1	AP_SET: unordered set of ITADs a route in the advertisement message has traversed
2	AP_SEQUENCE: ordered set of ITADs a route in the advertisement message has traversed

The path segment length is a 1-octet long field containing the number of ITADs in the path segment value field.

The path segment value field contains one or more ITAD numbers, each encoded as a 4-octets long field. ITAD numbers uniquely identify an Internet Telephony Administrative Domain, and must be obtained from IANA. See Section 13 for procedures to obtain an ITAD number from IANA.

5.4.2. Route Origination and AdvertisementPath

When an LS originates a route then:

- The originating LS shall include its own ITAD number in the AdvertisementPath attribute of all advertisements sent to LSs located in neighboring ITADs. In this case, the ITAD number of the originating LS's ITAD will be the only entry in the AdvertisementPath attribute.
- The originating LS shall include an empty AdvertisementPath attribute in all advertisements sent to LSs located in its own ITAD. An empty AdvertisementPath attribute is one whose length field contains the value zero.

5.4.3. Route Selection and AdvertisementPath

The AdvertisementPath may be used for route selection. Possible criteria to be used are the number of hops on the path and the presence or absence of particular ITADs on the path.

As discussed in Section 10, the AdvertisementPath is used to prevent routing information from looping. If an LS receives a route with its

own ITAD already in the AdvertisementPath, the route MUST be discarded.

5.4.4. Aggregation and AdvertisementPath

The rules for aggregating AdvertisementPath attributes are given in the following sections, where the term "path" used in Section 5.4.4.1 and 5.4.4.2 is understood to mean AdvertisementPath.

5.4.4.1. Aggregating Routes with Identical Paths

If all routes to be aggregated have identical path attributes, then the aggregated route has the same path attribute as the individual routes.

5.4.4.2. Aggregating Routes with Different Paths

For the purpose of aggregating path attributes we model each ITAD within the path as a pair <type, value>, where "type" identifies a type of the path segment (AP_SEQUENCE or AP_SET), and "value" is the ITAD number. Two ITADs are said to be the same if their corresponding <type, value> are the same.

If the routes to be aggregated have different path attributes, then the aggregated path attribute shall satisfy all of the following conditions:

- All pairs of the type AP_SEQUENCE in the aggregated path MUST appear in all of the paths of routes to be aggregated.
- All pairs of the type AP_SET in the aggregated path MUST appear in at least one of the paths of the initial set (they may appear as either AP_SET or AP_SEQUENCE types).
- For any pair X of the type AP_SEQUENCE that precedes pair Y in the aggregated path, X precedes Y in each path of the initial set that contains Y, regardless of the type of Y.
- No pair with the same value shall appear more than once in the aggregated path, regardless of the pair's type.

An implementation may choose any algorithm that conforms to these rules. At a minimum a conformant implementation MUST be able to perform the following algorithm that meets all of the above conditions:

- Determine the longest leading sequence of tuples (as defined above) common to all the paths of the routes to be aggregated. Make this sequence the leading sequence of the aggregated path.
- Set the type of the rest of the tuples from the paths of the routes to be aggregated to AP_SET, and append them to the aggregated path.
- If the aggregated path has more than one tuple with the same value (regardless of tuple's type), eliminate all but one such tuple by deleting tuples of the type AP_SET from the aggregated path.

  An implementation that chooses to provide a path aggregation algorithm that retains significant amounts of path information may wish to use the procedure of Section 5.4.4.3.

5.4.4.3. Example Path Aggregation Algorithm

An example algorithm to aggregate two paths works as follows:

- Identify the ITADs (as defined in Section 5.4.1) within each path attribute that are in the same relative order within both path attributes. Two ITADs, X and Y, are said to be in the same order if either X precedes Y in both paths, or if Y precedes X in both paths.
- The aggregated path consists of ITADs identified in (a) in exactly the same order as they appear in the paths to be aggregated. If two consecutive ITADs identified in (a) do not immediately follow each other in both of the paths to be aggregated, then the intervening ITADs (ITADs that are between the two consecutive ITADs that are the same) in both attributes are combined into an AP_SET path segment that consists of the intervening ITADs from both paths; this segment is then placed in between the two consecutive ITADs identified in (a) of the aggregated attribute. If two consecutive ITADs identified in (a) immediately follow each other in one attribute, but do not follow in another, then the intervening ITADs of the latter are combined into an AP_SET path segment; this segment is then placed in between the two consecutive ITADs identified in (a) of the aggregated path.

If as a result of the above procedure a given ITAD number appears more than once within the aggregated path, all, but the last instance (rightmost occurrence) of that ITAD number should be removed from the aggregated path.

5.4.5. Route Dissemination and AdvertisementPath

When an LS propagates a route which it has learned from another LS, it shall modify the route's AdvertisementPath attribute based on the location of the LS to which the route will be sent.

- When a LS advertises a route to another LS located in its own ITAD, the advertising LS MUST NOT modify the AdvertisementPath attribute associated with the route.
- When a LS advertises a route to an LS located in a neighboring ITAD, then the advertising LS MUST update the AdvertisementPath attribute as follows:

  * If the first path segment of the AdvertisementPath is of type AP_SEQUENCE, the local system shall prepend its own ITAD number as the last element of the sequence (put it in the leftmost position).
  * If the first path segment of the AdvertisementPath is of type AP_SET, the local system shall prepend a new path segment of type AP_SEQUENCE to the AdvertisementPath, including its own ITAD number in that segment.

5.5. RoutedPath

Conditional Mandatory: True (if ReachableRoutes attribute is present).
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 5.

This attribute identifies the ITADs through which messages sent using this route would pass. The ITADs in this path are a subset of those in the AdvertisementPath.

5.5.1. RoutedPath Syntax

The syntax of the RoutedPath attribute is the same as that of the AdvertisementPath attribute. See Section 5.4.1.

5.5.2. Route Origination and RoutedPath

When an LS originates a route it MUST include the RoutedPath attribute.

- The originating LS shall include its own ITAD number in the RoutedPath attribute of all advertisements sent to LSs located in neighboring ITADs. In this case, the ITAD number of the originating LS's ITAD will be the only entry in the RoutedPath attribute.
- The originating LS shall include an empty RoutedPath attribute in all advertisements sent to LSs located in its own ITAD. An empty RoutedPath attribute is one whose length field contains the value zero.

5.5.3. Route Selection and RoutedPath

The RoutedPath MAY be used for route selection, and in most cases is preferred over the AdvertisementPath for this role. Some possible criteria to be used are the number of hops on the path and the presence or absence of particular ITADs on the path.

5.5.4. Aggregation and RoutedPath

The rules for aggregating RoutedPath attributes are given in Section 5.4.4.1 and 5.4.4.2, where the term "path" used in Section 5.4.4.1 and 5.4.4.2 is understood to mean RoutedPath.

5.5.5. Route Dissemination and RoutedPath

When an LS propagates a route that it learned from another LS, it modifies the route's RoutedPath attribute based on the location of the LS to which the route is sent.

- When a LS advertises a route to another LS located in its own ITAD, the advertising LS MUST NOT modify the RoutedPath attribute associated with the route.
- If the LS has not changed the NextHopServer attribute, then the LS MUST NOT change the RoutedPath attribute.
- Otherwise, the LS changed the NextHopServer and is advertising the route to an LS in another ITAD. The advertising LS MUST update the RoutedPath attribute as follows:

  * If the first path segment of the RoutedPath is of type AP_SEQUENCE, the local system shall prepend its own ITAD number as the last element of the sequence (put it in the leftmost position).

* If the first path segment of the RoutedPath is of type AP_SET, the local system shall prepend a new path segment of type AP_SEQUENCE to the RoutedPath, including its own ITAD number in that segment.

5.6. AtomicAggregate

Conditional Mandatory: False.
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 6.

The AtomicAggregate attribute indicates that a route may traverse domains not listed in the RoutedPath. If an LS, when presented with a set of overlapping routes from a peer LS, selects the less specific route without selecting the more specific route, then the LS includes the AtomicAggregate attribute with the routing object.

5.6.1. AtomicAggregate Syntax

This attribute has length zero (0); the value field is empty.

5.6.2. Route Origination and AtomicAggregate

Routes are never originated with the AtomicAggregate attribute.

5.6.3. Route Selection and AtomicAggregate

The AtomicAggregate attribute may be used in route selection - it indicates that the RoutedPath may be incomplete.

5.6.4. Aggregation and AtomicAggregate

If any of the routes to aggregate has the AtomicAggregate attribute, then so MUST the resultant aggregate.

5.6.5. Route Dissemination and AtomicAggregate

If an LS, when presented with a set of overlapping routes from a peer LS, selects the less specific route (see Section 0) without selecting the more specific route, then the LS MUST include the AtomicAggregate attribute with the routing object (if it is not already present).

An LS receiving a routing object with an AtomicAggregate attribute MUST NOT make the set of destinations more specific when advertising it to other LSs, and MUST NOT remove the attribute when propagating this object to a peer LS.

5.7. LocalPreference

Conditional Mandatory: False.
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 7.

The LocalPreference attribute is only used intra-domain, it indicates the local LS's preference for the routing object to other LSs within the same domain. This attribute MUST NOT be included when communicating to an LS in another domain, and MUST be included over intra-domain links.

5.7.1. LocalPreference Syntax

The LocalPreference attribute is a 4-octet unsigned numeric value. A higher value indicates a higher preference.

5.7.2. Route Origination and LocalPreference

Routes MUST NOT be originated with the LocalPreference attribute to inter-domain peers. Routes to intra-domain peers MUST be originated with the LocalPreference attribute.

5.7.3. Route Selection and LocalPreference

The LocalPreference attribute allows one LS in a domain to calculate a preference for a route, and to communicate this preference to other LSs within the domain.

5.7.4. Aggregation and LocalPreference

The LocalPreference attribute is not affected by aggregation.

5.7.5. Route Dissemination and LocalPreference

An LS MUST include the LocalPreference attribute when communicating with peer LSs within its own domain. An LS MUST NOT include the LocalPreference attribute when communicating with LSs in other domains. LocalPreference attributes received from inter-domain peers MUST be ignored.

5.8. MultiExitDisc

Conditional Mandatory: False.
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 8.

When two ITADs are connected by more than one set of peers, the MultiExitDisc attribute may be used to specify preferences for routes received over one of those links versus routes received over other links. The MultiExitDisc parameter is used only for route selection.

5.8.1. MultiExitDisc Syntax

The MultiExitDisc attribute carries a 4-octet unsigned numeric value. A higher value represents a more preferred routing object.

5.8.2. Route Origination and MultiExitDisc

Routes originated to intra-domain peers MUST NOT be originated with the MultiExitDisc attribute. When originating a route to an inter-domain peer, the MultiExitDisc attribute may be included.

5.8.3. Route Selection and MultiExitDisc

The MultiExitDisc attribute is used to express a preference when there are multiple links between two domains. If all other factors are equal, then a route with a higher MultiExitDisc attribute is preferred over a route with a lower MultiExitDisc attribute.

5.8.4. Aggregation and MultiExitDisc

Routes with differing MultiExitDisc parameters MUST NOT be aggregated. Routes with the same value in the MultiExitDisc attribute MAY be aggregated and the same MultiExitDisc attribute attached to the aggregated object.

5.8.5. Route Dissemination and MultiExitDisc

If received from a peer LS in another domain, an LS MAY propagate the MultiExitDisc to other LSs within its domain. The MultiExitDisc attribute MUST NOT be propagated to LSs in other domains.

An LS may add the MultiExitDisc attribute when propagating routing objects to an LS in another domain. The inclusion of the MultiExitDisc attribute is a matter of policy, as is the value of the attribute.

5.9. Communities

Conditional Mandatory: False.
Required Flags: Not Well-Known, Independent Transitive.
Potential Flags: None.
TRIP Type Code: 9.

A community is a group of destinations that share some common property.

The Communities attribute is used to group destinations so that the routing decision can be based on the identity of the group. Using the Communities attribute should significantly simplify the distribution of routing information by providing an administratively defined aggregation unit.

Each ITAD administrator may define the communities to which a particular route belongs. By default, all routes belong to the general Internet Telephony community.

As an example, the Communities attribute could be used to define an alliance between a group of Internet Telephony service providers for a specific subset of routing information. In this case, members of that alliance would accept only routes for destinations in this group that are advertised by other members of the alliance. Other destinations would be more freely accepted. To achieve this, a member would tag each route with a designated Community attribute value before disseminating it. This relieves the members of such an

alliance from the responsibility of keeping track of the identities of all other members of that alliance.

Another example use of the Communities attribute is with aggregation. It is often useful to advertise both the aggregate route and the component more-specific routes that were used to form the aggregate. These component information are only useful to the neighboring TRIP peer, and perhaps the ITAD of the neighboring TRIP peer, so it is desirable to filter out the component routes. This can be achieved by specifying a Community attribute value that the neighboring peers will match and filter on. That way it can be assured that the more specific routes will not propagate beyond their desired scope.

5.9.1. Syntax of Communities

The Communities attribute is of variable length. It consists of set of 8-octet values, each of which specifies a community. The first 4 octets of the Community value are the Community ITAD Number and the next 4 octets are the Community ID.

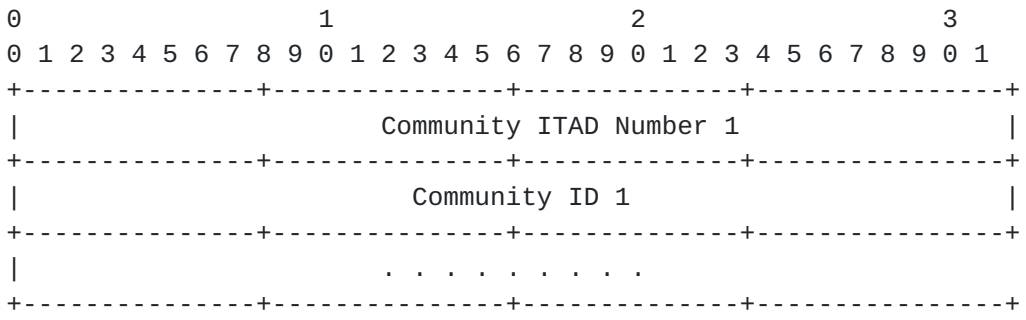

Figure 14: Communities Syntax

For administrative assignment, the following assumptions may be made:

> The Community attribute values starting with a Community ITAD Number of 0x00000000 are hereby reserved.

The following communities have global significance and their operation MUST be implemented in any Community attribute-aware TRIP LS.

- NO_EXPORT (Community ITAD Number = 0x00000000 and Community ID = 0xFFFFFF01). Any received route with a community attribute containing this value MUST NOT be advertised outside of the receiving TRIP ITAD.

Other community values MUST be encoded using an ITAD number in the

four most significant octets. The semantics of the final four octets (the Community ID octets) may be defined by the ITAD (e.g., ITAD 690 may define research, educational, and commercial community IDs that may be used for policy routing as defined by the operators of that ITAD).

5.9.2. Route Origination and Communities

The Communities attribute is not well-known. If a route has a Communities attribute associated with it, the LS MUST include that attribute in advertisement it originates.

5.9.3. Route Selection and Communities

The Communities attribute may be used for route selection. A route that is a member of a certain community may be preferred over another route that is not a member of that community. Likewise, routes without a certain community value may be excluded from consideration.

5.9.4. Aggregation and Communities

If a set of routes is to be aggregated and the resultant aggregate does not carry an Atomic_Aggregate attribute, then the resulting aggregate should have a Communities attribute that contains the union of the Community attributes of the aggregated routes.

5.9.5. Route Dissemination and Communities

An LS may manipulate the Communities attribute before disseminating a route to a peer. Community attribute manipulation may include adding communities, removing communities, adding a Communities attribute (if none exists), deleting the Communities attribute, etc.

5.10. ITAD Topology

Conditional Mandatory: False.
Required Flags: Well-known, Link-State encapsulated.
Potential Flags: None.
TRIP Type Code: 10.

Within an ITAD, each LS must know the status of other LSs so that LS failure can be detected. To do this, each LS advertises its internal topology to other LSs within the domain. When an LS detects that

another LS is no longer active, the information sourced by that LS can be deleted (the Adj-TRIB-In for that peer may be cleared). The ITAD Topology attribute is used to communicate this information to other LSs within the domain.

An LS MUST send a topology update each time it detects a change in its internal peer set. The topology update may be sent in an UPDATE message by itself or it may be piggybacked on an UPDATE message which includes ReachableRoutes and/or WithdrawnRoutes information.

When an LS receives a topology update from an internal LS, it MUST recalculate to which LSs are active within their domain via a connectivity algorithm on the topology.

5.10.1. ITAD Topology Syntax

The ITAD Topology attribute indicates the LSs with which the LS is currently peering. The attribute consists of a list of the TRIP Identifiers with which the LS is currently peering, the format is given in Figure 15. This attribute MUST use the link-state encapsulation as defined in Section 4.3.2.4.

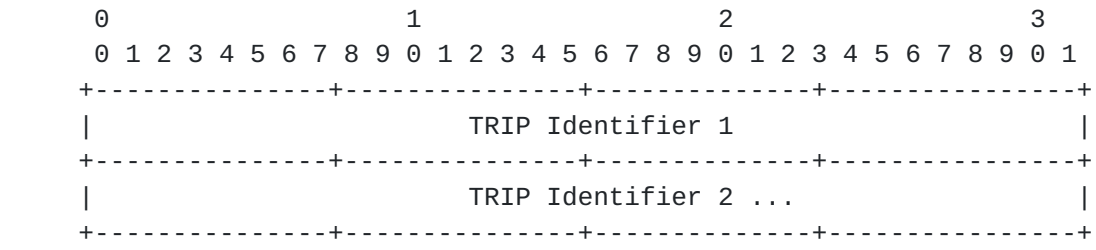

Figure 15: ITAD Topology Syntax

5.10.2. Route Origination and ITAD Topology

The ITAD Topology attribute is independent of any routes in the UPDATE. Whenever the set of internal peers of a LS changes, it MUST originate an UPDATE with the ITAD Topology Attribute included listing the current set of internal peers. The LS MUST include this attribute in the first UPDATE it sends to a peer after the peering session is established.

5.10.3. Route Selection and ITAD Topology

This attribute is independent of any routing information in the UPDATE. When an LS receives an UPDATE with an ITAD Topology attribute, it MUST compute the set of LSs currently active in the domain by performing a connectivity test on the ITAD topology as given by the set of originated ITAD Topology attributes. The LS MUST locally purge the Adj-TRIB-In for any LS that is no longer active in the domain. The LS MUST NOT propagate this purging information to other LSs as they will make a similar decision.

5.10.4. Aggregation and ITAD Topology

This information is not aggregated.

5.10.5. Route Dissemination and ITAD Topology

An LS MUST ignore the attribute if received from a peer in another domain. An LS MUST NOT send this attribute to an inter-domain peer.

5.11. ConvertedRoute

Conditional Mandatory: False.
Required Flags: Well-known.
Potential Flags: None.
TRIP Type Code: 12.

The ConvertedRoute attribute indicates that an intermediate LS has altered the route by changing the route's Application Protocol. For example, if an LS receives a route with Application Protocol X and changes the Application Protocol to Y before advertising the route to an external peer, the LS MUST include the ConvertedRoute attribute. The attribute is an indication that the advertised application protocol will not be used end-to-end, i.e., the information advertised about this route is not complete.

5.11.1. ConvertedRoute Syntax

This attribute has length zero (0); the value field is empty.

5.11.2. Route Origination and ConvertedRoute

Routes are never originated with the ConvertedRoute attribute.

5.11.3. Route Selection and ConvertedRoute

The ConvertedRoute attribute may be used in route selection - it indicates that advertised routing information is not complete.

5.11.4. Aggregation and ConvertedRoute

If any of the routes to aggregate has the ConvertedRoute attribute, then so MUST the resultant aggregate.

5.11.5. Route Dissemination and ConvertedRoute

If an LS changes the Application Protocol of route before advertising the route to an external peer, the LS MUST include the ConvertedRoute attribute.

5.12. Considerations for Defining New TRIP Attributes

Any proposal for defining new TRIP attributes should specify the following:

- the use of this attribute,
- the attribute's flags,
- the attribute's syntax,
- how the attribute works with route origination,
- how the attribute works with route aggregation, and
- how the attribute works with route dissemination and the attribute's scope (e.g., intra-domain only like LocalPreference)

IANA will manage the assignment of TRIP attribute type codes to new attributes.

6. TRIP Error Detection and Handling

This section describes errors to be detected and the actions to be taken while processing TRIP messages.

When any of the conditions described here are detected, a NOTIFICATION message with the indicated Error Code, Error Subcode, and Data fields MUST be sent, and the TRIP connection MUST be closed. If no Error Subcode is specified, then a zero Subcode MUST be used.

The phrase "the TRIP connection is closed" means that the transport protocol connection has been closed and that all resources for that TRIP connection have been de-allocated. If the connection was inter-domain, then routing table entries associated with the remote peer MUST be marked as invalid. Routing table entries MUST NOT be marked as invalid if an internal peering session is terminated. The fact that the routes have been marked as invalid is passed to other TRIP peers before the routes are deleted from the system.

Unless specified explicitly, the Data field of the NOTIFICATION message that is sent to indicate an error MUST be empty.

6.1. Message Header Error Detection and Handling

All errors detected while processing the Message Header are indicated by sending the NOTIFICATION message with Error Code Message Header Error. The Error Subcode elaborates on the specific nature of the error. The error checks in this section MUST be performed by each LS on receipt of every message.

If the Length field of the message header is less than 3 or greater than 4096, or if the Length field of an OPEN message is less than the minimum length of the OPEN message, or if the Length field of an UPDATE message is less than the minimum length of the UPDATE message, or if the Length field of a KEEPALIVE message is not equal to 3, or if the Length field of a NOTIFICATION message is less than the minimum length of the NOTIFICATION message, then the Error Subcode MUST be set to Bad Message Length. The Data field contains the erroneous Length field.

If the Type field of the message header is not recognized, then the Error Subcode MUST be set to "Bad Message Type." The Data field contains the erroneous Type field.

6.2. OPEN Message Error Detection and Handling

All errors detected while processing the OPEN message are indicated by sending the NOTIFICATION message with Error Code "OPEN Message Error." The Error Subcode elaborates on the specific nature of the error. The error checks in this section MUST be performed by each LS on receipt of every OPEN message.

If the version number contained in the Version field of the received OPEN message is not supported, then the Error Subcode MUST be set to "Unsupported Version Number." The Data field is a 1-octet unsigned integer, which indicates the largest locally supported version number less than the version the remote TRIP peer bid (as indicated in the received OPEN message).

If the ITAD field of the OPEN message is unacceptable, then the Error Subcode MUST be set to "Bad Peer ITAD." The determination of acceptable ITAD numbers is outside the scope of this protocol.

If the Hold Time field of the OPEN message is unacceptable, then the Error Subcode MUST be set to "Unacceptable Hold Time." An implementation MUST reject Hold Time values of one or two seconds. An implementation MAY reject any proposed Hold Time. An implementation that accepts a Hold Time MUST use the negotiated value for the Hold Time.

If the TRIP Identifier field of the OPEN message is not valid, then the Error Subcode MUST be set to "Bad TRIP Identifier." A TRIP identifier is 4-octets and can take any value. An LS considers the TRIP Identifier invalid if it has an already open connection with another peer LS that has the same ITAD and TRIP Identifier.

Any two LSs within the same ITAD MUST NOT have equal TRIP Identifier values. This restriction does not apply to LSs in different ITADs since the purpose is to uniquely identify an LS using its TRIP Identifier and its ITAD number.

If one of the Optional Parameters in the OPEN message is not recognized, then the Error Subcode MUST be set to "Unsupported Optional Parameters."

If the Optional Parameters of the OPEN message include Capability Information with an unsupported capability (unsupported in either capability type or value), then the Error Subcode MUST be set to "Unsupported Capability," and the entirety of the unsupported capabilities MUST be listed in the Data field of the NOTIFICATION message.

If the Optional Parameters of the OPEN message include Capability Information which do not match the receiving LS's capabilities, then the Error Subcode MUST be set to "Capability Mismatch," and the entirety of the mismatched capabilities MUST be listed in the Data field of the NOTIFICATION message.

6.3. UPDATE Message Error Detection and Handling

All errors detected while processing the UPDATE message are indicated by sending the NOTIFICATION message with Error Code "UPDATE Message Error." The Error Subcode elaborates on the specific nature of the error. The error checks in this section MUST be performed by each LS on receipt of every UPDATE message. These error checks MUST occur before flooding procedures are invoked with internal peers.

If any recognized attribute has Attribute Flags that conflict with the Attribute Type Code, then the Error Subcode MUST be set to "Attribute Flags Error." The Data field contains the erroneous attribute (type, length and value).

If any recognized attribute has Attribute Length that conflicts with the expected length (based on the attribute type code), then the Error Subcode MUST be set to "Attribute Length Error." The Data field contains the erroneous attribute (type, length and value).

If any of the mandatory (i.e., conditional mandatory attribute and the conditions for including it in the UPDATE message are fulfilled) well-known attributes are not present, then the Error Subcode MUST be set to "Missing Well-known Mandatory Attribute." The Data field contains the Attribute Type Code of the missing well-known conditional mandatory attributes.

If any of the well-known attributes are not recognized, then the Error Subcode MUST be set to "Unrecognized Well-known Attribute." The Data field contains the unrecognized attribute (type, length and value).

If any attribute has a syntactically incorrect value, or an undefined value, then the Error Subcode is set to "Invalid Attribute." The Data field contains the incorrect attribute (type, length and value). Such a NOTIFICATION message is sent, for example, when a NextHopServer attribute is received with an invalid address.

The information carried by the AdvertisementPath attribute is checked for ITAD loops. ITAD loop detection is done by scanning the full AdvertisementPath, and checking that the ITAD number of the local ITAD does not appear in the AdvertisementPath. If the local ITAD

number appears in the AdvertisementPath, then the route MAY be stored in the Adj-TRIB-In, but unless the LS is configured to accept routes with its own ITAD in the advertisement path, the route MUST not be passed to the TRIP Decision Process. The operation of an LS that is configured to accept routes with its own ITAD number in the advertisement path are outside the scope of this document.

If the UPDATE message was received from an internal peer and either the WithdrawnRoutes, ReachableRoutes, or ITAD Topology attribute does not have the Link-State Encapsulation flag set, then the Error Subcode is set to "Invalid Attribute" and the data field contains the attribute. Likewise, the attribute is invalid if received from an external peer and the Link-State Flag is set.

If any attribute appears more than once in the UPDATE message, then the Error Subcode is set to "Malformed Attribute List."

6.4. NOTIFICATION Message Error Detection and Handling

If a peer sends a NOTIFICATION message, and there is an error in that message, there is unfortunately no means of reporting this error via a subsequent NOTIFICATION message. Any such error, such as an unrecognized Error Code or Error Subcode, should be noticed, logged locally, and brought to the attention of the administration of the peer. The means to do this, however, are outside the scope of this document.

6.5. Hold Timer Expired Error Handling

If a system does not receive successive messages within the period specified by the negotiated Hold Time, then a NOTIFICATION message with "Hold Timer Expired" Error Code MUST be sent and the TRIP connection MUST be closed.

6.6. Finite State Machine Error Handling

An error detected by the TRIP Finite State Machine (e.g., receipt of an unexpected event) MUST result in sending a NOTIFICATION message with Error Code "Finite State Machine Error" and the TRIP connection MUST be closed.

6.7. Cease

In the absence of any fatal errors (that are indicated in this section), a TRIP peer MAY choose at any given time to close its TRIP connection by sending the NOTIFICATION message with Error Code "Cease." However, the Cease NOTIFICATION message MUST NOT be used when a fatal error indicated by this section exists.

6.8. Connection Collision Detection

If a pair of LSs try simultaneously to establish a transport connection to each other, then two parallel connections between this pair of speakers might well be formed. We refer to this situation as connection collision. Clearly, one of these connections must be closed.

Based on the value of the TRIP Identifier a convention is established for detecting which TRIP connection is to be preserved when a collision occurs. The convention is to compare the TRIP Identifiers of the peers involved in the collision and to retain only the connection initiated by the LS with the higher-valued TRIP Identifier.

Upon receipt of an OPEN message, the local LS MUST examine all of its connections that are in the OpenConfirm state. An LS MAY also examine connections in an OpenSent state if it knows the TRIP Identifier of the peer by means outside of the protocol. If among these connections there is a connection to a remote LS whose TRIP Identifier equals the one in the OPEN message, then the local LS MUST perform the following collision resolution procedure:

The TRIP Identifier and ITAD of the local LS is compared to the TRIP Identifier and ITAD of the remote LS (as specified in the OPEN message). TRIP Identifiers are treated as 4-octet unsigned integers for comparison.

If the value of the local TRIP Identifier is less than the remote one, or if the two TRIP Identifiers are equal and the value of ITAD of the local LS is less than value of the ITAD of the remote LS, then the local LS MUST close the TRIP connection that already exists (the one that is already in the OpenConfirm state), and accepts the TRIP connection initiated by the remote LS:

1. Otherwise, the local LS closes newly created TRIP connection continues to use the existing one (the one that is already in the OpenConfirm state).

2. If a connection collision occurs with an existing TRIP connection that is in the Established state, then the LS MUST unconditionally close of the newly created connection. Note that a connection collision cannot be detected with connections that are in Idle, Connect, or Active states.
3. To close the TRIP connection (that results from the collision resolution procedure), an LS MUST send a NOTIFICATION message with the Error Code "Cease" and the TRIP connection MUST be closed.

7. TRIP Version Negotiation

Peer LSs may negotiate the version of the protocol by making multiple attempts to open a TRIP connection, starting with the highest version number each supports. If an open attempt fails with an Error Code "OPEN Message Error" and an Error Subcode "Unsupported Version Number," then the LS has available the version number it tried, the version number its peer tried, the version number passed by its peer in the NOTIFICATION message, and the version numbers that it supports. If the two peers support one or more common versions, then this will allow them to rapidly determine the highest common version. In order to support TRIP version negotiation, future versions of TRIP must retain the format of the OPEN and NOTIFICATION messages.

8. TRIP Capability Negotiation

An LS MAY include the Capabilities Option in its OPEN message to a peer to indicate the capabilities supported by the LS. An LS receiving an OPEN message MUST NOT use any capabilities that were not included in the OPEN message of the peer when communicating with that peer.

9. TRIP Finite State Machine

This section specifies TRIP operation in terms of a Finite State Machine (FSM). Following is a brief summary and overview of TRIP operations by state as determined by this FSM. A condensed version of the TRIP FSM is found in Appendix 1. There is a TRIP FSM per peer and these FSMs operate independently.

Idle state:
Initially TRIP is in the Idle state for each peer. In this state, TRIP refuses all incoming connections. No resources are allocated to the peer. In response to the Start event (initiated by either the system or the operator), the local system initializes all TRIP

resources, starts the ConnectRetry timer, initiates a transport connection to the peer, starts listening for a connection that may be initiated by the remote TRIP peer, and changes its state to Connect. The exact value of the ConnectRetry timer is a local matter, but should be sufficiently large to allow TCP initialization.

If an LS detects an error, it closes the transport connection and changes its state to Idle. Transitioning from the Idle state requires generation of the Start event. If such an event is generated automatically, then persistent TRIP errors may result in persistent flapping of the LS. To avoid such a condition, Start events MUST NOT be generated immediately for a peer that was previously transitioned to Idle due to an error. For a peer that was previously transitioned to Idle due to an error, the time between consecutive Start events, if such events are generated automatically, MUST exponentially increase. The value of the initial timer SHOULD be 60 seconds, and the time SHOULD be at least doubled for each consecutive retry up to some maximum value.

Any other event received in the Idle state is ignored.

Connect state:
In this state, an LS is waiting for a transport protocol connection to be completed to the peer, and is listening for inbound transport connections from the peer.

If the transport protocol connection succeeds, the local LS clears the ConnectRetry timer, completes initialization, sends an OPEN message to its peer, sets its Hold Timer to a large value, and changes its state to OpenSent. A Hold Timer value of 4 minutes is suggested.

If the transport protocol connect fails (e.g., retransmission timeout), the local system restarts the ConnectRetry timer, continues to listen for a connection that may be initiated by the remote LS, and changes its state to Active state.

In response to the ConnectRetry timer expired event, the local LS cancels any outstanding transport connection to the peer, restarts the ConnectRetry timer, initiates a transport connection to the remote LS, continues to listen for a connection that may be initiated by the remote LS, and stays in the Connect state.

If the local LS detects that a remote peer is trying to establish a connection to it and the IP address of the peer is not an expected one, then the local LS rejects the attempted connection and continues to listen for a connection from its expected peers without changing state.

If an inbound transport protocol connection succeeds, the local LS clears the ConnectRetry timer, completes initialization, sends an OPEN message to its peer, sets its Hold Timer to a large value, and changes its state to OpenSent. A Hold Timer value of 4 minutes is suggested.

The Start event is ignored in the Connect state.

In response to any other event (initiated by either the system or the operator), the local system releases all TRIP resources associated with this connection and changes its state to Idle.

Active state:
In this state, an LS is listening for an inbound connection from the peer, but is not in the process of initiating a connection to the peer.

If an inbound transport protocol connection succeeds, the local LS clears the ConnectRetry timer, completes initialization, sends an OPEN message to its peer, sets its Hold Timer to a large value, and changes its state to OpenSent. A Hold Timer value of 4 minutes is suggested.

In response to the ConnectRetry timer expired event, the local system restarts the ConnectRetry timer, initiates a transport connection to the TRIP peer, continues to listen for a connection that may be initiated by the remote TRIP peer, and changes its state to Connect.

If the local LS detects that a remote peer is trying to establish a connection to it and the IP address of the peer is not an expected one, then the local LS rejects the attempted connection and continues to listen for a connection from its expected peers without changing state.

Start event is ignored in the Active state.

In response to any other event (initiated by either the system or the operator), the local system releases all TRIP resources associated with this connection and changes its state to Idle.

OpenSent state:
In this state, an LS has sent an OPEN message to its peer and is waiting for an OPEN message from its peer. When an OPEN message is received, all fields are checked for correctness. If the TRIP message header checking or OPEN message checking detects an error (see Section 6.2) or a connection collision (see Section 6.8), the local system sends a NOTIFICATION message and changes its state to Idle.

If there are no errors in the OPEN message, TRIP sends a KEEPALIVE message and sets a KeepAlive timer. The Hold Timer, which was originally set to a large value (see above), is replaced with the negotiated Hold Time value (see Section 4.2). If the negotiated Hold Time value is zero, then the Hold Time timer and KeepAlive timers are not started. If the value of the ITAD field is the same as the local ITAD number, then the connection is an "internal" connection; otherwise, it is "external" (this will affect UPDATE processing). Finally, the state is changed to OpenConfirm.

If the local LS detects that a remote peer is trying to establish a connection to it and the IP address of the peer is not an expected one, then the local LS rejects the attempted connection and continues to listen for a connection from its expected peers without changing state.

If a disconnect notification is received from the underlying transport protocol, the local LS closes the transport connection, restarts the ConnectRetry timer, continues to listen for a connection that may be initiated by the remote TRIP peer, and goes into the Active state.

If the Hold Timer expires, the local LS sends NOTIFICATION message with Error Code "Hold Timer Expired" and changes its state to Idle.

In response to the Stop event (initiated by either system or operator) the local LS sends NOTIFICATION message with Error Code "Cease" and changes its state to Idle.

The Start event is ignored in the OpenSent state.

In response to any other event the local LS sends NOTIFICATION message with Error Code "Finite State Machine Error" and changes its state to Idle.

Whenever TRIP changes its state from OpenSent to Idle, it closes the transport connection and releases all resources associated with that connection.

OpenConfirm state:
In this state, an LS has sent an OPEN to its peer, received an OPEN from its peer, and sent a KEEPALIVE in response to the OPEN. The LS is now waiting for a KEEPALIVE or NOTIFICATION message in response to its OPEN.

If the local LS receives a KEEPALIVE message, it changes its state to Established.

If the Hold Timer expires before a KEEPALIVE message is received, the local LS sends NOTIFICATION message with Error Code "Hold Timer Expired" and changes its state to Idle.

If the local LS receives a NOTIFICATION message, it changes its state to Idle.

If the KeepAlive timer expires, the local LS sends a KEEPALIVE message and restarts its KeepAlive timer.

If a disconnect notification is received from the underlying transport protocol, the local LS closes the transport connection, restarts the ConnectRetry timer, continues to listen for a connection that may be initiated by the remote TRIP peer, and goes into the Active state.

In response to the Stop event (initiated by either the system or the operator) the local LS sends NOTIFICATION message with Error Code "Cease" and changes its state to Idle.

Start event is ignored in the OpenConfirm state.

In response to any other event the local LS sends NOTIFICATION message with Error Code "Finite State Machine Error" and changes its state to Idle.

Whenever TRIP changes its state from OpenConfirm to Idle, it closes the transport connection and releases all resources associated with that connection.

Established state:
In the Established state, an LS can exchange UPDATE, NOTIFICATION, and KEEPALIVE messages with its peer.

If the negotiated Hold Timer is zero, then no procedures are necessary for keeping a peering session alive. If the negotiated Hold Time value is non-zero, the procedures of this paragraph apply. If the Hold Timer expires, the local LS sends a NOTIFICATION message with Error Code "Hold Timer Expired" and changes its state to Idle. If the KeepAlive Timer expires, then the local LS sends a KeepAlive message and restarts the KeepAlive Timer. If the local LS receives an UPDATE or KEEPALIVE message, then it restarts its Hold Timer. Each time the LS sends an UPDATE or KEEPALIVE message, it restarts its KeepAlive Timer.

If the local LS receives a NOTIFICATION message, it changes its state to Idle.

If the local LS receives an UPDATE message and the UPDATE message error handling procedure (see Section6.3) detects an error, the local LS sends a NOTIFICATION message and changes its state to Idle.

If a disconnect notification is received from the underlying transport protocol, the local LS changes its state to Idle.

In response to the Stop event (initiated by either the system or the operator), the local LS sends a NOTIFICATION message with Error Code "Cease" and changes its state to Idle.

The Start event is ignored in the Established state.

In response to any other event, the local LS sends NOTIFICATION message with Error Code "Finite State Machine Error" and changes its state to Idle.

Whenever TRIP changes its state from Established to Idle, it closes the transport) connection, releases all resources associated with that connection. Additionally, if the peer is an external peer, the LS deletes all routes derived from that connection.

10. UPDATE Message Handling

An UPDATE message may be received only in the Established state. When an UPDATE message is received, each field is checked for validity as specified in Section 6.3. The rest of this section presumes that the UPDATE message has passed the error-checking procedures of Section 6.3.

If the UPDATE message was received from an internal peer, the flooding procedures of Section 10.1 MUST be applied. The flooding process synchronizes the Loc-TRIBs of all LSs within the domain. Certain routes within the UPDATE may be marked as old or duplicates by the flooding process and are ignored during the rest of the UPDATE processing.

If the UPDATE message contains withdrawn routes, then the corresponding previously advertised routes shall be removed from the Adj-TRIB-In. This LS MUST run its Decision Process since the previously advertised route is no longer available for use.

If the UPDATE message contains a route, then the route MUST be placed in the appropriate Adj-TRIB-In, and the following additional actions MUST be taken:

1. If its destinations are identical to those of a route currently stored in the Adj-TRIB-In, then the new route MUST replace the older route in the Adj-TRIB-In, thus implicitly withdrawing the older route from service. The LS MUST run its Decision Process since the older route is no longer available for use.
2. If the new route is more specific than an earlier route contained in the Adj-TRIB-In and has identical attributes, then no further actions are necessary.
3. If the new route is more specific than an earlier route contained in the Adj-TRIB-In but does not have identical attributes, then the LS MUST run its Decision Process since the more specific route has implicitly made a portion of the less specific route unavailable for use.
4. If the new route has destinations that are not present in any of the routes currently stored in the Adj-TRIB-In, then the LS MUST run its Decision Process.
5. If the new route is less specific than an earlier route contained in the Adj-TRIB-In, the LS MUST run its Decision Process on the set of destinations that are described only by the less specific route.

10.1. Flooding Process

When an LS receives an UPDATE message from an internal peer, the LS floods the new information from that message to all of its other internal peers. Flooding is used to efficiently synchronize all of the LSs within a domain without putting any constraints on the domain's internal topology. The flooding mechanism is based on the techniques used in OSPF [4] and SCSP [6]. One may argue that TRIP's flooding process is in reality a controlled broadcast mechanism.

10.1.1. Database Information

The LS MUST maintain the sequence number and originating TRIP identifier for each link-state encapsulated attribute in an internal Adj-TRIB-In. These values are included with the route in the ReachableRoutes, WithdrawnRoutes, and ITAD Topology attributes. The originating TRIP identifier gives the internal LS that originated this route into the ITAD, the sequence number gives the version of this route at the originating LS.

10.1.2. Determining Newness

For each route in the ReachableRoutes or WithdrawnRoutes field, the LS decides if the route is new or old. This is determined by comparing the Sequence Number of the route in the UPDATE with the Sequence Number of the route saved in the Adj-TRIB-In. The route is new if either the route does not exist in the Adj-TRIB-In for the originating LS, or if the route does exist in the Adj-TRIB-In but the Sequence Number in the UPDATE is greater than the Sequence Number saved in the Adj-TRIBs-In. Note that the newness test is independently applied to each link-state encapsulated attribute in the UPDATE (WithdrawnRoutes or ReachableRoutes).

10.1.3. Flooding

Each route in the ReachableRoutes or WithdrawnRoutes field that is determined to be old is ignored in further processing. If the route is determined to be new then the following actions occur.

If the route is being withdrawn, then the LS MUST flood the withdrawn route to all other internal peers, and MUST mark the route as withdrawn. An LS MUST maintain routes marked as withdrawn in its databases for MaxPurgeTime seconds.

If the route is being updated, then the LS MUST update the route in the Adj-TRIB-In and MUST flood it to all other internal peers.

If these procedures result in changes to the Adj-TRIB-In, then the route is also made available for local route processing as described early in Section 10.

To implement flooding, the following is recommended. All routes received in a single UPDATE message that are determined to be new should be forwarded to all other internal peers in a single UPDATE message. Other variations on flooding are possible, but the local LS MUST ensure that each new route (and any associated attributes) received from an internal peer get forwarded to every other internal peer.

10.1.4. Sequence Number Considerations

The Sequence Number is used to determine when one version of a Route is newer than another version of a route. A larger Sequence Number indicates a newer version. The Sequence Number is assigned by the LS originating the route into the local ITAD. The Sequence Number is an unsigned 4-octet integer in the range of 1 thru 2^31-1 MinSequenceNum

thru MaxSequenceNum). The value 0 is reserved. When an LS first originates a route (including when the LS restarts/reboots) into its ITAD, it MUST originate it with a Sequence Number of MinSequenceNum. Each time the route is updated within the ITAD by the originator, the Sequence Number MUST be increased.

If it is ever the case that the sequence number is MaxSequenceNum-1 and it needs to be increased, then the TRIP module of the LS MUST be disabled for a period of TripDisableTime so that all routes originated by this LS with high sequence numbers can be removed.

10.1.5. Purging a Route Within the ITAD

To withdraw a route that it originated within the ITAD, an LS includes the route in the WithdrawnRoutes field of an UPDATE message. The Sequence Number MUST be greater than the last valid version of the route. The LS MAY choose to use a sequence number of MaxSequenceNum when withdrawing routes within its ITAD, but this is not required.

After withdrawing a route, an LS MUST mark the route as "withdrawn" in its database, and maintain the withdrawn route in its database for MaxPurgeTime seconds. If the LS needs to re-originate a route that had been purged but is still in its database, it can either re-originate the route immediately using a Sequence Number that is greater than that used in the withdraw, or the LS may wait until MaxPurgeTime seconds have expired since the route was withdrawn.

10.1.6. Receiving Self-Originated Routes

It is common for an LS to receive UPDATES for routes that it originated within the ITAD via the flooding procedure. If the LS receives an UPDATE for a route that it originated that is newer (has a higher sequence number) than the LSs current version, then special actions must be taken. This should be a relatively rare occurrence and indicates that a route still exists within the ITAD since the LSs last restart/reboot.

If an LS receives a self-originated route update that is newer than the current version of the route at the LS, then the following actions MUST be taken. If the LS still wishes to advertise the information in the route, then the LS MUST increase the Sequence Number of the route to a value greater than that received in the UPDATE and re-originate the route. If the LS does not wish to continue to advertise the route, then it MUST purge the route as described in Section 10.1.5.

10.1.7. Removing Withdrawn Routes

An LS SHOULD ensure that routes marked as withdrawn are removed from the database in a timely fashion after the MaxPurgeTime has expired. This could be done, for example, by periodically sweeping the database, and deleting those entries that were withdrawn more than MaxPurgeTime seconds ago.

10.2. Decision Process

The Decision Process selects routes for subsequent advertisement by applying the policies in the local Policy Information Base (PIB) to the routes stored in its Adj-TRIBs-In. The output of the Decision Process is the set of routes that will be advertised to all peers; the selected routes will be stored in the local LS's Adj-TRIBs-Out.

The selection process is formalized by defining a function that takes the attributes of a given route as an argument and returns a non-negative integer denoting the degree of preference for the route. The function that calculates the degree of preference for a given route shall not use as its inputs any of the following: the existence of other routes, the non-existence of other routes, or the attributes of other routes. Route selection then consists of individual application of the degree of preference function to each feasible route, followed by the choice of the one with the highest degree of preference.

All internal LSs in an ITAD MUST run the Decision Process and apply the same decision criteria, otherwise it will not be possible to synchronize their Loc-TRIBs.

The Decision Process operates on routes contained in each Adj-TRIBs-In, and is responsible for:

- selection of routes to be advertised to internal peers
- selection of routes to be advertised to external peers
- route aggregation and route information reduction

The Decision Process takes place in three distinct phases, each triggered by a different event:

- Phase 1 is responsible for calculating the degree of preference for each route received from an external peer.
- Phase 2 is invoked on completion of phase 1. It is responsible for choosing the best route out of all those available for each distinct destination, and for installing each chosen route into the Loc-TRIB.

- Phase 3 is invoked after the Loc-TRIB has been modified. It is responsible for disseminating routes in the Loc-TRIB to each external peer, according to the policies contained in the PIB. Route aggregation and information reduction can optionally be performed within this phase.

10.2.1. Phase 1: Calculation of Degree of Preference

The Phase 1 decision function shall be invoked whenever the local LS receives from a peer an UPDATE message that advertises a new route, a replacement route, or a withdrawn route.

The Phase 1 decision function is a separate process that completes when it has no further work to do.

The Phase 1 decision function shall lock an Adj-TRIB-In prior to operating on any route contained within it, and shall unlock it after operating on all new or replacement routes contained within it.

The local LS MUST determine a degree of preference for each newly received or replacement route. If the route is learned from an internal peer, the value of the LocalPreference attribute MUST be taken as the degree of preference. If the route is learned from an external peer, then the degree of preference MUST be computed based on pre-configured policy information and used as the LocalPreference value in any intra-domain TRIP advertisement. The exact nature of this policy information and the computation involved is a local matter.

The output of the degree of preference determination process is the local preference of a route. The local LS computes the local preference of routes learned from external peers or originated internally at that LS. The local preference of a route learned from an internal peer is included in the LocalPreference attribute associated with that route.

10.2.2. Phase 2: Route Selection

The Phase 2 decision function shall be invoked on completion of Phase 1. The Phase 2 function is a separate process that completes when it has no further work to do. Phase 2 consists of two sub- phases: 2a and 2b. The same route selection function is applied in both sub-phases, but the inputs to each phase are different. The Phase 2a process MUST consider as inputs all external routes, that are present in the Adj-TRIBs-In of external peers, and all local routes. The output of Phase 2a is inserted into the Ext-TRIB. The Phase 2b

process shall be invoked upon completion of Phase 2a and it MUST consider as inputs all routes in the Ext-TRIB and all routes that are present in the Adj-TRIBs-In of internal LSs. The output of Phase 2b is stored in the Loc-TRIB.

The Phase 2 decision function MUST be blocked from running while the Phase 3 decision function is in process. The Phase 2 function MUST lock all Adj-TRIBs-In and the Ext-TRIB prior to commencing its function, and MUST unlock them on completion.

If the LS determines that the NextHopServer listed in a route is unreachable, then the route MAY be excluded from the Phase 2 decision function. The means by which such a determination is made is not mandated here.

For each set of destinations for which one or more routes exist, the local LS's route selection function MUST identify the route that has:

- the highest degree of preference, or
- is selected as a result of the tie breaking rules specified in 10.2.2.1.

Withdrawn routes MUST be removed from the Loc-TRIB, Ext-TRIB, and the Adj-TRIBs-In.

10.2.2.1. Breaking Ties (Phase 2)

Several routes to the same destination that have the same degree of preference may be input to the Phase 2 route selection function. The local LS can select only one of these routes for inclusion in the associated Ext-TRIB (Phase 2a) or Loc-TRIB (Phase 2b). The local LS considers all routes with the same degrees of preference. The following algorithm shall be used to break ties.

- If the local LS is configured to use the MultiExitDisc attribute to break ties, and candidate routes received from the same neighboring ITAD differ in the value of the MultiExitDisc attribute, then select the route that has the larger value of MultiExitDisc.
- If at least one of the routes was originated by an internal LS, select the route route that was advertised by the internal LS that has the lowest TRIP ID.
- Otherwise, select the route that was advertised by the neighbor domain that has the lowest ITAD number.

10.2.3. Phase 3: Route Dissemination

The Phase 3 decision function MUST be invoked upon completion of Phase 2 if Phase 2 results in changes to the Loc-TRIB or when a new LS-to-LS peer session is established.

The Phase 3 function is a separate process that completes when it has no further work to do. The Phase 3 routing decision function MUST be blocked from running while the Phase 2 decision function is in process.

All routes in the Loc-TRIB shall be processed into a corresponding entry in the associated Adj-TRIBs-Out. Route aggregation and information reduction techniques (see 10.3.4) MAY optionally be applied.

When the updating of the Adj-TRIBs-Out is complete, the local LS MUST run the external update process of 10.3.2.

10.2.4. Overlapping Routes

When overlapping routes are present in the same Adj-TRIB-In, the more specific route shall take precedence, in order from more specific to least specific.

The set of destinations described by the overlap represents a portion of the less specific route that is feasible, but is not currently in use. If a more specific route is later withdrawn, the set of destinations described by the more specific route will still be reachable using the less specific route.

If an LS receives overlapping routes, the Decision Process MUST take into account the semantics of the overlapping routes. In particular, if an LS accepts the less specific route while rejecting the more specific route from the same peer, then the destinations represented by the overlap may not forward along the domains listed in the AdvertisementPath attribute of that route. Therefore, an LS has the following choices:

1. Install both the less and the more specific routes
2. Install the more specific route only
3. Install the non-overlapping part of the less specific route only (that implies disaggregation of the less-specific route)
4. Aggregate the two routes and install the aggregated route

5. Install the less specific route only
6. Install neither route

If an LS chooses 5), then it SHOULD add AtomicAggregate attribute to the route. A route that carries AtomicAggregate attribute MUST NOT be de-aggregated. That is, the route cannot be made more specific. Forwarding along such a route does not guarantee that route traverses only domains listed in the RoutedPath of the route. If an LS chooses 1), then it MUST NOT advertise the more general route without the more specific route.

10.3. Update-Send Process

The Update-Send process is responsible for advertising UPDATE messages to all peers. For example, it distributes the routes chosen by the Decision Process to other LSs that may be located in either the same ITAD or a neighboring ITAD. Rules for information exchange between peer LSs located in different ITADs are given in 10.3.2; rules for information exchange between peer LSs located in the same ITAD are given in 10.3.1.

Before forwarding routes to peers, an LS MUST determine which attributes should be forwarded along with that route. If a not well-known non-transitive attribute is unrecognized, it is quietly ignored. If a not well-known dependent-transitive attribute is unrecognized, and the NextHopServer attribute has been changed by the LS, the unrecognized attribute is quietly ignored. If a not well-known dependent-transitive attribute is unrecognized, and the NextHopServer attribute has not been modified by the LS, the Partial bit in the attribute flags octet is set to 1, and the attribute is retained for propagation to other TRIP speakers. Similarly, if an not well-known independent-transitive attribute is unrecognized, the Partial bit in the attribute flags octet is set to 1, and the attribute is retained for propagation to other TRIP speakers.

If a not well-known attribute is recognized, and has a valid value, then, depending on the type of the not well-known attribute, it is updated, if necessary, for possible propagation to other TRIP speakers.

10.3.1. Internal Updates

The Internal update process is concerned with the distribution of routing information to internal peers.

When an LS receives an UPDATE message from another TRIP LS located in

its own ITAD, it is flooded as described in Section 10.1.

When an LS receives a new route from an LS in a neighboring ITAD, or if a local route is injected into TRIP, the LS determines the preference of that route. If the new route has the highest degree of preference for all external routes and local routes to a given destination (or if the route was selected via a tie-breaking procedure as specified in 10.3.1.1), the LS MUST insert that new route into the Ext-TRIB database and the LS MUST advertise that route to all other LSs in its ITAD by means of an UPDATE message. The LS MUST advertise itself as the Originator of that route within the ITAD.

When an LS receives an UPDATE message with a non-empty WithdrawnRoutes attribute from an external peer, or if a local route is withdrawn from TRIP, the LS MUST remove from its Adj-TRIB-In all routes whose destinations were carried in this field. If the withdrawn route was previously selected into the Ext-TRIB, the LS MUST take the following additional steps:

- If a new route is selected for advertisement for those destinations, then the LS MUST insert the replacement route into Ext-TRIB to replace the withdrawn route and advertise it to all internal LSs.
- If a replacement route is not available for advertisement, then the LS MUST include the destinations of the route in the WithdrawnRoutes attribute of an UPDATE message, and MUST send this message to each internal peer. The LS MUST also remove the withdrawn route from the Ext-TRIB.

10.3.1.1. Breaking Ties (Routes Received from External Peers)

If an LS has connections to several external peers, there will be multiple Adj-TRIBs-In associated with these peers. These databases might contain several equally preferable routes to the same destination, all of which were advertised by external peers. The local LS shall select one of these routes according to the following rules:

- If the LS is configured to use the MultiExitDisc attribute to break ties, and the candidate routes differ in the value of the MultiExitDisc attribute, then select the route that has the lowest value of MultiExitDisc, else
- Select the route that was advertised by the external LS that has the lowest TRIP Identifier.

10.3.2. External Updates

The external update process is concerned with the distribution of routing information to external peers. As part of Phase 3 route selection process, the LS has updated its Adj-TRIBs-Out. All newly installed routes and all newly unfeasible routes for which there is no replacement route MUST be advertised to external peers by means of UPDATE messages.

Any routes in the Loc-TRIB marked as withdrawn MUST be removed. Changes to the reachable destinations within its own ITAD SHALL also be advertised in an UPDATE message.

10.3.3. Controlling Routing Traffic Overhead

The TRIP protocol constrains the amount of routing traffic (that is, UPDATE messages) in order to limit both the link bandwidth needed to advertise UPDATE messages and the processing power needed by the Decision Process to digest the information contained in the UPDATE messages.

10.3.3.1. Frequency of Route Advertisement

The parameter MinRouteAdvertisementInterval determines the minimum amount of time that must elapse between advertisements of routes to a particular destination from a single LS. This rate limiting procedure applies on a per-destination basis, although the value of MinRouteAdvertisementInterval is set on a per LS peer basis.

Two UPDATE messages sent from a single LS that advertise feasible routes to some common set of destinations received from external peers MUST be separated by at least MinRouteAdvertisementInterval. Clearly, this can only be achieved precisely by keeping a separate timer for each common set of destinations. This would be unwarranted overhead. Any technique which ensures that the interval between two UPDATE messages sent from a single LS that advertise feasible routes to some common set of destinations received from external peers will be at least MinRouteAdvertisementInterval, and will also ensure a constant upper bound on the interval is acceptable.

Two UPDATE messages, sent from a single LS to an external peer, that advertise feasible routes to some common set of destinations received from internal peers MUST be separated by at least MinRouteAdvertisementInterval.

Since fast convergence is needed within an ITAD, this rate limiting

procedure does not apply to routes received from internal peers and being broadcast to other internal peers. To avoid long-lived black holes, the procedure does not apply to the explicit withdrawal of routes (that is, routes whose destinations explicitly withdrawn by UPDATE messages.

This procedure does not limit the rate of route selection, but only the rate of route advertisement. If new routes are selected multiple times while awaiting the expiration of MinRouteAdvertisementInterval, the last route selected shall be advertised at the end of MinRouteAdvertisementInterval.

10.3.3.2. Frequency of Route Origination

The parameter MinITADOriginationInterval determines the minimum amount of time that must elapse between successive advertisements of UPDATE messages that report changes within the advertising LS's own ITAD.

10.3.3.3. Jitter

To minimize the likelihood that the distribution of TRIP messages by a given LS will contain peaks, jitter should be applied to the timers associated with MinITADOriginationInterval, KeepAlive, and MinRouteAdvertisementInterval. A given LS shall apply the same jitter to each of these quantities regardless of the destinations to which the updates are being sent; that is, jitter will not be applied on a "per peer" basis.

The amount of jitter to be introduced shall be determined by multiplying the base value of the appropriate timer by a random factor that is uniformly distributed in the range from 0.75 to 1.0.

10.3.4. Efficient Organization of Routing Information

Having selected the routing information that it will advertise, a TRIP speaker may use methods to organize this information in an efficient manner. These methods are discussed in the following sections.

10.3.4.1. Information Reduction

Information reduction may imply a reduction in granularity of policy control - after information is collapsed, the same policies will

apply to all destinations and paths in the equivalence class.

The Decision Process may optionally reduce the amount of information that it will place in the Adj-TRIBs-Out by any of the following methods:

- ReachableRoutes: A set of destinations can be usually represented in compact form. For example, a set of E.164 phone numbers can be represented in more compact form using E.164 prefixes.
- AdvertisementPath: AdvertisementPath information can be represented as ordered AP_SEQUENCEs or unordered AP_SETs. AP_SETs are used in the route aggregation algorithm described in Section 5.4.4. They reduce the size of the AP_PATH information by listing each ITAD number only once, regardless of how many times it may have appeared in multiple advertisement paths that were aggregated.

An AP_SET implies that the destinations advertised in the UPDATE message can be reached through paths that traverse at least some of the constituent ITADs. AP_SETs provide sufficient information to avoid route looping; however their use may prune potentially feasible paths, since such paths are no longer listed individually as in the form of AP_SEQUENCEs. In practice this is not likely to be a problem, since once a call arrives at the edge of a group of ITADs, the LS at that point is likely to have more detailed path information and can distinguish individual paths to destinations.

10.3.4.2. Aggregating Routing Information

Aggregation is the process of combining the characteristics of several different routes in such a way that a single route can be advertised. Aggregation can occur as part of the decision process to reduce the amount of routing information that is placed in the Adj-TRIBs-Out.

Aggregation reduces the amount of information an LS must store and exchange with other LSs. Routes can be aggregated by applying the following procedure separately to attributes of like type.

Routes that have the following attributes shall not be aggregated unless the corresponding attributes of each route are identical: MultiExitDisc, NextHopServer.

Attributes that have different type codes cannot be aggregated. Attributes of the same type code may be aggregated. The rules for aggregating each attribute MUST be provided together with attribute definition. For example, aggregation rules for TRIP's basic

attributes, e.g., ReachableRoutes and AdvertisementPath, are given in Section 5.

10.4. Route Selection Criteria

Generally speaking, additional rules for comparing routes among several alternatives are outside the scope of this document. There are two exceptions:

- If the local ITAD appears in the AdvertisementPath of the new route being considered, then that new route cannot be viewed as better than any other route. If such a route were ever used, a routing loop could result (see Section 6.3).
- In order to achieve successful distributed operation, only routes with a likelihood of stability can be chosen. Thus, an ITAD must avoid using unstable routes, and it must not make rapid spontaneous changes to its choice of route. Quantifying the terms "unstable" and "rapid" in the previous sentence will require experience, but the principle is clear.

10.5. Originating TRIP Routes

An LS may originate local routes by injecting routing information acquired by some other means (e.g. via an intra-domain routing protocol or through manual configuration or some dynamic registration mechanism/protocol) into TRIP. An LS that originates TRIP routes shall assign the degree of preference to these routes by passing them through the Decision Process (see Section 10.2). To TRIP local routes are identical to external routes and are subjected to the same two phase route selection mechanism. A local route which is selected into the Ext-TRIB MUST be advertised to all internal LSs. The decision whether to distribute non-TRIP acquired routes within an ITAD via TRIP or not depends on the environment within the ITAD (e.g. type of intra-domain routing protocol) and should be controlled via configuration.

11. TRIP Transport

This specification defines the use of TCP as the transport layer for TRIP. TRIP uses TCP port 6069. Running TRIP over other transport protocols is for further study.

12. ITAD Topology

There are no restrictions on the intra-domain topology of TRIP LSs. For example, LSs in an ITAD can be configured in a full mesh, star, or any other connected topology. Similarly, there are no restrictions on the topology of TRIP ITADs. For example, the ITADs can be organized in a flat topology (mesh or ring) or in multi-level hierarchy or any other topology.

The border between two TRIP ITADs may be located either on the link between two TRIP LSs or it may coincide on a TRIP LS. In the latter case, the same TRIP LS will be member in more than one ITAD, and it appears to be an internal peer to LSs in each ITAD it is member of.

13. IANA Considerations

This document creates a new IANA registry for TRIP parameters. The following TRIP parameters are included in the registry:

- TRIP Capabilities
- TRIP Attributes
- TRIP Address Families
- TRIP Application Protocols
- TRIP ITAD Numbers

Protocol parameters are frequently initialized/reset to 0. This document reserves the value 0 of each of the above TRIP parameters in order to clearly distinguish between an unset parameter and any other registered values for that parameter.

The sub-registries for each of the above parameters are discussed in the sections below.

13.1. TRIP Capabilities

Requests to add TRIP capabilities other than those defined in Section 4.2.1.1 must be submitted to iana@iana.org. Following the assigned number policies outlined in [11], Capability Codes in the range 32768-65535 are reserved for Private Use (these are the codes with the first bit of the code value equal to 1). This document reserves value 0. Capability Codes 1 and 2 have been assigned in Section 4.2.1.1. Capability Codes in the range 2-32767 are controlled by IANA, and are allocated subject to the Specification Required (IETF RFC or equivalent) condition. The specification MUST include a description of the capability, the possible values it may take, and what constitutes a capability mismatch.

13.2. TRIP Attributes

This document reserves Attribute Type Codes 224-255 for Private Use (these are the codes with the first three bits of the code equal to 1). This document reserves value 0. Attribute Type Codes 1 through 11 have already been allocated by this document. Attribute Type Codes 1 through 11 are defined in Sections 5.1 through 5.11.

Attribute Type Codes in the range 12-223 are controlled by IANA, and require a Specification document (RFC or equivalent). The specification MUST provide all information required in Section 5.12 of this document.

Attribute Type Code registration requests must be sent to iana@iana.org. In addition to the specification requirement, the request MUST include an indication of who has change control over the attribute and contact information (postal and email address).

13.3. Destination Address Families

This document reserves address family 0. Requests to add TRIP address families other than those defined in Section 5.1.1.1 (address families 1, 2, and 3), i.e., in the range 3-32767, must be submitted to iana@iana.org. The request MUST include a brief description of the address family, its alphabet, and special processing rules and guidelines, such as guidelines for aggregation, if any. The requests are subject to Expert Review. This document reserves addresss family codes 32768-65535 for vendor-specific applications.

13.4. TRIP Application Protocols

This document creates a new IANA registry for TRIP application protocols. This document reserves application protocol code 0. Requests to add TRIP application protocols other than those defined in Section 5.1.1.1 (application protocols 1 through 4), i.e., in the range 5- 32767 must be submitted to iana@iana.org. The request MUST include a brief background on the application protocol, and a description of how TRIP can be used to advertise routes for that protocol. The requests are subject to Expert Review. This document reserves application protocol codes 32768-65535 for vendor-specific applications.

13.5. ITAD Numbers

This document reserves ITAD number 0. ITAD numbers in the range 1-255 are designated for Private Use. ITAD numbers in the range from 256 to (2**32-1) are allocated by IANA on a First-Come-First-Serve basis. Requests for ITAD numbers must be submitted to iana@iana.org. The requests MUST include the following:

- Information about the organization that will administer the ITAD.
- Contact information (postal and email address).

14. Security Considerations

This section covers security between peer TRIP LSs when TRIP runs over TCP in an IP environment.

A security mechanism is clearly needed to prevent unauthorized entities from using the protocol defined in this document for setting up unauthorized peer sessions with other TRIP LSs or interfering with authorized peer sessions. The security mechanism for the protocol when transported over TCP in an IP network is IPsec [12]. IPsec uses two protocols to provide traffic security: Authentication Header (AH) [13] and Encapsulating Security Payload (ESP) [14].

The AH header affords data origin authentication, connectionless integrity and optional anti-replay protection of messages passed between the peer LSs. The ESP header provides origin authentication, connectionless integrity, anti-replay protection, and, in addition, confidentiality of messages.

Implementations of the protocol defined in this document employing the ESP header SHALL comply with section 5 of [14], which defines a minimum set of algorithms for integrity checking and encryption. Similarly, implementations employing the AH header SHALL comply with section 5 of [13], which defines a minimum set of algorithms for integrity checking using manual keys.

Implementations SHOULD use IKE [15] to permit more robust keying options. Implementations employing IKE SHOULD support authentication with RSA signatures and RSA public key encryption.

A Security Association (SA) [12] is a simplex "connection" that affords security services to the traffic carried by it. Security services are afforded to an SA by the use of AH, or ESP, but not both. Two types of SAs are defined: transport mode and tunnel mode [12]. A transport mode SA is a security association between two hosts, and is appropriate for protecting the TRIP session between two peer LSs.

Appendix 1: TRIP FSM State Transitions and Actions

This Appendix discusses the transitions between states in the TRIP FSM in response to TRIP events. The following is the list of these states and events when the negotiated Hold Time value is non-zero.

TRIP States:

1 - Idle
2 - Connect
3 - Active
4 - OpenSent
5 - OpenConfirm
6 - Established

TRIP Events:

1 - TRIP Start
2 - TRIP Stop
3 - TRIP Transport connection open
4 - TRIP Transport connection closed
5 - TRIP Transport connection open failed
6 - TRIP Transport fatal error
7 - ConnectRetry timer expired
8 - Hold Timer expired
9 - KeepAlive timer expired
10 - Receive OPEN message
11 - Receive KEEPALIVE message
12 - Receive UPDATE messages
13 - Receive NOTIFICATION message

The following table describes the state transitions of the TRIP FSM and the actions triggered by these transitions.

Event	Actions	Message Sent	Next State
Idle (1)			
1	Initialize resources Start ConnectRetry timer Initiate a transport connection	none	2
others	none	none	1
Connect(2)			
1	none	none	2
3	Complete initialization Clear ConnectRetry timer	OPEN	4
5	Restart ConnectRetry timer	none	3
7	Restart ConnectRetry timer Initiate a transport connection	none	2
others	Release resources	none	1

Active (3)			
1	none	none	3
3	Complete initialization Clear ConnectRetry timer	OPEN	4
5	Close connection Restart ConnectRetry timer		3
7	Restart ConnectRetry timer Initiate a transport connection	none	2
others	Release resources	none	1
OpenSent(4)			
1	none	none	4
4	Close transport connection Restart ConnectRetry timer	none	3
6	Release resources	none	1
10	Process OPEN is OK	KEEPALIVE	5
	Process OPEN failed	NOTIFICATION	1
others	Close transport connection Release resources	NOTIFICATION	1
OpenConfirm (5)			
1	none	none	5
4	Release resources	none	1
6	Release resources	none	1
9	Restart KeepAlive timer	KEEPALIVE	5
11	Complete initialization Restart Hold Timer	none	6
13	Close transport connection Release resources		1
others	Close transport connection Release resources	NOTIFICATION	1
Established (6)			
1	none	none	6
4	Release resources	none	1
6	Release resources	none	1
9	Restart KeepAlive timer	KEEPALIVE	6
11	Restart Hold Timer	none	6
12	Process UPDATE is OK	UPDATE	6
	Process UPDATE failed	NOTIFICATION	1
13	Close transport connection Release resources		1
others	Close transport connection Release resources	NOTIFICATION	1

The following is a condensed version of the above state transition table.

Events	Idle (1)	Connect (2)	Active (3)	OpenSent (4)	OpenConfirm (5)	Estab (6)
1	2	2	3	4	5	6
2	1	1	1	1	1	1
3	1	4	4	1	1	1
4	1	1	1	3	1	1
5	1	3	3	1	1	1
6	1	1	1	1	1	1
7	1	2	2	1	1	1
8	1	1	1	1	1	1
9	1	1	1	1	5	6
10	1	1	1	1 or 5	1	1
11	1	1	1	1	6	6
12	1	1	1	1	1	1 or 6
13	1	1	1	1	1	1

Appendix 2: Implementation Recommendations

This section presents some implementation recommendations.

A.2.1: Multiple Networks Per Message

The TRIP protocol allows for multiple address prefixes with the same advertisement path and next-hop server to be specified in one message. Making use of this capability is highly recommended. With one address prefix per message there is a substantial increase in overhead in the receiver. Not only does the system overhead increase due to the reception of multiple messages, but the overhead of scanning the routing table for updates to TRIP peers is incurred multiple times as well. One method of building messages containing many address prefixes per advertisement path and next hop from a

routing table that is not organized per advertisement path is to build many messages as the routing table is scanned. As each address prefix is processed, a message for the associated advertisement path and next hop is allocated, if it does not exist, and the new address prefix is added to it. If such a message exists, the new address prefix is just appended to it. If the message lacks the space to hold the new address prefix, it is transmitted, a new message is allocated, and the new address prefix is inserted into the new message. When the entire routing table has been scanned, all allocated messages are sent and their resources released. Maximum compression is achieved when all the destinations covered by the address prefixes share a next hop server and common attributes, making it possible to send many address prefixes in one 4096-byte message.

When peering with a TRIP implementation that does not compress multiple address prefixes into one message, it may be necessary to take steps to reduce the overhead from the flood of data received when a peer is acquired or a significant network topology change occurs. One method of doing this is to limit the rate of updates. This will eliminate the redundant scanning of the routing table to provide flash updates for TRIP peers. A disadvantage of this approach is that it increases the propagation latency of routing information. By choosing a minimum flash update interval that is not much greater than the time it takes to process the multiple messages this latency should be minimized. A better method would be to read all received messages before sending updates.

A.2.2: Processing Messages on a Stream Protocol

TRIP uses TCP as a transport mechanism. Due to the stream nature of TCP, all the data for received messages does not necessarily arrive at the same time. This can make it difficult to process the data as messages, especially on systems where it is not possible to determine how much data has been received but not yet processed.

One method that can be used in this situation is to first try to read just the message header. For the KEEPALIVE message type, this is a complete message; for other message types, the header should first be verified, in particular the total length. If all checks are successful, the specified length, minus the size of the message header is the amount of data left to read. An implementation that would "hang" the routing information process while trying to read from a peer could set up a message buffer (4096 bytes) per peer and fill it with data as available until a complete message has been received.

A.2.3: Reducing Route Flapping

To avoid excessive route flapping an LS which needs to withdraw a destination and send an update about a more specific or less specific route SHOULD combine them into the same UPDATE message.

A.2.4: TRIP Timers

TRIP employs seven timers: ConnectRetry, Hold Time, KeepAlive, MaxPurgeTime, TripDisableTime, MinITADOriginationInterval, and MinRouteAdvertisementInterval The suggested value for the ConnectRetry timer is 120 seconds. The suggested value for the Hold Time is 90 seconds. The suggested value for the KeepAlive timer is 30 seconds. The suggested value for the MaxPurgeTime timer is 10 seconds. The suggested value for the TripDisableTime timer is 180 seconds. The suggested value for the MinITADOriginationInterval is 30 seconds. The suggested value for the MinRouteAdvertisementInterval is 30 seconds.

An implementation of TRIP MUST allow these timers to be configurable.

A.2.5: AP_SET Sorting

Another useful optimization that can be done to simplify this situation is to sort the ITAD numbers found in an AP_SET. This optimization is entirely optional.

Acknowledgments

We wish to thank Dave Oran for his insightful comments and suggestions.

References

[1] S. Bradner, "Keywords for use in RFCs to Indicate Requirement Levels," IETF RFC 2119, March 1997.

[2] J. Rosenberg and H. Schulzrinne, "A Framework for a Gateway Location Protocol," IETF RFC 2871, June 2000.

[3] Y. Rekhter and T. Li, "Border Gateway Protocol 4 (BGP-4)," IETF RFC 1771, March 1995.

[4] J. Moy, "Open Shortest Path First Version 2," IETF RFC 2328,

April, 1998.

[5] "Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol for use in Conjunction with the Protocol for Providing the Connectionless-mode Network Service (ISO 8473)," ISO DP 10589, February 1990.

[6] J. Luciani, et al, "Server Cache Synchronization Protocol (SCSP)," IETF RFC 2334, April, 1998.

[7] International Telecommunication Union, "Visual Telephone Systems and Equipment for Local Area Networks which Provide a Non-Guaranteed Quality of Service," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.

[8] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol," IETF RFC 2543, March 1999.

[9] R. Braden, "Requirements for Internet Hosts -- Application and Support," IETF RFC 1123, October 1989.

[10] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," IETF RFC 2373, July 1998.

[11] T. Narten and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," IETF RFC 2434, October 1998.

[12] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," IETF RFC 2401, November 1998.

[13] S. Kent and R. Atkinson, "IP Authentication Header," IETF RFC 2402, November 1998.

[14] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," IETF RFC 2406, November 1998.

[15] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," IETF RFC 2409, November 1998.

Authors' Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936

973-952-5000
email: jdrosen@dynamicsoft.com

Hussein F. Salama
Cisco Systems
Mail Stop SJ-6/3
170 W. Tasman Drive
San Jose, CA 95134
408-527-7147
email: hsalama@cisco.com

Matt Squire
WindWire
4825 Creekstone Drive
Durham, NC 27703
919-247-0820
email: msquire@windwire.com

Intellectual Property Notice

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

Full Copyright Statement

Copyright (C) The Internet Society (1999). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.