# [MS-LLTD]:
# Link Layer Topology Discovery (LLTD) Protocol Specification

---



## Revision Summary

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 12/18/2006 | 0.01 | | MCPP Milestone 2 Initial Availability |
| 03/02/2007 | 1.0 | | MCPP Milestone 2 |
| 04/03/2007 | 1.1 | | Monthly release |
| 05/11/2007 | 1.2 | | Monthly release |

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 06/01/2007 | 1.2.1 | Editorial | Revised and edited the technical content. |
| 07/03/2007 | 1.2.2 | Editorial | Revised and edited the technical content. |
| 07/20/2007 | 1.2.3 | Editorial | Revised and edited the technical content. |
| 08/10/2007 | 1.2.4 | Editorial | Revised and edited the technical content. |
| 09/28/2007 | 1.2.5 | Editorial | Revised and edited the technical content. |
| 10/23/2007 | 1.2.6 | Editorial | Revised and edited the technical content. |
| 11/30/2007 | 1.3 | Minor | Added introduction. |
| 01/25/2008 | 1.3.1 | Editorial | Revised and edited the technical content. |

# Table of Contents

# 1 Introduction

This document specifies the Link Layer Topology Discovery (LLTD) Protocol, a proprietary Microsoft protocol that an application or higher-layer protocol can use to facilitate discovery of link-layer topology and to diagnose various problems that are associated with a network's signal strength and bandwidth.

## 1.1 Glossary

**Access Point (AP):** A device that connects wireless devices to form a wireless network.

**Basic Service Set Identifier (BSSID):** A **MAC address** that is used to identify an entity (such as the **access point**) in a wireless network.

**Controller:** A **station** that initiates a **network test** request.

**Cross-Traffic Analysis:** A technique used by Quality of Service (QoS) applications to understand the nature of network activity, usually resulting in the identification of the hosts that are responsible for most of this activity.

**Cross-Traffic Analysis Initiator:** A **station** that initiates a **cross-traffic analysis** request.

**Enumerator:** A **station** that wants to find all LLTD–capable **stations** on the link by using **quick discovery**.

**Flooding:** A **switch's** sending of a frame to all **segments** to which it is connected. A **switch** will flood a frame containing a **MAC address** for which the **switch** does not know the corresponding **segment**.

**Friendly Name:** A human-readable name that identifies a network resource.

**Ethernet Broadcast Domain:** The portion of a network that can receive frames destined for the special broadcast **MAC address** (that is, consisting of all binary 1s).

**Generation Number:** A number used by a **mapper** to generate fresh **MAC addresses** from a private range.

**Hub:** A data link-layer network device that acts as a shared bus. All **stations** that are connected to a **hub** are on the same **segment**; therefore, each **station** that is connected to a **hub** sees all frames that are sent to or from all other **stations** on that **hub**. Compare this term with **router** and **switch**.

**Interrupt Moderation:** The delay of central processing unit (CPU) interrupts by a local network interface to allow collection and notification of multiple interrupt events to the CPU. This delay improves system efficiency and usage, but it impacts the measurement accuracy of timed events.

**Mapper:** A **station** that initiates a **topology discovery test**.

**Media Access Control (MAC) Address:** A unique link-layer address that identifies a network interface.

**Network Test:** Generic term to describe any technique (for example, probegap or timed probe) that is used to estimate the throughput of a network.

**Quick Discovery:** The process of discovering **responders** on a network.

**Real MAC Address:** A **MAC address** provided by the network interface vendor to uniquely identify the device on the network, as specified in [IEEE802.3].

**RepeatBAND:** A fast and scalable **station** enumeration algorithm as specified in section 3.5.6.2.

**Responder:** An LLTD–capable **station** to which **mappers** and **enumerators** send LLTD commands.

**Router:** A network-layer device that defines the limit of an **Ethernet broadcast domain**. Compare with **hub** and **switch**.

**Segment:** A set of **stations** that see each others' link-layer frames without being changed by any device in the middle, such as a **switch**.

**Service Set Identifier (SSID):** A unique identifier that is used to differentiate one wireless network from another.

**Session:** A context for managing communication over LLTD among **stations**.

**Sink:** A **responder** that is the target of a **network test session**.

**Station:** Any device that implements LLTD.

**Switch:** A data link-layer device that propagates frames between **segments** and allows communication among **stations** on different **segments**. **Stations** that are connected through a **switch** see only those frames destined for their **segments**. Compare this term with **hub** and **router**.

**Topology Discovery Test:** A test that an application or higher-layer protocol can use to facilitate discovering the link-layer topology of a single link in a network. That is, to facilitate discovering the set of **segments** and **switches**, and determining which **responders** are on which segments. Compare this term with **quick discovery**.

**Type-Length-Value (TLV):** A property, as used in this protocol, of a network interface, so named because each property is composed of a **Type** field, a **Length** field, and a value. All LLTD attributes are **TLVs**, as specified in section 2.2.1.1.

**Universally Unique Identifier (UUID):** A 128-bit value that is assigned to an object and is guaranteed to be unique.

**Wireless Band:** A term used to identify an IEEE 802.11 protocol family. For example, 802.11a is a **wireless band**.

**MAY, SHOULD, MUST, SHOULD NOT, MUST NOT:** These terms (in all caps) are used as specified in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

## 1.2 References

### 1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[IANAifType] Internet Assigned Numbers Authority, "IANAifType-MIB Definitions", January 2007, http://www.iana.org/assignments/ianaiftype-mib

[IEEE EtherType] IEEE Standards Association, "IEEE EtherType Field Registration Authority", February 2007, http://standards.ieee.org/regauth/ethertype/eth.txt

[IEEE OUI] IEEE Standards Association, "IEEE OUI Registration Authority", February 2007, http://standards.ieee.org/regauth/oui/oui.txt

[IEEE802.11] Institute of Electrical and Electronics Engineers, "IEEE Standards for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Network - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", ANSI/IEEE Std 802.11, 1999, http://standards.ieee.org/getieee802/download/802.11-1999.pdf

[IEEE802.1Q] Institute of Electrical and Electronics Engineers, "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks", IEEE Std 802.1Q, May 2003, http://ieeexplore.ieee.org/iel5/8557/27089/01203093.pdf

[IEEE802.3] Institute of Electrical and Electronics Engineers, "Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications–Description", IEEE Std 802.2, 2002, http://standards.ieee.org/getieee802/download/802.3-2002.pdf

[ISO-10646] International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO/IEC 10646:2003, December 2003, http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=39921&ICS1

[RFC826] Plummer, D., "An Ethernet Address Resolution Protocol - or -Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", RFC 826, November 1982, http://www.ietf.org/rfc/rfc826.txt

[RFC1123] Braden, R., "Requirements for Internet Hosts–Application and Support", RFC 1123, October 1989, http://www.ietf.org/rfc/rfc1123.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

[RFC2461] Narten, T., Nordmark, E., and Simpson, W., "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998, http://www.ietf.org/rfc/rfc2461.txt

[RFC3022] Srisuresh, P., and Egevang, K., "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, January 2001, http://www.ietf.org/rfc/rfc3022.txt

[RFC3513] Hinden, R. and Deering, S., "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003, http://www.ietf.org/rfc/rfc3513.txt

[UPnP] UPnP Forum, "Standards", http://www.upnp.org/standardizeddcps/default.asp

### 1.2.2 Informative References

[BAND] Black, R., Donnelly, A., Gavrilescu, A., and Thaler, D., "Fast Scalable Robust Node Enumeration", http://research.microsoft.com/~dthaler/BAND.pdf

[MSDN-ICO] Microsoft Corporation, "Icons in Win32", http://msdn2.microsoft.com/en-us/library/ms997538.aspx

[RALLY] Microsoft Corporation, "Windows Rally: Connectivity Technologies for Devices", http://www.microsoft.com/whdc/rally/default.mspx

## 1.3 Protocol Overview (Synopsis)

This document specifies the Link Layer Topology Discovery (LLTD) Protocol, which operates over Ethernet-like media, including both wired (802.3 Ethernet) and wireless (802.11) media. As the protocol name suggests, the core functions of LLTD enable applications to discover the link-layer topology of a single link in a network. That is, it is used to facilitate discovering the set of **switches** and **segments** that constitute the link. LLTD also has Quality of Service (QoS) extensions that applications can use to diagnose problems, such as those problems that involve signal strength on wireless networks or bandwidth constraints in home networks.

LLTD offers the following services, which operate independently on the network (except as noted in this document):

- **Quick discovery**.
- **Topology discovery test**.
- QoS diagnostics for **network tests**.
- QoS diagnostics for **cross-traffic analysis**.

There are no dependencies or ordering restrictions between these services, except that the topology discovery test requires that quick discovery is performed first.

### 1.3.1 Quick Discovery

Quick discovery is the method of enumerating LLTD-capable **stations** on the network and their various properties. Throughout this document, these LLTD-capable stations are referred to as **responders**. That is, the roles of stations involved in the quick-discovery process are as the **enumerator** and the responders. All responders that participate in quick discovery implement a distributed network load balancing algorithm called **RepeatBAND**, as specified in section 3.5.6.2.

RepeatBAND is a scalable enumeration algorithm that allows responders to advertise their presence to enumerators without overloading the network. In this scheme, each responder independently throttles its outbound network traffic by counting the LLTD frames it sees. Responders measure the network load due to LLTD over a number of loosely synchronized rounds, also called blocks, of approximately fixed duration. Each responder uses these load measurements to calculate a current estimate of the number of responders that are active on the network. Each responder then sends a frame in a block with a probability that depends on this estimate (for an analysis of an earlier version of this algorithm which did not accommodate multiple simultaneous enumerators, see [BAND]). These frames each contain a set of properties (or **Type-Length-Values (TLVs)**) that the responders want to advertise to the enumerator.

### 1.3.2 Topology Discovery Tests

In Topology Discovery Tests, the roles of stations are as the **mapper** and the responders. Topology discovery tests are an extension of quick discovery, and they can only be performed after quick discovery is complete. During quick discovery, a mapper temporarily fulfills the role of an enumerator while negotiating its intention to perform topology discovery tests with all responders involved.

Each responder that participates in quick discovery associates itself with a mapper if it does not already have an active association. It is only through this association that a responder accepts and

responds to the associated mapper's topology discovery test commands. This association is also reported by each responder in all quick discovery packet exchanges. While it is the ultimate goal to have only one mapper associated with all responders in a specific **Ethernet broadcast domain**, this mechanism puts the onus on the mapper to ensure that it stops itself completely (and releases any active associations) if it sees a quick discovery packet from any responder that is reporting an association to another mapper.

The mapping **session** makes assumptions about the behavior of the network infrastructure that interconnects the available responders, such as switches and **hubs**. Information about network interfaces and results from particular operations on responders provide the mapper with information to assess the network's topology. One key assumption made is that after a switch has learned a responder's segment, it does not forward traffic destined to that responder's Ethernet address to other segments.

After quick discovery, the mapper knows of available responders and the types of networks they are connected to (such as Ethernet or 802.11 wireless). If the application or higher-layer protocol sees two responders on Ethernet, it could direct the LLTD Protocol to request a responder to send Ethernet frames on the wire by using different source and destination **media access control (MAC) addresses** and ask the other responder which of the Ethernet frames it received. The MAC addresses used are dedicated for use by LLTD.

The choice of which responder to use and the parameters of the topology discovery test are up to the application or higher-layer protocol. An LLTD implementation merely allows applications to learn link details, with which they can construct topology maps using application-specific algorithms.

The LLTD Protocol is used by such an application to request that a chosen responder send LLTD frames with a specified source and destination MAC address, where the source MAC address may or may not be the responder's own MAC address. To avoid interfering with other nodes' MAC addresses, the LLTD Protocol defines a reserved range of MAC addresses that applications can use when they request that a responder use a source MAC address that it does not own.

The LLTD Protocol is also used by such an application to ask other responders which test frames they have seen. This information allows the application to infer the existence of switches and hubs. For example, because a switch will remember a segment it has seen, forwarding frames with the corresponding MAC address to that segment and flooding all segments for frames with previously-unseen MAC addresses, applications can generate tests to determine whether a switch or a hub interconnects two responders.

For example, the application using LLTD might do the following. The application might direct one responder to use a specific LLTD MAC address and train a switch about the segment to which it is connected by sending a frame from that MAC address. The application might then invoke LLTD to request that a second responder and then send a frame to that MAC address. Finally, the application could ask a third responder for the test frames that it saw. If the third responder did not see the test frame (after multiple such tests to reduce the chance of packet loss), the application can conclude that the first and third responders are on different segments; that is, that a switch separates them.

A responder must also be able to perform both quick discovery and topology discovery tests with different stations, where one is functioning as an enumerator and the other is functioning as a mapper.

In addition, this service also allows the mapper to ask a responder for additional property data that is too large to fit into the quick discovery responses.

### 1.3.3 QoS Diagnostics: Network Test

QoS diagnostics for network tests facilitates the determination of a network path's bottleneck bandwidth (or "capacity"), its available bandwidth, and the existence of a prioritization mechanism in a network equipment over a network path. Each of these is a form of network test operation that can be achieved by the use of two roles: a **controller** and a **sink**. The controller role is initiated by a local application. The sink role is implemented in a responder.

The controller's job is to manage a network test session by initializing and resetting the sink, sending test frames to the sink, and accepting test frames that the sink sends back.

For each network path (defined as the network link between a controller and a sink), a higher-layer application may use the time stamp and success code that is returned via the controller to compute the bandwidth and existence of a prioritization mechanism.

### 1.3.4 QoS Diagnostics: Cross-Traffic Analysis

QoS diagnostics for cross-traffic analysis facilitates the detection of network traffic congestion by means of analyzing network packet counters. An application that is interested in analyzing these packet counters invokes the role of the **cross-traffic analysis initiator**. The application explicitly identifies each responder from which it wants to obtain the counters. (The application may have previously learned the responders via quick discovery, or any other method. Hence, this service does not necessarily require that quick discovery is performed first.) The initiator's role is simply to make these counters available to the application, where possible.

Responders that support this feature maintain a history of the following counters:

- Number of bytes received.
- Number of bytes sent.
- Number of frames received.
- Number of frames sent.

Intermediate devices, such as **access points (APs)** and bridges, can make per-network interface counters and aggregate link counters available through this protocol. These counters allow cross-traffic detection even in the absence of responders on the segment. Examples of available network interfaces on a typical AP device are:

- **Basic Service Set Identifier (BSSID)** of a **wireless band**. Note that multiband APs use separate BSSIDs for each band that they support.
- Wired Ethernet network interface that is usually connected to a built-in switch.

The aggregate (across all network interfaces on the same link) counters indicate the amount of traffic that is entering and leaving the link, which enables consideration of the capacity of the uplink in QoS wireless area network (WAN) admission decisions.

It is assumed that the bottleneck point for an AP is always the wireless link. As such, APs are not required to provide the wired local area network (LAN) counters.

## 1.4 Relationship to Other Protocols

The LLTD Protocol operates directly over Ethernet (including media such as 802.11 that support Ethernet encapsulation and hence appear as Ethernet to protocols) and is not used as a transport for other protocols. Therefore, it is a stand-alone protocol.

HTTP is often used in parallel with this protocol because this protocol transfers information that may be directly used by HTTP.

Link Layer Topology Discovery (LLTD) is part of the Windows Rally technologies for enhancing the user experience for computer and device interaction (for more information about Windows Rally, see [RALLY]). LLTD does not have a dependency on any of the other Rally technologies, nor do other Rally technologies depend on LLTD.

## 1.5 Prerequisites/Preconditions

This protocol requires that the implementation have a random number generator whose seed value does not depend solely on the current time because the time could be synchronized on the network. Indeed, for a computer with multiple network interfaces, the time is identical on each network interface. An easily available alternate seed is to use the MAC address of the network interface.

## 1.6 Applicability Statement

This protocol operates at Layer 2 (the link-layer) in the OSI reference model and is therefore not routable. The protocol is suitable only for discovering the link-layer topology of networks that constitute a single link, such as a small office network or a home network. It is not applicable for discovering the Layer 3 (network-layer) topology of a larger network, or for discovering the Layer 2 topology of a link to which the LLTD implementation is not directly attached.

LLTD is designed to scale up to 10,000 nodes on the same link.

## 1.7 Versioning and Capability Negotiation

This protocol has no capability negotiation or versioning aspects, except that messages include a version number for future extensibility.

## 1.8 Vendor-Extensible Fields

This protocol defines a range of special MAC addresses that applications can use when they conduct network topology tests. This range is 0x000D3AD7F140 through 0x000D3AFFFFFF. These MAC addresses do not conflict with actual MAC addresses because the range is built from an assigned Organization Unique Identifier (OUI), as described in section 1.9. To minimize the probability of collisions between two such applications on the same link, while still allowing addresses that the same application uses to be similar (simply for ease in debugging), applications using LLTD SHOULD construct such MAC addresses by using the OUI, followed by a random number in the range 0xD7F2 to 0xFFFF, and leaving 8 bits that can be used to give 256 MAC addresses. The Link Layer Topology Discovery (LLTD) Protocol contains a **generation number** field that can be used as a seed in a pseudo-random number generator.

## 1.9 Standards Assignments

| Parameter | Value | Reference |
|---|---|---|
| Organizationally Unique Identifier (OUI) | 0x000D3A | [IEEE OUI] |
| Ether type | 0x88D9 | [IEEE EtherType] |

# 2 Messages

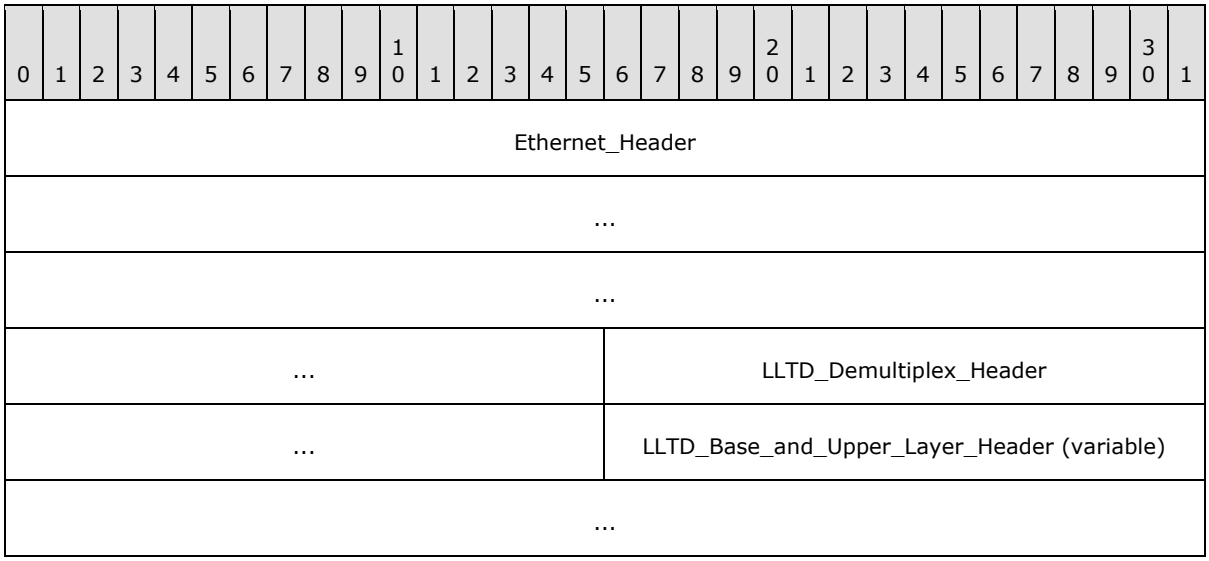
The following sections specify how Link Layer Topology Discovery (LLTD) messages are encapsulated on the wire and common LLTD data types.

## 2.1 Transport

LLTD messages MUST be transported over raw Ethernet, as specified in [IEEE802.3], with the value of the **Ethernet Header Ethertype** field set to 0x88D9.

## 2.2 Message Syntax

The following diagram shows the position of each layer of header in the Link Layer Topology Discovery (LLTD) Protocol.




**Ethernet_Header (14 bytes):** 802.3 defined frame format, as specified in [IEEE802.3], with Ethertype value set to 0x88D9.

**LLTD_Demultiplex_Header (4 bytes):** LLTD framing that indicates message types as specified in section 2.2.3.1.

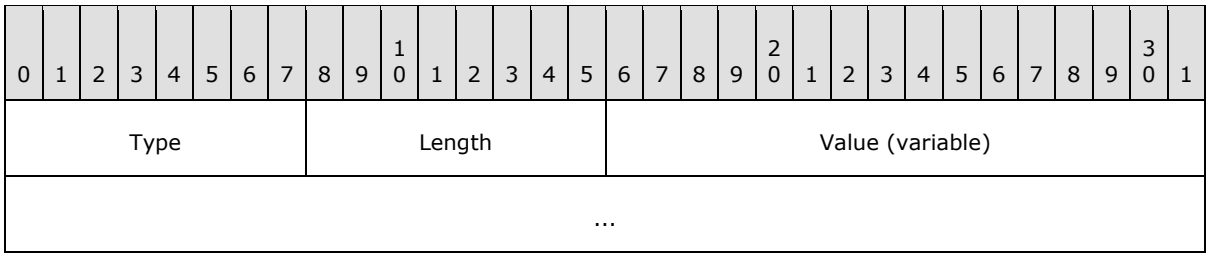
**LLTD_Base_and_Upper_Layer_Header (variable):** Service and message-specific framing header as specified in sections 2.2.4, 2.2.5, and 2.2.6.

### 2.2.1 Common Data Types

#### 2.2.1.1 Attributes

Attributes are used in Hello frames (as specified in section 2.2.4.3) that responders send to enumerators during quick discovery.

All attributes are Type-Length-Values (TLVs) and MUST comply with the following format, except when **Type** is 0x00.




**Type (1 byte):** The **Type** field identifies each attribute. Legal values are specified in the following table, and each attribute is specified in its own subsection.

| Value | Meaning |
|---|---|
| 0x00 | End-of-Property List marker (section 2.2.1.1.1). |
| 0x01 | Host ID (section 2.2.1.1.2) that uniquely identifies the host on which the responder is running. |
| 0x02 | Characteristics (section 2.2.1.1.3). |
| 0x03 | Physical Medium (section 2.2.1.1.4). |
| 0x04 | Wireless Mode (section 2.2.1.1.5). |
| 0x05 | 802.11 Basic Service Set Identifier (BSSID) (section 2.2.1.1.6). |
| 0x06 | 802.11 Service Set Identifier (SSID) (section 2.2.1.1.7). |
| 0x07 | IPv4 Address (section 2.2.1.1.8). |
| 0x08 | IPv6 Address (section 2.2.1.1.9). |
| 0x09 | 802.11 Maximum Operational Rate (section 2.2.1.1.10). |
| 0x0A | Performance Counter Frequency (section 2.2.1.1.11). |
| 0x0C | Link Speed (section 2.2.1.1.12). |
| 0x0D | 802.11 Received Signal Strength Indication (RSSI) (section 2.2.1.1.13). |
| 0x0E | Icon Image (section 2.2.1.1.14). |
| 0x0F | Machine Name (section 2.2.1.1.15). |
| 0x10 | Support Information (section 2.2.1.1.16) that identifies the device manufacturer's support information. |
| 0x11 | Friendly Name (section 2.2.1.1.17). |
| 0x12 | Device Universally Unique Identifier (UUID) (section 2.2.1.1.18). |
| 0x13 | Hardware ID (section 2.2.1.1.19). |
| 0x14 | QoS Characteristics (section 2.2.1.1.20). |
| 0x15 | 802.11 Physical Medium (section 2.2.1.1.21). |
| 0x16 | AP Association Table (section 2.2.1.1.22). |

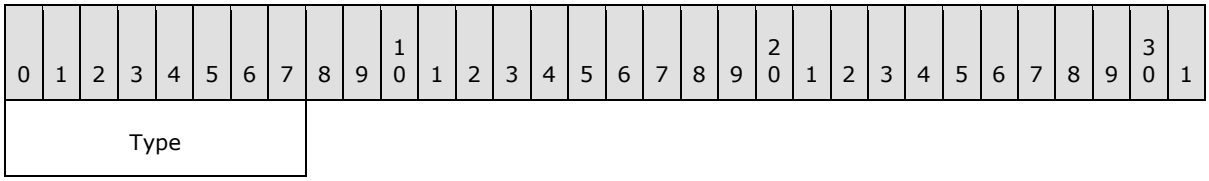
| Value | Meaning |
|---|---|
| 0x18 | Detailed Icon Image (section 2.2.1.1.23). |
| 0x19 | Sees-List Working Set (section 2.2.1.1.24). |
| 0x1A | Component Table (section 2.2.1.1.25). |
| 0x1B | Repeater AP Lineage (section 2.2.1.1.26). |
| 0x1C | Repeater AP Table (section 2.2.1.1.27). |

**Length (1 byte):** This field specifies the length, in bytes, of the **Value** field.

**Value (variable):** This field specifies information that is specific to the attribute, as specified in the corresponding subsection.

### 2.2.1.1.1 End-of-Property List Marker

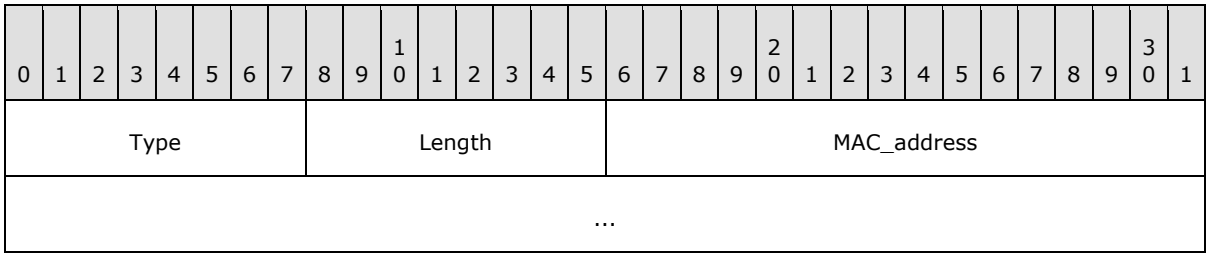
The End-of-Property List Marker attribute signals the end of the TLV list. All responders MUST include this marker in every Hello frame.




**Type (1 byte):** This field MUST be set to 0x00.

### 2.2.1.1.2 Host ID

The Host ID attribute uniquely identifies the host on which the responder is running. All responders MUST include this attribute in all Hello frames.



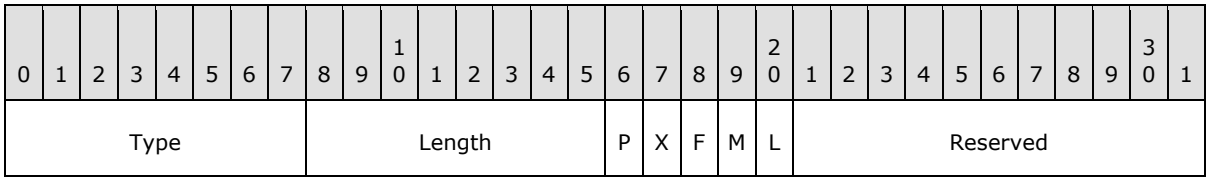

**Type (1 byte):** This field MUST be set to 0x01.

**Length (1 byte):** This field MUST be set to 0x06.

**MAC_address (6 bytes):** This field MUST be the MAC address of the host upon which the responder is running. On a host with multiple network interfaces, this field SHOULD be the lowest MAC address across the network interfaces.

#### 2.2.1.1.3 Characteristics

The Characteristics attribute identifies various characteristics of the responder host and network interface. This attribute is mandatory. All responders MUST include this attribute in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x02.

**Length (1 byte):** This field MUST be set to 0x04.

**P (1 bit):** Network interface is the public side of a network address translation (NAT), as specified in [RFC3022].

**X (1 bit):** Network interface is the private side of a NAT.

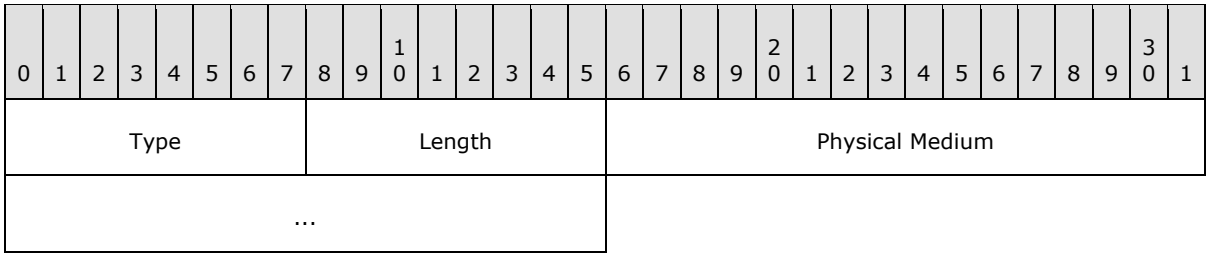
**F (1 bit):** Network interface is in full duplex mode.

**M (1 bit):** Responder MUST set this field if it has a management Web page accessible via the HTTP protocol. A management Web page is optional. A responder MAY support it. The mapper SHOULD construct a URL from the reported IPv6 address. If one is not available, the IPv4 address MUST be used instead. The URL MUST be of the form: "http://<ip-address>/", where "<ip-address>" is either an IPv6 address in IPv6 literal notation (as specified in [RFC3513] section 2.2) or an IPv4 address in four part dotted decimal notation (as specified in [RFC1123] section 2.1).

**L (1 bit):** Network interface is looping back outbound packets.

#### 2.2.1.1.4 Physical Medium

The Physical Medium attribute identifies the physical medium of a network interface by using one of the IANA-published ifType object enumeration values. This attribute is mandatory. All responders MUST include this attribute in all Hello frames.



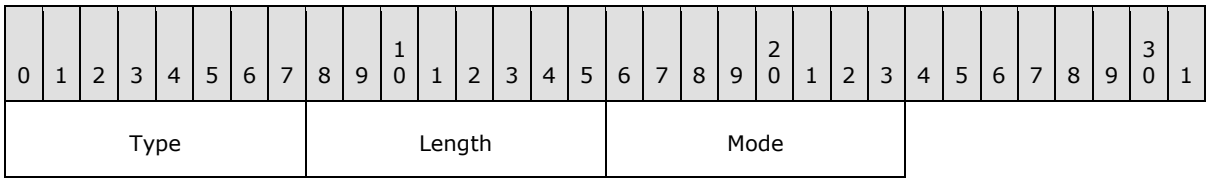

**Type (1 byte):** This field MUST be set to 0x03.

**Length (1 byte):** This field MUST be set to 0x04.

**Physical Medium (4 bytes):** This field MUST be set to the physical medium type of the network interface that the responder is using. The values are published by the Internet Assigned Numbers Authority (IANA) for the ifType object, as specified in [IANAifType].

#### 2.2.1.1.5 Wireless Mode

The Wireless Mode attribute identifies how an Institute of Electrical and Electronics Engineers (IEEE) 802.11, as specified in [IEEE802.11], network interface connects to the network. Implementations with responders in 802.11 stations MUST include this attribute in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x04.

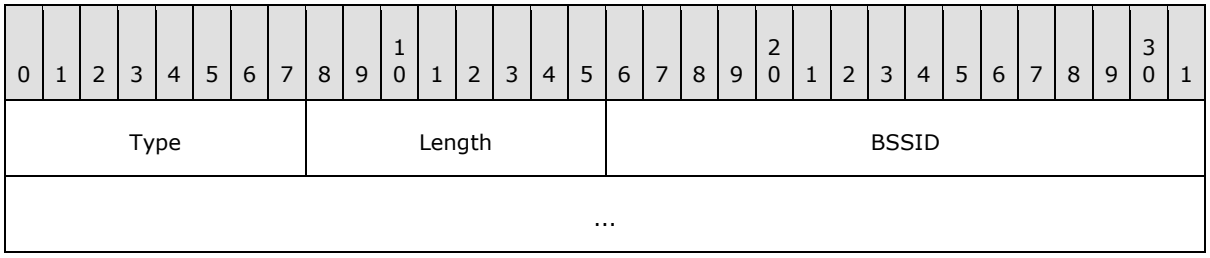
**Length (1 byte):** This field MUST be set to 0x01.

**Mode (1 byte):** This field specifies the method by which a responder's IEEE 802.11 network interface connects to the network. The following table shows valid values.

| Value | Meaning |
|---|---|
| 0x00 | 802.11 IBSS or ad-hoc mode, as specified in [IEEE802.11]. |
| 0x01 | 802.11 infrastructure mode, as specified in [IEEE802.11]. |

#### 2.2.1.1.6 802.11 BSSID

The 802.11 BSSID attribute specifies an IEEE 802.11 network interface's associated AP. Implementations with responders in 802.11 stations MUST include this attribute in all Hello frames.



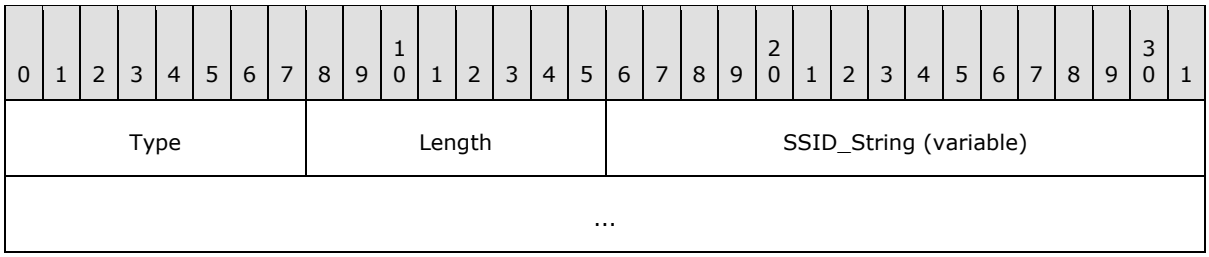

**Type (1 byte):** This field MUST be set to 0x05.

**Length (1 byte):** This field MUST be set to 0x06.

**BSSID (6 bytes):** This field specifies the MAC address of the AP with which a wireless responder's wireless network interface is associated.

#### 2.2.1.1.7 802.11 SSID

The 802.11 SSID attribute specifies an IEEE 802.11 network interface's associated AP. Implementations with responders in 802.11 stations MUST include this attribute in all Hello frames.



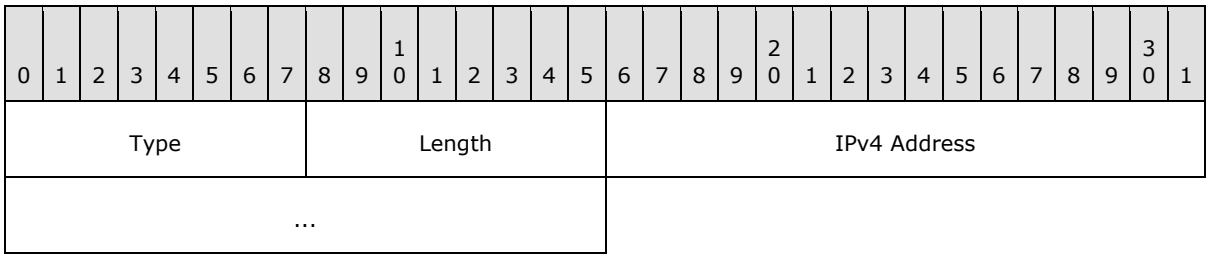

**Type (1 byte):** This field MUST be set to 0x06.

**Length (1 byte):** This field specifies the length in bytes of the **SSID_String** field.

**SSID_String (variable):** This field specifies the **SSID** of the basic service set with which a wireless responder's wireless network interface associates. Note that the string MUST NOT be null terminated and MUST be treated as case sensitive. The maximum length of the string is 32 characters.

#### 2.2.1.1.8 IPv4 Address

The IPv4 Address attribute specifies an IPv4 network address of the responder. This attribute is optional; implementations SHOULD include it in Hello frames if they have an IPv4 address.



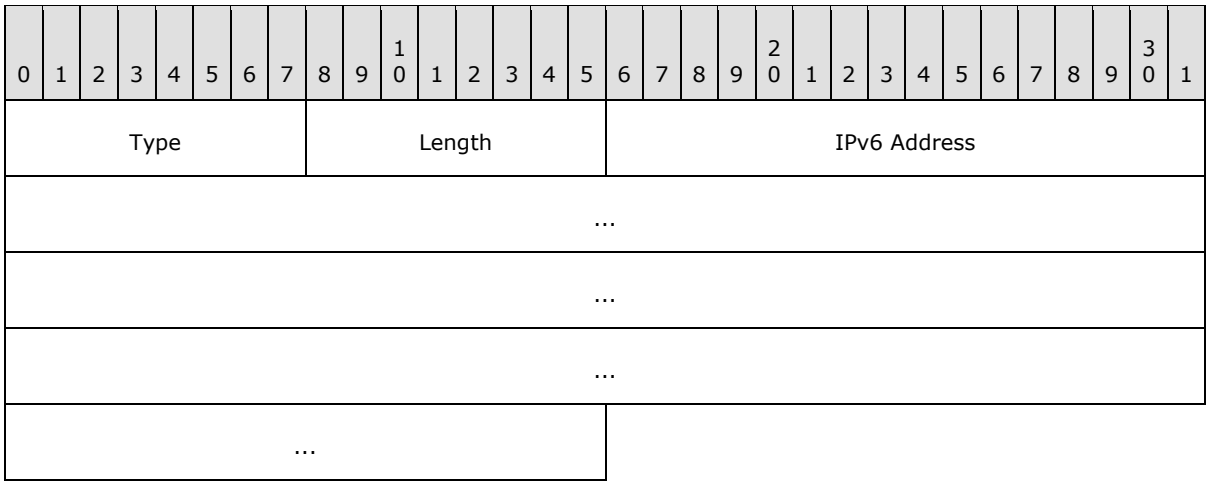

**Type (1 byte):** This field MUST be set to 0x07.

**Length (1 byte):** This field MUST be set to 0x04.

**IPv4 Address (4 bytes):** This field specifies an IPv4 address of the responder. This field's value MUST be an address of the network interface over which the frame is sent, if it has an IPv4 address. If there are multiple IPv4 addresses on the network interface, the device is free to choose any one of them. If an IPv4 address is not available on the network interface over which the frame is sent, the device MAY use an IPv4 address on a different network interface. However, if the responder sets the MW bit in the Characteristics attribute, the address MUST be one which is reachable via the interface over which the frame is sent (for example, if the device is a **router**), and if there is no such address, the responder MUST NOT include the IPv4 **Address** attribute.

#### 2.2.1.1.9 IPv6 Address

The IPv6 Address attribute specifies an IPv6 network address of the responder. This attribute is optional; implementations SHOULD include it in all Hello frames if they have an IPv6 address.



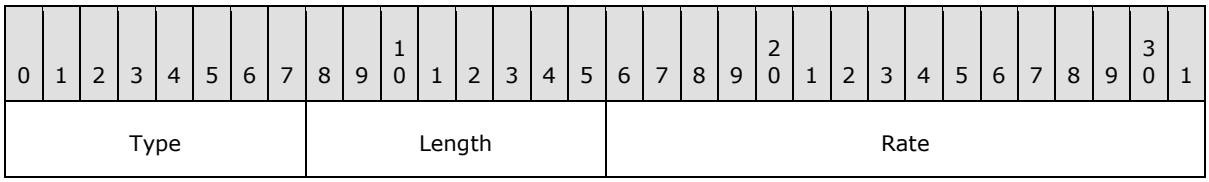

**Type (1 byte):** This field MUST be set to 0x08.

**Length (1 byte):** This field MUST be set to 0x10.

**IPv6 Address (16 bytes):** This field specifies an IPv6 address of the responder. This field's value MUST be an address of the network interface over which the frame is sent, if it has an IPv6 address. If there are multiple IPv6 addresses on the network interface, the device is free to choose any one of them. If an IPv6 address is not available on the network interface over which the frame is sent, the device MAY use an IPv6 address on a different network interface. However, if the responder sets the MW bit in the Characteristics attribute, the address MUST be one which is reachable via the interface over which the frame is sent, and if there is no such address, the responder MUST NOT include the IPv6 Address attribute.

#### 2.2.1.1.10 802.11 Maximum Operational Rate

The 802.11 Maximum Operational Rate attribute specifies the maximum data rate at which the radio can run. This attribute is optional; implementations that support 802.11 MAY include it in Hello frames.



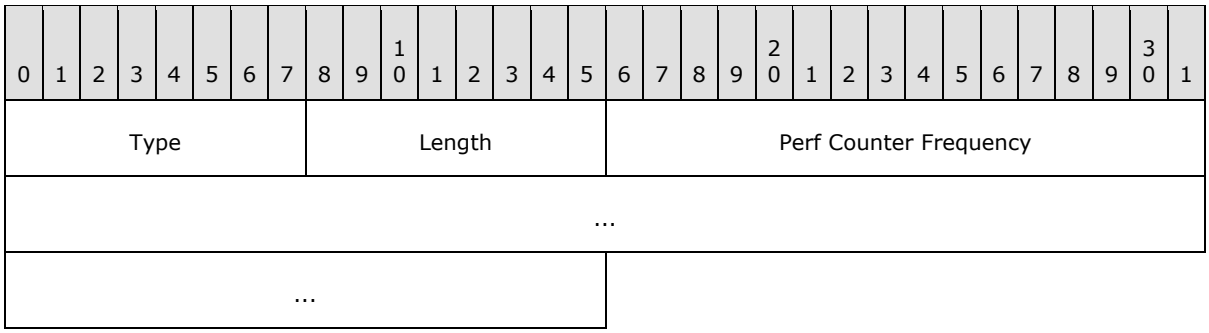

**Type (1 byte):** This field MUST be set to 0x08.

**Length (1 byte):** This field MUST be set to 0x02.

**Rate (2 bytes):** This field specifies the maximum data rate, in network byte order, at which the 802.11 interface can run, in units of 0.5 megabits per second (Mbps).

#### 2.2.1.1.11 Performance Counter Frequency

The Performance Counter Frequency attribute specifies how fast the time stamp counters run in ticks per second. This information is particularly useful for deciphering the results from timed probe and probegap tests in the QoS diagnostics type of service. This attribute is optional; implementations SHOULD include it in all Hello frames.



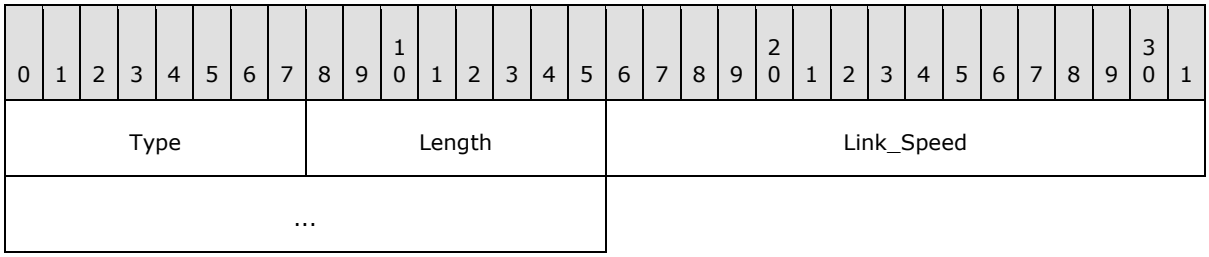

**Type (1 byte):** This field MUST be set to 0x0A.

**Length (1 byte):** This field MUST be set to 0x08.

**Perf Counter Frequency (8 bytes):** This field specifies the number of ticks per second, in network byte order, at which the responder's time stamp counters function.

### 2.2.1.1.12 Link Speed

The Link Speed attribute specifies the network interface's maximum speed in units of 100 bits per second (bps). This attribute is optional; implementations SHOULD include it in all Hello frames.



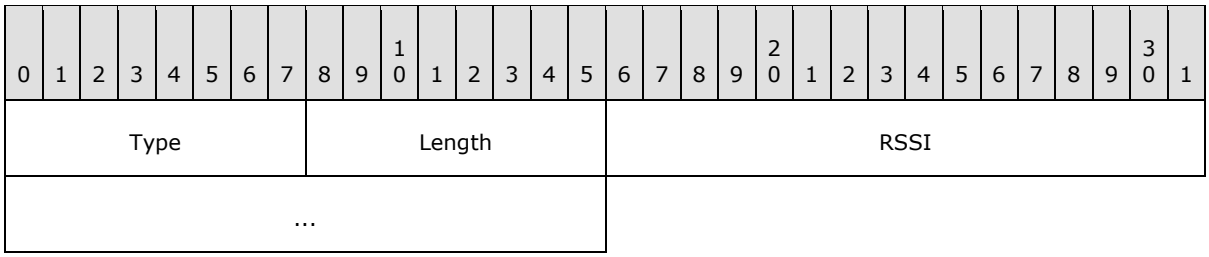

**Type (1 byte):** This field MUST be set to 0x0C.

**Length (1 byte):** This field MUST be set to 0x04.

**Link_Speed (4 bytes):** This field specifies the maximum speed, in network byte order, of the sender's network interface, in units of 100 bps.

### 2.2.1.1.13 802.11 RSSI

The 802.11 RSSI attribute specifies an IEEE 802.11 network interface's RSSI, as specified in [IEEE802.11]. This attribute is optional; responders with 802.11 network interfaces SHOULD include it in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x0D.

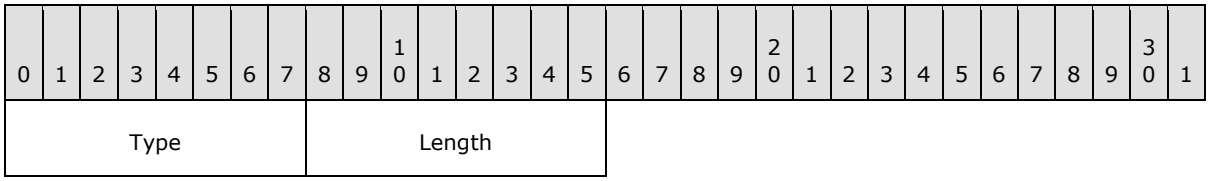
**Length (1 byte):** This field MUST be set to 0x04.

**RSSI (4 bytes):** This field specifies an aligned integer that identifies the IEEE 802.11 network interfaces' RSSI. If the actual RSSI value is available, this field MUST be a negative value (the normal range for an RSSI value is -10 through -200), in dBm in network byte order.

If the actual RSSI value is not available, but the implementation has some other estimate of the signal strength,<1> this field MUST be a value in the range 0 to 100, where a value of 50 means an "average" link quality, and a value of 100 means a "perfect" link.

#### 2.2.1.1.14 Icon Image

The Icon Image attribute specifies that the responder has an icon image that represents the host running the responder and is willing to provide it if a QueryLargeTLV frame requests it. This attribute is optional; implementations SHOULD include it in all Hello frames.

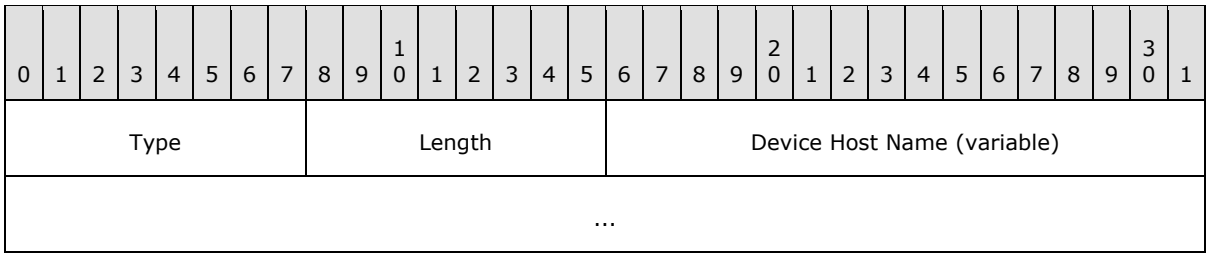



**Type (1 byte):** This field MUST be set to 0x0E.

**Length (1 byte):** This field MUST be set to 0x00.

#### 2.2.1.1.15 Machine Name

The Machine Name attribute specifies an unterminated UCS-2LE string, as specified in [ISO-10646], that identifies the device's host name. This attribute is mandatory; implementations MUST include it in Hello frames.



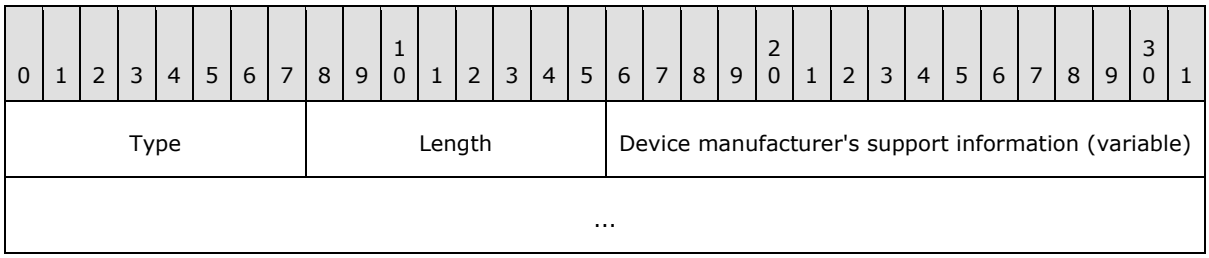

**Type (1 byte):** This field MUST be set to 0x0F.

**Length (1 byte):** This field specifies the length of the **Device Host Name** field, in bytes. This field's value MUST be in the range 2 to 32 (that is, 1 to 16 Unicode characters).

**Device Host Name (variable):** This field specifies a UCS-2LE string that specifies the device's host name, where host name SHOULD be a non-fully qualified domain name. The string MUST NOT be null terminated.

### 2.2.1.1.16 Support Information

The Support Information attribute specifies the device manufacturer's support information (for example, telephone number and support URL). This attribute is optional; implementations SHOULD include it in Hello frames.



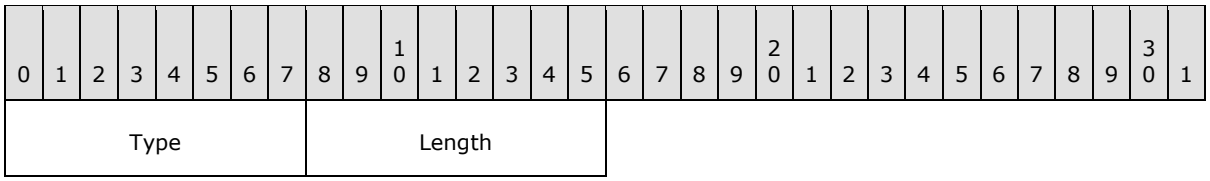

**Type (1 byte):** This field MUST be set to 0x10.

**Length (1 byte):** This field MUST specify a length of 64 octets or less.

**Device manufacturer's support information (variable):** This field specifies a UCS-2LE string that specifies the device manufacturer's support information (such as telephone number). The maximum length of the string is 32 characters or 64 octets. Note that the string MUST NOT be null terminated.

### 2.2.1.1.17 Friendly Name

The Friendly Name attribute indicates that the device has a friendly name and is willing to provide it if a QueryLargeTLV frame requests it. This attribute is optional; implementations MAY include it in Hello frames.

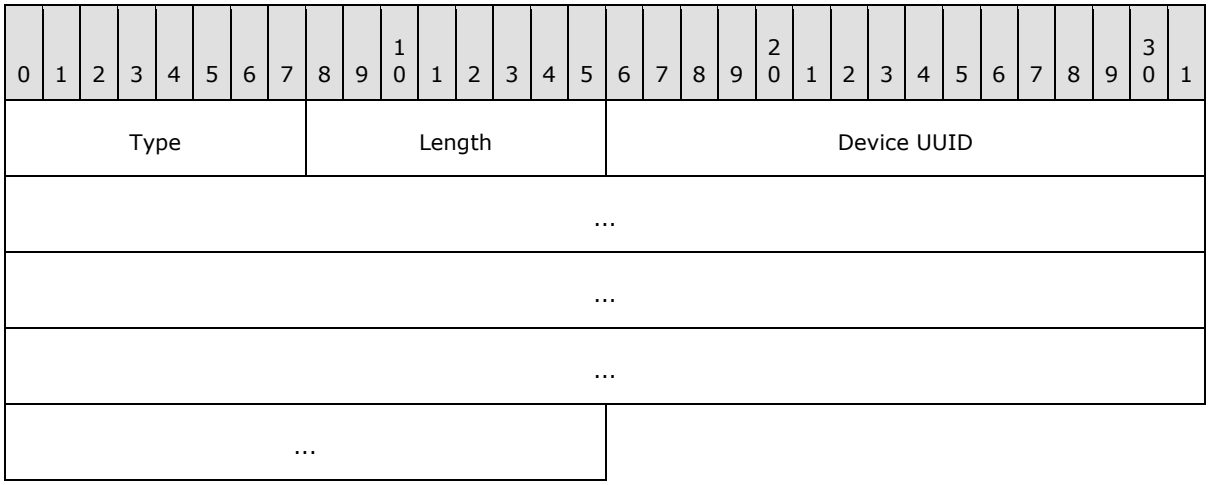



**Type (1 byte):** This field MUST be set to 0x11.

**Length (1 byte):** This field MUST be set to 0x00.

### 2.2.1.1.18 Device UUID

The Device UUID attribute specifies a **UUID** and uniquely identifies a device that supports UPnP, as specified in [UPnP]. This attribute is mandatory for responders in UPnP devices; that is, implementations that include UPnP device functionality MUST include it in all Hello frames.



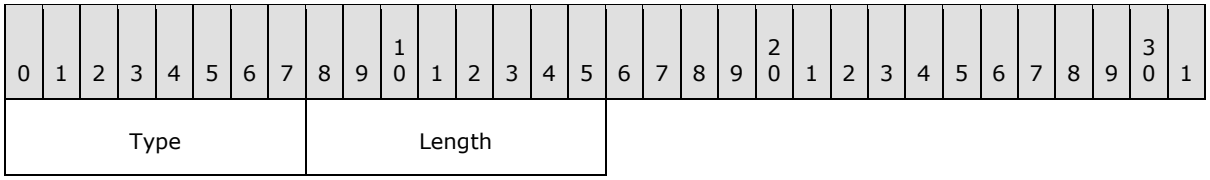

**Type (1 byte):** This field MUST be set to 0x12.

**Length (1 byte):** This field MUST be set to 0x16.

**Device UUID (16 bytes):** This field specifies the UUID that is found in the device unique service name (USN) portion of an Simple Service Discovery Protocol (SSDP) discovery response(as specified in [UPnP] section 1.2.3) in UUID binary format.

### 2.2.1.1.19 Hardware ID

The Hardware ID attribute is used by a responder to indicate that it has a Hardware ID property (see section 2.2.2.3) and is willing to provide it if a QueryLargeTLV frame requests it. This attribute is optional; implementations that support UPnP MAY include it in Hello frames.

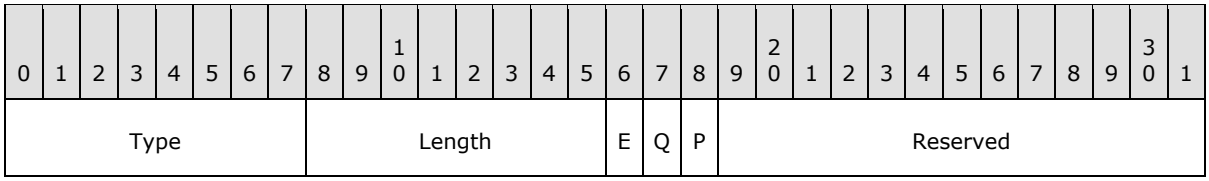



**Type (1 byte):** This field MUST be set to 0x13.

**Length (1 byte):** This field MUST be set to 0x00.

### 2.2.1.1.20 QoS Characteristics

The QoS Characteristics attribute specifies various QoS–related characteristics of the responder host and network interface. This attribute is mandatory for responders that support QoS extensions; implementations MUST include it in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x14.

**Length (1 byte):** This field MUST be set to 0x04.

**E (1 bit):** This field MUST be set if the responder is not providing any Layer 2 forwarding between segments on this link.

**Q (1 bit):** This field MUST be set if the interface supports 802.1q virtual local area network (VLAN) tagging, as specified in [IEEE802.1Q] section 9.

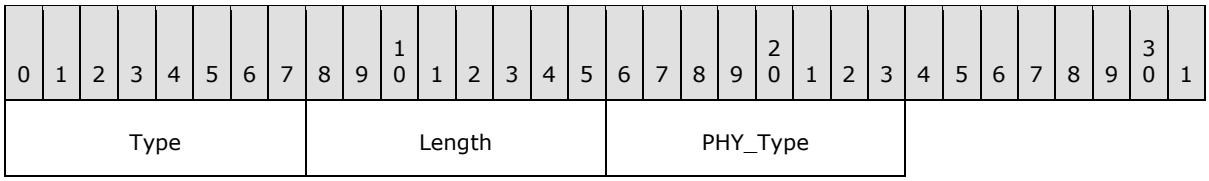
**P (1 bit):** This field MUST be set if the network interface supports 802.1p priority tagging, as specified in [IEEE802.1Q] section 9.

**Reserved (13 bits):** This field MUST be set to 0x00 and ignored on receipt.

0x00
0x0

### 2.2.1.1.21 802.11 Physical Medium

The 802.11 Physical Medium attribute specifies the wireless physical medium. This attribute is mandatory for responders in 802.11 stations; implementations MUST include it in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x15.

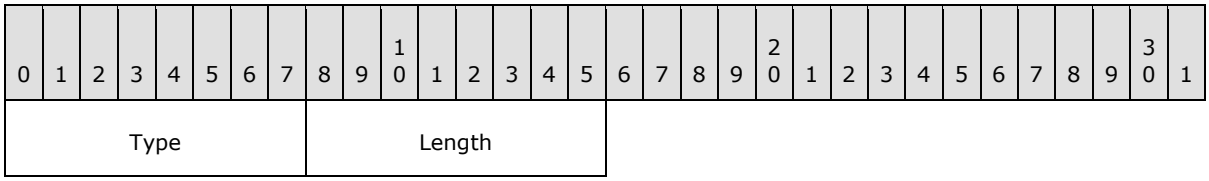
**Length (1 byte):** This field MUST be set to 0x01.

**PHY_Type (1 byte):** A wireless responder MUST use this field to report the 802.11 physical medium in use per dot11PHYType in 802dot11-MIB, as specified in [IEEE802.11] Annex D. The following table shows the valid values.

| Value | Meaning |
|---|---|
| 0x00 | Unknown |
| 0x01 | FHSS 2.4 gigahertz (GHz) |
| 0x02 | DSSS 2.4 GHz |
| 0x03 | IR Baseband |
| 0x04 | OFDM 5 GHz |
| 0x05 | HRDSSS |
| 0x06 | ERP |
| 0x07 — 0xFF | Reserved for future use. |

#### 2.2.1.1.22 AP Association Table

The AP Association Table attribute indicates that the responder is an AP with an AP Association Table that lists wireless hosts that are associated with it, and is willing to provide it if a QueryLargeTLV frame requests it. This attribute is mandatory for 802.11 Access Point responders; APs MUST include it in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x16.

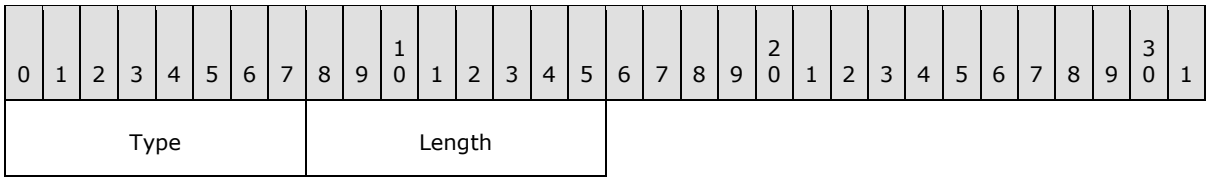
**Length (1 byte):** This field MUST be set to 0x00.

#### 2.2.1.1.23 Detailed Icon Image

The presence of a Detailed Icon Image attribute indicates that the responder has a Detailed Icon Image and is willing to provide it if a QueryLargeTLV requests it. A Detailed Icon Image is a high-resolution graphical representation of the device running the responder, as opposed to an Icon Image attribute, which is lower resolution. This attribute is optional; implementers MAY include it in Hello frames.

If a responder includes this attribute, it SHOULD also include the smaller Icon Image attribute. If space is restricted such that only one icon image is available in the responder, the responder MUST return the Icon Image in the Hello frame if the image is less than or equal to 32,768 octets, or it MUST return this Detailed Icon Image attribute in the Hello frame if the icon image is greater than 32,768 octets and less than or equal to 262,144 octets.

The Detailed Icon Image attribute MUST have the following format.

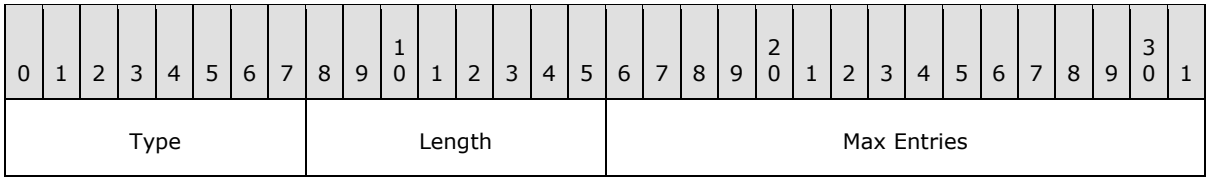



**Type (1 byte):** This field MUST be set to 0x18.

**Length (1 byte):** This field MUST be set to 0x00.

### 2.2.1.1.24 Sees-List Working Set

The Sees-List Working Set attribute specifies the maximum entry count in the responder's sees-list database. This attribute is optional; implementations SHOULD include it in all Hello frames.



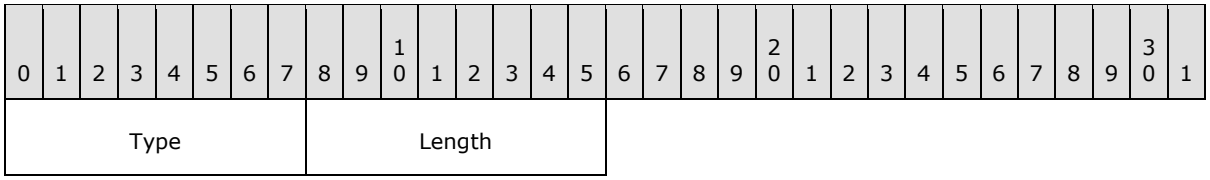

**Type (1 byte):** This field MUST be set to 0x19.

**Length (1 byte):** This field MUST be set to 0x02.

**Max Entries (2 bytes):** A responder MUST include this field to report the maximum count, in network byte order of **RecveeDesc** entries (as specified in section 2.2.4.9) that may be stored in its sees-list database.

### 2.2.1.1.25 Component Table

The presence of the Component Table attribute indicates that the responder has a Component Table that specifies a responder's internal components, allowing the mapper to generate a more accurate topology map, and that the responder is willing to provide it if a QueryLargeTLV requests it. Responder implementations in multifunction devices MUST include this attribute in all Hello frames.



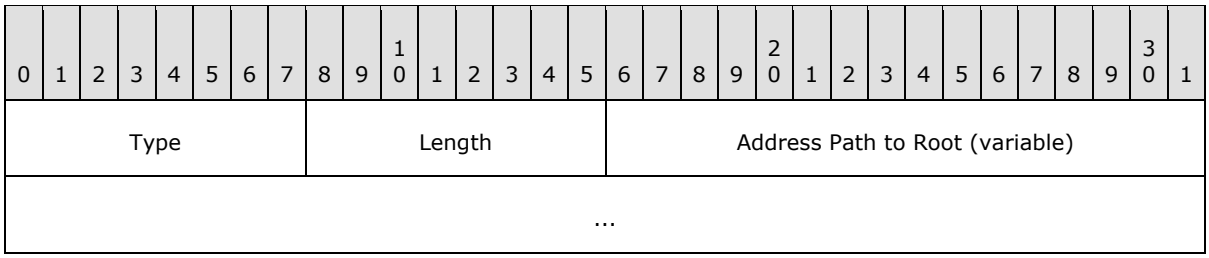

**Type (1 byte):** This field MUST be set to 0x1A.

**Length (1 byte):** This field MUST be set to 0x00.

### 2.2.1.1.26 Repeater AP Lineage

The Repeater AP Lineage attribute specifies the address of the parent and may optionally hold the chain of parents up to the root of the 802.11 Distribution System, as specified in [IEEE802.11] section 5.2.2. A responder in an access point operating in repeater mode MUST use this attribute to provide the address of the parent (which MUST be the same as the reported BSSID because this device is also a client) and each subsequent parent toward the root, if available.

Responders in 802.11 access points MUST include this attribute in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x1B.

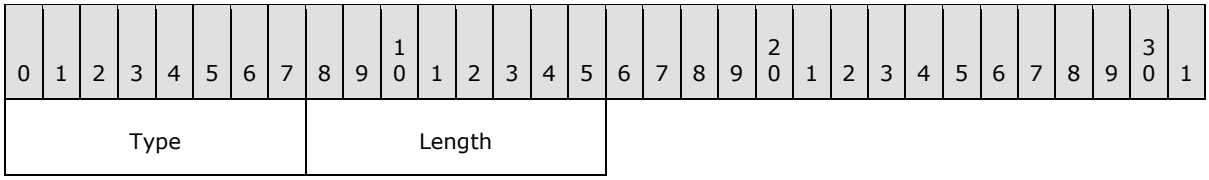
**Length (1 byte):** This field MUST be set to a multiple of 6, with a maximum of 36.

6
12
18
24
30
36

**Address Path to Root (variable):** If the sender is the root of the 802.11 Distribution System, this field MUST be empty (not present). Otherwise, it MUST contain a list of up to six MAC addresses, where the first address is the parent AP address, the second address is that AP's parent, and so forth until either the root MAC address is reached or six addresses have been included.

### 2.2.1.1.27 Repeater AP Table

The Repeater AP Table attribute indicates that the responder has the routing table that a responder is using for packets to addresses that are not directly associated, and that the responder is willing to provide it if a QueryLargeTLV requests it. If the access point is a repeater AP as part of a Wireless Distribution System, this information permits the mapper to generate a more accurate topology map. This attribute is mandatory for responders in 802.11 repeater access points; such implementations MUST include it in all Hello frames.




**Type (1 byte):** This field MUST be set to 0x1C.

**Length (1 byte):** This field MUST be set to 0x00.

## 2.2.2 Large Data Properties

The **QueryLargeTlvResp** frame, as specified in 2.2.4.14, is used to return (portions of) data properties that are declared as zero length in Hello frames.

#### 2.2.2.1 Icon Image

The property data MUST be an icon image, at most 32,768 bytes long. The image MUST be in any image format that has a unique signature at the beginning, so the receiver can detect the image format purely by inspecting the image. There are many file formats that meet this requirement, including GIF and JPEG. A responder that chooses to support this property MAY use any such format, and the mapper MAY<2> recognize any such formats it chooses. If the image is not in a format the mapper recognizes, the mapper MUST use a default image it has, in place of the one it received from the responder.

#### 2.2.2.2 Friendly Name

The Friendly Name property contains a non-NULL-terminated UCS-2LE string that identifies the device's **friendly name**. This property's value MUST be between 2 and 64 bytes (1 and 32 characters) in length.

#### 2.2.2.3 Hardware ID

The Hardware ID property contains a non-NULL-terminated UCS-2LE string. This information MUST come from the UPnP device description phase, as specified in [UPnP] section 2.1.<3>

The Hardware ID MUST follow these formatting rules:

- Characters with an ASCII value less than 0x20 are not allowed.
- Characters with an ASCII value greater than 0x80 are not allowed.
- Commas are not allowed.
- All spaces " " MUST be replaced with an underscore character "_".

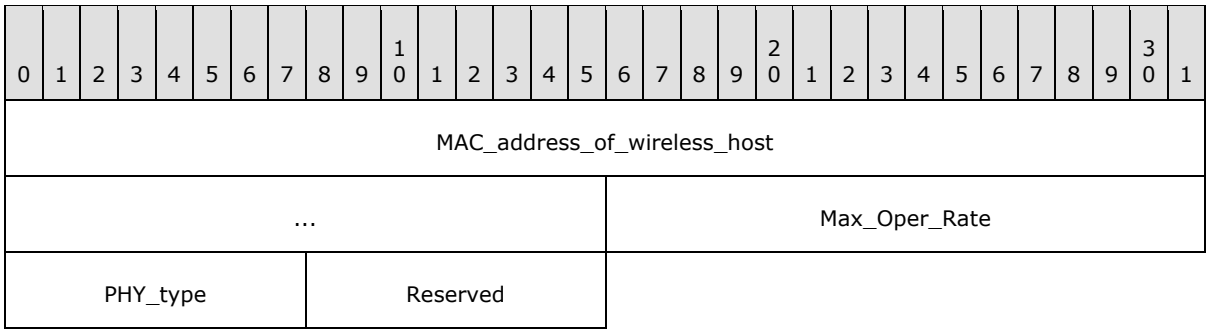
Note that the string MUST NOT be null terminated.

The maximum length of the string is 200 characters (400 octets) and MUST be provided in UCS-2LE format.

#### 2.2.2.4 AP Association Table

A wireless access point responder uses this data object to report the wireless hosts that are associated with it. This information is particularly useful for discovering legacy wireless devices that do not implement the responder. Additionally, it allows the mapper to conclusively match wireless hosts that are associated with the same access point via different BSSIDs (for example, one for each supported band).

The table MUST contain 0 or more entries for associated stations, where each entry MUST have the following format.




**MAC_address_of_wireless_host (6 bytes):** MAC address of the particular 802.11 station that is associated with the AP.

**Max_Oper_Rate (2 bytes):** The maximum operational data rate at which the selected radio can run to the given host, in network byte order. The data rate MUST be encoded in units of 0.5 Mbps.

**PHY_type (1 byte):** The physical medium type for the given host. Valid values are defined in section 2.2.1.1.21.

**Reserved (1 byte):** This field MUST be set to 0, and it MUST be ignored upon receipt.

If the size of the actual AP association table exceeds 409 entries, the responder MUST make only 409 entries available in this data object. It is up to the implementor to choose which stations to make available in that case.

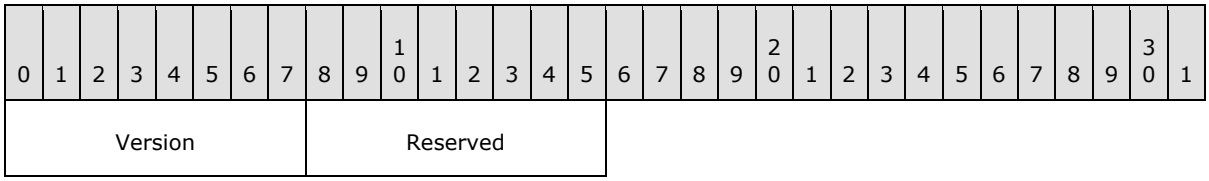
### 2.2.2.5 Detailed Icon Image

The Detailed Icon Image property's data MUST be a high-resolution icon image, at most 262,144 bytes in length. The image format requirements are the same as specified in section 2.2.2.1.

### 2.2.2.6 Component Table

The Component Table data object is used by multifunction devices such as APs to report their internal components.

The table MUST be at most 4096 bytes in size and contain 0 or more entries for the sender's components, where each entry MUST begin with a header that is two octets in length.

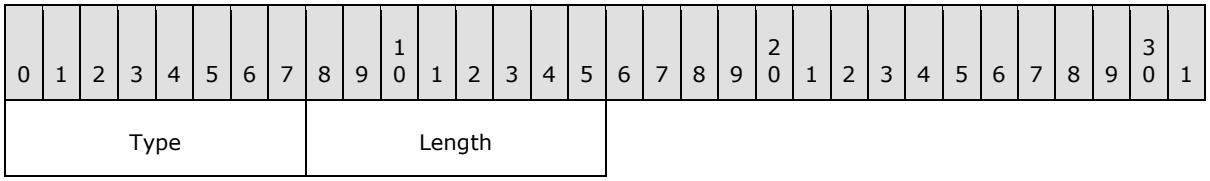



**Version (1 byte):** This field MUST be set to 0x01.

**Reserved (1 byte):** This field MUST be set to 0x00 on transmission and MUST be ignored on receipt.

#### 2.2.2.6.1 Component Descriptors

The Component Table header MUST be followed by an arbitrary number of component descriptors, each carrying a mandatory header.



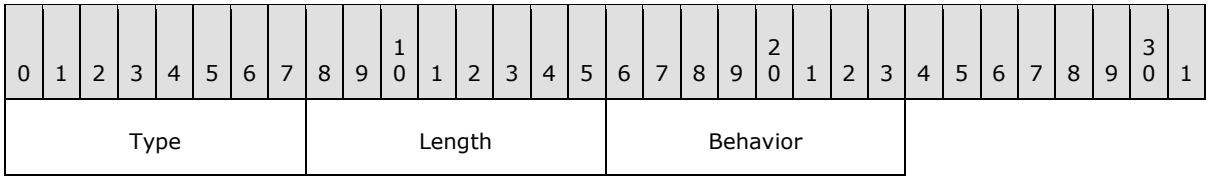

**Type (1 byte):** This field is the component type. The following table shows the valid values.

| Value | Meaning |
|---|---|
| 0x00 | A bridge that interconnects all identified wireless local area network (WLAN) and LAN segments. It is assumed that the responder reporting the Component Table attribute is connected directly into this bridge. |
| 0x01 | This field is the 802.11 access point. |
| 0x02 | This field is a built-in switch. If a bridge component (type 0x00) also exists, it indicates that this switch connects directly into the bridge. If a bridge component does not exist, it indicates that the switch is connected directly to the built-in responder. |

**Length (1 byte):** This field specifies the length (in octets) of the descriptor payload immediately following this header.

##### 2.2.2.6.1.1 Bridge Component Descriptor

A bridge component descriptor with **Type** value 0x00 MUST have the following format.




**Type (1 byte):** This field MUST be set to 0x00.

**Length (1 byte):** This field MUST be set to 0x01.

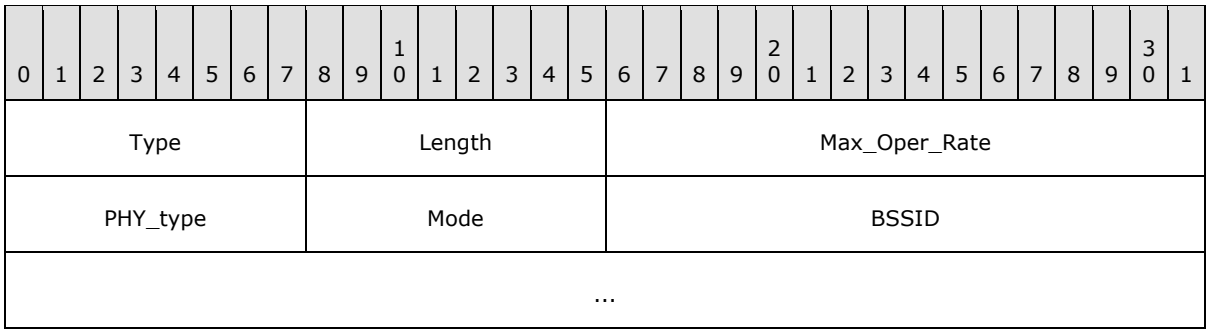
**Behavior (1 byte):** This field identifies the behavior of the bridge. Valid values are the following.

| Value | Meaning |
|---|---|
| Hub<br>0x00 | All packets transitioning through the bridge are seen on the responder. |
| Switch<br>0x01 | Packets from LAN or WLAN are seen only on the responder if they are broadcast or explicitly targeted at the responder. |
| Internal_hub_switch<br>0x02 | Packets transitioning through the bridge are seen on the responder; however, the bridge learns addresses like a switch, provided that they |

| Value | Meaning |
|---|---|
| | initiate on components other than the responder. |

### 2.2.2.6.1.2 802.11 Access Point Component Descriptor

An 802.11 AP component descriptor with **Type** value 0x01 MUST have the following format.




**Type (1 byte):** This field MUST be set to 0x01.

**Length (1 byte):** This field MUST be set to 0x0A.

**Max_Oper_Rate (2 bytes):** The maximum operational data rate at which the radio can function, encoded in units of 0.5 Mbps in network byte order.

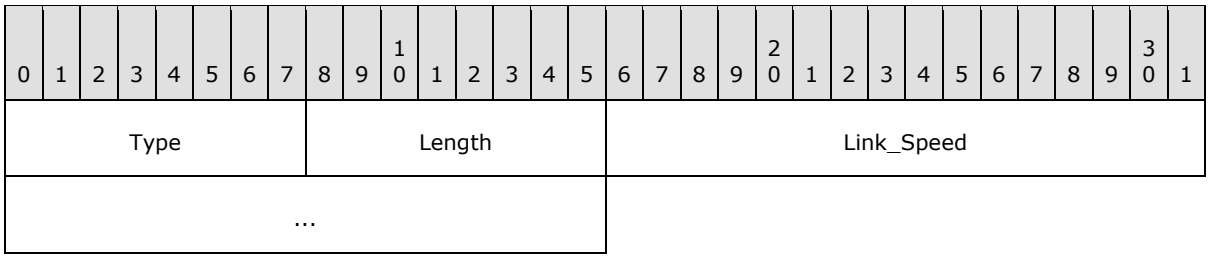
**PHY_type (1 byte):** This field is the physical medium type. Valid values are defined in section 2.2.1.1.21.

**Mode (1 byte):** This field specifies how the radio connects to the wireless network. Valid values are defined in section 2.2.1.1.5.

**BSSID (6 bytes):** The MAC address of the AP that is hosting the SSID.

### 2.2.2.6.1.3 Built-in Switch Component Descriptor

A built-in switch component descriptor with **Type** value 0x02 MUST have the following format.




**Type (1 byte):** This field MUST be set to 0x01.

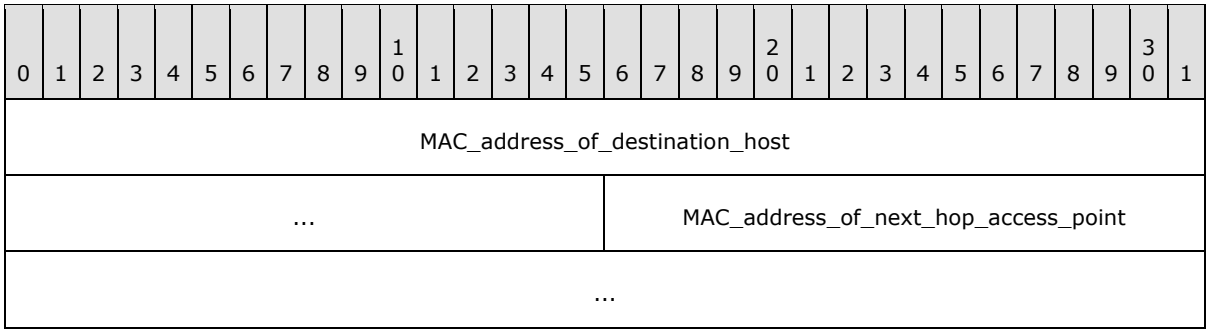
**Length (1 byte):** This field MUST be set to 0x04.

**Link_Speed (4 bytes):** The maximum speed of the switch, in units of 100 bps in network byte order.

### 2.2.2.7 Repeater AP Table

The Repeater AP Table data object is used by repeater access points to report station routing information.

The table MUST contain a list of 0 or more entries where each entry represents a host and AP pair. Each table entry is 12 octets in length, and the format MUST be the following.




**MAC_address_of_destination_host (6 bytes):** This field specifies the MAC address of the particular 802.11 station that is associated with another AP.

**MAC_address_of_next_hop_access_point (6 bytes):** This field MUST be one of the BSSID addresses that are listed in the AP Association Table through which the AP can reach the destination host. The implementor is free to choose any such BSSID address.

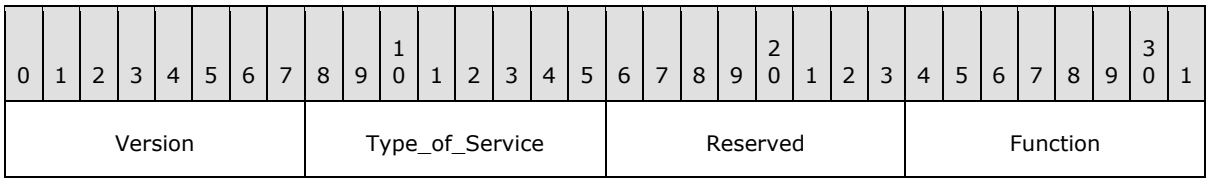
If the size of the actual Repeater AP Table exceeds 256 entries, the responder MUST make only 256 entries available in this property. It is up to the implementor to choose which host and AP pairs are made available in that case.

## 2.2.3 Base Specification

All Link Layer Topology Discovery (LLTD) Protocol implementations MUST use and accept the following base specification format.

### 2.2.3.1 Demultiplex Header Format

The Demultiplex header format is defined as follows.




**Version (1 byte):** This field specifies the version of the Demultiplex header. This field MUST be set to 0x01.

**Type_of_Service (1 byte):** This field specifies the purpose of the frame. This field MUST be one of the following.

| Value | Meaning |
|---|---|
| 0x00 | Topology discovery |

| Value | Meaning |
|---|---|
| 0x01 | Quick discovery |
| 0x02 | QoS diagnostics |

**Reserved (1 byte):** This field is reserved for future use. It MUST be set to 0x00 on transmission and ignored on receipt.

0x00
0x0

**Function (1 byte):** This field is the type of message for a given type of service. The following functions are valid for service type 0x00.

| Value | Meaning |
|---|---|
| 0x00 | Discover |
| 0x01 | Hello |
| 0x02 | Emit |
| 0x03 | Train |
| 0x04 | Probe |
| 0x05 | Ack |
| 0x06 | Query |
| 0x07 | QueryResp |
| 0x08 | Reset |
| 0x09 | Charge |
| 0x0A | Flat |
| 0x0B | QueryLargeTlv |
| 0x0C | QueryLargeTlvResp |

The following functions are valid for service type 0x01.

| Value | Meaning |
|---|---|
| 0x00 | Discover |
| 0x01 | Hello |
| 0x08 | Reset |

The following functions are valid for service type 0x02.

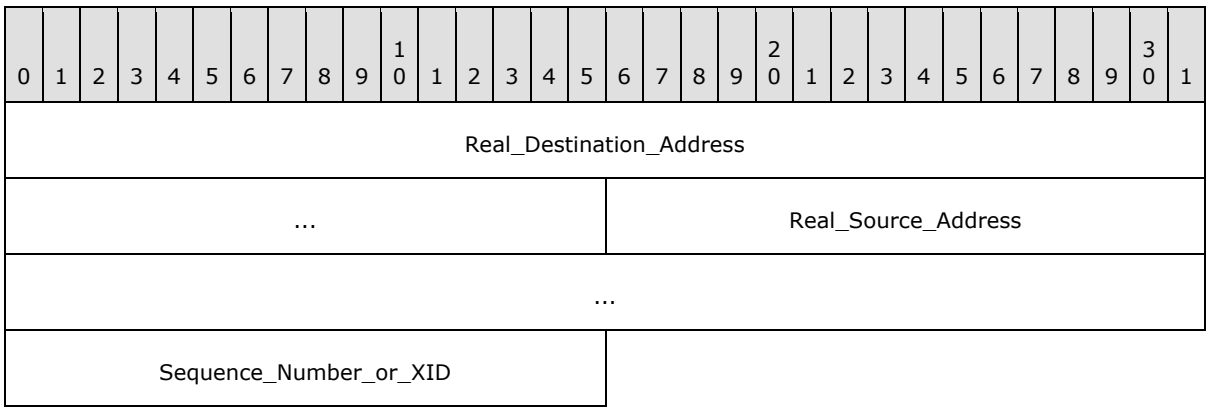
| Value | Meaning |
|---|---|
| 0x00 | QosInitializeSink |
| 0x01 | QosReady |
| 0x02 | QosProbe |
| 0x03 | QosQuery |
| 0x04 | QosQueryResp |
| 0x05 | QosReset |
| 0x06 | QosError |
| 0x07 | QosAck |
| 0x08 | QosCounterSnapshot |
| 0x09 | QosCounterResult |
| 0x0A | QosCounterLease |

### 2.2.4 Topology Discovery Tests and Quick Discovery

#### 2.2.4.1 Base Header Format

This base header MUST be used when the Type of Service value in the Demultiplex header is set to 0x00 (quick discovery) or 0x01 (topology discovery).

The Base header MUST be the following.




**Real_Destination_Address (6 bytes):** This field specifies the intended destination's real MAC address.

**Real_Source_Address (6 bytes):** This field specifies the sender's real MAC address. A sender MUST set the real source and destination MAC addresses to its own MAC address and its intended destination MAC address, respectively. These fields are required because the source and destination address fields of the Ethernet header are rewritten by some network devices and thus may not survive an end-to-end transmission.

**Sequence_Number_or_XID (2 bytes):** If the frame is a Discover frame, this field MUST contain a transaction ID (XID). Otherwise, it MUST contain a sequence number.

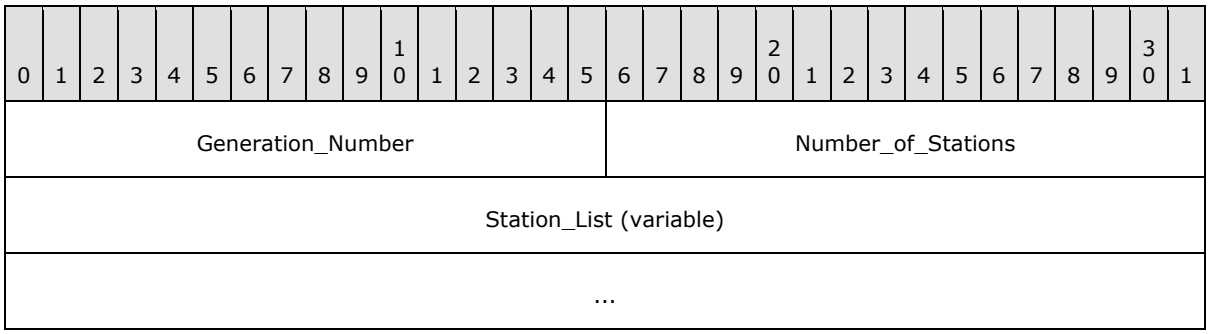
A sequence number, in network byte order, correlates a response to a specific request and increments using ones-complement arithmetic. The sequence number ensures reliability of certain packets in the protocol.

An XID is used to uniquely identify the mapper or enumerator session. A mapper MUST randomly generate two XIDs at initialization: one MUST be used for topology discovery tests, and one MUST be used for quick discovery. With stable storage, XID values MUST be sequential; without stable storage, XID values MAY be assigned at random.

### 2.2.4.2 Discover Upper-Level Header Format

A Discover frame is broadcast by an enumerator to all responders to initiate quick discovery and cause responders to start responding with Hello frames.

The Discover header MUST immediately follow the Base header.



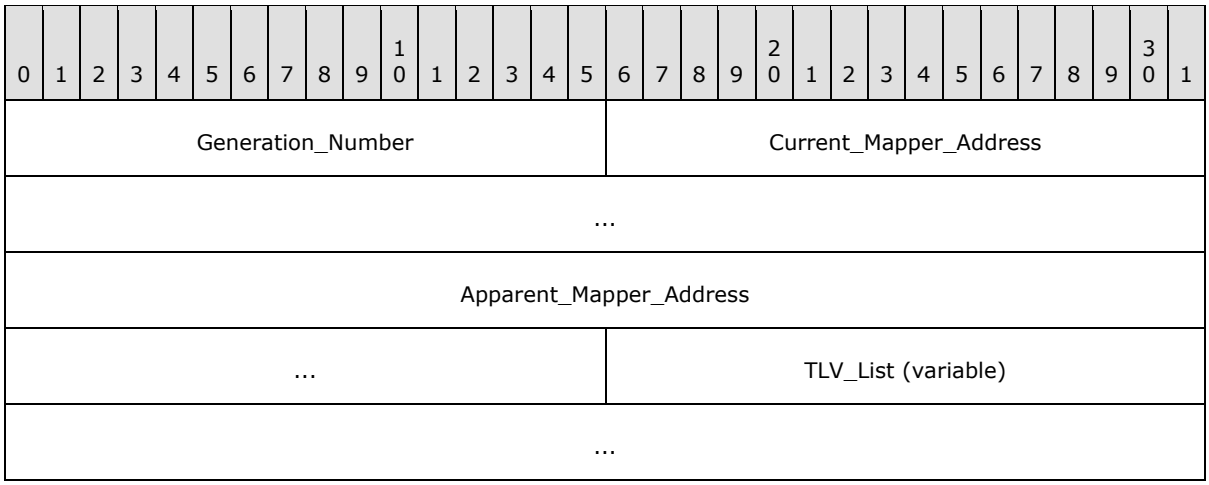

**Generation_Number (2 bytes):** This field contains an unsigned integer in network byte order. This field allows the mapper to negotiate a generation number with the responders that respond to a Discover frame. Ultimately, this number allows the mapper to generate a unique range of MAC addresses from the reserved topology discovery address pool, as specified in [IEEE OUI], that does not conflict with those from a recent topology discovery test. This value MUST NOT be zero.

**Number_of_Stations (2 bytes):** This field specifies an unsigned integer. This field indicates the number of station addresses that are present in the following station list.

**Station_List (variable):** This field MUST be a sequence of 6-octet MAC addresses where the number of addresses in the sequence is given by the **Number_of_Stations** field.

### 2.2.4.3 Hello Upper-Level Header Format

Hello frames MUST be sent to the Ethernet all-ones broadcast address so all switches can learn the source port of all responders. The Hello header following a Base header MUST be the following.




**Generation_Number (2 bytes):** This field specifies an unsigned integer that indicates the responder's current generation number.

**Current_Mapper_Address (6 bytes):** The active mapper's real MAC address as given in the **Real Source Address** field in the Base header of the Discover frame that initiated the active topology mapping request. This field MUST be set to zero if there is no active topology mapping session.

**Apparent_Mapper_Address (6 bytes):** This field specifies the mapper's MAC address as given in the **Source Address** field in the Ethernet header of the Discover frame that initiated the active topology mapping request. This field MUST be set to zero if there is no active topology mapping session.

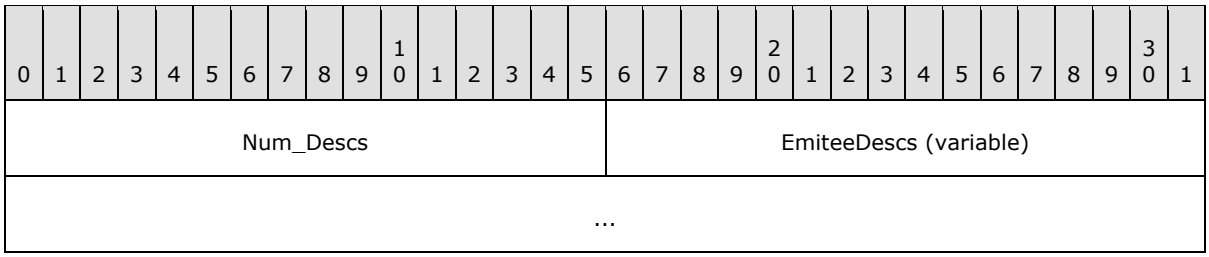
The **Real Destination Address** field in the Base header of the Hello frame MUST be set to the mapper's actual MAC address, so if more than one mapper is active, mappers can ignore replies from responders other than theirs. All but one mapper is eventually reset and thus wants to abort their associated responders, so it is vital that each responder is associated with only one mapper.

**TLV_List (variable):** This field specifies properties (as specified in section 2.2.1.1) that the responder knows about the network interface on which it is running. A TLV MUST NOT occur in the list more than once.

### 2.2.4.4 Emit Upper-Level Header Format

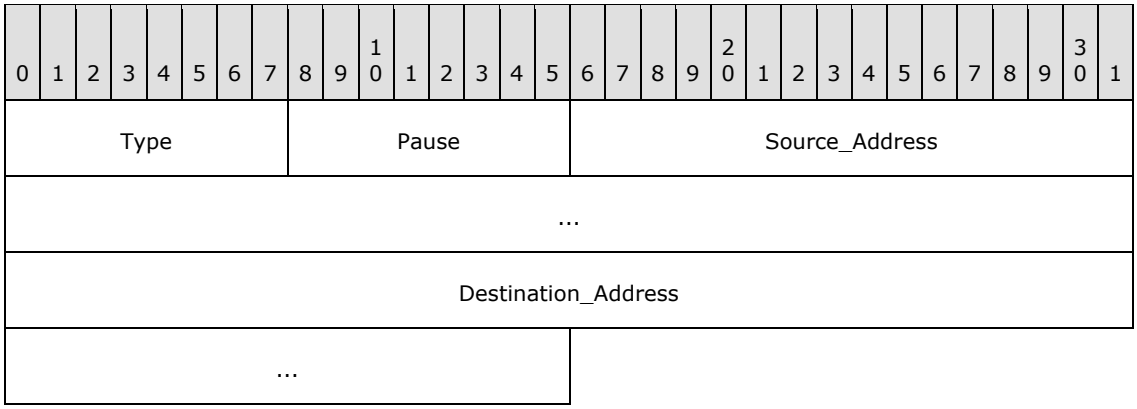
A mapper sends an Emit frame to a responder to request that the responder transmit a set of Train or Probe frames, each with a specified source and destination MAC addresses after a specified pause time, and that the responder immediately acknowledge the Emit frame with an Ack frame (this pause is used because some switches require approximately 150 milliseconds to update their port filtering databases, so back-to-back Train and Probe frames are not forwarded correctly).

The Emit frame following a Base header MUST have the following format.




**Num_Descs (2 bytes):** This field specifies the unsigned integer count, in network byte order, of the number of EmiteeDesc items in the **EmiteeDescs** field. This field's value MUST be in the range 1 to 105.

**EmiteeDescs (variable):** This field specifies a list of EmiteeDesc items, where each EmiteeDesc item is a 14-octet structure.




**Type (1 byte):** This field specifies the type of packet to emit. The following table shows valid values.

| Value | Meaning |
|---|---|
| 0x00 | Train |
| 0x01 | Probe |

**Pause (1 byte):** This field specifies the number of milliseconds to pause before the associated packet is emitted. The sum of the Pause values in all EmiteeDesc entries in an Emit frame MUST NOT exceed 1 second.

**Source_Address (6 bytes):** This field specifies the source MAC address of the packet to emit. The source MAC address MUST be either the responder's real MAC address to which the frame is sent or a MAC address from the special LLTD–specific MAC address range 0x000D3AD7F140 to 0x000D3AFFFFFF.

**Destination_Address (6 bytes):** This field specifies the destination MAC address of the packet to emit. The destination address MUST NOT be a multicast address because these addresses could amplify traffic.

### 2.2.4.5 Train Upper-Level Header Format

A mapper sends an Emit request to a responder, sometimes commanding it to send the Train frame. This Train frame is intended to allow a switch that is connected to the responder to learn the origin of a MAC address. The Train frame is ignored by all responders on reception.

The Train frame has no upper-level header other than the Base header itself.

### 2.2.4.6 Probe Upper-Level Header Format

A mapper sends an Emit request to a responder, sometimes commanding it to send a Probe frame to another responder. This Probe frame is meant to be seen and recorded by that responder.

The Probe frame has no upper-level header other than the Base header itself.

### 2.2.4.7 Ack Upper-Level Header Format

A responder sends an Ack frame to a mapper in response to an Emit request that contains a non-zero sequence number.

Ack frames are not acknowledged, but the **Sequence Number** field in the Base header MUST be nonzero, that is, the sequence number of the request that is being acknowledged.

The Ack frame has no upper-level header other than the Base header itself.

### 2.2.4.8 Query Upper-Level Header Format

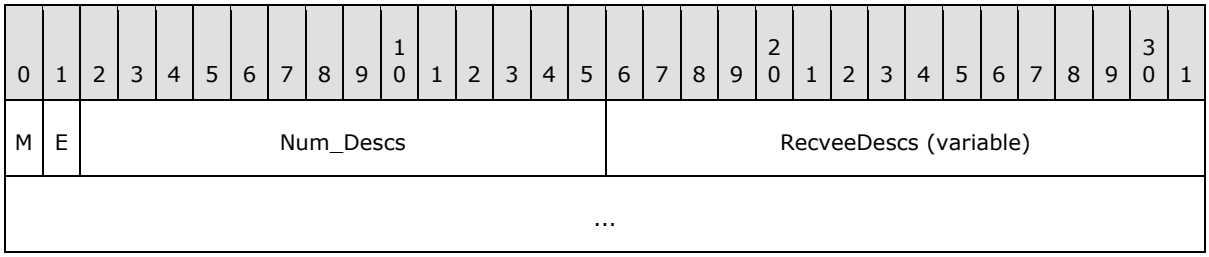
A mapper sends a Query frame to a responder to retrieve Probe events that the responder has observed on the wire.

The Query frame has no upper-level header other than the Base header itself.

### 2.2.4.9 QueryResp Upper-Level Header Format

A responder sends a QueryResp frame to a mapper in response to a Query request. It lists which recordable events (such as Ethernet source and Ethernet destination addresses from Probe frames that the responder has observed on the wire during a session) are available since the previous Query frame. QueryResp frames are not acknowledged but MUST set the Base header's **Sequence Number** field to match the Query frame to which they are generated in response.

The QueryResp frame that follows a Base header MUST have the following format.



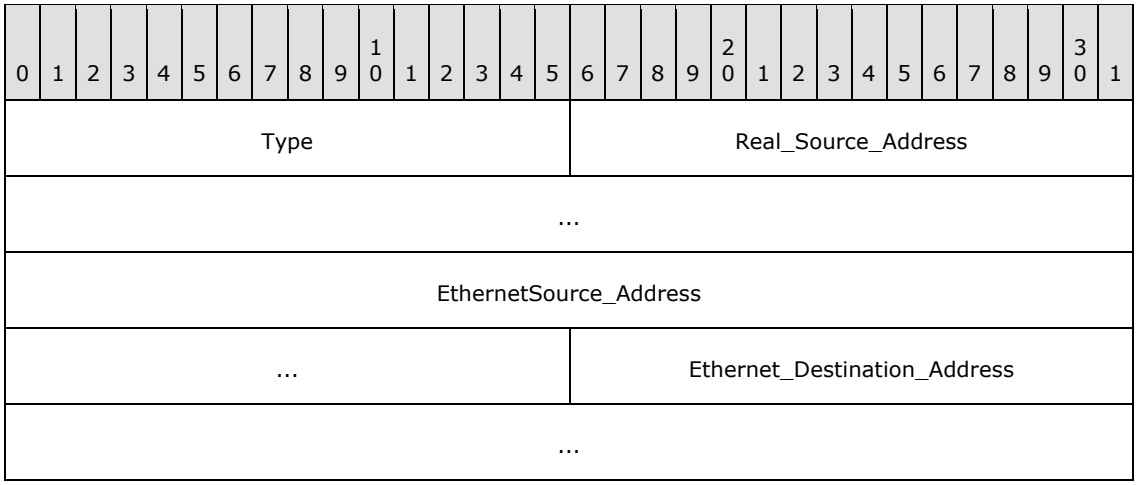

**M (1 bit):** This field MUST be set if there are more **RecveeDescs** than will fit in this frame.

**E (1 bit):** This field MUST be set if the responder is unable to store a RecveeDesc record due to lack of memory.

**Num_Descs (14 bits):** This field specifies the count of returned **RecveeDesc** structures that are included in the frame.

**RecveeDescs (variable):** This field specifies a list of **RecveeDesc** items, where each Recvee item is formatted as specified in the following table. Responders that are sending this frame MUST NOT merge identical recordable events (RecveeDescs items) even if they occur multiple times. The ordering of RecveeDesc items in this frame MUST represent arrival-time ordering.

Each RecveeDesc item MUST have the following 20-octet structure.




**Type (2 bytes):** This field specifies the recorded protocol type. The following table shows the valid values.

| Value | Meaning |
|---|---|
| 0x00 | Probe |
| 0x01 | Address Resolution Protocol (ARP), as specified in [RFC826], or Internet Message Control Protocol for the Internet Protocol Version 6 (ICMPv6) Neighbor Discovery, as specified in [RFC2461]. |

**Real_Source_Address (6 bytes):** This field specifies the real source MAC address.

For ARP, this field corresponds to the **ar$sha** field in an ARP response packet, as specified in [RFC826].

For ICMPv6, this corresponds to the optional target link-layer address option in a neighbor discovery packet, as specified in [RFC2461] section 4.

**EthernetSource_Address (6 bytes):** This field specifies the source MAC address in the Ethernet frame.

**Ethernet_Destination_Address (6 bytes):** This field specifies the destination MAC address in the Ethernet frame.

### 2.2.4.10 Reset Upper-Level Header Format

A mapper broadcasts a Reset frame to all responders to abort a mapping generation either because someone else is mapping or because mapping is over.

The Reset frame has no upper-level header other than the Base header itself.

### 2.2.4.11 Charge Upper-Level Header Format

A mapper sends a Charge frame to a responder to match the number of frames and amount of bytes that is to be requested in an upcoming Emit frame. This action is intended to prevent bandwidth amplification attacks.

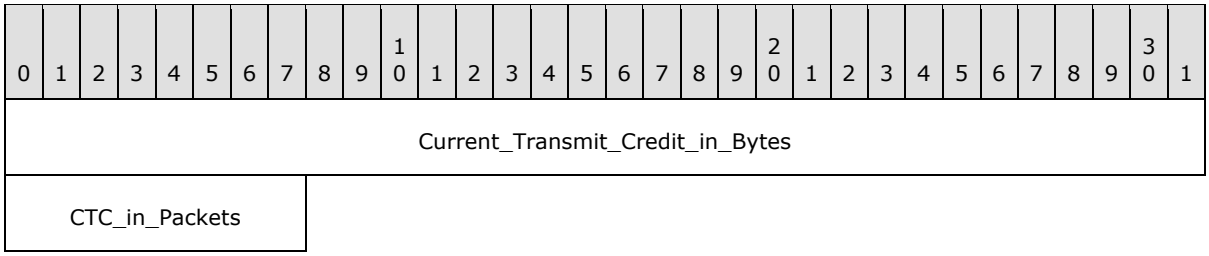
The Charge frame has no upper-level header other than the Base header itself.

### 2.2.4.12 Flat Upper-Level Header Format

A responder sends a Flat frame to a mapper in response to the following:

- An Emit frame that has a non-zero sequence number and requires more charges than the responder has. The Flat frame tells the mapper to retry the Emit request, preceded by a fixed count of Charge frames to build up the needed charge.
- A Charge frame that has a non-zero sequence number and effectively forces the responder to report its current charge count.

The Flat frame following a Base header MUST have the following format.



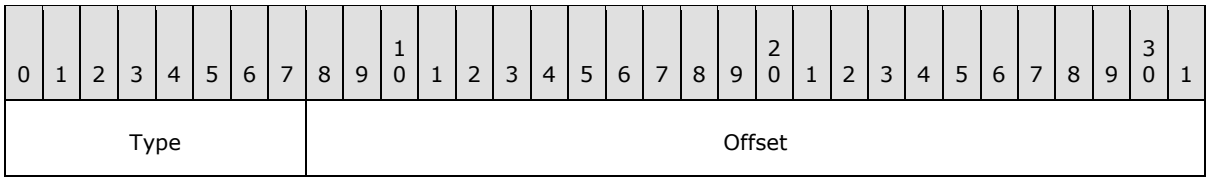

**Current_Transmit_Credit_in_Bytes (4 bytes):** (CTC) This field specifies the value of the CTC byte counter at the responder, in network byte order.

**CTC_in_Packets (1 byte):** This field specifies the value of the CTC packet counter at the responder, in network byte order.

### 2.2.4.13 QueryLargeTlv Upper-Level Header Format

The QueryLargeTlv frame allows the mapper to query a responder for TLV data that is too large to be included in a Hello frame. The inclusion of a zero-length TLV in the Hello frame indicates that such data is available and that the responder is willing to provide the data in a QueryLargeTlvResp response. Each QueryLargeTlv request results in a maximum of one QueryLargeTlvResp response. Repeated QueryLargeTlv requests have to be made for sufficiently large TLVs that do not fit in a single QueryLargelvVResp response frame.

The QueryLargeTlv frame that follows a Base header MUST have the following format.




**Type (1 byte):** This field specifies the type of TLV that is requested. It MUST be one of the following values.

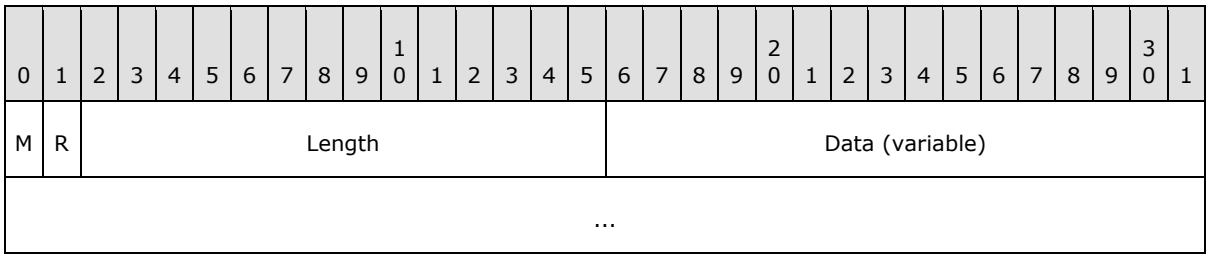
| Value | Meaning |
|---|---|
| 0x0E | Icon image (section 2.2.2.1) |
| 0x11 | Friendly Name (section 2.2.2.2) |
| 0x13 | Hardware ID (section 2.2.2.3) |
| 0x16 | AP Association Table (section 2.2.2.4) |
| 0x18 | Detailed Icon Image (section 2.2.2.5) |
| 0x1A | Component Table (section 2.2.2.6) |
| 0x1C | Repeater AP Table (section 2.2.2.7) |

**Offset (3 bytes):** This field specifies the offset in octets, in network byte order, within the TLV data to query.

### 2.2.4.14 QueryLargeTlvResp Upper-Level Header Format

A responder sends the QueryLargeTlvResp frame to a mapper in response to a QueryLargeTlv request. It returns up to the maximum number of octets that fit into a response frame over the Ethernet media, starting from a requested offset.

The QueryLargeTlvResp header MUST immediately follow the Base header and have the following format.




**M (1 bit):** This field MUST be set if there is more data than will fit in this frame.

**R (1 bit):** This field MUST be 0 on transmission and ignored on receipt.

**Length (14 bits):** This field specifies the octet count, in network byte order, of data that is returned in the QueryLargeTlvResp frame. This field MUST be set to 0x00 if the QueryLargeTlv request is for an unsupported TLV type.

| Value | Meaning |
|---|---|
| QueryLargeTlv<br>0x00 | An unsupported TLV type |

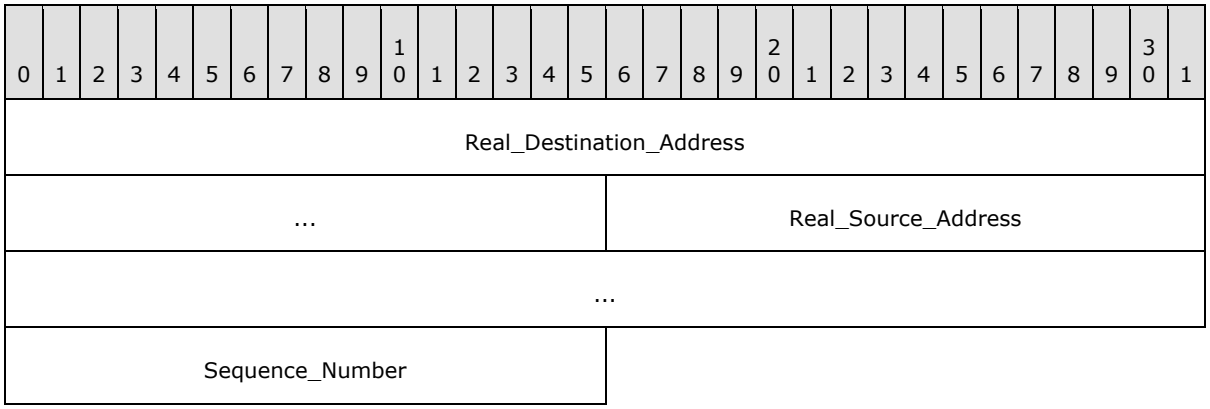
**Data (variable):** This field specifies the information that was requested in the QueryLargeTlv frame. The format of the data objects are specified in section 2.2.2. This field MUST contain a portion of the requested data object, starting at the offset requested in the QueryLargeTlv frame, and contain as many bytes of the data object as will fit in the frame.

### 2.2.5 QoS Diagnostics Specification for Network Test

#### 2.2.5.1 Base Header Format

This Base header MUST be used when the Type of Service value in the Demultiplex header is set to 0x02 (QoS diagnostics) and the Function value is in the range 0x00 (QosInitializeSink) to 0x07 (**QosAck**).

The Base header format MUST be the following.




**Real_Destination_Address (6 bytes):** This field specifies the intended destination's real MAC address.

**Real_Source_Address (6 bytes):** This field specifies the sender's real MAC address.

A sender MUST set the real source and destination MAC addresses to its own MAC address and its intended destination MAC address, respectively. This field is required because some network devices rewrite the **Source Address** and **Destination Address** fields of the Ethernet header and thus may not survive an end-to-end transmission.

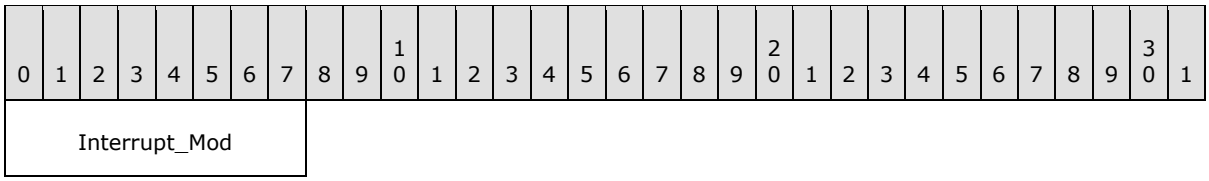
**Sequence_Number (2 bytes):** This field specifies the sequence number that correlates a response to a specific request. The sequence number ensures reliability of certain packets in the protocol.

For function codes 0x07 and 0x08, this field MUST be non-zero.

#### 2.2.5.2 QosInitializeSink Upper-Level Header Format

A controller sends the QosInitializeSink frame to a sink to set up a network test session.

The QosInitializeSink header that follows the Base header MUST have the following format.



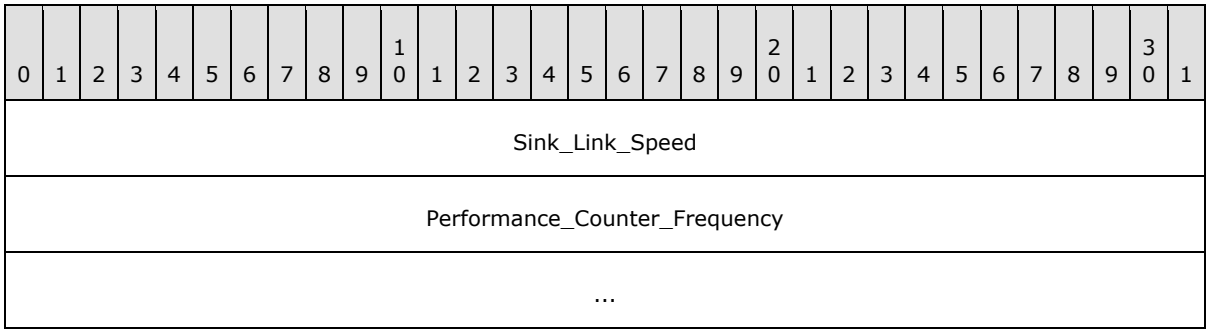

**Interrupt_Mod (1 byte):** This field specifies the interrupt moderation requirement of a network test session. The following table shows the possible values.

| Value | Meaning |
|---|---|
| 0x00 | Disable interrupt moderation. |
| 0x01 | Enable interrupt moderation. |
| 0xFF | Use the existing interrupt moderation setting. |

### 2.2.5.3 QosReady Upper-Level Header Format

A sink sends a QosReady frame to a controller, in reply to a QosInitializeSink frame, to notify the controller that a network test session is successfully established.

The QosReady header that follows a Base header MUST have the following format.



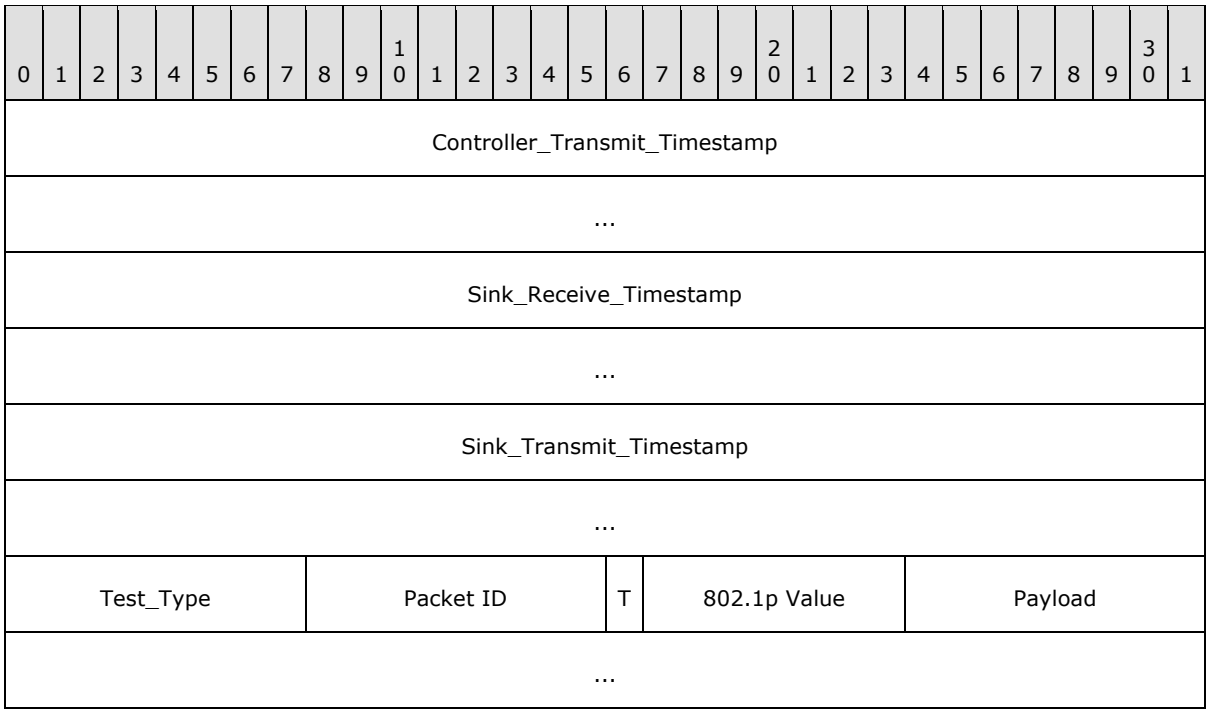

**Sink_Link_Speed (4 bytes):** This field specifies the responder's link speed in 100-bit-per-second units in network byte order.

**Performance_Counter_Frequency (8 bytes):** This field allows a responder to identify how fast its time stamp counters run in ticks per second in network byte order.

### 2.2.5.4 QosProbe Upper-Level Header Format

A controller sends a QosProbe frame to a sink, and by a sink back to a controller. It carries time stamp values that an application can use on the controller to calculate network bandwidth.

The QosProbe header that follows the Base header MUST have the following format.




**Controller_Transmit_Timestamp (8 bytes):** This field specifies the time stamp, in network byte order, of the controller on transmission, in units per Performance Counter Frequency.

**Sink_Receive_Timestamp (8 bytes):** This field specifies the time stamp, in network byte order, of the sink on receipt in units per Performance Counter Frequency. This field MUST be set to zero in a timed probe test. In a probegap test, this field MUST be set to zero on transmission from the controller.

**Sink_Transmit_Timestamp (8 bytes):** This field specifies the time stamp, in network byte order, of the sink on transmission in units per Performance Counter Frequency. This field MUST be set to zero in a timed probe test. In a probegap test, this field MUST be set to zero on transmission from the controller.

**Test_Type (1 byte):** This field specifies the test type in which this packet is involved. The following table shows the possible values.

| Value | Meaning |
|---|---|
| 0x00 | Timed probe. |
| 0x01 | Probegap originating from the controller. |
| 0x02 | Probegap originating from the sink. |

**Packet ID (1 byte):** The controller MUST assign an ID to the packet so it can be uniquely identified when it is returned in either a **QoSProbe** or **QosQueryResp**.

**T (1 bit):** If set, this field specifies the presence of an 802.1p value in the 802.1q tag for each packet.

**802.1p Value (7 bits):** If the T flag is set, this field contains the 802.1p value to be included in the 802.1q tag for each QosProbe packet that is reflected to the controller in the case of a probegap test. If the T flag is not set, this field MUST be set to zero and ignored on receipt.

**Payload (5 bytes):** This field specifies arbitrary data that is used to pad the frame to the correct frame size. In a probegap experiment, the payload content that a sink receives MUST be duplicated on the sink's send path.

### 2.2.5.5 QosQuery Upper-Level Header Format

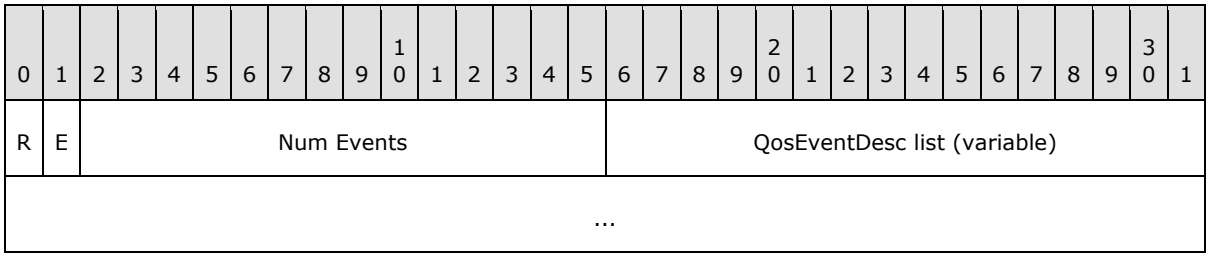
A controller sends a QosQuery frame to a sink following the last QosProbe frame in a timed probe test.

The QosQuery frame has no upper-level header other than the Base header itself. The sequence number MUST be non-zero.

### 2.2.5.6 QosQueryResp Upper-Level Header Format

A sink sends the QosQueryResp frame to the controller, in response to a QosQuery frame. It lists QosProbe events (also known as **QosEventDesc** structures) that have been observed since the previous QosQuery frame. QosQueryResp frames MUST NOT be acknowledged. The Base header's **Identifier** field of the QosQueryResp MUST match the QosQuery frame that is generated in response to the QosQueryResp frame. The ordering of QosEventDesc items in this frame MUST represent the ordering of the arrival time.

The QosQueryResp header that follows the Base header MUST have the following format.

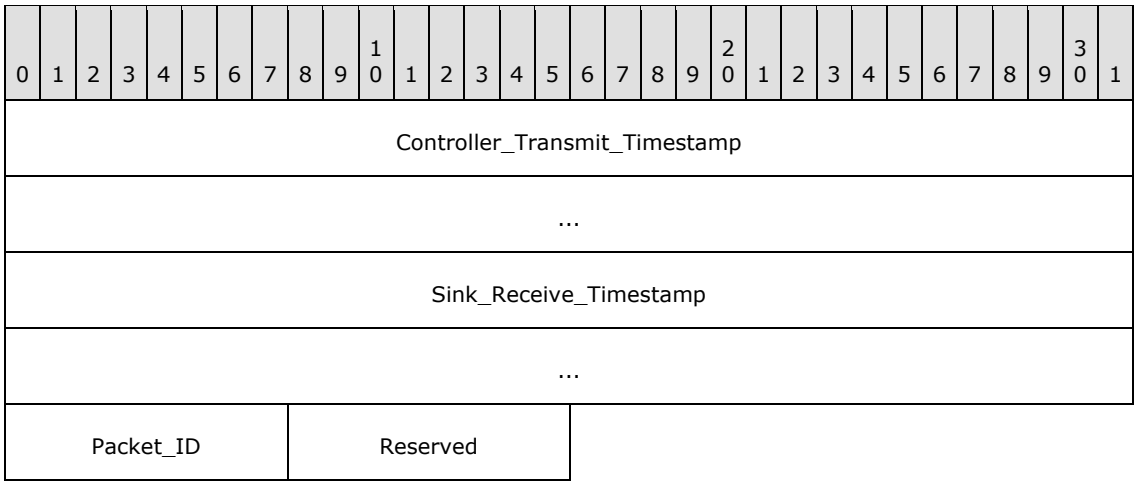



**R (1 bit):** This field MUST be set to 0x00 and MUST be ignored upon receipt.

**E (1 bit):** This field MUST be set if the responder is unable to allocate enough memory for one or more **QosEventDesc** structures. The **Num Events** field MUST be zero, and **QosEventDesc** structures MUST NOT follow the field.

**Num Events (14 bits):** This field specifies the count, in network byte order, of QosEventDesc items that follow.

**QosEventDesc list (variable):** This field specifies a set of **QosEventDesc** items, where each **QosEventDesc** item is an 18-octet structure.




**Controller_Transmit_Timestamp (8 bytes):** This field specifies the time stamp, in network byte order, of the controller on event transmission in units per Performance Counter Frequency.

**Sink_Receive_Timestamp (8 bytes):** This field specifies the time stamp, in network byte order, of the sink on event reception in units per Performance Counter Frequency.

**Packet_ID (1 byte):** This field specifies the value of the **Packet ID** field from the QosProbe frame that generated the event.

**Reserved (1 byte):** This field is not currently used, but it exists only to pad the structure to an even size. This field MUST be set to 0 on transmit and ignored on receipt.

### 2.2.5.7 QosReset Upper-Level Header Format

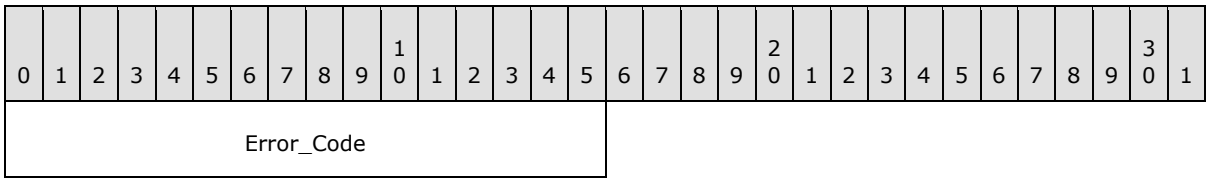
A controller sends a QosReset frame to a sink to terminate a network test session.

The QosReset frame has no upper-level header other than the Base header itself.

### 2.2.5.8 QosError Upper-Level Header Format

A sink sends the QosError frame to notify a controller that a network test session cannot be initiated.

The QosError header that follows the Base header MUST have the following format.




**Error_Code (2 bytes):** This field specifies an error code that identifies the reason that a request failed, resulting in this response. The following table shows valid error code values.

| Value | Meaning |
|---|---|
| 0x00 | Insufficient resources. The responder ran out of resources while attempting to set up the session. |
| 0x01 | Busy; try again later. The responder has reached its session limit. |
| 0x02 | Interrupt moderation not available. The interrupt moderation requirement cannot be satisfied, or the ability to control it is not available. |

#### 2.2.5.9 QosAck Upper-Level Header Format

A sink sends the QosAck frame to a controller to notify it that a QosReset request has been processed.

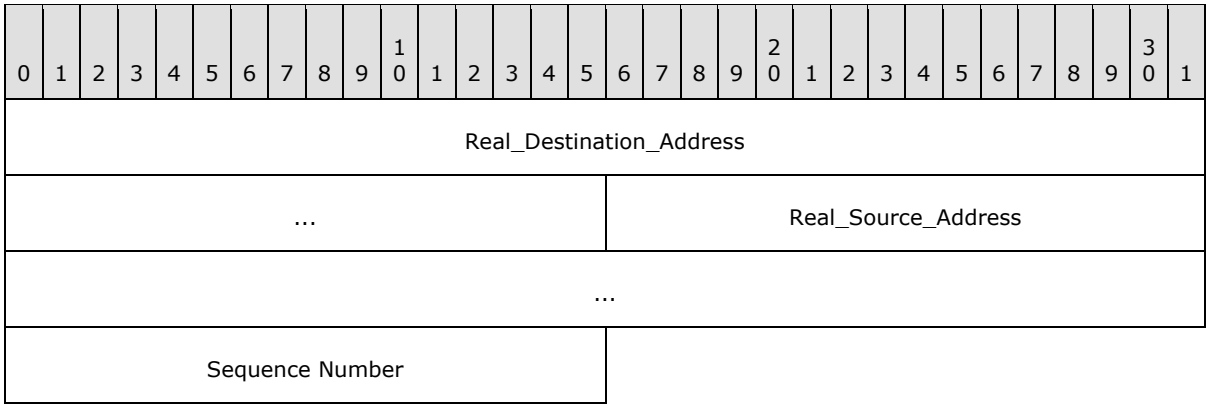
The QosAck frame has no upper-level header other than the Base header itself.

### 2.2.6 QoS Diagnostics Specification for Cross-Traffic Analysis

#### 2.2.6.1 Base Header Format

This Base header MUST be used when the Type of Service value in the Demultiplex header is set to 0x02 (QoS diagnostics) and the Function value is in the range 0x08 (QosCounterSnapshot) to 0x0A (QosCounterLease).

The Base header format MUST be the following.




**Real_Destination_Address (6 bytes):** This field specifies the intended destination's real MAC address. This field allows querying of per-network interface counters in wireless access points. For these devices, this address field MUST identify the BSSID.

**Real_Source_Address (6 bytes):** This field specifies the sender's real MAC address. This field is necessary because the **Source Address** field of the Ethernet header is translated by some network devices and thus may not survive an end-to-end transmission.

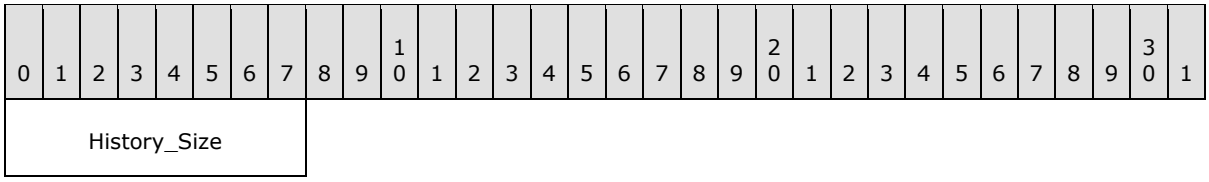
**Sequence Number (2 bytes):** This field specifies the sequence number that correlates a response to a specific request.

For function codes 0x07 and 0x08, this field MUST be non-zero.

### 2.2.6.2 QosCounterSnapshot Upper-Level Header Format

A cross-traffic analysis initiator sends a QosCounterSnapshot frame to a responder to retrieve its history of network performance counters.

The QosCounterSnapshot header MUST immediately follow the Base header, and it MUST have the following format.



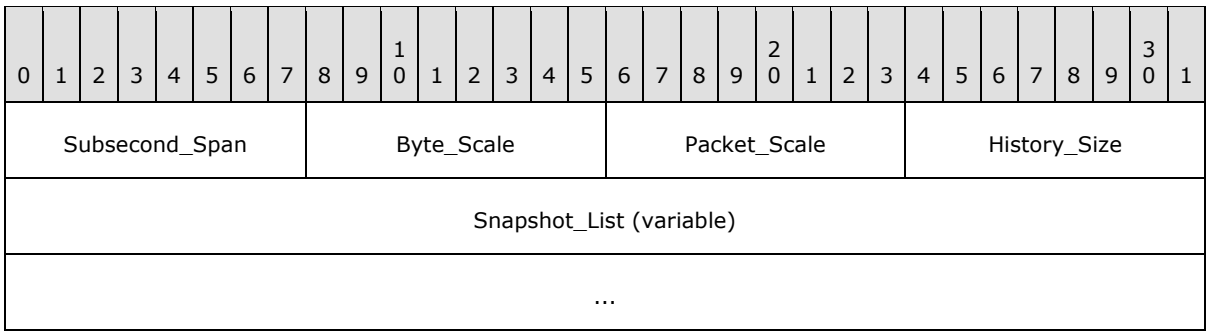

**History_Size (1 byte):** This field specifies the maximum number of most recent full 4-tuples to return from the history.

### 2.2.6.3 QosCounterResult Upper-Level Header Format

A responder sends a QosCounterResult frame to a cross-traffic analysis initiator in response to a QosCounterSnapshot frame.

Each QosCounterResult frame reports as many full 4-tuples as are requested in the preceding QosCounterSnapshot request. When the QosCounterSnapshot request is received, a snapshot of the 4-tuples is also taken, and the time span since the last sampling interval is recorded. This subsecond sample is also returned in the QosCounterResult frame.

The QosCounterResult header immediately follows the Base header, and it MUST have the following format.




**Subsecond_Span (1 byte):** This field specifies the time span (expressed as 1/256ths of a second) since the last sampling interval, taken at the time that the QosCounterSnapshot request is received. If the subsecond sample is still present in the snapshot list, this field MUST be set to zero.

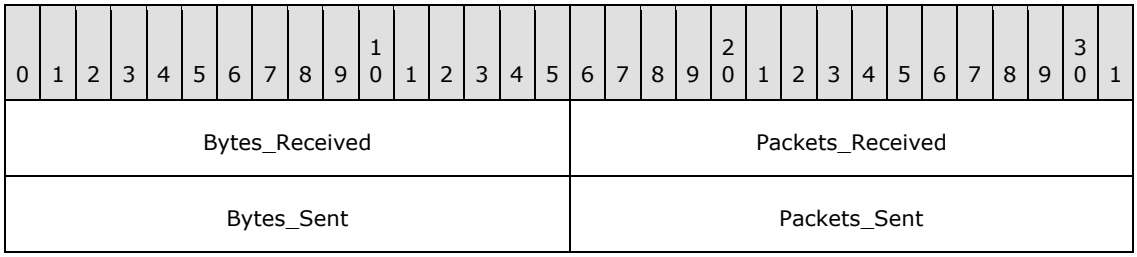
**Byte_Scale (1 byte):** This field's value MUST be in the range 0 to 255, where a value of n indicates that all byte counters are expressed in units of (n+1) kilobytes.

**Packet_Scale (1 byte):** This field's value MUST be in the range 0 to 255, where a value of n indicates that all packet counters are expressed in units of (n+1) packets.

**History_Size (1 byte):** This field specifies the number of full 4-tuples that the responder can return. This number MUST NOT include the subsecond sample that is taken when the QosCounterSnapshot request is received.

**Snapshot_List (variable):** This field MUST include the 4-tuple snapshots that were counted by the **History Size** field, plus the subsecond snapshot. Entries in the snapshot list MUST be arranged starting with the oldest 4-tuple snapshot and ending with the subsecond 4-tuple snapshot.

Each snapshot has the following format:




**Note** A 1500-byte Ethernet frame is large enough to fit 184 entries, which is over 3 minutes of historical data.

### 2.2.6.4 QosCounterLease Upper-Level Header Format

A cross-traffic analysis initiator broadcasts a QosCounterLease frame to all responders to request that they start collecting the network performance counters that are returned in the QosCounterResult frame.

The QosCounterLease frame has no upper-level header other than the Base header itself.

# 3 Protocol Details

As described in section 1.3, this protocol defines the following roles:

- Enumerator: This role MAY<4> be supported by LLTD implementations.
- Mapper: This role MAY be supported by LLTD implementations. If supported, the implementation MUST also support the Enumerator role.
- QoS Controller: This role MAY be supported by LLTD implementations. If supported, the implementation MUST also support the Enumerator role.
- Cross-Traffic Analysis Initiator: This role MAY be supported by LLTD implementations. If supported, the implementation MUST also support the Enumerator role.
- Responder (Quick Discovery): This role MUST be supported by LLTD implementations.
- Responder (Topology Discovery): This role MUST be supported by LLTD implementations.
- QoS Sink: This role MUST be supported by LLTD implementations.
- Responder (QoS Cross-Traffic): This role MUST be supported by LLTD implementations.

Each role is described in the following sections.

## 3.1 Enumerator Details

This section details the role of an enumerator that is used in LLTD quick discovery. An enumerator seeks to discover all LLTD-capable stations (responders) on the network. The enumerator starts by broadcasting a Discover frame. This frame contains a set of responder MAC addresses that the enumerator has seen (initially the empty set) and an XID value that helps all responders detect an enumerator that has reset itself without notifying other responders via the Reset frame. A station MUST NOT have more than one instance of an enumerator active at any time.

An important aspect of quick discovery is avoiding the network overload that is caused by either a very large network or one of the more malicious mappers. The RepeatBAND algorithm (as specified in section 3.5.6.2) is used for this purpose, and it forces responders to throttle their own transmissions based on seeing other responders' frames.

Message request/response pairs that are sent during quick discovery are defined as follows.

| Sent by enumerator | Sent by responder |
|---|---|
| Discover | Hello |
| Reset | N/A |

### 3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those described in this document.

The data elements required in any enumerator implementation are:

- Current Generation Number: This data element specifies the most recently accepted generation number that a responder volunteered in the Hello frame. This data element is an unsigned 16-bit value, and it MUST be incremented by using ones-complement arithmetic; that is, it MUST advance from 0xFFFF to 0x0001 and skip 0x0000.
- Last-Seen Station List: This list holds all unique responder MAC addresses seen via Hello frames since the enumerator sent the last Discover frame.
- Seen Station List: This list holds an entry for each unique responder that was seen since the start of the quick discovery process. It is keyed by the responder's MAC address and also contains a list of TLVs for the responder.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

### 3.1.2 Timers

The Enumerator role has one timer:

- Block timer: This recurring timer is used to periodically broadcast Discover frames. The timer SHOULD be set to fire at 300-millisecond intervals.

### 3.1.3 Initialization

During initialization, the following conditions must be met:

- The Block timer MUST be stopped.
- The Last-Seen Station List MUST be empty.
- The Current Generation Number MUST be set to zero.

### 3.1.4 Higher-Layer Triggered Events

#### 3.1.4.1 Quick Discovery Startup

When a higher-layer protocol or application triggers startup of the quick discovery process, the Block timer MUST be started and its logic (as specified in section 3.1.6.1) MUST be triggered immediately. The Last-Seen Station List MUST be initialized to empty.

#### 3.1.4.2 Quick Discovery Shutdown

When the higher-layer protocol or application that initially triggered the startup triggers a shutdown of the quick discovery process, the enumerator MUST broadcast a Reset frame, as specified in section 2.2.4.10.

### 3.1.5 Message Processing Events and Sequencing Rules

When an enumerator receives an LLTD frame, it MUST check the LLTD header to see if it is a valid Hello frame. If not, the message MUST be ignored.

#### 3.1.5.1 Receiving a Hello Frame

The source Ethernet MAC address of the Hello frame (that is, the responder's MAC address) MUST first be recorded in the Last-Seen Station List, if it is not already listed.

Also, a similar check MUST be made on the Seen Station List. If there is no existing entry in this list, the Hello frame MUST then be parsed for its TLV list (that is, the **TLV_List** field). If any entry in this TLV list is malformed, the frame MUST be ignored and the corresponding entry removed from the Last-Seen Station List. If the TLV list is valid, the enumerator MUST attempt to add a new entry containing all of these newly-discovered details into the Seen Station List. If the enumerator cannot allocate enough memory for this new entry, it MUST immediately shut down the protocol and broadcast a Reset frame.

#### 3.1.5.1.1 Enumerator Also Functioning in the Mapper Role

If the enumerator is also functioning as a mapper, it MUST also do the following:

First, upon receipt of the Hello message, it MUST immediately check if the **Current Mapper Address** field in the Hello header is equal to the MAC address of the network interface that it received the message about. In case of inequality, the mapper MUST immediately shutdown the protocol and broadcast a Reset frame.

Next, after all of the normal enumerator tasks are performed, it MUST decide which generation number (**Generation Number** field in Hello frame) to use for mapping, as follows. If the Current Generation Number is zero, the generation number from the Hello frame MUST be incremented by one and stored as the current generation number. Otherwise, the current generation number MUST be subtracted from the generation number in the Hello frame. If the resulting value is less than or equal to 0x7FFF, the generation number from the Hello frame MUST be incremented by one and stored as current generation number. If the resulting value is greater than 0x7FFF, the generation number that the responder volunteers MUST be ignored.

If no responder volunteered a non-zero generation number, the mapper MUST select a new, non-zero generation number at random and broadcast a final Discover frame to disseminate the generation number to all responders.

This process permits a mapper to select a generation number before knowing that all possible responders have sent a Hello frame. The mapper MUST follow this process because it never knows when it will receive a late Hello frame.

For more information about generation numbers, see section 3.2.1.

### 3.1.6 Timer Events

#### 3.1.6.1 Block Timer Expiry

When the Block timer fires, the enumerator MUST construct a Discover frame by filling the **Station List** field with entries from the Last-Seen Station List. If there are more entries in the list than will fit in the Discover frame, additional Discover frames MUST be created to hold these additional entries. All Discover frames are then broadcast over the network. Finally, the Last-Seen Station List MUST be cleared.

If the enumerator is not satisfied that it has given enough time for all responders to respond, the timer MUST be restarted. How the enumerator determines whether or not enough time has passed can be done in any implementation-specific<5> way. For example, the RepeatBAND algorithm (as specified in section 3.5.6.2) predicts that if the Seen Station List does not grow for three consecutive Block timer expirations, it can be assumed that all responders have reported. Stopping this timer implies stopping the quick discovery process.

#### 3.1.6.1.1 Enumerator Also Functioning in the Mapper Role

If the enumerator is also functioning as a mapper, it MUST populate the **Generation Number** field in the Discover header with the current generation number. Otherwise, the field MUST be set to zero.

### 3.1.7 Other Local Events

None.

## 3.2 Mapper Details

This section details the role of a mapper station that is used in LLTD topology discovery tests. A station MUST NOT have more than one instance of a mapper operational at any time. In addition to performing the role of an enumerator, a mapper also seeks to achieve the following:

- Associate with all responders that are discovered via the Enumerator role.
- Negotiate a generation number with the responders.
- Determine if another mapper is active.
- Infer the network topology by sending zero or more Emit requests to one or more responders.

Message request/response pairs applicable to topology discovery tests are defined as follows.

| Sent by mapper | Sent by responder |
|---|---|
| Emit | Ack / Flat (*) |
| Query | QueryResp |
| QueryLargeTlv | QueryLargeTlvResp |
| Charge | Flat (*) |
| Reset | N/A |

*If the request frame does not contain a non-zero sequence number, the responder does not send a response.

### 3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those specified in this document.

The data elements required in any mapper implementation are:

- Generation Number: The mapper uses generation numbers to generate fresh MAC addresses that are unknown to switches in the network. This avoids the requirement of restarting switches between mapping runs, so it is critical to choose an as-yet-unused generation number. Note that mappers do not store previous generation numbers because multiple mappers may be operating

on a network, and mappers do not participate in any process to keep their generation numbers synchronized.

- Network Topology Test Session List: This data element specifies a list of network topology test sessions in progress. Each entry is identified by a responder's MAC address and also contains the following additional fields:
  - **Sequence Number**: This field specifies a 16-bit unsigned value that is assigned to each responder for use in commands and requests. Sequence numbers MUST be advanced by using increments in ones-complement arithmetic; that is, they MUST advance from 0xFFFF to 0x0001 and skip 0x0000. The first sequence number that the mapper uses for each responder MUST be a non-zero value.
  - **Charge**: To prevent denial-of-service (DoS) attacks, the mapper needs to send as many bytes and packets (via the Charge frame) to a responder because it can trigger the responder to send on its behalf via the Emit request. The mapper maintains byte (unsigned 16-bit value) and packet (unsigned 8-bit value) charge counters for each responder.
  - **Pended Request**: This field specifies a request per responder for which a corresponding request is expected. A pended request is uniquely identified by its function code (the **Function** field in the Demultiplex header) and sequence number.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

### 3.2.2 Timers

The Mapper role has one timer:

(Per-Responder) Response timer: This one-shot timer, per entry in the Seen Station List, is used to ensure timely response (or non-response) to an Emit, Query, or QueryLargeTlv request that contains a non-zero sequence number. This process works because only one such request can be pended per responder.

### 3.2.3 Initialization

During initialization, the following conditions must be met:

- All timers must be disabled.

### 3.2.4 Higher-Layer Triggered Events

#### 3.2.4.1 Startup Trigger

When a higher-layer application or protocol triggers startup of topology discovery tests, the mapper MUST assume the role of an enumerator and begin quick discovery, as specified in section 3.1.4.1.

#### 3.2.4.2 Retrieve a Large Data Property

When an application or higher-layer protocol requests a large data property for a given Type and responder MAC address, the mapper MUST send a QueryLargeTlv frame to that responder, store the frame as the Pended request in the topology discovery test session, and set the Per-Responder Response timer to expire in 350 milliseconds.

#### 3.2.4.3 Perform a Network Topology Test

When a higher-layer application or protocol requests LLTD to perform a network topology test with a set of Train and Probe commands (specified by the application), the mapper MUST send one Charge frame per command to the responder that the application specifies, followed by an Emit frame to this responder that contains the commands that the application specifies. The Train and Probe commands that the application specifies MUST indicate the source and destination MAC addresses for the responder to use.

#### 3.2.4.4 Perform a Test Result Query

When a higher-layer application or protocol directs LLTD to request a list of Probe frames seen by a given responder, the mapper MUST send a Query frame to that responder, store the frame as the Pended request in the topology discovery test session, and set the Per-Responder Response timer to expire in 350 milliseconds.

#### 3.2.4.5 Shutdown Trigger

When the higher-layer application or protocol that initially triggered the startup shuts down the topology discovery tests, the mapper MUST broadcast a Reset frame (see section 2.2.4.10).

### 3.2.5 Message Processing Events and Sequencing Rules

When a message arrives, the mapper MUST first check whether it is a valid Ack, Flat, QueryResp, or QueryLargeTlvResp frame or not. If not, it MUST be dropped.

#### 3.2.5.1 Receiving an Ack Frame

Upon receipt of an Ack frame, the mapper MUST first validate the Ack frame by verifying that all of the following statements are true:

- The mapper did indeed solicit the response via an Emit frame, as tracked by the pended request state.
- The **Real Source Address** field in the Base header of the Ack frame matches the MAC address of the destination responder in the Emit request.
- The **Sequence Number** field in the Base header of the Ack frame matches that used in the Emit request.

Upon successful validation, the relevant Per-Responder Response timer MUST be stopped, and the sequence number for the affected responder MUST be incremented for the next request with a non-zero sequence number.

If the Ack frame completes the last test that the application requests, the mapper MUST delete the old Pended request and indicate to the application that the network tests have been completed.

#### 3.2.5.2 Receiving a Flat Frame

Upon receiving a Flat Frame, the mapper MUST validate it by verifying that the following are true:

- The mapper did indeed solicit the response via an Emit or Charge frame, as tracked by the pended request state.
- The **Real Source Address** field in the Base header of the Flat frame matches the MAC address of the destination responder in the original request.

- The **Sequence Number** field in the Base header of the Flat frame matches that used in the original request.

Upon successful validation, the relevant Per-Responder Response timer MUST be stopped, and the sequence number for the affected responder MUST be incremented for the next request with a non-zero sequence number.

#### 3.2.5.3 Receiving a QueryResp Frame

A responder sends a QueryResp frame in response to a valid Query request with a non-zero sequence number. The mapper MUST validate the QueryResp frame by verifying that the following are true:

- The mapper did indeed solicit the response via a Query frame, as tracked by the pended request state.
- The **Real Source Address** field in the Base header of the QueryResp frame matches the MAC address of the destination responder in the Query request.
- The **Sequence Number** field in the Base header of the QueryResp frame matches that used in the Query request.

If the QueryResp frame is not valid, it MUST be ignored. Otherwise, it MUST be processed as follows.

The relevant Per-Responder Response timer MUST be stopped. The sequence number for the affected responder MUST be incremented for the next request with a non-zero sequence number.

If the More flag in the QueryResp header is set, the mapper SHOULD follow up with a subsequent Query request. This action MUST continue until either a QueryResp frame is returned without the More flag set or the responder returns more total records than the mapper is prepared to handle.

#### 3.2.5.4 Receiving a QueryLargeTlvResp Frame

Upon receiving a QueryLargeTlvResp, the mapper MUST first validate it by verifying that the following are true:

- The mapper did indeed solicit the response via a QueryLargeTlv frame as tracked by the pended request state.
- The **Real Source Address** field in the Base header of the QueryLargeTlvResp frame matches the MAC address of the destination responder in the QueryLargeTlv request.
- The **Sequence Number** field in the Base header of the QueryLargeTlvResp frame matches that used in the QueryLargeTlv request.

Upon successful validation, the relevant Per-Responder Response timer MUST be stopped. The sequence number for the affected responder MUST be incremented for the next request with a non-zero sequence number.

If the More flag in the QueryLargeTlvResp header is set, the mapper SHOULD follow up with a subsequent QueryLargeTlv request. This action MUST continue until a QueryLargeTlvResp frame is returned without the More flag set or if the responder returns more bytes than the mapper is required to accommodate for the given TLV type.

If a subsequent QueryLargeTlv request is sent, the mapper MUST store the frame as the Pended request in the topology discovery test session and set the Per-Request Response timer to expire in

350 milliseconds. Otherwise, the mapper MUST pass the retrieved data back to the application or higher-layer protocol.

### 3.2.6 Timer Events

#### 3.2.6.1 Per-Responder Response Timer Expiry

When a Per-Responder Response Timer fires, the mapper MUST retransmit the pended request frame (the sequence number MUST be unchanged), and the timer MUST be restarted in that case.

The mapper MAY<6> give up retrying communication with the responder if the timer has fired more than once. If the mapper opts to continue with the topology discovery tests, it SHOULD NOT communicate with this responder for the duration of the discovery process since the sequence numbering is likely tainted, and the responder will likely not respond.

### 3.2.7 Other Local Events

#### 3.2.7.1 Enumerator Finishes Enumerating Responders

After the Enumerator role is fulfilled (that is, when the Block timer is stopped, as specified in section 3.1.6.1), for each responder that is discovered, a non-zero sequence number MUST be selected (by means of any random number generator) and remembered. All subsequent requests with a non-zero sequence number that the mapper sends MUST adhere to the defined sequence numbering rule.

At this point, the mapper MUST indicate to the application or higher-layer protocol that it is ready to perform network topology tests.

## 3.3 QoS Controller Details

This section details the role of a controller station that is used in the LLTD QoS network test type-of-service.

Message request/response pairs applicable to a controller are defined as follows.

| Sent by controller | Sent by sink |
|---|---|
| QosInitializeSink | QosError / QosReady |
| QosProbe | QosProbe (*) |
| QosQuery | QosQueryResp |
| QosReset | QosAck |

*If the request frame does not contain a non-zero sequence number, the sink does not send a response.

### 3.3.1 Abstract Data Model

This section specifies a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The specified organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those specified in this document.

The data elements required in any controller implementation are:

- Network Test Session Table : A list of network test sessions MAY<7> be maintained if the controller chooses to support more than one simultaneous sink. Otherwise, the controller MUST instead support a single network test session.

  Each network test session is identified by the MAC address of the sink station and MUST have the following additional fields:

  - **Probegap Request Table**: When a probegap test is requested by a higher-layer application or protocol, it is registered in this table. Each entry in this table MUST be identified by a unique sequence number (unsigned 16-bit value) that is then used in the received QosProbe response from the sink station. For more information about the probegap test, see section 3.3.4.

  - **Timed-Probe Request Table**: When a timed probe test is requested by a higher-layer application or protocol, it is registered in this table. Each entry in this table MUST be identified by a unique sequence number (unsigned 16-bit value) that is then used in the received QosQueryResp response from the sink stations. For more information about the timed-probe test, see section 3.3.4.

  - **Sequence Number**: Each network test session MUST be able to generate a unique new sequence number every time a network test is requested. In order to do so, an unsigned 16-bit value is stored in each network test session and incremented every time it is used. If in the process of incrementing the value overflows to 0x0000, it MUST be automatically incremented to 0x0001, so at no point in time can it be zero.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

### 3.3.2 Timers

Each Network Test session has the following timers:

- Per-QosInitializeSink Response timer: This one-shot timer is used to ensure response (or non-response) to a QosInitializeSink request. This timer is only valid while the controller attempts to establish a network test session with the sink.

- Per-QosReset Response timer: This one-shot timer is used to ensure response (or non-response) to a QosReset request. This timer is only valid while the controller attempts to shut down a network test session.

Each entry in a **Probegap Request Table** has the following timer:

- Per-QosProbe Response timer: This one-shot timer is used to ensure response (or non-response) to a QosProbe request where the **Test Type** field in the QosProbe header is set to 0x01 (that is, a probegap test). This timer MUST be tied to the originating QosProbe frame by means of the corresponding entry in the **Probegap Request Table**. In other words, as long as the sink has not responded to the QosProbe frame, the timer MUST remain active.

Each entry in a **Timed-Probe Request Table** has the following timer:

- Per-QosQuery Response timer: This one-shot timer is used to ensure timely response (or non-response) to a QosQuery request. This timer MUST be tied to the originating QosQuery frame by means of the corresponding entry in the **Timed-Probe Request Table**. In other words, as long as the sink has not responded to the QosQuery frame, the timer MUST remain active.

### 3.3.3 Initialization

During initialization, the following conditions must be met:

- All timers must be disabled.

### 3.3.4 Higher-Layer Triggered Events

#### 3.3.4.1 Start Network Test Session

A higher-layer application or protocol must first instantiate a network test session with a sink identified by a responder before it can request subsequent timed probe or probegap tests with the sink.

When a higher-layer application or protocol requests a network test session with a given sink, the controller MUST first check if it already has a network test session in progress to the same sink station, and, if so, it MUST fail the request.

Otherwise, it MUST attempt to create a network test session state with a random sequence number. If it cannot create the state, it MUST fail the request.

It MUST then send a QosInitializeSink frame (see section 2.2.5.2) to the specified sink and set the Per-QosInitializeSink Response timer to expire after 100 milliseconds.

A timed probe test requires that the higher-layer application or protocol submit to the controller a set of one or more descriptors that identify the content of each QosProbe frame that it wants to send to the sink. When the controller receives this set, it MUST construct a QosProbe frame for each descriptor in the set. When all the frames are constructed, the controller MUST assign the next available sequence number to all of the frames and then time stamp each frame (**Controller Transmit Timestamp** field in QosProbe header) as it is transmitted. Immediately following the last frame, the controller MUST construct a QosQuery frame with the same sequence number to be sent to the sink. The controller MUST attempt to create a new entry for the newly-chosen sequence number and place it in the **Probegap Request Table**, before the QosQuery frame is sent. If a new entry cannot be created due to the lack of memory, the test request MUST be failed and all of the frames that were created MUST be deleted. The frames MUST be sent only after the appropriate entry can be created and placed in the **Probegap Request Table**. After the QosQuery frame is sent, the Per-QosQuery Response timer must be enabled and set to expire after 100 milliseconds.

A probegap test requires that the higher-layer application or protocol submit just one descriptor to be used for a timed-probe. When the controller receives this descriptor, it MUST construct a QosProbe frame using the next available sequence number. The **Controller Transmit Timestamp** field in the QosProbe header MUST be updated as the frame is transmitted. The controller MUST attempt to create a new entry for the newly-chosen sequence number and place it in the **Timed-Probe Request Table**, before the QosProbe frame is sent. If a new entry cannot be created due to lack of memory, the test request MUST be failed and all of the frames that were created MUST be deleted. The frames MUST be sent only after the appropriate entry can be created and placed in the **Timed-Probe Request Table**. After the QosProbe is sent, the Per-QosProbe Response timer MUST be set to expire after 100 milliseconds.

An example of different specifications that MAY<8> be applied on QosProbe frames sent by the controller is in the size or content (the data following the QosProbe header itself; this is ignored by the controller and sink, but it can be used to exercise the network equipment in interesting ways).

#### 3.3.4.2 Stop Network Test Session

When the higher-layer application or protocol for a previously-established network test session requests that the session be stopped, the Per-QosReset Response timer MUST be set to expire in 100 milliseconds and its logic (see section 3.3.6.4) invoked immediately. The request to shut down a session MUST always succeed, even if the QosAck response is not received from the sink. The next available sequence number MUST be used by all of these QosReset frames.

### 3.3.5 Message Processing Events and Sequencing Rules

When a message arrives, the controller MUST first check whether or not it is a valid QosError, QosReady, QosProbe, QosQueryResp, or QosAck frame. If not, it MUST be dropped.

#### 3.3.5.1 Receiving a QosProbe Frame

When a QosProbe frame is received, the controller MUST first verify that the **Test Type** field in the QosProbe header is set to 0x02 (that is, the sink returns the probegap test result). If not, the frame MUST be ignored.

Otherwise, the controller MUST attempt to locate a corresponding entry in the **Probegap Request Table** by matching its identifier against the **Sequence Number** field in the Base header of the received frame. If one is not found, the frame MUST be ignored.

Otherwise, the associated Per-QosProbe Response timer MUST be stopped. The controller MUST ensure that a high-resolution time stamp is sampled at the time the frame is received. It MUST then return this time stamp with the contents of the **Sink Receive Timestamp** and **Sink Transmit Timestamp** fields in the QosProbe header to the higher-layer application or protocol that requested the probegap test. The associated Per-QosProbe Response timer MUST then be stopped, and the corresponding entry must be removed from the **Probegap Request Table**.

#### 3.3.5.2 Receiving a QosQueryResp Frame

When a QosQueryResp frame is received, the controller MUST attempt to match the sequence number of this QosQueryResp to the identifier of an entry in the **Timed-Probe Request Table**. If one cannot be found, the QosQueryResp frame MUST be ignored.

If the count of **QosEventDesc** structures in the QosQueryResp header is greater than the count of descriptors in the array (as specified in section 2.2.5.6) given to the controller to start the test, the QosQueryResp MUST be ignored.

Otherwise, the **QosEventDesc List** field in the QosQueryResp header MUST be returned to the higher-layer application or protocol that initiated the timed probe test.

If the QosQueryResp is processed successfully, the associated Per-QosQuery Response timer MUST be stopped and the corresponding entry MUST be removed from the **Timed-Probe Request Table**.

#### 3.3.5.3 Receiving a QosError Frame

When a QosError frame is received, the controller MUST attempt to match the **Sequence Number** field in the Base header and the **Source MAC address** field in the Ethernet header of the received frame against an existing network test session. If a session cannot be found, the frame MUST be ignored.

Otherwise, the **Error Code** field in the QosError header MUST be used to inform the higher-layer application or protocol of why the request failed. The Per-QosInitializeSink Response timer MUST be stopped, and the corresponding network test session MUST be deleted.

#### 3.3.5.4 Receiving a QosReady Frame

When a QosReady frame is received, the controller MUST attempt to match the **Sequence Number** field in the Base header and the **Source MAC address** field in the Ethernet header of the received frame against an existing network test session. If a session cannot be found, the frame MUST be ignored.

Otherwise, the controller MUST notify the higher-layer application or protocol that the network test session has been established. The Per-QosInitializeSink Response timer MUST be stopped.

#### 3.3.5.5 Receiving a QosAck Frame

When a QosAck frame is received, the controller MUST attempt to match the **Sequence Number** field in the Base header and the **Source MAC address** field in the Ethernet header of the received frame against an existing network test session. If a session cannot be found, the frame MUST be ignored.

Otherwise, the controller MUST delete the associated network test session and MUST stop the Per-QosReset Response timer.

### 3.3.6 Timer Events

#### 3.3.6.1 Per-QosInitializeSink Response Timer Expiry

When this timer fires, the controller SHOULD attempt to send another QosInitializeSink frame to the sink and restart the timer to expire after 100 milliseconds. The fifth consecutive time that the timer expires, the controller MUST instead stop and return a time-out error result to the higher-layer application or protocol that originally requested the creation of the network test session. The associated network test session MUST also be deleted.

#### 3.3.6.2 Per-QosProbe Response Timer Expiry

When this timer fires, the controller MUST NOT attempt to resend the associated QosProbe frame. Instead, it MUST return a time-out error result to the higher-layer application or protocol that initiated the probegap test and the associated entry from the **Probegap Request Table** MUST be deleted.

#### 3.3.6.3 Per-QosQuery Response Timer Expiry

When this timer fires, the controller SHOULD attempt to send another QosQuery frame to the sink and restart the timer to expire after 100 milliseconds. The fifth consecutive time the timer expires, the controller MUST instead stop and return a time-out error result to the higher-layer application or protocol that initiated the timed probe test, and the associated entry from the **Timed-Probe Request Table** MUST be deleted.

#### 3.3.6.4 Per-QosReset Response Timer Expiry

When this timer fires, the controller SHOULD attempt to send another QosReset frame to the sink and restart the timer to expire after 100 milliseconds The fifth consecutive time the timer expires, it MUST stop sending the QosReset and delete the associated network test session.

### 3.3.7 Other Local Events

None.

## 3.4 Cross-Traffic Analysis Initiator Details

This section details the role of a controller station used in the LLTD QoS cross-traffic analysis type-of-service.

Applicable message request/response pairs are defined as follows.

| Sent by initiator | Sent by sink |
|---|---|
| QosCounterSnapshot | QosCounterResult |
| QosCounterLease | N/A |

### 3.4.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with what is described in this document.

The data elements required in any implementation are:

- Lease Period: This data element specifies the time period over which the cross-traffic analysis is performed. The period SHOULD be at least 5 minutes long.
- Sequence Number: Each time a higher-layer application or protocol requests the values of the cross-traffic analysis counters from a responder, the initiator MUST generate a unique sequence number for the QosCounterSnapshot request that it sends to the responder. This sequence number is an unsigned 16-bit value and MUST be incremented by using ones-complement arithmetic. This sequence number is a global value.
- Snapshot Request Table: This global table tracks the counter snapshot requests that higher-layer applications or protocols issue. Each entry in the table is identified by a unique sequence number.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation can implement such data in any way.

### 3.4.2 Timers

The Cross-Traffic Analysis Initiator role has one timer—the Per-Interface Lease Renewal timer. This recurring timer broadcasts the QosCounterLease frame. This timer remains active for as long as any higher-layer application or protocol is interested in performing cross-traffic analysis. This timer SHOULD have a lower period than the lease period so responders can keep collecting their counter histories.

Each entry in the **Snapshot Request Table** has a Per-Snapshot Response timer. This one-shot timer ensures a timely response (or non-response) to a QosCounterSnapshot request.

### 3.4.3 Initialization

During initialization, the following conditions must be met:

- All timers MUST be disabled.

### 3.4.4 Higher-Layer Triggered Events

#### 3.4.4.1 Start Cross-Traffic Analysis

When a higher-layer application or protocol indicates that it wants to start cross-traffic analysis on a given interface, the initiator MUST broadcast a QosCounterLease frame over that interface, and start the interface's periodic Lease Renewal timer. The timer SHOULD be set to expire every 3 minutes.

#### 3.4.4.2 Request Counters

When a higher-layer application or protocol requests the values of the cross-traffic analysis counters for a specific responder (specified by the MAC address), the initiator MUST transmit a QosCounterSnapshot request to that responder. The next available sequence number MUST be assigned to the request, and an entry MUST be created in the Snapshot Request Table before it is sent. The Per-Snapshot Response timer MUST be set to expire in 100 milliseconds.

The higher-layer application can also specify that a special MAC address is set in the **Real Destination Address** field of the Base header of the QosCounterSnapshot request to further refine the scope of the counters that are returned. For more information, see section 3.8.5.2. Unless this special MAC address is provided, the Cross-Traffic Analysis Initiator MUST always set this particular field to be equal to that used in the **Source MAC Address** field in the Ethernet header.

#### 3.4.4.3 Stop Cross-Traffic Analysis

When a higher-layer application or protocol indicates that it is finished with cross-traffic analysis on a given interface, the initiator MUST stop the interface's Lease Renewal timer.

### 3.4.5 Message Processing Events and Sequencing Rules

When a message arrives, the initiator MUST first check whether it is a valid QosCounterResult frame or not. If not, it MUST be dropped.

#### 3.4.5.1 Receiving a QosCounterResult Frame

When a QosCounterResult frame is received, the **Sequence Number** field in the Base header of the received frame MUST be used to look up a matching sequence number identifier in the **Snapshot Request Table**. If a matching sequence number is not found, the frame MUST be ignored.

Otherwise, the result MUST be passed back to the higher-layer application or protocol that requested the snapshot in the first place. The associated entry in the Snapshot Request Table MUST then be deleted, and the Per-Snapshot Response timer MUST be disabled.

### 3.4.6 Timer Events

#### 3.4.6.1 Per-Interface Lease Renewal Timer Expiry

When this timer fires, a QosCounterLease frame MUST be broadcast over the network.

#### 3.4.6.2 Per-Snapshot Response Timer Expiry

When this timer fires, the controller SHOULD attempt to send another QosCounterSnapshot frame to the responder and reset the timer to expire after 100 milliseconds. The fifth consecutive time the timer expires, the controller MUST instead stop and return a time-out error result to the higher-layer

application or protocol that initiated the request, and the associated entry from the **Snapshot Request Table** MUST be deleted.

### 3.4.7 Other Local Events

None.

## 3.5 Responder (Quick Discovery) Details

The following figure shows the workings of a responder's quick discovery state engine, also known as the enumeration state engine.

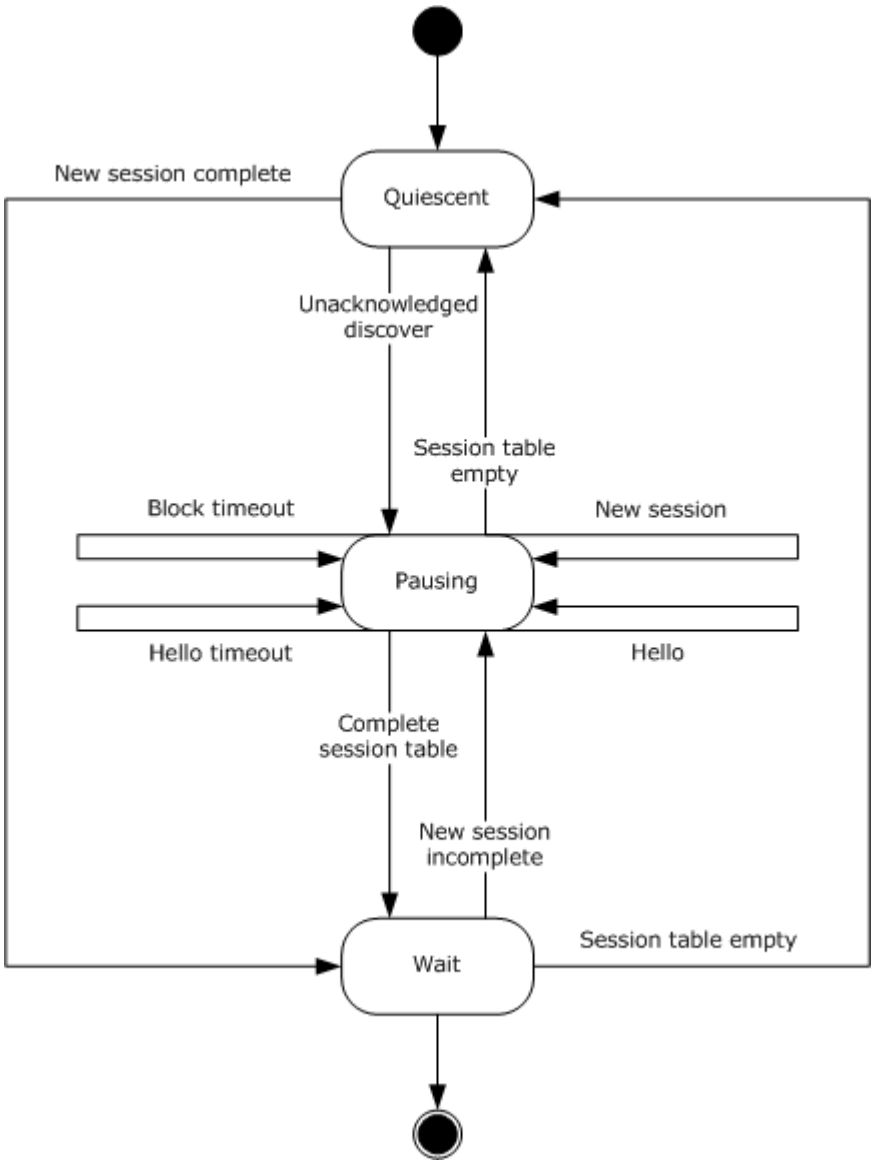



**Figure 1: Possible Responder's Quick Discovery states**

While in Quiescent state, responders only listen to broadcast frames and wait for a Discover frame to trigger an association with a mapper (only for topology discovery) or initiate enumeration session.

The Pausing state is critical to scalable discovery of the responders. During the Wait state, the responder waits for enumerators or the mapper to finalize their sessions via the Reset frame. Responders leave the Wait state for the Quiescent state when all enumerators have either timed out due to inactivity or have successfully sent the Reset command.

Message request/response pairs applicable to quick discovery are defined as follows.

| Sent by mapper | Sent by responder |
|---|---|
| Discover (as BROADCAST) | Hello (as BROADCAST) |
| Reset (as either UNICAST or BROADCAST) | N/A |

### 3.5.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those described in this document.

The data elements required in any responder implementation are:

- Topology State Machine State: This data element specifies the current state in the Topology State Machine.
- Generation Number: Knowing the correct generation number for a mapping iteration is necessary because of the way switches are forced to learn addresses. By the end of quick discovery, at most one mapper is active, and the mapper knows the correct generation number and all responders that are associated to it.
- Alpha: This data element specifies a RepeatBAND constant, and it MUST be set to 45.
- Beta: This data element specifies a RepeatBAND constant, and it MUST be set to 2.
- Gamma: This data element specifies a RepeatBAND constant, and it MUST be set to 10.
- Nmax: This data element specifies a RepeatBAND constant, being the maximum number of responders on a link, and it MUST be set to 10,000.
- r: This data element specifies the observed count of Discover and Hello frames over the network.
- I: This data element specifies the ideal time spacing between two Hello frames seen on the network. This data element MUST be set to 6.67 milliseconds.
- N: This data element specifies an estimate of the number of responders that have yet to respond.
- Begun flag: This data element flags the presence of a new enumerator or mapper station.
- Tb: This data element specifies a RepeatBAND constant that MUST be set to 300 milliseconds.
- Session Table: This data element stores enumerator state information and thereby enables the enumeration state engine to decide when to transmit Hello frames and when to transition to the Wait state. The table is indexed by the enumerator station's MAC address and the type-of-service

identifier (that is, quick discovery or topology discovery). Each entry MUST have the following fields:

- **Transaction ID (XID)**: The **XID** field is an unsigned 16-bit integer that uniquely identifies the mapper or enumerator session.
- **State**: This field specifies the current state in the Session Table State Machine, as shown in the following figure.

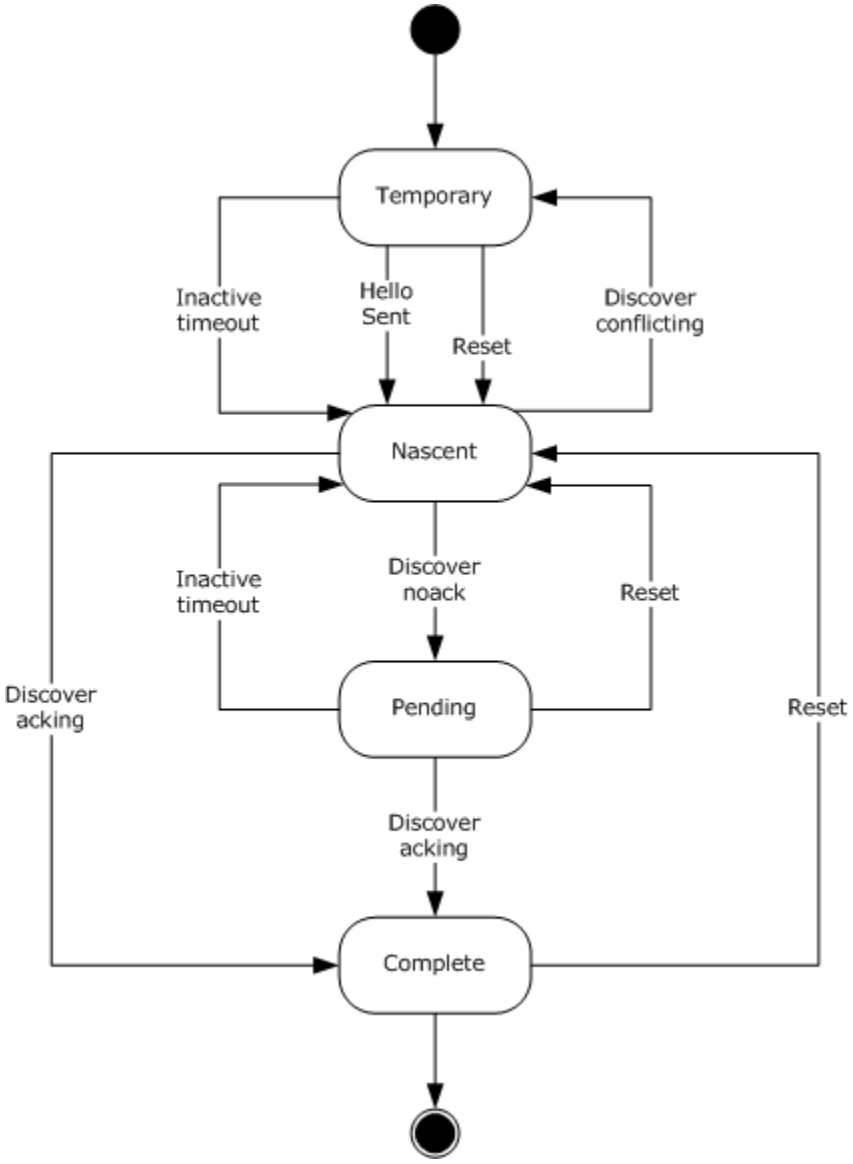



**Figure 2: Responder's Quick Discovery state**

- **Active Time**: This field specifies the time at which the last Discover frame was received.
- **Txc**: This field specifies the per-session Hello frame retransmission counter.

- **TXC**: This field specifies the maximum number of Hellos to retransmit per session.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation is at liberty to implement such data in any way it pleases.

### 3.5.2 Timers

The Responder (Quick Discovery) role has three timers:

- Session Inactivity timer: This periodic timer checks each session in the session table for inactivity.
- Block timer: This periodic timer operates the RepeatBAND network load control algorithm (section 3.5.6.2). It is only active when the enumeration state engine is in the Pausing state.
- Hello timer: This one-shot timer delays the sending of a Hello frame until RepeatBAND (section 3.5.6.2) determines that it is time to send one.

### 3.5.3 Initialization

During initialization, the following conditions must be met:

- All timers MUST be disabled.
- The enumeration state engine MUST be in Quiescent state.
- The Session Table MUST be empty.

### 3.5.4 Higher-Layer Triggered Events

None.

### 3.5.5 Message Processing Events and Sequencing Rules

The enumeration state engine MUST ignore any arriving message that is not explicitly identified in the following sections and pass them on to the topology discovery state engine, as detailed in section 3.6.

#### 3.5.5.1 Receiving a Discover Frame

When a Discover frame arrives, the responder MUST attempt to match the MAC address and type-of-service code of the sender against an entry in the Session Table.

If no entry exists, or the entry has a different XID, the responder MUST then attempt to create a session entry with the session state set depending on whether the request contains an acknowledgment for the responder (either pending or complete) or not. The active time MUST also be updated. If a session entry cannot be created due to the lack of memory, the Discover frame MUST be ignored silently.

If a session table entry exists (and has the same XID), the active time MUST be updated. If the responder's MAC address exists in the **Station List** field in the Discover frame (indicating that the responder's Hello request is being acknowledged), the entry state MUST be set to complete.

In a Discover frame for topology discovery type-of-service, only one such session can be marked as pending or complete. If the responder does not know of an active mapper, it MUST remember the MAC address of the sender of the current Discover frame as the current mapper.

If a current mapper already exists, the Session Table entry MUST be set to the temporary state (notwithstanding the previous paragraph).

Lastly, the enumeration state engine MUST transition to the Pausing, Wait, or Quiescent state, as indicated in section 3.5.5.1.1.

#### 3.5.5.1.1 State Transition Rules

The following subsections explain the exact state transition logic used from the perspective of each of the three possible states: Quiescent, Pausing, and Wait.

##### 3.5.5.1.1.1 Quiescent State

If a Discover frame arrives while the enumerator is in the Quiescent State, but does not acknowledge the responder, the enumeration state engine MUST proceed to the Pausing state. In other words, the Session Table MUST have an incomplete entry.

Otherwise, the enumeration state engine MUST proceed to the Wait state.

##### 3.5.5.1.1.2 Pausing State

If a Discover frame arrives, acknowledging the responder's Hello, but the Session Table still has one or more sessions that are not in the complete state, the enumeration state engine MUST remain in the Pausing state.

If at any time the Session Table changes and all session entries are in the complete state, the enumeration state engine MUST proceed to the Wait state.

If the Session Table is empty (for example, due to Reset or inactivity time out), the enumeration state engine MUST proceed to the Quiescent state.

##### 3.5.5.1.1.3 Wait State

If a Discover frame arrives, acknowledging the responder's Hello request, the enumeration state engine MUST remain in the Wait state (note that at this point, all entries in the Session Table are in the complete state).

Otherwise, the enumeration state engine MUST transition to the Pausing state.

If at any time the Session Table changes and becomes empty, the enumeration state engine MUST proceed to the Quiescent state.

#### 3.5.5.1.2 Network Load Control

Network load control and scalability of the enumeration process are handled by an algorithm called RepeatBAND (see section 3.5.6.2). Responders send Hello frames in the Pausing state, but they do not send them immediately. Instead, responders MUST measure the network load over a number of loosely synchronized rounds, also called blocks, of approximately fixed duration Tb (the "block time").

##### 3.5.5.1.2.1 Load Initialization

When the enumeration state engine transitions to the Pausing state, it MUST initialize N to 10,000 and set r to 0. It then MUST begin the first block round.

The responder MUST NOT begin to monitor the network load until it is ready to transmit; otherwise, many similar machines might think that the network load is low and become ready simultaneously.

#### 3.5.5.1.2.2 Dynamic Behavior

At the start of each round (triggered by the expiration of the Block timer) in the Pausing state, a responder MUST sample its random number generator and choose a time that is uniformly distributed between 0 and N times I. If the chosen time is less than Tb, the responder MUST set the Hello timer to the chosen time. If the time is greater than or equal to Tb, the responder MUST NOT send a Hello frame in this round (because the Hello timer will not expire during the round).

During the block, the responder MUST count the Hello and Discover messages on the network (including its own transmission if any) in the variable r, so at the end of the block, the responder can use this information to update its estimate of the number of active responders, as specified in section 3.5.6.2.

#### 3.5.5.1.2.3 Effect of Discover over Network Load Control

Discover frames are handled differently, depending on whether the enumerator is known to the responder (that is, a session already exists in the Session Table) and the responder is acknowledged. Discover frames are counted toward the load estimation.

If a new session is created directly into the complete state, it has no effect on the load control system. If an already existing session transitions to the complete state, it has no effect on load control (unless it causes simultaneous transition of the enumeration state engine out of the Pausing state). A Discover frame for an existing session that does not acknowledge the responder also does not change load control.

The **Txc** counter for the session MUST be set to **TXC**. If the session is causing a transition to Pausing state, the load control MUST be initialized as specified in section 3.5.5.1.2.1. If this new session is not causing a transition to Pausing state, the Begun flag MUST be set, which impacts load control at the end of the current block.

### 3.5.5.2 Receiving a Hello Frame

For each Hello frame received, the responder MUST increment r by one. For further specifications about the use of this counter, see section 3.5.5.1.2.2.

### 3.5.5.3 Receiving a Reset Frame

When a Reset frame is received, the responder MUST first look for a corresponding session entry in the Session Table by matching the **Real Source Address** field from the Base header to the enumerator's MAC address and the **Type of Service** field from the Demultiplex header to the entry's type-of-service identifier.

If no corresponding session entry is found, the Reset frame MUST be ignored. If a corresponding session entry is found, it MUST be deleted. If the session table becomes empty as a result, the enumeration state engine MUST proceed to the Quiescent state.

If the Reset is for a topology discovery session entry, the Topology State Machine MUST also be reset to the Quiescent state. In addition, all sessions in the temporary state MUST also be reset.

### 3.5.6 Timer Events

#### 3.5.6.1 Session Inactivity Timer Expiry

When this timer fires, each entry in the Session Table MUST be checked for inactivity as follows. If the session is not in the temporary state and its type-of-service identifier is topology discovery, and the topology discovery state engine is in the Command state, the session MUST be considered inactive if 60 seconds or more have elapsed since the active time. Otherwise, the session MUST be considered inactive if 30 seconds or more have elapsed since the active time.

If a session is considered inactive, it MUST be removed, and the enumeration state engine's state MUST be updated as specified in section 3.5.5.1.1.

This timer MUST be reset so it continues firing until the enumeration state engine transitions back to the Quiescent state.

#### 3.5.6.2 Block Timer Expiry

When the Block timer fires (signaling the end of the block), the responder MUST update the estimate of the number of active responders on the network based on the count of frames during the block and the measured length of the block (in milliseconds), which is called Ta (note that Ta is likely about the same as the period of the block timer (Tb), but on some platforms, it can be longer due to scheduling delays). The estimate MUST be calculated by using the RepeatBAND algorithm as follows.

```
Value = RoundUp( r * Nold * I / Ta )
Bound = RoundUp( Nold * Gamma / (Beta * Alpha) )
Nnew = Max( Bound, Min( 100 * Nold , Value ) )
```

If the implementation is accomplished carefully, this value is never zero or negative and can be implemented entirely in integer arithmetic.

The responder then MUST check the Begun flag. If it is set and the estimate N is less than half of Nmax, it MUST be doubled. Otherwise, if the Begun flag is set and N is less than Nmax, it MUST be set to Nmax. The Begun flag MUST then be cleared.

Finally, the responder MUST begin the next round.

#### 3.5.6.3 Hello Timer Expiry

After this timer fires, a Hello frame MUST be sent, the Txc counter MUST be decremented for each pending session in the Session Table, and each session in the temporary state MUST be deleted. When this counter reaches zero, the session MUST be marked complete even if it has not been acknowledged.

### 3.5.7 Other Local Events

#### 3.5.7.1 Media Disconnect Event

When the Media Disconnect event is received, all timers MUST be disabled. The enumeration state engine MUST transition to the Quiescent state. The Session Table MUST be cleared. If the topology discovery state engine is not already in Quiescent state, it MUST transition to the Quiescent state.

### 3.5.7.2 Entering Quiescent State

When the enumeration state engine enters the Quiescent state, all timers MUST be disabled. It is assumed that the Session Table is already empty before entering this state.

### 3.5.7.3 Entering Pausing State

When the enumeration state engine enters the Pausing State, the Begun flag MUST be set to false. N MUST be set to Nmax. All per-session Txc counters SHOULD be set to TXC; this is implied from the moment a session is first created. The Block timer MUST be started and set to expire after 300 milliseconds. The Session Inactivity timer MUST also be started and SHOULD be set to expire after 30 seconds.

The enumeration state engine MUST immediately decide, as specified in section 3.5.5.1.2.2, if a Hello timer is to be set.

### 3.5.7.4 Entering Wait State

When the enumeration state engine enters the Wait State, the Block timer and any pending Hello timer MUST be disabled.

## 3.6 Responder (Topology Discovery) Details

This section details the workings of a responder's topology discovery state engine. This state engine operates in one of three states, as shown in the following figure.

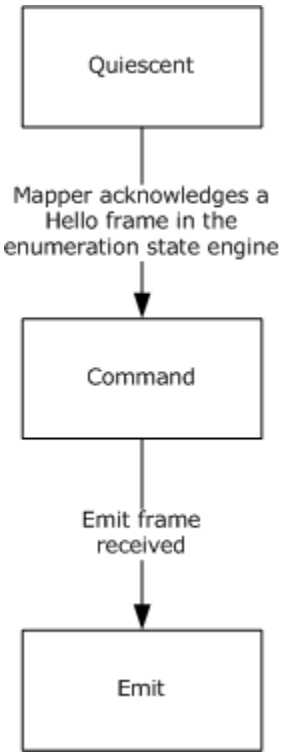



**Figure 3: Possible Responder's Topology Discovery states**

Responders in the Quiescent state ignore all frames that are marked for topology discovery. The Command state is reached when the enumeration state engine (see section 3.5) successfully associates with a mapper (and only one mapper). The Command state is where responders spend most of their time during topology discovery tests. In the Command state, responders execute Emit and Query commands from the mapper and operate with the network interface in promiscuous mode. The Emit state is reached only if responders receive the Emit command. As soon as the command is fully processed, responders return to the Command state. Responders return to the Quiescent state after the Reset command or after timing out due to inactivity.

It is important to note that the topology discovery state engine only processes frames after the enumeration state engine ignores them. By definition, the topology discovery state engine does not process Discover, Hello, and Reset frames. Moreover, when the topology discovery state engine is not in the Quiescent state, upon receipt of a Charge, Emit, Query, or QueryLargeTlv frame from the currently associated mapper, it must update the current topology discovery session's active time field in the enumeration state engine's Session Table.

Message request/response pairs applicable to topology discovery are defined as follows.

| Sent by mapper | Sent by responder |
|---|---|
| Emit | Ack / Flat (*) |
| Query | QueryResp |
| Charge | Flat (*) |
| QueryLargeTlv | QueryLargeTlvResp |

*If the request frame has a sequence number of zero, the responder does not send a response.

### 3.6.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those described in this document.

The data elements required in any responder implementation are:

- Topology State: This data element stores the current state of the topology discovery state engine, as shown in the following figure.

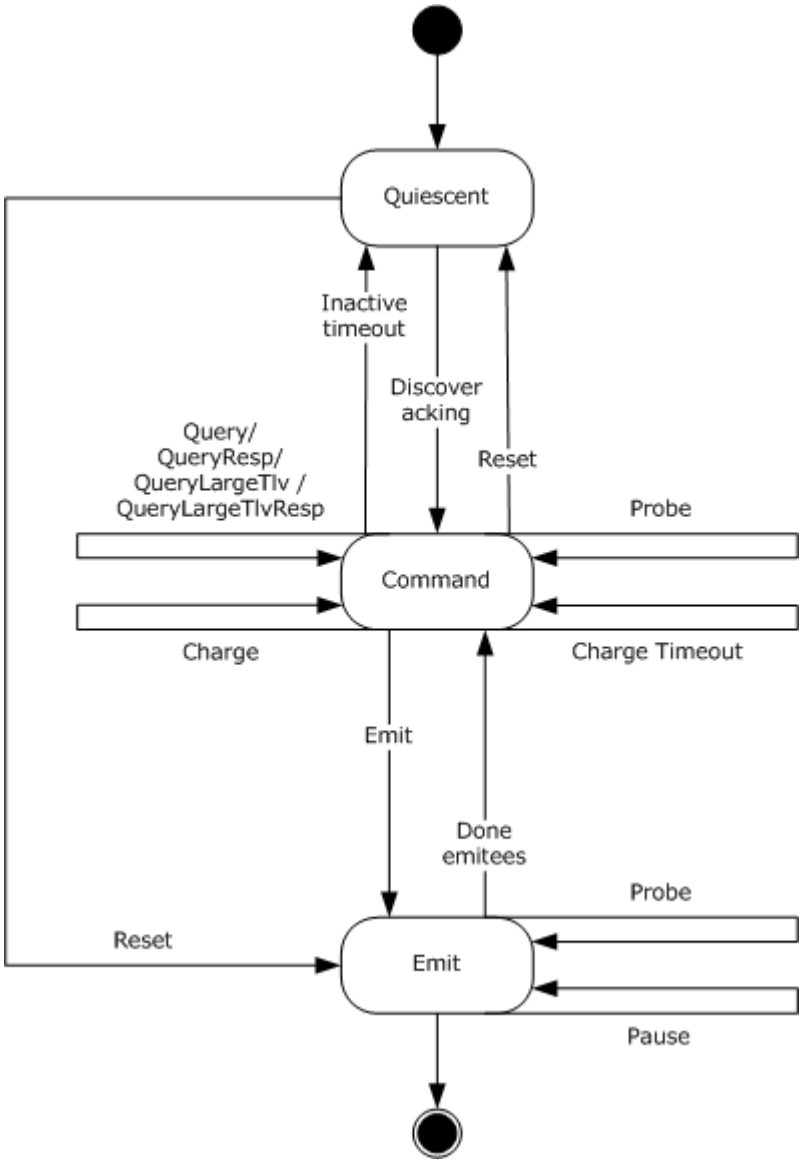



**Figure 4: Responder's Topology Discovery state**

- Generation Number: Knowing the correct generation number for a mapping iteration is necessary because of the way switches are forced to learn addresses. By the end of quick discovery, at most one mapper is active, and the mapper knows the correct generation number and all responders that are associated to it.
- Next Sequence Number: This data element is a 16-bit unsigned value that is incremented by using ones-complement arithmetic. This data element MUST be initially set to zero, which indicates an invalid sequence number value. The first request that the mapper sends (via one of the Charge, Emit, Query, or QueryLargeTLV frames) that has a non-zero **Sequence Number** field in the Base header is incremented and stored in this data element.
- Sees-List: This list MUST hold all of the information that is required to construct one or more **RecveeDesc** structures that are returned in the QueryResp packet (as specified in section

2.2.4.9). Entries in the list MUST be stored in such a way that the oldest entry can be returned first. A responder SHOULD support up to 10,000 entries in the Sees-List.

- Last-Sent Response: The Last-Sent Response MUST be identified by the **Function Number** field in the Demultiplex header of the Request frame and the **Sequence Number** field in the Base header of the original request frame. Each time the responder sends out an Ack, Flat, QueryResp, or QueryLargeTlvResp frame, it updates this value as well as a copy of the response frame that it sent.
- Charge/CTC Counters: A responder MUST maintain the current transmit credit (CTC) count, in both frames and bytes. The CTC (or charge) frame count is an unsigned 8-bit integer, and the CTC (or charge) byte count is an unsigned 16-bit integer.
- Emit List: This data element is a list that stores the remaining **EmiteeDescs** fields in the Emit header that need to be processed when the topology state is set to Emit.
- Emit Sequence Value: This 16-bit unsigned value stores the sequence number of the Emit frame that is being processed when the topology state is set to Emit.
- Error Flag: This is a global flag. It MUST be set to FALSE initially. It MUST be set to TRUE when a Probe frame arrives, and the responder is not able to accommodate it in the Sees-List.
- Broadcast Flag: This data element is a global flag. It MUST initially be set to false. It MUST be set to TRUE when any supported frame is received in which the real source address (**Real Source Address** field in the Base header) is not equal to the Ethernet header's source address. If this flag is set, the responder MUST broadcast all of its response frames.
- Large Data Property List: This data element is a set of large data properties, as specified in section 2.2.2, for the responder itself.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation can implement such data in any way.

### 3.6.2 Timers

The Responder (Topology Discovery) role has two timers:

- Charge timer: This one-shot timer zeroes out the CTC counters.
- Emit timer: This one-shot timer processes each **EmiteeDesc** field in an Emit request.

### 3.6.3 Initialization

During initialization, the following conditions must be met:

- All timers MUST be disabled.
- The topology discovery state engine (topology state) MUST be in the Quiescent state.
- CTC byte and frame counters MUST be zero. The Sees-List MUST be empty.
- The Error flag MUST be cleared.
- The Broadcast flag MUST be cleared.
- The Last-Sent response must be zeroed.

### 3.6.4 Higher-Layer Triggered Events

None.

### 3.6.5 Message Processing Events and Sequencing Rules

When a message arrives, the responder MUST first check whether it is a valid Charge, Emit, Query, or QueryLargeTlv frame or not. If not, it MUST be dropped.

#### 3.6.5.1 Receiving a Charge Frame

If the topology state is not in the Command state, the Charge frame MUST be ignored.

If the **Sequence Number** field in the Base header of the received Charge frame is nonzero, the responder MUST check this sequence number and function number (**Function Number** field in the Demultiplex header) against the Last-Sent Response. If there is a match, the frame saved MUST be resent and the Charge request MUST be ignored.

If there is no match, and the sequence number in the frame is nonzero, the responder MUST validate this sequence number against the next sequence number. If the next sequence number is zero or if the numbers match, the sequence number from the Charge frame is incremented by one and stored. Otherwise, if the numbers do not match, the Charge frame MUST be ignored.

The responder MUST then zero out the Last-Sent Response (and delete any saved frame) and proceed to increase the CTC counters by incrementing the frame count by one and the byte count by the total size of the Charge frame (including any payload). The amount of charge that is available via the CTC counters MUST be capped at a maximum value to prevent a rogue mapper from accumulating a large amount of charge at multiple responders and releasing this charge at the same time against a target. The limits SHOULD be 65,536 bytes and 64 frames.

If the Charge frame contains a nonzero sequence number, the responder MUST also send a Flat frame in response, carrying the current CTC values (before taking into account the charge that is required for the Flat frame). Note that such a Charge frame has the net effect of increasing only the CTC byte count (if at all). In other words, the Flat frame MUST consume one frame charge and a byte charge equivalent to the size of the Flat frame itself.

When the responder sends the Flat frame, it MUST record this information in the Last-Sent Response.

The Charge timer MUST be started if the value of either the CTC byte or frame counters is non-zero, after the Flat frame charge has been accounted for. The time out SHOULD be set to 1,000 milliseconds.

#### 3.6.5.2 Receiving an Emit Frame

A responder that is not in the Command state MUST ignore received Emit frames.

If the **Sequence Number** field in the Base header of the received Emit frame is non-zero, the responder MUST check this sequence number and function number (**Function Number** field in Demultiplex header) against the Last-Sent Response. If there is a match, the frame that is saved MUST be resent, and the Emit request MUST be ignored.

If there is no match, and the frame's sequence number is nonzero, the responder MUST validate this sequence number against the next sequence number. If the next sequence number is nonzero and does not match the sequence number of the Emit frame, the frame MUST be ignored.

Next, the responder MUST zero out the Last-Sent Response (and delete any saved frame). The Emit frame MUST then be checked for validity by testing whether all of the following are true:

- The Emit frame was not sent to the broadcast address.
- Train and Probe **Source Address** field values equal the responder's MAC address or are within the range of the organizationally unique identifier (OUI) that is allocated for this protocol (see section 1.9).
- Trains and Probe **Destination Address** field values are not an Ethernet broadcast or multicast address.
- The cumulative Pause value from all quadruples in the Emit frame MUST NOT exceed 1 second.

If any of the previous statements are not true, the responder MUST ignore the Emit frame.

If validation succeeds, the responder MUST increase the CTC counters by incrementing the frame count by one, and the byte count by the total size of the Emit frame (including any payload). Small amounts of bytes charge can be transferred simply by padding an Emit frame appropriately.

To avoid amplification attacks, the responder MUST require enough CTC (in both packets and bytes) to handle an Emit frame (including the cost of sending a possible acknowledgment frame). If not enough CTC exists (and the Emit frames is intended to be reliable; that is, a sequence number is present), a Flat frame (which contains the current CTC values) MUST be returned. An Emit frame always contains enough inherent charge to send a Flat frame. When the responder sends the Flat frame, it MUST record this information in the Last-Sent Response.

After it is determined that an Emit frame will be accepted, the CTC counters MUST then be zeroed, regardless of whether or not the Emit frame has a non-zero sequence number. The topology discovery state engine MUST then transition into the Emit state (by setting the topology state to Emit) while the Emit frame is being processed. The responder MUST attempt to copy the entire **EmiteeDescs** field in the Emit header into the emit list. The sequence number of the Emit frame is copied into Emit Sequence Value (even if it is zero). The Emit timer MUST be started, with the expiration time delta set to the Pause value in the first quadruple in the Emit header. If the responder fails to copy the **EmiteeDescs** field, it MUST silently ignore the Emit frame.

### 3.6.5.3 Receiving a Probe Frame

Upon receiving a Probe frame, if the topology state is not in the Command or Emit state, the Probe frame MUST be ignored.

Otherwise, the topology discovery state engine MUST attempt to add a new **RecveeDesc** field to its Sees-List. If it runs out of memory, or reaches the maximum size of the Sees-List, it MUST indicate this by setting the Error flag when responding to the Query request.

The responder MUST record the following information in the Sees-List entry:

- **Real Source Address** field from the Base header.
- **Source Address** field from the Ethernet header.
- **Destination Address** field from the Ethernet header.

### 3.6.5.4 Receiving a Query Frame

When a Query frame is received, if the topology state is not in the Command state, the Query frame MUST be ignored.

Otherwise, if the **Sequence Number** field in the Base header of the received Query frame is zero, the Query frame MUST be ignored. The responder MUST check this sequence number and function number (**Function Number** field in Demultiplex header) against the Last-Sent Response. If there is a match, the saved frame MUST be resent, and the Query request MUST be ignored.

If there is no match, the responder MUST validate this sequence number against the next sequence number. If the next sequence number is zero, or if the numbers match, the sequence number from the Query frame is incremented by one and stored. Otherwise, if the numbers do not match, the Query frame MUST be ignored.

The responder MUST then zero out the Last-Sent Response (and delete any saved frame).

The responder MUST now send a QueryResp frame to the mapper, including as many entries in its Sees-List as will fit in the frame. The responder MUST then remove the transmitted entries from its Sees-List. If the list contains more entries than will fit in a single QueryResp frame, the responder MUST set the More bit in the QueryResp header so that the mapper will continue sending Query frames until it has gathered all of the entries.

If the Error flag is set, the responder MUST set the Error bit in the QueryResp header. If the Sees-List is empty, the Error flag MUST then be cleared.

When the responder sends the QueryResp frame, it MUST record this information in the Last-Sent Response.

### 3.6.5.5 Receiving a QueryLargeTlv Frame

Some TLV pairs can be too large to return in a single Hello frame. These TLVs are returned by using the QueryLargeTlv header. For a list of these TLVs, see the Hello and QueryLargeTlv frame formats in sections 2.2.4.3 and 2.2.4.13, respectively.

The QueryLargeTlv and kQueryLargeTlvResp frames (see section 2.2.4.14) operate in a very similar way to the Query and QueryResp frames. A QueryLargeTlv frame is sent to the responder's topology discovery state engine and asks it to return as many octets as possible, starting from a specific offset, for a specific TLV type.

When a QueryLargeTlv frame is received, if the topology state is in Command state, the QueryLargeTlv frame MUST be ignored.

If the **Sequence Number** field in the Base header of the received QueryLargeTlv frame is zero, the QueryLargeTlv frame MUST be ignored. Otherwise, the responder MUST check this sequence number and function number (**Function Number** field in Demultiplex header) against the Last-Sent Response. If there is a match, the saved frame MUST be resent, and the QueryLargeTlv request MUST be ignored.

If there is no match, the responder MUST validate this sequence number against the next sequence number. If the next sequence number is zero, or if the numbers match, the sequence number from the QueryLargeTlv frame is incremented by one and stored. Otherwise, if the numbers do not match, the QueryLargeTlv frame MUST be ignored.

The responder MUST then zero out the Last-Sent Response (and delete any saved frame).

The responder then MUST check whether it has a Large Data Property for the requested TLV type. If not, the responder SHOULD ignore the frame, but MAY<9> instead respond with a QueryLargeTlvResp, where the **Length** field is set to zero.

Otherwise, the responder MUST now acknowledge the QueryLargeTlv by returning the maximum possible number of octets of the requested Large Data Property that fit in a single Ethernet frame,

starting from the specified offset. If there are more octets to return, the responder MUST set the More bit in the QueryLargeTlvResp frame to prompt the mapper to continue sending QueryLargeTlv frames with updated offset values until it has gathered the full TLV. The mapper does not know how large the TLV is until the final QueryLargeTlvResp frame is returned (with the More bit set to zero).

When the responder sends the QueryLargeTlvResp frame, it MUST record this information in the Last-Sent Response.

### 3.6.6 Timer Events

#### 3.6.6.1 Charge Timer Expiry

When the Charge Timer expires, the responder MUST zero out the CTC counters.

#### 3.6.6.2 Emit Timer Expiry

When the Emit timer expires, the first EmiteeDesc entry (as specified in section 2.2.4.4) in the Emit List MUST be processed, which results in the sending of either a Train or Probe frame, as specified in the EmiteeDesc entry. The **Source MAC Address** field in the Ethernet header MUST be the source MAC address that is specified in the EmiteeDesc entry. The **Real Source Address** field in the Base header MUST be the MAC address of the responder itself on the network interface over which the frame is sent. Next, the processed entry MUST be removed from the Emit List. If the responder fails to transmit the Train or Probe frame, it MUST transition the topology state to the Command state (at this point, it is expected that the mapper retries the Emit command again with the same sequence number).

If the Emit List is not empty, the Emit timer MUST be reactivated with the expiration time delta set to the **Pause** field of the next entry in the Emit List.

If the Emit List is empty, the Emit Sequence Value is checked. If this value is zero, the topology state MUST transition to the Command state and the Emit timer MUST be stopped.

Otherwise, if the Emit Sequence Value is non-zero, the responder MUST send an Ack response to the mapper by setting the **Sequence Number** field in the Base header of the Ack frame to the Emit Sequence Value. It MUST record this information in the Last-Sent Response. The topology state MUST now transition to the Command state and, the Emit timer MUST be stopped.

If at any time the topology state transitions to the Command state, and if Emit Sequence Value is non-zero, it MUST be incremented by one and stored in Next Sequence Number.

### 3.6.7 Other Local Events

#### 3.6.7.1 Media Disconnect Event

When the Media Disconnect event is received, the topology discovery state engine MUST transition to the Quiescent state (and all the side effects of entering this state MUST be observed as specified in section 3.6.7.2).

#### 3.6.7.2 Entering Quiescent State

When the topology discovery state engine enters the Quiescent state, all timers MUST be disabled. CTC byte and frame counters MUST be zero. The Sees-List MUST be cleared. The Error flag MUST be cleared. The Broadcast flag MUST be cleared. The Last-Sent Response must be zeroed.

### 3.6.7.3 Entering Command State

When the topology discovery state engine enters the Command state, the Emit timer MUST be stopped.

## 3.7 QoS Sink Details

This section details the workings of a responder's QoS network test engine.

Message request/response pairs applicable to a sink are defined as follows.

| Sent by controller | Sent by sink |
|---|---|
| QosInitializeSink | QosError / QosReady |
| QosProbe | QosProbe (*) |
| QosQuery | QosQueryResp |
| QosReset | QosAck |

*If the request frame does not contain a non-zero sequence number, the responder does not send a response.

### 3.7.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those described in this document.

The data elements required in any sink implementation are:

- Session List: This data element is a list of active network test sessions. A sink MUST support at least three unique network test sessions, up to a recommended maximum of ten sessions. Each network test session is identified by the MAC address of the controller station, and each network test session also contains the following fields:
  - **Error Flag**: This field is initially set to FALSE. Use this flag when the sink cannot allocate the memory for a sequence bucket.
  - **Last Active Time**: This field specifies the time at which the last QosProbe or QosQuery frame for this network test session was received.
  - **Sequence Bucket**: This field is a list of entries holding information that was obtained from incoming QosProbe frames with the **Test Type** field set to 0x00, all belonging to the same sequence number. For further specifications about using this field, see section 3.7.5.2.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation can implement such data in any way.

### 3.7.2 Timers

The QoS Sink role has one timer—the Inactivity timer. This is a periodic timer that MUST operate at a period of 30 seconds. It is used to expire inactive network test sessions.

### 3.7.3 Initialization

During initialization, the following conditions must be met:

- All timers MUST be disabled.
- The Session List MUST be initialized to empty.

### 3.7.4 Higher-Layer Triggered Events

None.

### 3.7.5 Message Processing Events and Sequencing Rules

When a message arrives, the sink MUST first check whether it is a valid QosInitializeSink, QosProbe, QosQuery, or QosReset frame or not. If not, it MUST be dropped.

#### 3.7.5.1 Receiving a QosInitializeSink Frame

When a sink receives a QosInitializeSink frame, it MUST first find a matching network test session that already exists in the sink's Session List. If one exists, it MUST immediately reply with a QosReady frame. Otherwise, processing continues as follows.

The **Interrupt Mod** field in the QosInitializeSink header specifies the interrupt moderation mode that needs to be honored for the session to be established. If the sink cannot support the requirement, it MUST send a QosError frame with the Error Code value equal to 0x02.

Otherwise, if the sink successfully honors the interrupt moderation request, it MUST attempt to create a network test session and add it to the Session List. A sink MUST use the **Real Source Address** field in the Base header of the QosInitializeSink frame to identify the controller station. If a sink cannot support additional sessions, it MUST return a QosError frame with the Error Code value equal to 0x01.

Otherwise, the sink MUST return a QosReady frame.

If the Inactivity timer has not already been started, it MUST be started as soon as the Session List is not empty.

#### 3.7.5.2 Receiving a QosProbe Frame

After a network test session is established, the controller sends one or more QosProbe frames to the sink over a period of time. The exact action the sink takes in response to this frame depends on the **Test Type** field in the QosProbe header.

If the value of the field is 0x01, the controller has requested that the sink participate in a probegap test. On receipt of such a frame, the sink MUST immediately copy the QosProbe frame as-is and return it to the controller with the following modifications:

- The **Source Address** and **Destination Address** fields in the Ethernet header MUST be exchanged.
- The **Real Source Address** and **Real Destination Address** fields in the Base header MUST be exchanged.
- Any existing Ethernet 802.1p tag MUST be added or removed as per directions in the **T** and **802.1p Value** fields of the received QosProbe header.

- The **Sink Receive Timestamp** field MUST be updated with a high-resolution time stamp sampled at the earliest time possible when the QosProbe frame was received.
- The **Sink Transmit Timestamp** field MUST be updated with a high-resolution time stamp sampled at the last possible moment before the outgoing QosProbe frame is sent.
- The **Test Type** field in the outgoing QosProbe header MUST be changed to the value 0x02, which indicates to the controller that the QosProbe is sourced from a sink.

If the value of the **Test Type** field is 0x00, the controller has requested that the sink participate in a timed probe test. This test requires that a sink receive and record up to 82 consecutive QosProbe frames, all of the same sequence number. All timed probe frames following the eighty-second frame MUST be ignored completely. The collection of QosProbe records for a specific sequence number is called a sequence bucket. The sink MUST attempt to record specific bits of information from each frame in the form of an 8-octet high-resolution time stamp of the send operation on the controller side, an 8-octet high-resolution time stamp of the receive operation on the sink side, and a 1-octet identifier. The controller requests this recorded information immediately after the last QosProbe frame in the sequence is sent via the QosQuery frame. The exact number of QosProbe frames sent will vary.

In some rare cases, the QosQuery frame may be dropped and the controller may resend it if needed. However, such a retransmission implies the overlapping arrival of the next series of QosProbe frames under a subsequent sequence number. Meanwhile, the QosQuery frame for the previous sequence bucket can still arrive in the near future. In view of this possibility, the sink MUST be prepared to handle at least two sequence buckets worth of recordings at any point in time up to a maximum of ten sequence buckets where possible. As a new sequence bucket is needed, the oldest one SHOULD be cleared and reused.

In case of memory allocation failure preventing the information in the frame from being recorded, the sink MUST set the network test session's Error flag to TRUE, so it reports the error condition in the Error bit in the QosQueryResp header when replying to a QosQuery request.

The applicable network test session's last active time MUST be updated on receipt of this frame.

### 3.7.5.3 Receiving a QosQuery Frame

Upon receipt of a QosQuery frame, the sink MUST first match the **Real Source Address** field in the Base header against an existing network test session's controller MAC address. If one cannot be found, the QosQuery frame MUST be ignored.

Next, the sink MUST match the **Sequence Number** field in the Base header against the sequence bucket in the associated network test session. If one cannot be found, the QosQuery frame MUST be ignored.

The sink MUST send only one QosQueryResp frame in response because there are no more records that are stored in a sequence bucket than will fit in a standard 1514-octet Ethernet frame.

If at any time the sink encounters a memory allocation failure while attempting to allocate storage for the sequence bucket, it MUST set the network test session's Error flag.

The applicable network test session's last active time MUST be updated on receipt of this frame.

#### 3.7.5.4 Receiving a QosReset Frame

Upon receipt of a QosReset frame, the sink MUST attempt to match the **Real Source Address** field in the Base header of the QosReset frame against its Session List. If a session is found, it MUST send a QosAck response. Otherwise, the sink MUST NOT send a response.

If the Session List is empty, the Inactivity timer MUST be disabled.

### 3.7.6 Timer Events

#### 3.7.6.1 Inactivity Timer Expiry

When the Inactivity timer expires, the sink SHOULD<10> remove any network test sessions that have had at least two minutes of inactivity as computed from the last active time.

### 3.7.7 Other Local Events

#### 3.7.7.1 Media Disconnect Event

When the Media Disconnect event is received, the sink MUST remove all sessions from the Session List. All timers MUST be disabled.

## 3.8 Responder (QoS Cross-Traffic) Details

This section details the workings of a responder's QoS cross-traffic engine.

Applicable message request/response pairs are defined as follows.

| Sent by controller | Sent by responder |
|---|---|
| QosCounterSnapshot | QosCounterResult |
| QosCounterLease | N/A |

### 3.8.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behaviors are consistent with those described in this document.

The data elements required in any responder implementation are:

- Counter History: This data element specifies a collection of bytes, packets received, and sent counters for each network interface existing on the device that implements the responder. Each history buffer is normally implemented as a circular-buffer. Both byte counts and packet counts use a fixed scaling factor inclusively between 1 and 256 kilobytes or packet units respectively. Each individual implementation of the protocol is free.<11>

  All counters MUST be sampled at 1-second intervals, with each counter measured relative to that from the previous interval. At least 3 seconds of history MUST be maintained for each counter. Devices with sufficient spare memory SHOULD collect up to 30 seconds of history.

  Each row in the collection contains the following counters:

- Number of bytes received: This counter MUST be available.
- Number of bytes sent: This counter MUST be available.
- Number of packets received: Devices SHOULD choose to collect this counter.
- Number of packets sent: Devices SHOULD choose to collect this counter.

- Lease Period: This data element is the time period in which the counter history is being collected. The period SHOULD be 5 minutes in length.

**Note** The previous conceptual data can be implemented by using a variety of techniques. An implementation can implement such data in any way.

### 3.8.2 Timers

The Responder (QoS Cross-Traffic) role has two timers:

- Lease timer: A one-shot timer that is started and renewed when a QosCounterLease frame is received. This timer enforces the Lease Period. While this timer is active, the counter history is guaranteed to be available for query by the server via the QosCounterSnapshot request.
- Snapshot timer: This periodic timer is active only when the lease timer is running. This timer MUST have a period of 1 second.

### 3.8.3 Initialization

During initialization, the following conditions must be met:

- All timers MUST be disabled.

### 3.8.4 Higher-Layer Triggered Events

None.

### 3.8.5 Message Processing Events and Sequencing Rules

When a message arrives, the responder MUST first check whether it is a valid QosCounterLease or QosCounterSnapshot frame or not. If not, it MUST be dropped.

#### 3.8.5.1 Receiving a QosCounterLease Frame

On receipt of this request, a responder MUST set the Lease timer to expire after 5 minutes. If the Snapshot timer is not already running, it MUST be started as well, and set to expire after 1 second.

#### 3.8.5.2 Receiving a QosCounterSnapshot Frame

When a responder receives a QosCounterSnapshot frame, and a Lease Period is not in effect because the server failed to send sufficient QosCounterLease frames to keep it going, the responder MAY<12> simply ignore this frame.

Otherwise, the responder MUST send a QosCounterResult frame in response. The QosCounterSnapshot frame carries a sequence number that MUST be quoted in the transmission of the QosCounterResult response. The QosCounterResult response MUST return at most the **History Size** field's (from the QosCounterSnapshot header) count of snapshots from the counter history, starting with the oldest snapshot available. The last snapshot in the QosCounterResult response

MUST be the sub-second sample, whose existence is always implied and not reflected by the **History Size** field.

The **Real Destination Address** field in the Base header of the QosCounterSnapshot frame indicates the network interface for which the counter history is to be returned. In some cases, the **Real Destination Address** field in the Base header does not equal the destination MAC address in the Ethernet header. This is intended to be used in the case where the responder is an access point device, where the **Real Destination Address** is the BSSID address of one of its wireless bands or a special FF:FF:FF:FF:FF:FF address. The responder SHOULD<13> return only the relevant counter history given the specified BSSID (if the address is not recognizable, the QosCounterSnapshot request SHOULD be ignored), or in the case of the special address, return the aggregate of the counter histories for all of its network interfaces, including the wireless bands it supports.

### 3.8.6 Timer Events

#### 3.8.6.1 Lease Timer Expiry

When the Lease timer fires, the Snapshot timer MUST be stopped. Any existing counter history MUST be cleared.

#### 3.8.6.2 Snapshot Timer Expiry

When the Snapshot timer fires, the responder MUST take a snapshot of the current number of bytes and packets that were sent and received for each network interface that is available on the device. It MUST then add this value to the appropriate counter history. If a history reaches its maximum size, the oldest snapshot MUST be removed to make room for the new snapshot.

### 3.8.7 Other Local Events

None.

# 4 Protocol Examples

The following sections describe several operations as used in common scenarios to illustrate the function of the Link Layer Topology Discovery (LLTD) Protocol.

## 4.1 Example 1: Mapping a Network

The following figure shows a typical network that interconnects two computers and a printer. The user may have connection problems between the two computers or between a computer and the printer for a variety of reasons, including a mismatch in IP addressing on the network. Application problems may motivate the user to generate a map of his or her network to help discover the problem.

The user uses one of the computers as a Mapper, and the printer (R1) and laptop PC (R2) function as responders. They are interconnected with an Ethernet hub.

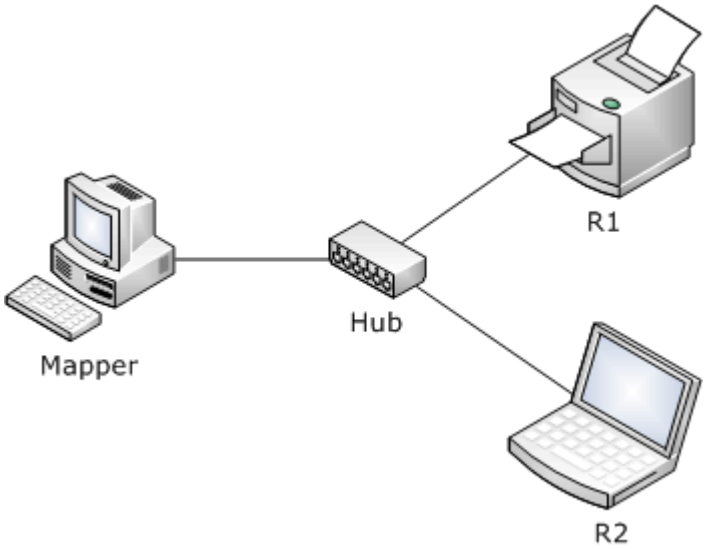



**Figure 5: Typical two-computers, one-printer network**

The following figure shows the protocol exchange between the mapper and the two responders that are on the network.

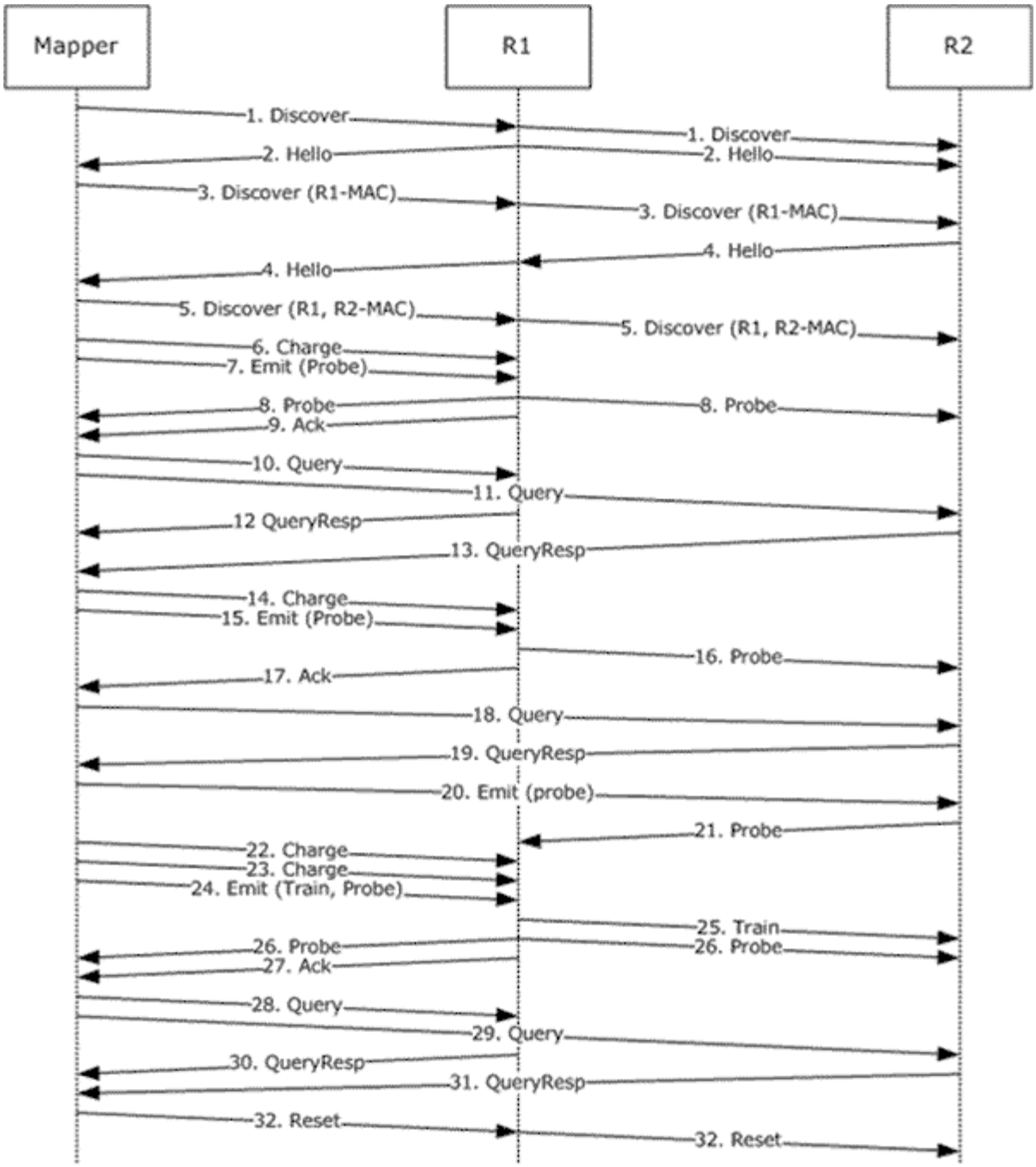



**Figure 6: Protocol exchange between networked mapper and two responders**

The following list describes each step in the protocol exchange:

1. The Mapper broadcasts a Discover frame with a Generation Number of zero to determine what responders are available on the network.

2. Responder 1 (R1) broadcasts a Hello frame that indicates its current Generation Number and basic information, such as Host ID, Characteristics, and Physical Medium (Ethernet in this case) in its **TLV_List**.

3. The Mapper broadcasts another Discover frame with the generation number given by R1, including R1's responder in the Station List.

4. Responder 2 (R2) broadcasts a Hello frame that indicates its current Generation Number of zero, and basic information, such as Host ID, Characteristics, and Physical Medium (Ethernet in this case) in its **TLV_List**.

   **Note** R2 used its RepeatBAND load control mechanism (section 3.5.6.2) to not respond to the first Discover with a Hello response.

5. The Mapper broadcasts another Discover frame with the generation number given by R1, including R2's MAC address in the Station List.

6. The application now decides it wants to test the network topology and invokes LLTD with a series of tests for R1. The LLTD Mapper sends a Charge frame to R1 to generate sufficient byte and frame credits in R1 for a request that will follow.

7. The Mapper sends an Emit frame to R1, indicating that R1 is to send a Probe frame with a Source MAC Address of 00-0D-3A-D7-F2-01 and a Destination MAC Address of 00-0D-3A-D7-F1-41.

8. R1 transmits the Probe frame.

   **Note** The Destination MAC Address does not address any machine in particular, so it traverses the network like a broadcast address.

9. R1 sends an Ack frame to the Mapper to indicate that it has completed the Emit request. At this point, the Mapper indicates to the application that the series of tests has completed.

10. The application asks LLTD to send a Query to R1 to get the list of MAC address seen by this responder.

11. The application also asks LLTD to send a Query to R2 to get the list of MAC address that this responder has seen.

12. R1 sends a QueryResp to the Mapper with no MAC address in the list, and the Mapper completes the application's request from step 10.

13. R2 sends a QueryResp to the Mapper with an entry that indicates it saw a frame with a Source MAC Address of 00-0D-3A-D7-F2-01 and a Destination MAC Address of 00-0D-3A-D7-F1-41. The Mapper completes the application's request from step 11.

14. The application decides to conduct another test and gives LLTD another set of commands for R1. The Mapper sends a Charge frame to R1 to generate sufficient byte and frame credits in R1 for a request that will follow.

15. The Mapper sends an Emit frame to R1, indicating that R1 is to send a Probe frame with a Source MAC Address of 00-0D-3A-D7-F2-02 and R2's Destination MAC Address.

16. R1 sends a Probe frame destined to R2 with a Source MAC Address of 00-0D-3A-D7-F2-02.

17. R1 sends an Ack frame to the Mapper to indicate it has completed the Emit request. At this point, the Mapper indicates to the application that the latest test has completed.

18. The application asks LLTD to send a Query to R2 to get the list of MAC addresses that this responder has seen.

19. R2 sends a QueryResp to the Mapper with an entry that indicates that it saw a frame with a Source MAC Address of 00-0D-3A-D7-F2-02 and a R2's MAC address as the destination. LLTD indicates this information to the application.

    **Note** R2 did not return the MAC address pair that it reported in step 12 because after sending that information in step 12, it cleared that information from memory.

20. The application asks LLTD to perform another test from R2, and the Mapper sends an Emit to R2 with a Sequence Number of zero and a request for R2 to send a Probe using R2's MAC address for the source and R1's MAC address for the destination.

    **Note** A zero sequence number indicates to R2 that it does not send an Ack frame to the Mapper when it has completed the Emit request. Hence, the Mapper completes the application's request immediately.

21. R2 sends a Probe using R2's MAC address for the source and R1's MAC address for the destination.

22. The application asks LLTD to perform another test from R1, and the Mapper sends a Charge frame to R1 to generate sufficient byte and frame credits in R1 for a request that will follow.

23. The Mapper sends a second Charge frame to R1 to generate sufficient byte and frame credits in R1 for a request that will follow.

24. The Mapper sends an Emit frame to R1 that requests R1 to send a Train frame using a Source MAC Address of 00-0D-3A-D7-F2-03 and R2's MAC address as the destination. The Mapper also sends a Probe frame using R1's MAC address as the Source and 00-0D-3A-D7-F2-03 as the Destination MAC Address.

25. R1 sends a Train frame using a Source MAC Address of 00-0D-3A-D7-F2-03 and R2's MAC address as the destination.

26. R1 sends a Probe using R1's MAC address for the Source and 00-0D-3A-D7-F2-03 as the Destination MAC Address.

27. R1 sends an Ack frame to the Mapper to indicate that it has completed the Emit request, and the Mapper completes the application's request from step 22.

28. The application asks LLTD to send a Query to R1 to get the list of MAC addresses that this responder has seen.

29. The application also asks LLTD to send a Query to R2 to get the list of MAC addresses that this responder has seen.

30. R1 sends a QueryResp to the Mapper with an entry that indicates that it saw a frame with R2's MAC address as the Source MAC address and R1's MAC address as the Destination MAC address. The Mapper completes the application's request from step 28.

31. R2 sends a QueryResp to the Mapper with an entry that indicates it saw a frame with R1's MAC address as the Source MAC Address and Destination MAC Address of 00-0D-3A-D7-F2-03. The Mapper completes the application's request from step 29.

32. The application finally directs LLTD to terminate the topology discovery session, and the Mapper broadcasts a Reset to indicate the mapping session is complete.

## 4.2 Example 2: Measuring Network Capacity

The following figure shows the layout of an example network that interconnects a media server and a TV with an integrated media player.

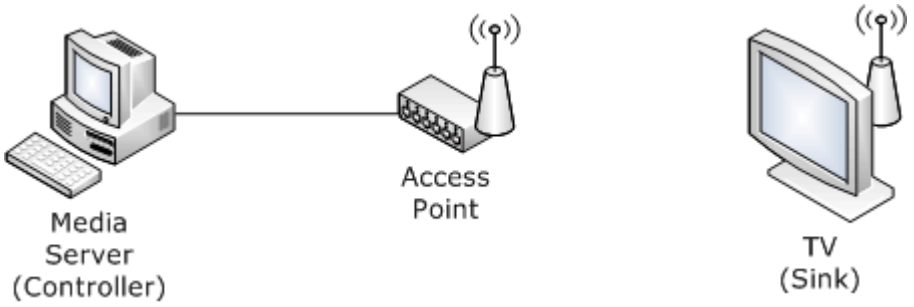



**Figure 7: Example media server and TV, integrated media player network**

The media server is used to stream media content to the TV. The QoS support in LLTD can be used to assess the capacity of the connection between the two endpoints to determine if adequate bandwidth is available for a requested stream. This example describes the LLTD QoS exchange for testing the bandwidth using a Test Type of Timed probes.

The following figure shows the protocol exchange between the media server and the TV. Communication between the controller and the sink is done using their real MAC addresses (no LLTD OUI–based MAC addresses) and the LLTD Ethertype.

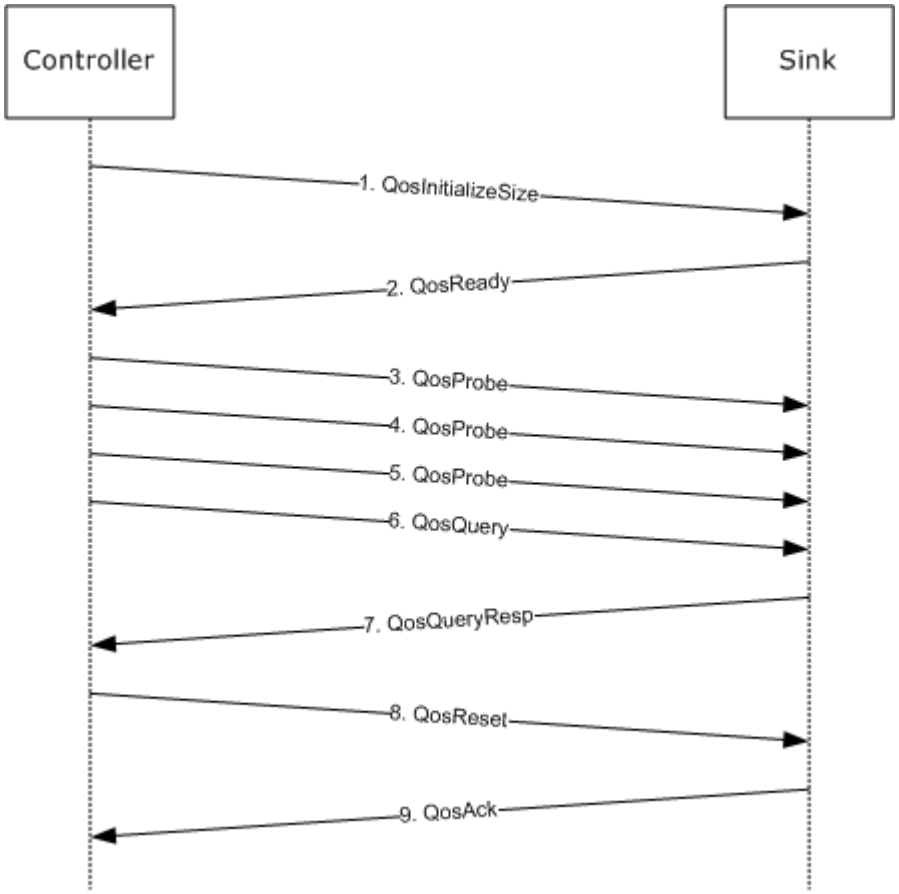



**Figure 8: Protocol exchange between media server and TV**

The following list describes each step in the protocol exchange:

1. The controller sends a QosInitializeSink header to the sink and indicates that the sink should use its existing interrupt moderation setting (Interrupt Mod is set to 0xFF).

2. The sink returns a QosReady header to confirm the creation of a network test session. The sink indicates that the Sink Link Speed is 54 Mbps (value of 540,000 or 0x83D60) and that its time-stamp counter has an accuracy of 1 microsecond (value 1,000,000 or 0xF4240).

3. The controller creates its first QosProbe frame, time stamps it, and then transmits it to the sink. The controller indicates in the QosProbe frame that the Test Type is a Timed probe. The **802.1p** field is indicated as not used. The sink time stamps this frame when it arrives and saves it for returning the header information to the controller when the controller requests it.

4. The controller immediately creates a second QosProbe frame, time stamps it, and transmits it to the sink using the same parameters as in step 3. The sink time stamps this frame when it arrives and saves it for returning the header information to the controller when the controller requests it.

5. The controller immediately creates a third QosProbe frame, time stamps it, and transmits it to the sink using the same parameters as in step 3. The sink time stamps this frame when it arrives and saves it for returning the header information to the controller when the controller requests it.

6. The controller sends a QosQuery to the sink to retrieve the header information from the QosProbe frames.

7. The sink sends a QosQueryResp to the controller and indicates that it has received three events. The QosProbe headers with both the controller and sink time stamps are included in the frame.

8. The controller sends a QosReset to the sink to indicate that it is done running QoS tests.

9. The sink confirms reception of the QosReset header with a QosAck header.

# 5 Security

The following sections specify security considerations for implementers of the Link Layer Topology Discovery (LLTD) Protocol.

## 5.1 Security Considerations for Implementers

While the LLTD Protocol performs no security checks, it includes measures (the RepeatBAND mechanism, as specified in section 3.5.6.2, and the Charge mechanism, as specified in section 3.6) to prevent traffic amplification that could be used in a DoS attack. The intent is that an attacker can do no more harm using the LLTD Protocol than the attacker could do by simply sending Ethernet frames in a non–LLTD environment.

## 5.2 Index of Security Parameters

None.

# 6 Appendix A: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows XP
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.2.1.1.13: Some wireless drivers do not expose the RSSI but do expose a signal strength indicator between 0 and 100. Windows XP only reports the RSSI if available and does not include the attribute if it is not. However, Windows Vista and Windows Server 2008 report the signal strength indicator as provided by the driver, when the driver does not provide the actual RSSI.

<2> Section 2.2.2.1: Windows only sends ICO format images as a responder, and only recognizes ICO format images as a mapper. For more information about ICO, see [MSDN-ICO].

<3> Section 2.2.2.3: For a Universal Plug and Play (UPnP) device, the required information comes from the UPnP device description phase that has the XML elements that Plug and Play Extension uses to derive the PnP hardware ID string. This property is the string that Plug and Play uses to match a device with an INF file on a Windows-based computer.

<4> Section 3: Windows Vista and Windows Server 2008 both support all eight roles. Windows XP only supports the mandatory ones.

<5> Section 3.1.6.1: Windows Vista and Windows Server 2008 stop the quick discovery process when the Seen Station List does not grow for three consecutive Block timer expirations.

<6> Section 3.2.6.1: Windows and Windows Server 2008 stop retrying communication with a responder after five consecutive Per-Responder Response timer expirations.

<7> Section 3.3.1: Windows Vista and Windows Server 2008 support up to ten network test sessions.

<8> Section 3.3.4.1: Windows Vista and Windows Server 2008 support transmission of arbitrary QosProbe frame size with randomized or zeroed content.

<9> Section 3.6.5.5: Windows sends a QueryLargeTlvResp with the **Length** field set to zero for all unrecognized TLV types.

<10> Section 3.7.6.1: Windows implements the period of inactivity at an aggressive five seconds.

<11> Section 3.8.1: Windows uses a byte scaling of 1 kilobyte and packet scaling of 1 packet to choose the most appropriate scaling factors.

<12> Section 3.8.5.2: Windows responds to all QosCounterSnapshot frames regardless of whether or not a lease period is in effect.

<13> Section 3.8.5.2: Windows always responds to all QosCounterSnapshot requests even if the **Real Destination Address** field in the Base header does not match the destination **MAC Address**

field in the Ethernet header. It responds with the counters for the network interface on which it received the QosCounterSnapshot.

# 7 Index

## I



## L



## M



## N



## O



## P



## Q

## R



## S

## T



## V



## W