CCAMP Working Group
Internet-Draft
Expires: March 2, 2005

R. Bonica
MCI
E. Rosen
D. Meyer
Cisco Systems
K. Kompella
Juniper Networks
R. Dube
Xebeo Communications

September 2004

Generic Tunnel Tracing Protocol (GTTP) Specification
draft-ietf-ccamp-tunproto-01

Status of this Memo

Copyright Notice

Abstract

This document describes the Generic Tunnel Tracing Protocol (GTTP). GTTP supports enhanced route-tracing applications. Network operators use enhanced route-tracing applications to trace the path between any two points in an IP network including tunnels that support the traced path.

Table of Contents

1. Introduction

This document describes the Generic Tunnel Tracing Protocol (GTTP). GTTP supports enhanced route-tracing applications. Network operators use enhanced route-tracing applications to trace the path between any two points in an IP network. Enhanced route-tracing applications can trace through a network's forwarding plane, its control plane or both. Furthermore, enhanced route-tracing applications can reveal details concerning tunnels that support the traced path. Tunnel details can be revealed regardless of tunneling technology. For example, enhanced route-tracing applications can trace through an MPLS LSP as well as through an IP-in-IP tunnel.

RFC 3609 [7] specifies requirements for enhanced route-tracing applications. It also describes protocol requirements for GTTP.

Currently, GTTP is specified for IPv4 addressing only. In future revisions, GTTP will also support IPv6 addressing.

Each section of this document presents a significant aspect of GTTP. Section 1 describes the generic route-tracing problem and provides an informal description of GTTP. Section 2 describes protocol mechanisms and Section 3 describes IANA considerations. Section 4 details security considerations.

1.1 Conventions Used In This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

1.2 The Tunnel Tracing Problem

This section illustrates how a route-tracing application can use GTTP to trace the path between two points in an IP network. It also

illustrates how a route-tracing application can use GTTP to discover tunnels that support the traced path.

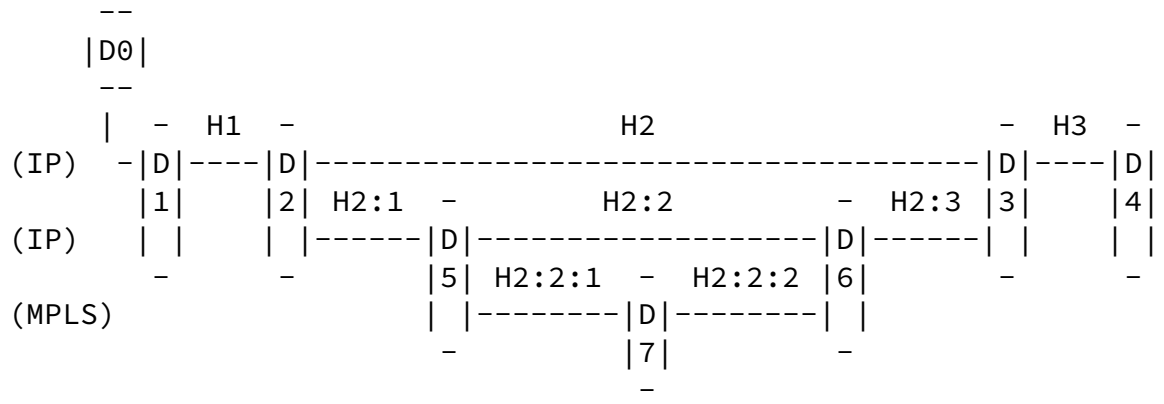


Figure 1: Tunnel Tracing Problem

Figure 1 depicts eight IP accessible devices (D0 through D7). An enhanced route-tracing application executes upon device D0. The route-tracing application must trace the route between devices D1 and D4. In this example, the traced route originates on D1's loopback interface and terminates on D4's loopback interface.

The route between D1 and D4 contains three top-level hops. These are H1, H2 and H3. No tunnel supports H1 or H3. An IP-in-IP tunnel supports H2. The IP-in-IP tunnel itself contains three hops. These hops, H2:1, H2:2 and H2:3, are subordinate to H2.

Finally, an MPLS LSP supports H2:2. The LSP contains two hops, H2:2:1 and H2:2:2. The MPLS LSP is configured for penultimate hop popping. Therefore, MPLS headers do not encapsulate datagrams

arriving at D6.

1.3 Informal Protocol Description

GTTP supports two PDU types. These PDU types represent a traceProbe and a traceResponse. Enhanced route-tracing applications emit a series of traceProbe messages. Each traceProbe elicits exactly one traceResponse and each traceResponse describes a component of the traced path. Specifically, the traceResponse can describe a top-level hop or a hop that is contained by a supporting tunnel.

Each traceProbe contains the following objects:

- Source Object
- Head-end Object
- Access Control Object (optional)
- Path Object
- Propagation Object
- Context Object (optional)

The Source Object identifies the IP interface and UDP port upon which the route-tracing application is listening for a traceResponse. It also contains a timestamp and sequence number. The route-tracing application provides these values and uses the sequence number to match traceProbes with traceResponses.

The Head-end Object identifies the head-end of the traced path by IP address. It also contains two timestamps. The device that supports the head-end of the traced path provides one timestamp while processing a traceProbe message, and the other timestamp while processing the corresponding traceResponse message. The difference between these two timestamps represents the round trip time between the head-end device and the device that provided the traceResponse.

The Access Control Object contains an access control token. Network elements use this token in order to determine whether the route-tracing application can access the information that it is requesting. The Access Control Object is optional.

The Path Object identifies the path being traced, from the perspective of the head-end device. The traced path can be a top-level path between two points in an IP network. It can also be a tunnel that supports the top-level path. When the Path Object represents a top-level path, it contains an IP Header Object. The IP Header Object contains an IP header. (In most cases, the traced path is identifiable by IP destination address only. In special cases, a router bases its forwarding decision upon multiple IP header fields, so the entire IP header is provided for completeness.) When the Path Object represents a tunnel, it contains a Tunnel Object and the Tunnel Object contains a tunnel identifier.

The Propagation Object determines which member of the traced path will respond to the traceProbe. If the traced path supports TTL decrement (as is the case for IP or MPLS) the propagation object contains a Hop Count. This Hop Count determines how far the traceProbe will travel along the traced before it expires and elicits a traceResponse message. If the traced path does not support TTL decrement (as is the case for some tunneling technologies), the Propagation Object identifies the device that will provide a traceResponse by IP address.

The Context Object identifies a VPN context. See Section 1.6 for details on Context Object processing and Section 2.3.11 for format details.

Each traceResponse message contains an error code and the following

objects:
- Source Object
- Head-end Object
- Arrival Object (optional)
- Next-hop Object (optional)
- Context Object (optional)

The Source and Head-end are described above.

The Arrival Object describes how the traceProbe arrived at the responding device. Specifically, the Arrival Object contains the following:

o an Interface Object that identifies the interface upon which the traceProbe arrived

- an optional Tunnel Object that identifies the tunnel upon which the traceProbe arrived
- an Expiration field that indicates whether the traceProbe was contained by a datagram whose TTL expired at the responding device

The traceResponse message MUST include an Arrival Object if it describes any interface other than the origination point of a traced path or tunnel. If it describes the origination point of a traced path or tunnel, it MUST NOT include an Arrival Object.

The Next-hop Object identifies the next hop along the traced path. Specifically, the Next-hop Object contains the following:

- the next hop, identified by IP address.
- an Interface Object that identifies the interface through which the responding device would forward the traceProbe if it were to forward it to its ultimate destination.
- an optional Tunnel Object that identifies the tunnel through which the responding device would forward the traceProbe if it were to forward it to its ultimate destination.

If a traced path does not terminate on the responding device and the responding device can forward datagrams through the traced path, the traceResponse message MUST contain a Next-hop Object. Otherwise, the traceResponse message MUST NOT contain a Next-Hop Object. If the responding device does not terminate the traced path and the responding device cannot forward datagrams through the traced path, the traceResponse MUST contain an error code indicating why the responding device cannot forward datagrams through the traced path.

If a device receives a traceProbe that contains a Context Object and responds with a traceResponse, the traceResponse MUST contain an identical Context Object. See Section 1.6 for details.

1.4 Theory of Operation

The enhanced route-tracing application operates in phases. During its initial phase, the application acquires information regarding the top-level path that connects two IP interfaces. Specifically, the application receives a traceResponse from each device that participates in the top-level path. Each traceResponse can contain

an Arrival Object and a Next-hop Object. The Arrival Object describes how the traceProbe arrived at the responding device. It contains an Interface Object, that describes the interface upon which the traceProbe arrived, and possibly a Tunnel Object, that describes the tunnel upon which the traceProbe arrived. The traceResponse message also contains a Next-Hop Object that describes how the responding device would forward the traceProbe if it were to do so. The Next-hop Object contains a Tunnel Object if a tunnel supports the downstream hop. The Tunnel Object contains a tunnel identifier that the application will use in subsequent phases to probe for tunnel details.

During the next phase, the application acquires information regarding tunnels that support top-level hops. Specifically, the application uses the Tunnel Object mentioned above to query tunnel details. If the application discovers nested tunnels, it executes subsequent phases as required.

During each phase, the route-tracing application sends probes to the device that serves as head-end of the traced object. For example, during the initial phase, the route-tracing application sends traceProbes to the head-end of the top-level path. If the route-tracing application discovers that a tunnel supports one top-level hop, it initiates a second phase. During the second phase, the route-tracing application sends traceProbes directly to the device that supports the tunnel head-end.

Therefore, the device that hosts the route-tracing application must maintain a route to the device that hosts the head-end of the top-level path. Conversely, the device that hosts the head-end of the top-level path must maintain a route to the device that hosts the route-tracing application. If the route-tracing application is to discover tunnel details, it must maintain a route to the tunnel ingress device and the tunnel ingress device must maintain a route back to the device that hosts the route-tracing application.

The following sections describe how an enhanced route-tracing application would trace the path described in Section 1.2.

1.4.1 Top Level Path Discovery

D0 formats a traceProbe, encapsulates it within UDP and IP headers, and sends it to D1. The traceProbe contains the following objects:

- a Source Object that identifies D0 as the traceProbe's source
- a Head-end Object that identifies D1 as the head-end of the traced path
- an Access Control Object that contains an access control token
- a Path Object that identifies D4 as the tail-end of the traced path
- a Propagation Object that contains a Hop Count of 0, indicating that the traceProbe is requesting information about the hop directly downstream from D1.

D1 receives the traceProbe and interrogates its Access Control Object. If D1 does not grant access, it sends D0 a traceResponse indicating that access has been denied. If D1 grants access, it interrogates the Head-end object and determines that it is the head-end of the traced path. D1 then interrogates the Path Object and determines that it is tracing the route towards D4. Finally, D1 interrogates the Propagation Object and determines that it should report on H1.

D1 sends D0 a traceResponse that contains the following objects:

- the Source Object, as it arrived in the traceProbe
- the Head-end Object, as it arrived in the traceProbe. D1 overwrites both timestamps, indicating the time at which it received the traceProbe and the time at which it sent the traceResponse.
- a Next-hop Object that identifies the next hop by IP address. The Next-hop Object also contains an Interface Object that describes the interface to the next hop. The Next-hop Object does not contain a Tunnel Object because no tunnel supports H1.

D1 directs the traceResponse to the UDP port specified by the Source Object.

Having received the traceResponse, D0 has acquired control plane information regarding H1. Now, D0 sends D1 another traceProbe. This traceProbe is identical to the previous traceProbe except that its Propagation Object contains a Hop Count of 1.

D1 receives the traceProbe and interrogates its Access Control Object. If D1 does not grant access, it sends D0 a traceResponse indicating that access has been denied. If D1 grants access, it interrogates the Head-end Object and determines that it is the head-end of the traced path. D1 then interrogates the Path Object and determines that it is tracing the route towards D4. Next, D1

interrogates the Propagation Object and determines that it should probe the path towards D4. Therefore, D1 timestamps the traceProbe's Head-end Object and encapsulates the traceProbe within UDP and IP headers. D1 sets all IP header values to those specified by the traceProbe's Path Object. It also sets the IP TTL value equal to the Propagation Object's Hop Count (i.e., 1). Finally, D1 recalculates the IP header checksum and forwards the traceProbe towards D4.

Because D1 set the IP TTL to 1, the traceProbe expires at D2. D2 emits an ICMP Time Expired message, which GTTP ignores. D2 then processes the traceProbe.

Specifically, D2 interrogates the traceProbe's Access Control Object. If D2 does not grant access, it sends D1 a traceResponse indicating that access has been denied. (D1 would relay this message to D0, updating timestamps on the Head-end object as appropriate.) If D2 grants access, it interrogates the Head-end object and determines that it is not the head-end of the traced-path. D2 then interrogates the Path Object and determines that it is tracing the IP route towards D4. Therefore, D2 determines that it should report forwarding plane information regarding H1 and control plane information regarding H2. Having made this determination, D2 sends D1 a traceResponse that contains the following objects:

- the Source Object, as it arrived in the traceProbe
- the Head-end Object, as it arrived in the traceProbe
- an Arrival Object indicating that the traceProbe arrived in an expired datagram (i.e., TTL = 0). The Arrival Object also describes the interface upon which the datagram arrived. It does not contain a tunnel object because no tunnel supports H1.
- a Next-hop Object that identifies the next hop by IP address. The Next-hop Object also contains an Interface Object that describes the interface to the next hop. Furthermore, the Next-hop Object contains a Tunnel Object that will be used when probing for details regarding the tunnel that supports H2.

D1 receives the traceResponse. If the traceResponse does not specify any errors, D1 updates the Head-end object's traceResponse timestamp. Whether or not any errors were detected, D1 relays the traceResponse to D0.

Having received the traceResponse, D0 has acquired forwarding plane information regarding H1 and control plane information regarding H2.

D0 repeats the process described above in order to discover the remaining hops of the top-level path.

1.4.2 Tunnel Discovery

Having discovered details regarding the top level path, the route-tracing application must obtain details regarding the IP-in-IP tunnel that supports H2. In order to obtain these details, D0 sends D2 a traceProbe. The traceProbe contains the following objects:

- a Source Object that identifies D0 as the traceProbe's source
- a Head-end Object that identifies D2 as the head-end of the traced tunnel
- an Access Control Object that contains an access control token
- a Path Object that contains the Tunnel Object obtained from D2, above
- a Propagation Object that contains a Hop Count of 0, indicating that the traceProbe is requesting information about the tunnel hop directly downstream from D2.

D2 receives the traceProbe and interrogates its Access Control Object. If D2 does not grant access, it sends D0 a traceResponse indicating that access has been denied. If D2 grants access, it interrogates the Head-end object and determines that it is the head-end of the traced tunnel. D2 then interrogates the Path Object and determines that it is tracing Tunnel H2. Finally, D2 interrogates the propagation object and determines that it should report on H2:1.

D2 sends D0 a traceResponse that contains the following objects:

- the Source Object, as it arrived in the traceProbe
- the Head-end Object, as it arrived in the traceProbe. D2 overwrites both timestamps, indicating the time at which it received the traceProbe and the time at which it sent the traceResponse.
- a Next-hop Object that identifies the next hop by IP address. The Next-hop Object also contains an Interface Object that describes the interface to the next hop. It does not contain a Tunnel Object because no tunnel supports H2:1.

Having received the traceResponse, D0 has acquired control plane information regarding H2:1. Now, D0 sends D2 another traceProbe. This traceProbe is identical to the previous traceProbe except that its Propagation Object contains a Hop Count of 1.

D2 receives the traceProbe and interrogates its Access Control Object. If D2 does not grant access, it sends D0 a traceResponse indicating that access has been denied. If D2 grants access, it interrogates the Head-end Object and determines that it is the head-end of the traced-path. D2 then interrogates the Path Object and determines that it is tracing Tunnel H2. Next, D2 interrogates the Propagation Object and determines that it should probe Tunnel H2.

Therefore, D2 timestamps the traceProbe's Head-end Object and encapsulates the traceProbe within UDP and IP headers. (IP header values are determined by the configuration of tunnel H2.) D2 sets the IP TTL value equal to the Propagation Object's Hop Count (i.e., 1), recalculates the checksum and forwards the traceProbe the tunnel endpoint.

Because D2 set the IP TTL to 1, the traceProbe expires at D5. D5 emits an ICMP Time Expired message, which GTTP ignores. D5 then processes the traceProbe.

Specifically, D5 interrogates the traceProbe's Access Control Object. If D5 does not grant access, it sends D2 a traceResponse indicating that access has been denied. (D2 would relay this message to D0.) If D5 grants access, it interrogates the Head-end object and determines that it is not the head-end of the traced-path. D5 then interrogates the Path Object and determines that it is tracing Tunnel H2. Therefore, D5 determines that it should report forwarding plane information regarding H2:1 and control plane information regarding H2:2. Having made this determination, D5 sends D2 a traceResponse that contains the following objects:

- the Source Object, as it arrived in the traceProbe
- the Head-end Object, as it arrived in the traceProbe.
- an Arrival Object indicating that the traceProbe arrived in an expired datagram (i.e., TTL = 0). The Arrival Object also describes the interface upon which the datagram arrived. It does not contain a tunnel object because no tunnel supports H2:1.
- a Next-hop Object that identifies the next hop by IP address. The Next-hop Object also contains an Interface Object that describes

the interface to the next hop. Furthermore, the Next-hop Object contains a Tunnel Object that will be used when probing for details regarding the tunnel that supports H2:2.

D2 receives the traceResponse. If the traceResponse does not specify any errors, D2 updates the Head-end object's traceResponse timestamp. Whether or not any errors were detected, D2 relays the traceResponse to D0.

Having received the traceResponse, D0 has acquired forwarding plane information regarding H2:1 and control plane information regarding H2:2.

D0 repeats the process described above in order to discover remaining tunnel details.

1.5 Timeout Processing

As demonstrated above, devices D1 through D7 are stateless with

regard to GTTP. Therefore, they need not maintain any timeout logic. However, the route-tracing application does maintain state and it must maintain timeout logic.

When the route tracing-application sends a traceProbe, it initiates a timer that will expire after a configurable period of time has elapsed. If the application receives a traceResponse before the timer expires, it destroys the timer. If the timer expires before the application receives a traceResponse, the application invokes timeout logic.

Because timeout logic is contained entirely by the route-tracing application, it is beyond the scope of this specification. However, the route-tracing applicationÆs timeout behavior should be similar to that of TRACEROUTE.

1.6 Context Object Processing

The Context Object allows VPN customers to trace through a Service Provider's network. It is used only when the service provider's policy is to allow such tracing.

Assume that a VPN customer wants to trace the path from one VPN site to another. Assume also that the VPN customer wants to discover details regarding the intervening Service Provider network and it is the service provider's policy to allow such discovery. The VPN customer executes a route-tracing application.

The route tracing application operates in phases. During the first phase, the route-tracing application discovers the top-level path between the two VPN sites. It also reveals that a tunnel (specifically, a VPN tunnel) connects the Service Provider ingress router to the Service Provider egress router.

During the second phase, the route-tracing application probes for details regarding the VPN tunnel. Specifically, the route-tracing application sends probes to the Service Provider ingress router, with each probe containing a Source Object. The Source Object identifies the route-tracing application by IP address and UDP port. However, because the Service Provider may support RFC 1918 [1] addressing, the Source Object may not uniquely identify the route-tracing application to the provider ingress router. Therefore, the Service Provider ingress router MUST append a Context Object to the traceProbe. The Context Object adds VPN context to the IP addresses specified in the Source Object.

The Service Provider ingress router forwards the traceProbe downstream and a downstream device responds to it. The responding

device MUST return the Context Object in a traceResponse, exactly as it was received. The Service Provider ingress router uses the Context Object, together with the Source Object, to forward the traceResponse message to its ultimate destination (i.e., route-tracing application). Before forwarding the traceResponse to the route-tracing application, the Service Provider ingress router MUST remove the Context Object.

The Context Object is only used when probing the VPN tunnel. It is not required when probing the top-level path between VPN sites. Furthermore, the Context Object MUST NOT be exposed outside of the Service Provider Network.

As mentioned above, the Service Provider ingress router generates the Context Object and appends it to the traceProbe. The same router

uses the Context Object when it returns on the traceResponse. Because a single router both produces and consumes the Context Object, its contents are not the subject of standardization.

1.7 Transient Changes in Topology

The route-tracing application SHOULD detect transient changes in network topology. In order to do this, the route-tracing application SHOULD probe each hop multiple times, as does the traditional TRACEROUTE [4] application.

1.8 Load Balancing

The route-tracing application SHOULD detect load balancing. In order to detect load balancing, the route-tracing application SHOULD probe each hop multiple times, as does the traditional TRACEROUTE [4] application. Also, when a device receives a traceProbe that it might forward over multiple downstream interfaces or tunnels, it SHOULD respond with a traceResponse that contains multiple Next-Hop Objects.

1.9 Incremental Deployment

GTTP may not be deployed on all devices that contribute to a traced path or tunnel. Therefore, the traceProbe message may elicit an ICMP message indicating that GTTP is not available on the responding device. Likewise, the traceProbe may elicit no response at all. When the traceProbe does not elicit a traceResponse, the route-tracing application should proceed to probe the next component of the trace path or tunnel. The trace should expire after probing a configurable number of path or tunnel components.

2. Protocol Mechanisms

2.1 Transport

GTTP obtains transport services from UDP. All GTTP messages are directed to UDP port 3693, except for traceResponse messages that are bound directly for the route-tracing application. Those messages are directed to the UDP port specified by the route-tracing application

in the traceProbe Source Object.

2.2 Message Formats

The following subsections detail GTTP message formats.

2.2.1 The TraceProbe Message

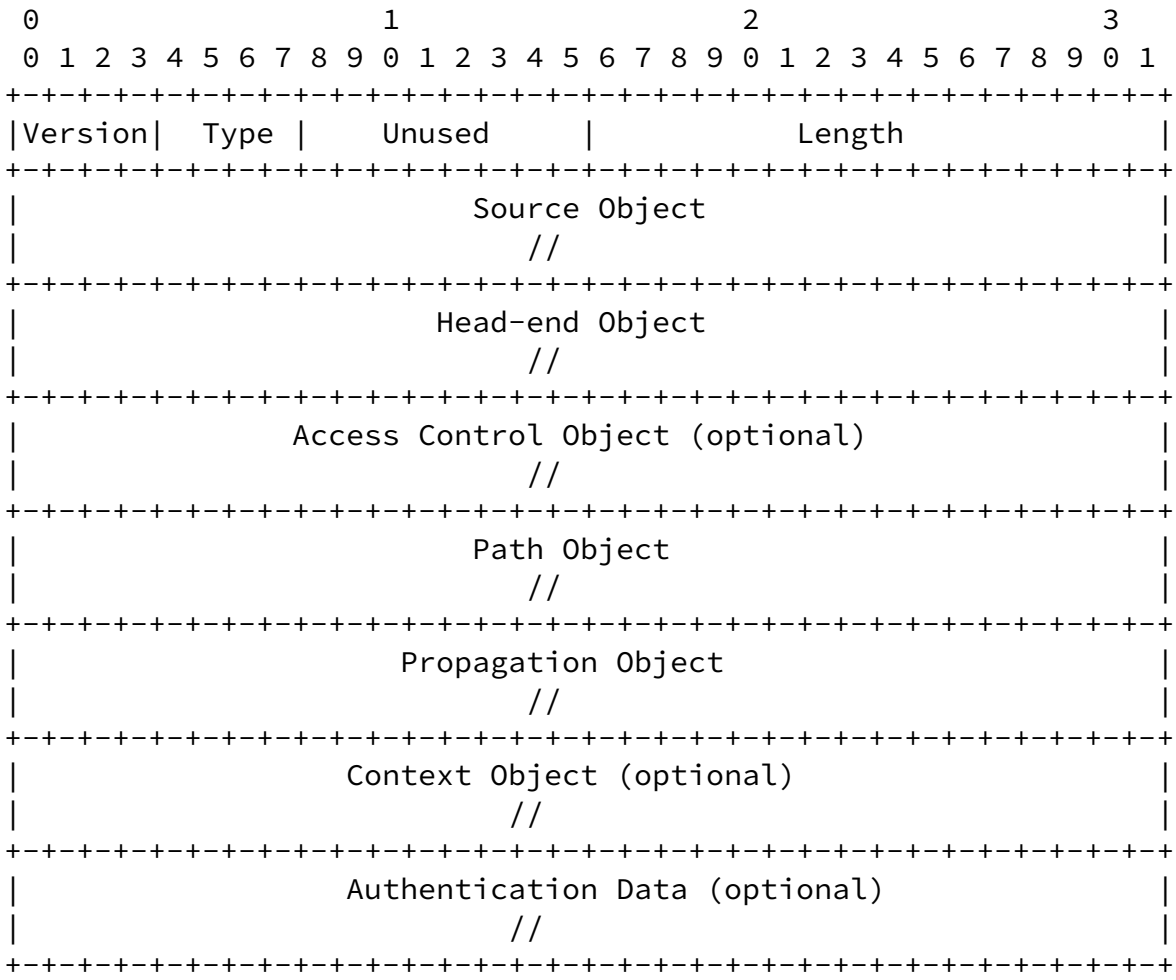


Figure 2: The TraceProbe Message

Figure 2 depicts the TraceProbe Message. Route-tracing applications send TraceProbe messages in order to solicit information regarding a component of a traced path. The following paragraphs describe each

field that contributes to the TraceProbe Message.

Version: 4 bits

The Version field specifies the GTTP message format. Value is equal to 1.

Type: 4 bits

The Type field specifies the GTTP message type. A value of 0 identifies a TraceProbe Message.

Length: 16 bits

The Length field specifies the number of four-octet words that follow.

The Source, Head-end, Access Control, Path, Propagation and Context Objects are described in dedicated sections of this document. Every traceProbe MUST contain the Source, Head-end, Path and Propagation objects. The Access Control and Context Objects are optional. Authentication Data is also optional. Authentication Data is REQUIRED when the Authentication Object specifies cryptographic Authentication. In all other cases, Authentication Data MUST NOT be included in the traceProbe message.

The object ordering illustrated above is REQUIRED.

2.2.2 The TraceResponse Message

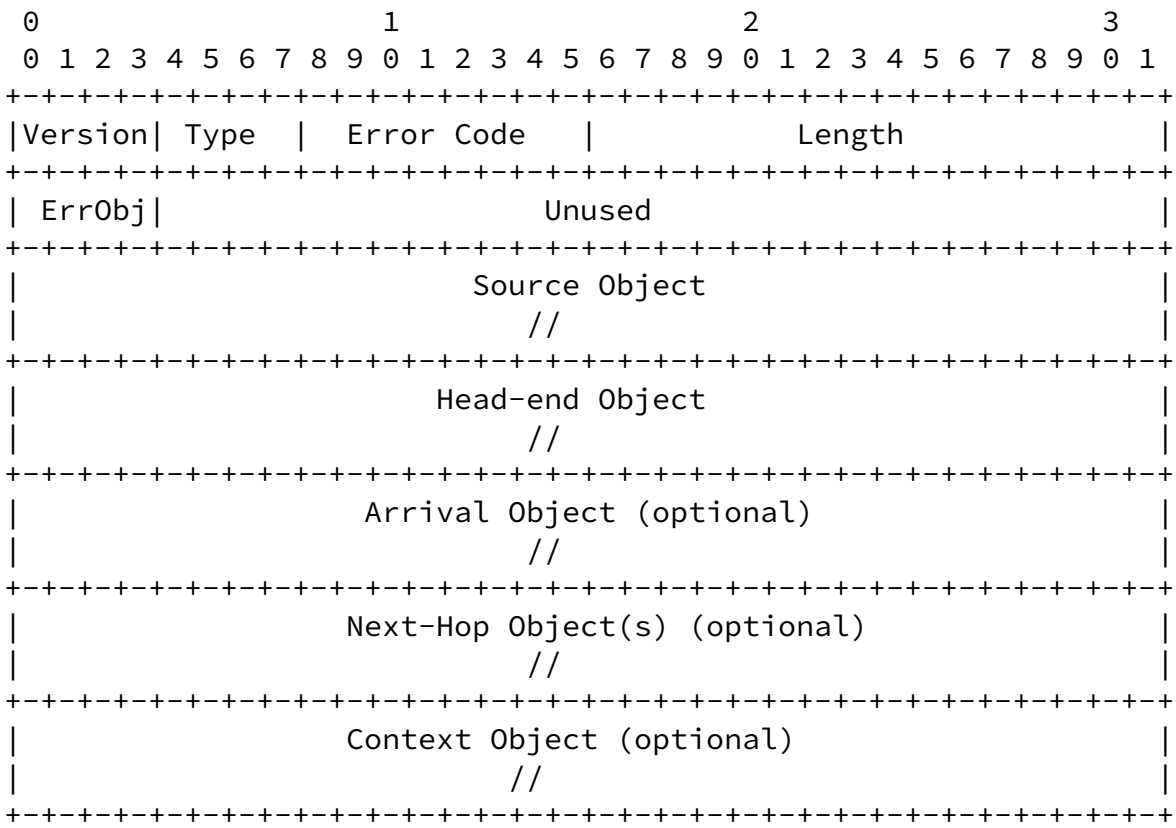


Figure 3: The TraceResponse Message

Figure 3 depicts the TraceResponse Message. Network devices send traceResponse messages in order to provide information regarding a component of a traced path. The following paragraphs describe each field that contributes to the TraceResponse Message.

Version: 4 bits

The Version field specifies the GTTP message format. Value is equal to 1.

Type: 4 bits

The Type field specifies the GTTP message type. A value of 1 identifies a TraceResponse Message.

Error Code: 8 bits

The Error Code field defines protocol errors. The following error codes are currently defined:

0 - gttp_no_error
1 - gttp_access_denied
2 - gttp_no_such_tunnel
3 - gttp_no_route_to_destination
4 - gttp_route_to_destination_administratively_blocked
5 - gttp_missing_object
6 - gttp_malformed_object

Length: 16 bits

The Length field specifies the number of four-octet words that follow.

ErrObj: 8 bits

The ErrObj field identifies a missing or malformed object. If no object is missing or malformed, this value must be set to 0.

The Source, Head-end, Arrival, Next-Hop and Context Objects are described in dedicated sections of this document. Every traceResponse message MUST contain the Source and Head-end Objects. The Arrival, Next-Hop and Context Objects are optional.

When a device receives a traceProbe that it might forward over multiple downstream interfaces or tunnels, it SHOULD respond with a traceResponse that contains multiple Next-Hop Objects.

The object ordering illustrated above is REQUIRED.

2.3 Object Formats

The following subsections detail GTTP object formats.

2.3.1 The Source Object

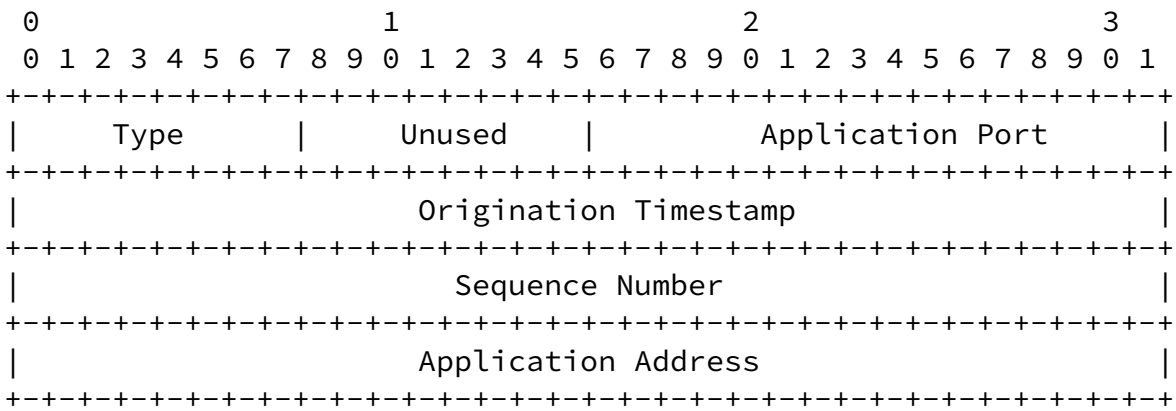


Figure 4: The Source Object

Figure 4 depicts the Source Object. The Source Object identifies the GTTP message and its source. The following paragraphs describe each field that contributes to the Source Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Source Object, the Type field is always equal to 1.

Application Port: 16 bits

The Application Port field identifies a UDP port upon which the route-tracing application is listening for a traceResponse.

Origination TimeStamp: 32 bits

The Origination Timestamp represents the time at which the traceProbe was issued. As the device that hosts the route-tracing application

supplies this value and is its only user, the unit of measure is not subject to standardization.

Sequence Number: 32 bits

The Sequence Number identifies the traceProbe. Applications use this field to match traceProbes with TraceResponses.

Application Address: 32 bits

The Application Address identifies an IP interface upon which the route-tracing application is awaiting a traceResponse.

All Unused bits MUST be set to 0 and ignored.

2.3.2 The Head-end Object

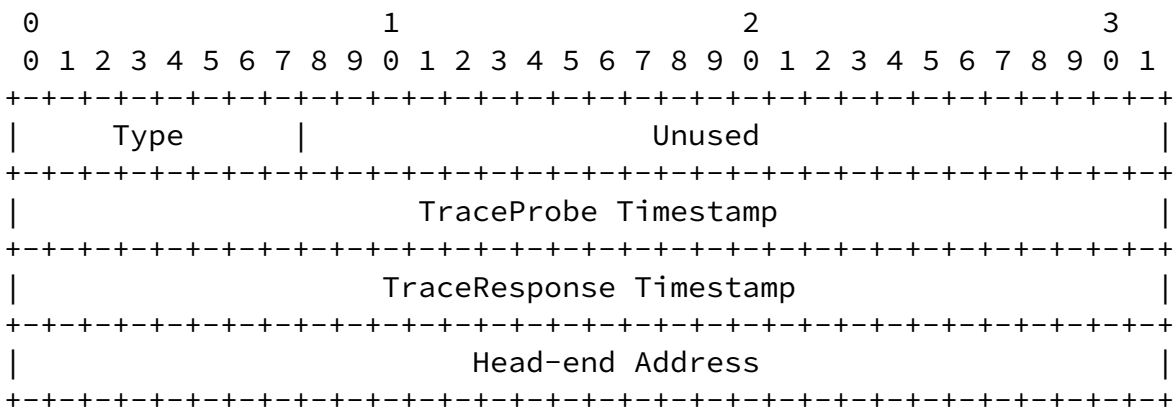


Figure 5: The Head-end Object

Figure 5 depicts the Head-end Object. The Head-end Object identifies the head-end of a top-level path or supporting tunnel. The following paragraphs describe each field that contributes to the Head-end Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Head-end Object, the Type field is always equal to 2.

TraceProbe TimeStamp: 32 bits

The TraceProbe Timestamp represents the time at which the Head-end device received the traceProbe. It represents the number of milliseconds since some arbitrarily chosen point in time.

When the route-tracing application originates a traceProbe, it MUST set this field to 0x0000. The head-end of a traced path or tunnel updates this field as described above. When the originator of a traceResponse copies the Head-end Object from a traceProbe to a traceResponse, it MUST NOT change the value of this field.

TraceResponse TimeStamp: 32 bits

The TraceResponse Timestamp represents the time at which the Head-end device sent the traceResponse. It represents the number of milliseconds since some arbitrarily chosen point in time.

When the route-tracing application originates a traceProbe, it MUST set this field to 0x0000. The head-end of a traced path or tunnel updates this field as described above. When the originator of a traceResponse copies the Head-end Object from a traceProbe to a

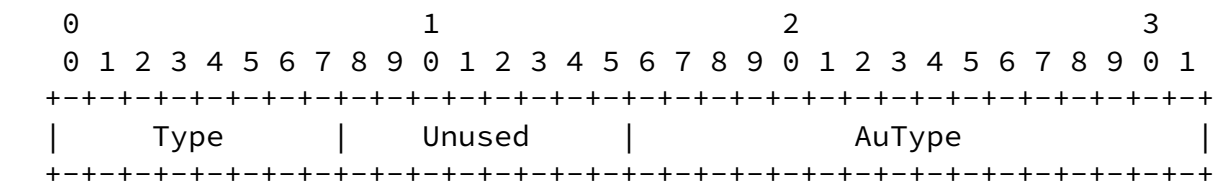
traceResponse, it MUST NOT change the value of this field.

Head-end Address: 32 bits

The Head-end Address identifies the head-end of the top-level path or tunnel that is being traced.

All Unused bits MUST be set to 0 and ignored. The TraceProbe Timestamp and TraceResponse Timestamp fields must use the same arbitrarily chosen point in time as a reference.

2.3.3 The Access Control Object



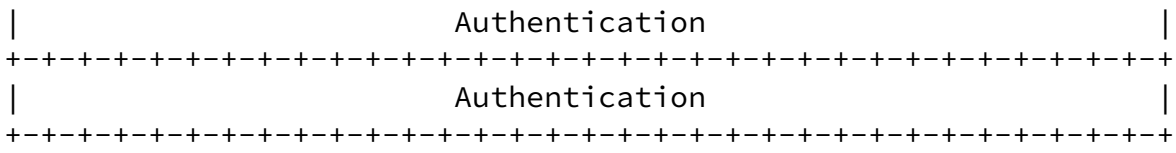


Figure 6: The Access Control Object

Figure 6 depicts the Access Control Object. GTTP entities use the access control object to enforce access control policy. The following paragraphs describe each field that contributes to the Access Control Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Access Control Object, the Type field is always equal to 3.

AuType: 16 bits

The AuType specifies the type of authentication used for this message. Encoding rules follow:

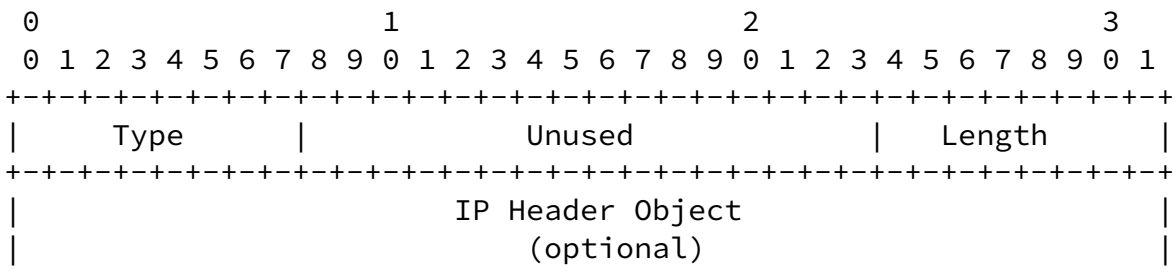
1 - Plaintext password
2 - Cryptographic Authentication

Authentication: 64 bits

The Authentication field contains authentication data. See Appendix D of RFC 2328 [5] for a description of this field. This field is used in conjunction with the Authentication Data field that is specified at the end of the traceProbe Message.

All Unused bits MUST be set to 0 and ignored.

2.3.4 The Path Object



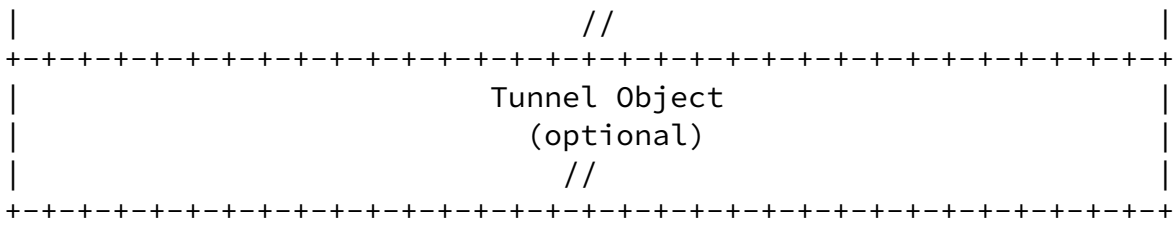


Figure 7: The Path Object

Figure 7 depicts the Path Object. The Path Object identifies the path being traced. The Path Object can represent the top-level path between two points in an IP network. It can also represent a tunnel that supports the top-level path. When the Path Object represents a top-level path, it contains an IP Header Object. The IP Header Object contains an IP header that identifies the traced path. (In most cases, the traced path is identifiable by IP destination address only. In special cases, the router bases its forwarding decision upon multiple IP header fields, so the entire IP header is provided for completeness.) When the Path Object represents a tunnel, it contains a Tunnel Object and the Tunnel Object contains a tunnel identifier.

The following paragraphs describe each field that contributes to the Path Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Path Object, the Type field is always equal to 4.

Length : 8 bits

The Length field specifies the number of four-octet words that follow.

The IP Header and Tunnel Objects are described in Section 2.3.8 and Section 2.3.10. The Path Object MUST contain an IP Header Object or

a Tunnel Object, but it MUST NOT contain both.

All Unused bits MUST be set to 0 and ignored. The object ordering illustrated above is REQUIRED.

2.3.5 The Propagation Object

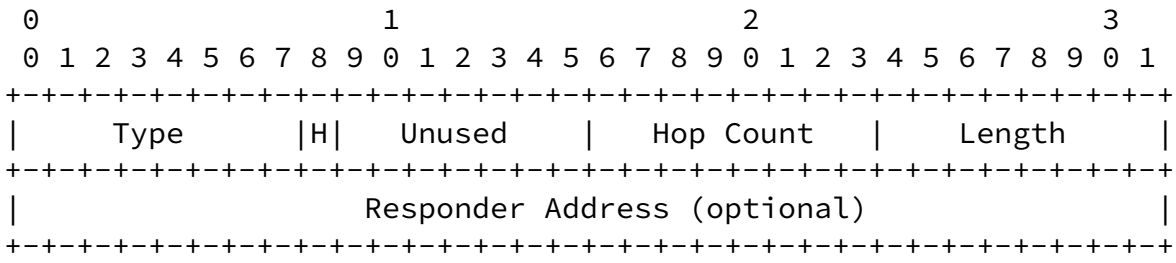


Figure 8: The Propagation Object

Figure 8 depicts the Propagation Object. The Propagation Object determines which member of the traced path will respond to the traceProbe. If the traced path supports TTL decrement (as is the case for IP or MPLS) the H-bit is set and the propagation object specifies a Hop Count. This Hop Count determines how far the traceProbe will travel along the traced path before it expires and elicits a traceResponse message. If the traced path does not support TTL decrement (as is the case for some tunneling technologies), the H-bit is clear and the Propagation Object identifies the device that will provide a traceResponse by IP address.

The following paragraphs describe each field that contributes to the Propagation Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Propagation Object, the Type field is always equal to 5.

H : 1 bit

The H flag determines whether GTTP will use TTL decrement to discover the traced path. If the H flag is set, the Hop Count field is significant and the Responder Address should not be specified. If the H flag is not set, the Hop Count is insignificant and the Responder Address is required.

Hop Count: 8 bits

The Hop Count field determines which member of the traced path should respond to the traceProbe. A value of 0 indicates that the head-end

device should respond. A value of 1 indicates that the device one hop beyond the head and should respond and so forth. The Hop Count field is only significant when the H Flag is set.

Length: 8 bits

The Length Field specifies the number of four-octet words that follow. In the Path Object, its value is equal to 0 or 1.

Responder Address : 32 bits

The Responder Address identifies the device that is to provide the traceResponse by IP address. When the H-bit is set, this field MUST NOT be present. When the H-bit is clear, this field MUST be present and contain a valid IPv4 address.

All Unused bits must be set to 0 and ignored.

2.3.6 The Arrival Object

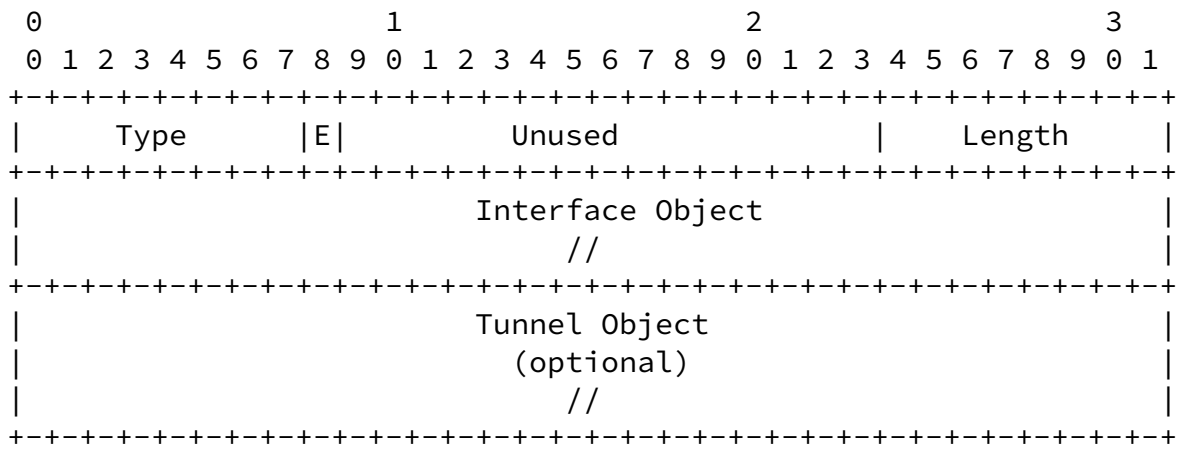


Figure 9: The Arrival Object

Figure 9 depicts the Arrival Object. The Arrival Object describes how a traceProbe arrived at the probed device. The following paragraphs describe each field that contributes to the Arrival Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Arrival Object, the Type field is always equal to 6.

E : 1 bit

The E flag indicates that GTTP is responding to a traceProbe that

arrived in an IP datagram whose TTL has expired.

Length : 8 bits

The Length Field specifies the number of four-octet words that follow.

The Interface Object and Tunnel Object are described in dedicated sections of this document.

All Unused bits must be set to 0 and ignored.

2.3.7 The Next-Hop Object

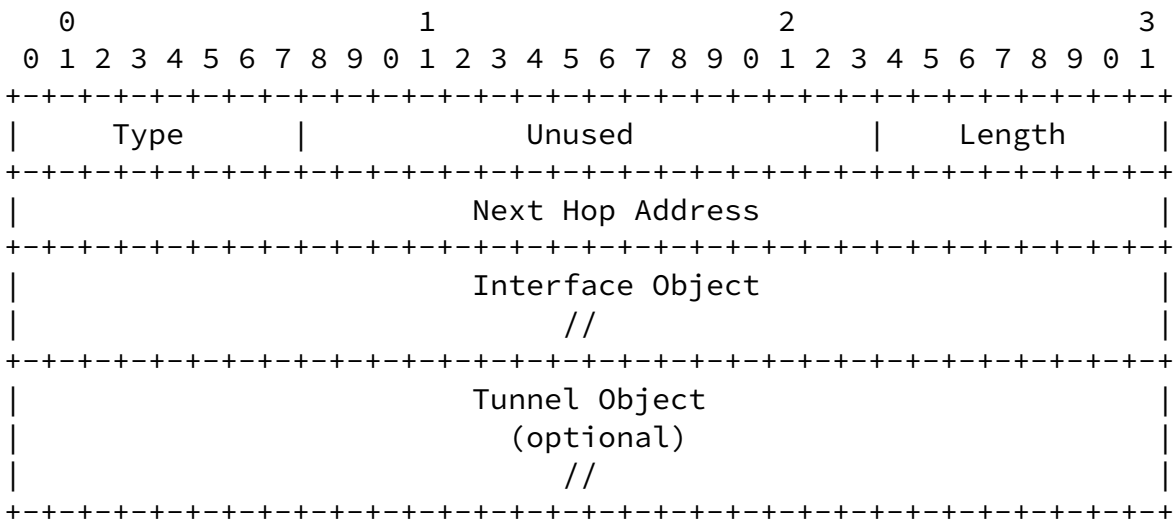


Figure 10: The Next-Hop Object

Figure 10 depicts the Next-hop Object. The Next-Hop Object describes how a device would forward a traceProbe toward its destination. The following paragraphs describe each field that contributes to the Next-hop Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Next-Hop Object, the Type field is always equal to 7.

Length : 8 bits

The Length Field specifies the number of four-octet words that follow.

Next Hop Address : 32 bits

The Next Hop Address identifies the device that supports the next hop by IP address.

The Interface and Tunnel Objects are described in dedicated sections of this document.

All Unused bits must be set to 0 and ignored.

2.3.8 The IP Header Object

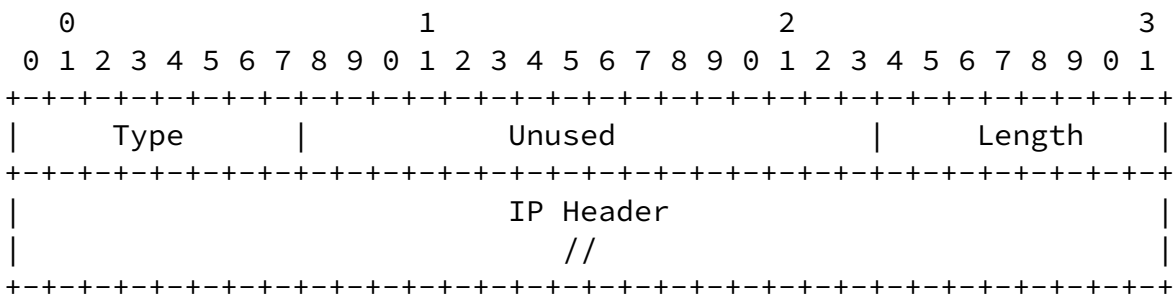


Figure 11: The IP Header Object

Figure 11 depicts the IP Header Object. The IP Header Object contains an IP Header that identifies the top-level path being traced. The following paragraphs describe each field that contributes to the IP Header Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the IP Header Object, the Type field is always equal to 8.

Length : 8 bits

The Length Field specifies the number of four-octet words that follow.

IP Header : Variable Length

This field contains an IP header. It can be 0 padded for word alignment.

All Unused bits must be set to 0 and ignored.

2.3.9 The Interface Object

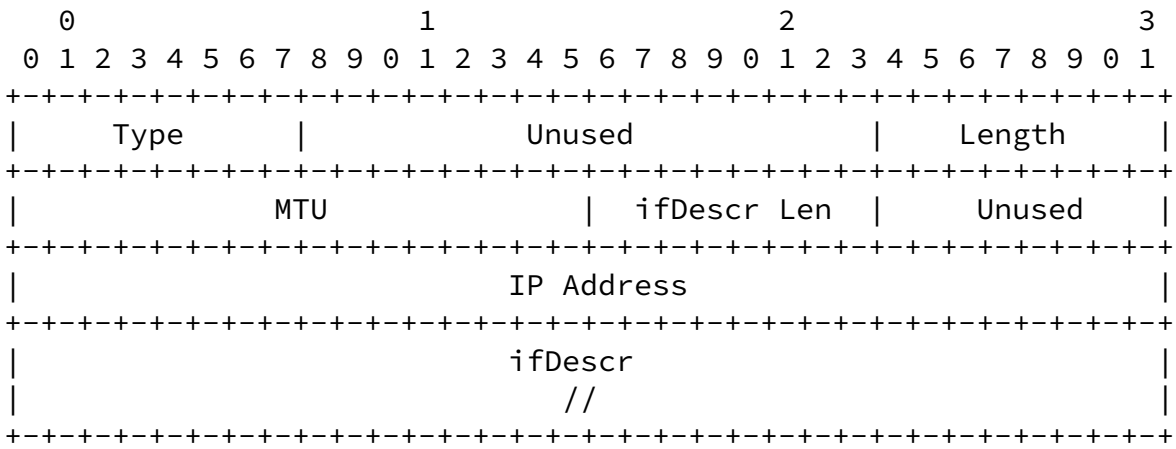


Figure 12: The Interface Object

Figure 12 depicts the Interface Object. The Interface Object identifies an interface and specifies its attributes. The Arrival Object and Next-hop Object can contain the Interface Object.

The following paragraphs describe each field that contributes to the Interface Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Interface Object, the Type field is always equal to 9.

Length : 8 bits

The Length Field specifies the number of four-octet words that follow.

MTU : 16 bits

The MTU field specifies interface MTU in octets.

ifDescr Length : 8 bits

The ifDescr Length Field specifies the length of the ifDescr field, in four-octet words.

IP Address: 32 bits

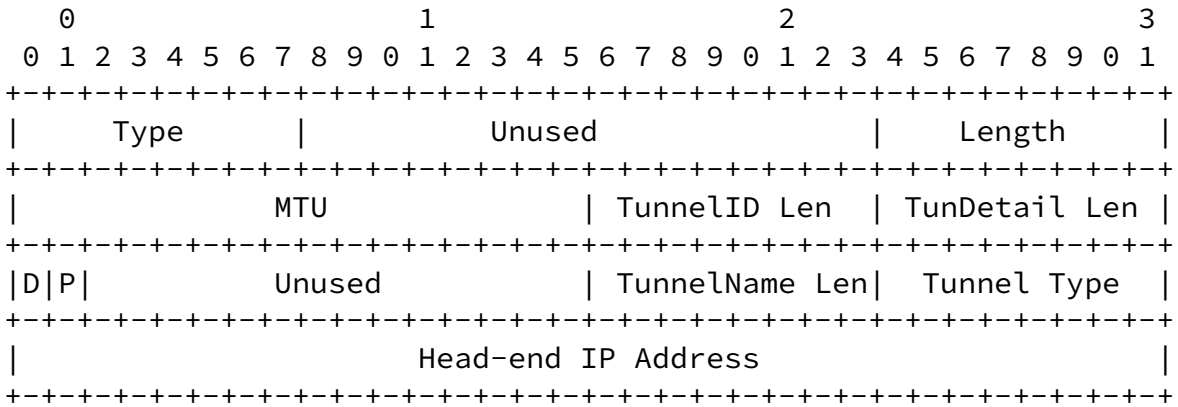
The IP Address field identifies the interface by IP Address.

ifDescr : Variable Length

The ifDescr field identifies the interface by a unique name. This field must contain ASCII printable characters followed by a NULL character. It can be 0 padded for word alignment.

All Unused bits must be set to 0 and ignored.

2.3.10 The Tunnel Object



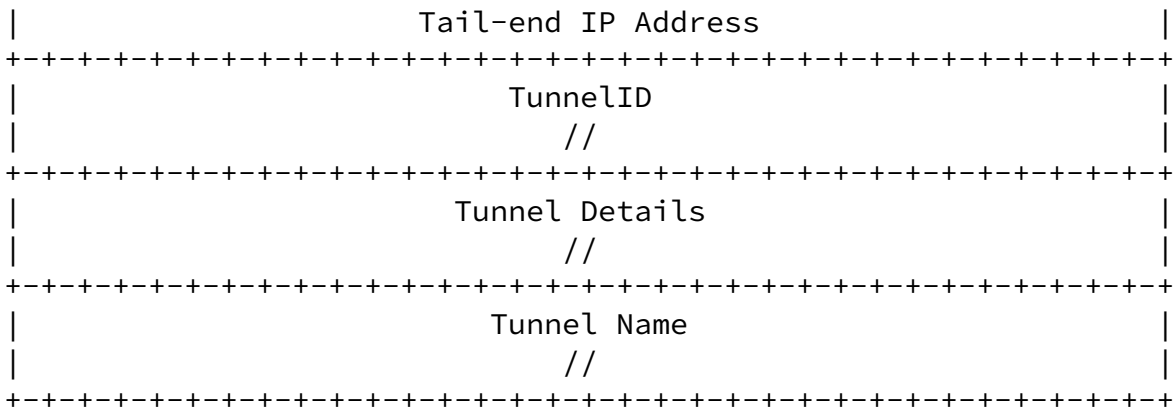


Figure 13: The Tunnel Object

Figure 13 depicts the Tunnel Object. The Tunnel Object identifies a tunnel and specifies its attributes. The Path Object, Arrival Object and Next-hop Object can contain the Tunnel Object.

The following paragraphs describe each field that contributes to the Tunnel Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Tunnel Object, the Type field is always equal to 10.

Length : 8 bits

The Length Field specifies the number of four-octet words that

follow.

MTU : 16 bits

The MTU field specifies tunnel MTU in octets.

TunnelID Length : 8 bits

The TunnelID Length Field specifies the length of the TunnelID field, in four-octet words.

TunDetail Length : 8 bits

The TunDetail Length Field specifies the length of the TunDetail field, in four-octet octet words.

D-Flag : 1 bit

The D-flag is set if the tunnel supports TTL decrement.

P-Flag : 1 bit

The P-Flag is set if the tunnel inherits its TTL value from that of its payload. The P-Flag is cleared if the tunnel always sets its TTL to an arbitrarily high value or does not support TTL.

TunnelName Length : 8 bits

The TunnelName Length Field specifies the length of the TunnelName field, in four-octet words.

Tunnel Type : 8 bits

The Tunnel Type field identifies a tunnel type. Currently, the following tunnel types are defined:

- 0 - IP-in-IP
- 1 - GRE
- 2 - GMPLS
- 3 - MPLS
- 4 - MPLS/LDP Signaled
- 5 - MPLS/RSVP-TE Signaled
- 6 - L2TPv2
- 7 - L2TPv3
- 8 - IPSEC

Head-end IP Address : 32 bits

The Head-end IP Address field identifies the tunnel head-end by IP

address.

Tail-end IP Address : 32 bits

The Tail-end IP Address field identifies the tunnel tail-end by IP

address.

TunnelID : Variable Length

The TunnelID field identifies the tunnel by a unique identifier. Because this field is produced and consumed by the same router, its contents are not the subject of standardization. It can be 0 padded for word alignment.

Tunnel Details : Variable Length

The Tunnel Details field provides tunnel specific details (e.g., MPLS label values). This field must contain ASCII printable characters followed by a NULL character. The route tracing application will print its contents, verbatim. It can be 0 padded for word alignment.

TunnelName : Variable Length

The TunnelName field identifies the tunnel by name. This field must contain ASCII printable characters followed by a NULL character. It can be 0 padded for word alignment.

All Unused bits must be set to 0 and ignored.

2.3.11 The Context Object

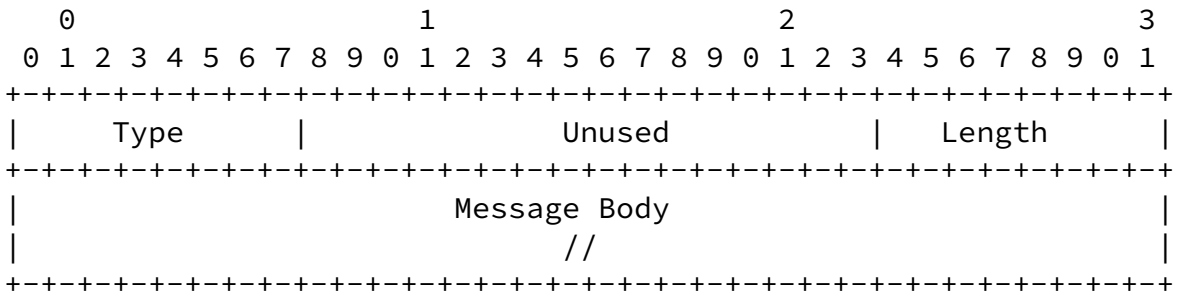


Figure 14: The Context Object

Figure 14 depicts the Context Object. The Context Object provides a context for the Source Object. It is required when the address provided in the Source Object is to be interpreted within the context of a VPN.

The following paragraphs describe each field that contributes to the

Context Object.

Type: 8 bits

The Type field specifies the type of the object that follows. For the Context Object, the Type field is always equal to 11.

Length : 8 bits

The Length Field specifies the length of the object that follows, in four-octet words.

Message Body : variable length

Because the same device that provides the context object interprets its meaning, the syntax of this field need not be standardized.

3. IANA Guidelines

IANA has assigned UDP port 3693 to GTTP. IANA will establish a registry for GTTP codepoints.

4. Security Considerations

The following are security considerations: 1) GTTP MUST enforce access control policy before disclosing any information. 2) GTTP entities should rate limit messages that they send and receive.

5 Normative References

[1] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G. and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[2] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

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

[4] Kessler, G. and S. Shepard, "A Primer On Internet and TCP/IP Tools and Utilities", RFC 2151, June 1997.

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

[6] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[7] Bonica, R., Kompella, K. and D. Meyer, "Tracing Requirements for Generic Tunnels", RFC 3609, September 2003.

Authors' Addresses

Ronald P. Bonica
MCI
22001 Loudoun County Pkwy

Ashburn, VA 20147
US

Phone: +1 703 886 1681
EMail: ronald.p.bonica@mci.com

Eric C. Rosen
Cisco Systems
250 Apollo Drive
Chelmsford, MA 01824
US

EMail: erosen@cisco.com

Dave Meyer
Cisco Systems

EMail: dmm@1-4-5.net

Kireeti Kompella
Juniper Networks
1194 N. Mathilda Ave
Sunnyvale, CA 94089
USA

EMail: kireeti@juniper.net

Rohit Dube
Xebeo Communications
1 Cragwood Road, Suite 100
South Plainfield, NJ 07080

USA

EMail: rohit@xebeo.com

Appendix A. Acknowledgements

Thanks to Richard Rabbat, Adrian Farrel, Randy Bush, Philip Matthews, Tricci So and Beth Alwin for their comments.

Intellectual Property Statement

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 implementors 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.

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the

Internet Society.