Document Identifier: DSP0236

Date: 2014-12-03

Version: 1.2.1

Management Component Transport Protocol (MCTP) Base Specification

Includes MCTP Control Specifications

Document Type: Specification

Document Status: DMTF Standard

Document Language: en-US

CONTENTS

Figures

Tables

Foreword

The *Management Component Transport Protocol (MCTP) Base Specification* (DSP0236) was prepared by the PMCI Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Introduction

The Management Component Transport Protocol (MCTP) defines a communication model intended to facilitate communication between:

- Management controllers and other management controllers
- Management controllers and management devices

The communication model includes a message format, transport description, message exchange patterns, and configuration and initialization messages.

MCTP is designed so that it can potentially be used on many bus types. The protocol is intended to be used for intercommunication between elements of platform management subsystems used in computer systems, and is suitable for use in mobile, desktop, workstation, and server platforms. Management controllers such as a baseboard management controller (BMC) can use this protocol for communication between one another, as well as for accessing management devices within the platform.

Management controllers can use this protocol to send and receive MCTP-formatted messages across the different bus types that are used to access management devices and other management controllers. Management devices in a system need to provide an implementation of the message format to facilitate actions performed by management controllers.

It is intended that different types of devices in a management system may need to implement different portions of the complete capabilities defined by this protocol. Where relevant, this is called out in the individual requirements

Management Component Transport Protocol (MCTP) Base Specification

1 Scope

The *MCTP Base Specification* describes the command protocol, requirements, and use cases of a transport protocol for communication between discrete management controllers on a platform, as well as between management controllers and the devices they manage.

This document is intended to meet the following objectives:

- Describe the MCTP Base transport protocol
- Describe the MCTP control message protocol

The MCTP specifies a transport protocol format. This protocol is independent of the underlying physical bus properties, as well as the "data-link" layer messaging used on the bus. The physical and data-link layer methods for MCTP communication across a given medium are defined by companion "transport binding" specifications, such as DSP0238, MCTP over PCIe® Vendor Defined Messaging, and DSP0237, MCTP over SMBus/I^{2}C. This approach enables future transport bindings to be defined to support additional buses such as USB, RMII, and others, without affecting the base MCTP specification.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 Approved References

DMTF, DSP0239, *Management Component Transport Protocol (MCTP) IDs and Codes*
http://www.dmtf.org/standards/published_documents/DSP0239_1.2.0.pdf

DMTF DSP2016, Management Component Transport Protocol (MCTP) Overview White Paper
http://www.dmtf.org/standards/published_documents/DSP2016.pdf

DMTF DSP4014, *DMTF Process for Working Bodies 2.0*
http://dmtf.org/sites/default/files/standards/documents/DSP4014_2.0.pdf

2.2 Other References

DMTF DSP0237, Management Component Transport Protocol SMBus/I2C Transport Binding Specification
http://www.dmtf.org/standards/published_documents/DSP0237_1.0.0.pdf

DMTF DSP0238, Management Component Transport Protocol (MCTP) PCIe VDM Transport Binding Specification
http://www.dmtf.org/standards/published_documents/DSP0238_1.0.0.pdf

Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, *Advanced Configuration and Power Interface Specification v5.0*, ACPI, December 6, 2011
http://www.acpi.info/downloads/ACPIspec50.pdf

IETF, RFC20, *ASCII format for Network Interchange*, October 16, 1969
http://tools.ietf.org/html/rfc20

IETF, RFC2119, *Key Words for use in RFCs to Indicate Requirement Levels*, March 1997
http://datatracker.ietf.org/doc/rfc2119/

IETF, RFC4122, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005
http://datatracker.ietf.org/doc/rfc4122/

Intel, Hewlett-Packard, NEC, and Dell, *Intelligent Platform Management Interface Specification: Second Generation v2.0*, IPMI, 2004
http://www.intel.com/design/servers/ipmi

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*
http://isotc.iso.org/livelink/livelink?func=ll&objId=4230456&objAction=browse&sort=subtype

PCI-SIG, PCI Express™ Specifications
http://www.pcisig.com/specifications/pciexpress/

NXP Semiconductors, *UM10204 I2C-bus specification and user manual,* Rev. 5, October 9, 2012
http://www.nxp.com/documents/user_manual/UM10204.pdf

SMBus, *System Management Bus (SMBus) Specification v2.0*, SMBus, 2000
http://www.smbus.org/specs/smbus20.pdf

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1 Requirement Term Definitions

This clause defines key phrases and words that denote requirement levels in this specification. These definitions are consistent with the terms defined in RFC2119.

3.1.1
can
used for statements of possibility and capability, whether material, physical, or causal

3.1.2
cannot
used for statements of possibility and capability, whether material, physical or causal

3.1.3
conditional
indicates requirements to be followed strictly to conform to the document when the specified conditions are met

3.1.4
deprecated
indicates that an element or profile behavior has been outdated by newer constructs

3.1.5

mandatory

indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.1.6

may

indicates a course of action permissible within the limits of the document

NOTE: An implementation that does *not* include a particular option must be prepared to interoperate with another implementation that *does* include the option, although perhaps with reduced functionality. An implementation that *does* include a particular option must be prepared to interoperate with another implementation that does *not* include the option (except for the feature that the option provides).

3.1.7

may not

indicates flexibility of choice with no implied preference

3.1.8

must

indicates that the item is an absolute requirement of the specification

3.1.9

must not

indicates that the definition is an absolute prohibition of the specification

3.1.10

need not

indicates a course of action permissible within the limits of the document

3.1.11

not recommended

indicates that valid reasons may exist in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and carefully weighed before implementing any behavior described with this label

3.1.12

obsolete

indicates that an item was defined in prior specifications but has been removed from this specification

3.1.13

optional

indicates a course of action permissible within the limits of the document

3.1.14

recommended

indicates that valid reasons may exist in particular circumstances to ignore a particular item, but the full implications should be understood and carefully weighed before choosing a different course

3.1.15

required

indicates that the item is an absolute requirement of the specification

3.1.16

shall

indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.1.17

shall not

indicates requirements to be followed strictly to conform to the document and from which no deviation is permitted

3.1.18

should

indicates that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.1.19

should not

indicates that a certain possibility or course of action is deprecated but not prohibited

3.2 MCTP Term Definitions

For the purposes of this document, the following terms and definitions apply.

3.2.1

Address Resolution Protocol

ARP

refers to the procedure used to dynamically determine the addresses of devices on a shared communication medium

3.2.2

baseline transmission unit

the required common denominator size of a transmission unit for packet payloads that are carried in an MCTP packet. Baseline Transmission Unit-sized packets are guaranteed to be routable within an MCTP network.

3.2.3

baseboard management controller

BMC

a term coined by the IPMI specifications for the main management controller in an IPMI-based platform management subsystem. Also sometimes used as a generic name for a motherboard resident management controller that provides motherboard-specific hardware monitoring and control functions for the platform management subsystem.

3.2.4

binary-coded decimal

BCD

indicates a particular binary encoding for decimal numbers where each four bits (*nibble*) in a binary number is used to represent a single decimal digit, and with the least significant four bits of the binary number corresponding to the least significant decimal digit. The binary values `0000b` through `1001b` represent decimal values 0 through 9, respectively. For example, with BCD encoding a byte can represent a two-digit decimal number where the most significant nibble (bits 7:4) of the byte contains the encoding for the most significant decimal digit and the least significant nibble (bits 3:0) contains the encoding for the least significant decimal digit (for example, 0010_1001b in BCD encoding corresponds to the decimal number 29).

3.2.5

bridge

generically, the circuitry and logic that connects one computer bus or interconnect to another, allowing an agent on one to access the other. Within this document, the term *bridge* shall refer to MCTP bridge, unless otherwise indicated.

3.2.6

bus

a physical addressing domain shared between one or more platform components that share a common physical layer address space

3.2.7

bus owner

the party responsible for managing address assignments (can be logical or physical addresses) on a bus (for example, in MCTP, the bus owner is the party responsible for managing EID assignments for a given bus). A bus owner may also have additional media-specific responsibilities, such as assignment of physical addresses.

3.2.8

byte

an 8-bit quantity. Also referred to as an *octet*.

NOTE: PMCI specifications shall use the term *byte*, not *octet*.

3.2.9

endpoint

see MCTP endpoint

3.2.10

endpoint ID

EID

see MCTP endpoint ID

3.2.11

Globally Unique Identifier

GUID

see UUID

3.2.12
host interface

a hardware interface and associated protocols that is used by software running locally on the host processors to access the hardware of a management subsystem within a managed system.

3.2.13
Inter-Integrated Circuit

I^{2}C

a multi-master, two-wire, serial bus originally developed by Philips Semiconductor; now maintained by NXP Semiconductors,

3.2.14
intelligent management device

IMD

a management device that is typically implemented using a microcontroller and accessed through a messaging protocol. Management parameter access provided by an IMD is typically accomplished using an abstracted interface and data model rather than through direct "register level" accesses.

3.2.15
Intelligent Platform Management Bus

IPMB

name for the architecture, protocol, and implementation of an I^{2}C bus that provides a communications path between "management controllers" in IPMI -based systems

3.2.16
Intelligent Platform Management Interface

IPMI

a set of specifications defining interfaces and protocols originally developed for server platform management by the IPMI Promoters Group: Intel, Dell, HP, and NEC

3.2.17
managed entity

the physical or logical entity that is being managed through management parameters. Examples of *physical* entities include fans, processors, power supplies, circuit cards, chassis, and so on. Examples of *logical* entities include virtual processors, cooling domains, system security states, and so on.

3.2.18
Management Component Transport Protocol

MCTP

The protocol defined in this specification.

3.2.19
management controller

a microcontroller or processor that aggregates management parameters from one or more management devices and makes access to those parameters available to local or remote software, or to other management controllers, through one or more management data models. Management controllers may also interpret and process management-related data, and initiate management-related actions on management devices. While a native data model is defined for PMCI, it is designed to be capable of supporting other data models, such as CIM, IPMI, and vendor-specific data models. The microcontroller or processor that serves as a management controller can also incorporate the functions of a management device.

3.2.20

management device

any physical device that provides protocol terminus for accessing one or more management parameters. A management device responds to management requests, but does not initiate or aggregate management operations except in conjunction with a management controller (that is, it is a *satellite* device that is subsidiary to one or more management controllers). An example of a simple management device would be a temperature sensor chip. A management controller that has I/O pins or built-in analog-to-digital converters that monitor state and voltages for a managed entity would also be a management device.

3.2.21

management parameter

a particular datum representing a characteristic, capability, status, or control point associated with a managed entity. Example management parameters include temperature, speed, volts, on/off, link state, uncorrectable error count, device power state, and so on.

3.2.22

MCTP bridge

an MCTP endpoint that can route MCTP messages not destined for itself that it receives on one interconnect onto another without interpreting them. The ingress and egress media at the bridge may be either homogeneous or heterogeneous. Also referred to in this document as a "bridge".

3.2.23

MCTP bus owner

responsible for EID assignment for MCTP or translation on the buses that it is a master of. The MCTP bus owner may also be responsible for physical address assignment. For example, for SMBus/I2C bus segments, the MCTP bus owner is also the ARP master. This means the bus owner assigns dynamic SMBus/I2C addresses to those devices requiring it.

3.2.24

MCTP control command

commands defined under the MCTP *control* message type that are used for the initialization and management of MCTP communications (for example, commands to assign EIDs, discover device MCTP capabilities, and so on)

3.2.25

MCTP endpoint

an MCTP communication terminus. An MCTP endpoint is a terminus or origin of MCTP packets or messages. That is, the combined functionality within a physical device that communicates using the MCTP transport protocol and handles MCTP control commands. This includes MCTP-capable management controllers and management devices. Also referred to in this document as "endpoint".

3.2.26

MCTP endpoint ID

the logical address used to route MCTP messages to a specific MCTP endpoint. A numeric handle (logical address) that uniquely identifies a particular MCTP endpoint within a system for MCTP communication and message routing purposes. Endpoint IDs are unique among MCTP endpoints that comprise an MCTP communication network within a system. MCTP EIDs are only unique within a particular MCTP network. That is, they can be duplicated or overlap from one MCTP network to the next. Also referred to in this document as "endpoint ID" and abbreviated "EID".

3.2.27
MCTP host interface

a host interface that enables host software to locally access an MCTP Network in the managed system.

3.2.28
MCTP management controller

a management controller that is an MCTP endpoint. Unless otherwise indicated, the term "management controller" refers to an "MCTP management controller" in this document.

3.2.29
MCTP management device

a management device that is an MCTP endpoint. Unless otherwise indicated, the term "management device" refers to an "MCTP management device" in this document.

3.2.30
MCTP message

a unit of communication based on the message type that is relayed through the MCTP Network using one or more MCTP packets

3.2.31
MCTP network

a collection of MCTP endpoints that communicate using MCTP and share a common MCTP endpoint ID space

3.2.32
MCTP network ID

a unique identifier to distinguish each independent MCTP network within a platform

3.2.33
MCTP packet

the unit of data transfer used for MCTP communication on a given physical medium

3.2.34
MCTP packet payload

refers to the portion of the message body of an MCTP message that is carried in a single MCTP packet

3.2.35
message

see MCTP message

3.2.36
message assembly

the process of receiving and linking together two or more MCTP packets that belong to a given MCTP message to allow the entire message header and message data (payload) to be extracted

3.2.37

message body

the portion of an MCTP message that carries the message type field and any message type-specific data associated with the message. An MCTP message spans multiple MCTP packets when the message body needs is larger than what can fit in a single MCTP packet. Thus, the message body portion of an MCTP message can span multiple MCTP packets.

3.2.38

message disassembly

the process of taking an MCTP message where the message's header and data (payload) cannot be carried in a single MCTP packet and generating the sequence of two or more packets required to deliver that message content within the MCTP network

3.2.39

message originator

the original transmitter (source) of a message targeted to a particular message terminus

3.2.40

message terminus

the name for a triplet of fields called the MCTP Source Endpoint ID, Tag Owner bit value, and Message Tag value. Together, these fields identify the packets for an MCTP message within an MCTP network for the purpose of message assembly. The message terminus itself can be thought of as identifying a set of resources within the recipient endpoint that is handling the assembly of a particular message.

3.2.41

most significant byte

MSB

refers to the highest order byte in a number consisting of multiple bytes

3.2.42

nibble

the computer term for a four-bit aggregation, or half of a byte

3.2.43

packet

see MCTP packet

3.2.44

packet payload

see MCTP packet payload

3.2.45

pass-through traffic/message/packets

non-control packets passed between the external network and the management controller through the network controller

3.2.46

payload

refers to the information bearing fields of a message. This is separate from those fields and elements that are used to transport the message from one point to another, such as address fields, framing bits, checksums, and so on. In some instances, a given field may be both a payload field and a transport field.

3.2.47

physical transport binding

refers to specifications that define how the MCTP base protocol and MCTP control commands are implemented on a particular physical transport type and medium, such as SMBus/I^2C, PCI Express™ Vendor Defined Messaging, and so on.

3.2.48

Platform Management Component Intercommunications

PMCI

name for a working group under the Distributed Management Task Force's Pre-OS Workgroup that is chartered to define standardized communication protocols, low level data models, and transport definitions that support communications with and between management controllers and management devices that form a platform management subsystem within a managed computer system

3.2.49

point-to-point

refers to the case where only two physical communication devices are interconnected through a physical communication medium. The devices may be in a master/slave relationship, or could be peers.

3.2.50

Reduced Media Independent Interface

RMII

a reduced signal count MAC to PHY interface, based on the IEEE Media Independent Interface (MII), which was specified by the RMII Consortium (3Com Corporation; AMD Inc.; Bay Networks, Inc.; Broadcom Corp.; National Semiconductor Corp.; and Texas Instruments Inc.)

3.2.51

simple endpoint

an MCTP endpoint that is not associated with either the functions of an MCTP bus owner or an MCTP bridge

3.2.52

Transmission Unit

refers to the size of the portion of the MCTP packet payload, which is the portion of the message body carried in an MCTP packet

3.2.53

transport binding

see physical transport binding

3.2.54

Universally Unique Identifier

UUID

refers to an identifier originally standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE). UUIDs are created using a set of algorithms that enables them to be independently generated by different parties without requiring that the parties coordinate to ensure that generated IDs do not overlap. In this specification, RFC4122 is used as the base specification describing the format and generation of UUIDs. Also sometimes referred to as a globally unique identifier (GUID).

4 Symbols and Abbreviated Terms

The following symbols and abbreviations are used in this document.

4.1

ACPI

Advanced Configuration and Power Interface

4.2

ARP

Address Resolution Protocol

4.3

BCD

binary-coded decimal

4.4

BMC

baseboard management controller

4.5

CIM

Common Information Model

4.6

EID

endpoint identifier

4.7

FIFO

first-in first-out

4.8

GUID

Globally Unique Identifier

4.9

I^2C

Inter-Integrated Circuit

4.10

IANA

Internet Assigned Numbers Authority

4.11

IMD

intelligent management device

4.12

IP

Internet Protocol

4.13

IPMB

Intelligent platform management bus

4.14

IPMI

Intelligent platform management interface

4.15

ISO/IEC

International Organization for Standardization/International Engineering Consortium

4.16

KCS

Keyboard Controller Style

4.17

MCTP

Management Component Transport Protocol

4.18

MSB

most significant byte

4.19

PCIe

Peripheral Component Interconnect (PCI) Express

4.20

PMCI

Platform Management Component Intercommunications

4.21

RMII

Reduced Media Independent Interface

4.22

SMBus

System Management Bus

4.23

TCP/IP

Transmission Control Protocol/Internet Protocol

4.24

USB

Universal Serial Bus

4.25

UUID

Universally Unique Identifier

4.26

VDM

Vendor Defined Message

5 Conventions

The conventions described in the following clauses apply to this specification.

5.1 Byte Ordering

Unless otherwise specified, byte ordering of multi-byte numeric fields or bit fields is "Big Endian" (that is, the lower byte offset holds the most significant byte, and higher offsets hold lesser significant bytes).

5.2 Reserved Fields

Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other numeric ranges are reserved for future definition by the DMTF.

Unless otherwise specified, numeric or bit fields that are designated as reserved shall be written as 0 (zero) and ignored when read.

6 Management Component Relationships

Figure 1 illustrates the relationship between devices, management controllers, management devices, and managed entities, which are described in Clause 3.2.

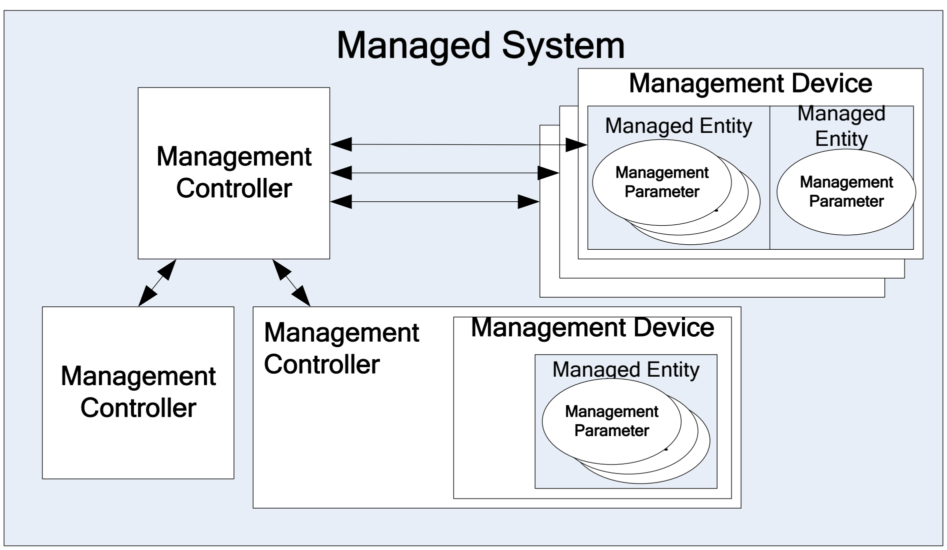


Figure 1 – Management Component Relationships

7 MCTP Overview

This clause provides an overview of the main elements of MCTP. Additional overview information is available in the MCTP white paper, DSP2016.

MCTP is a transport independent protocol that is used for intercommunication within an MCTP Network. An MCTP Network that consists of one of more physical transports that are used to transfer MCTP Packets between MCTP Endpoints. MCTP Transport Binding Specifications define how the MCTP protocol is implemented across a particular physical transport medium. For example, the DMTF has defined transport bindings for MCTP over SMBus/I^2C and MCTP over PCIe using PCIe Vendor Defined Messages (VDMs), and others.

An MCTP Endpoint is the terminus for MCTP communication. A physical device that supports MCTP may provide one or more MCTP Endpoints. Endpoints are addressed using a logical address called the Endpoint ID, or EID. EIDs in MCTP are analogous to IP Addresses in Internet Protocol networking. EIDs can be statically or dynamically allocated.

A system implementation can contain multiple MCTP Networks. Each MCTP Network has its own separate EID space. There is no coordination of EIDs between MCTP Networks. EIDs can overlap between MCTP Networks.

An MCTP Network may provide an MCTP Network ID that can be used to differentiate different MCTP Networks when more than one MCTP Network can be accessed by an entity such as system software. The Network ID is also used when an entity has more than one point of access to the MCTP Network. In this case, the MCTP Network ID enables the entity to tell whether the access points provide access to the same MCTP Network or to different MCTP Networks.

The DMTF MCTP specifications also include the definition of transport bindings for MCTP host interfaces. MCTP host interfaces are used by software that runs locally on the host processors of the managed system to access an MCTP Network.

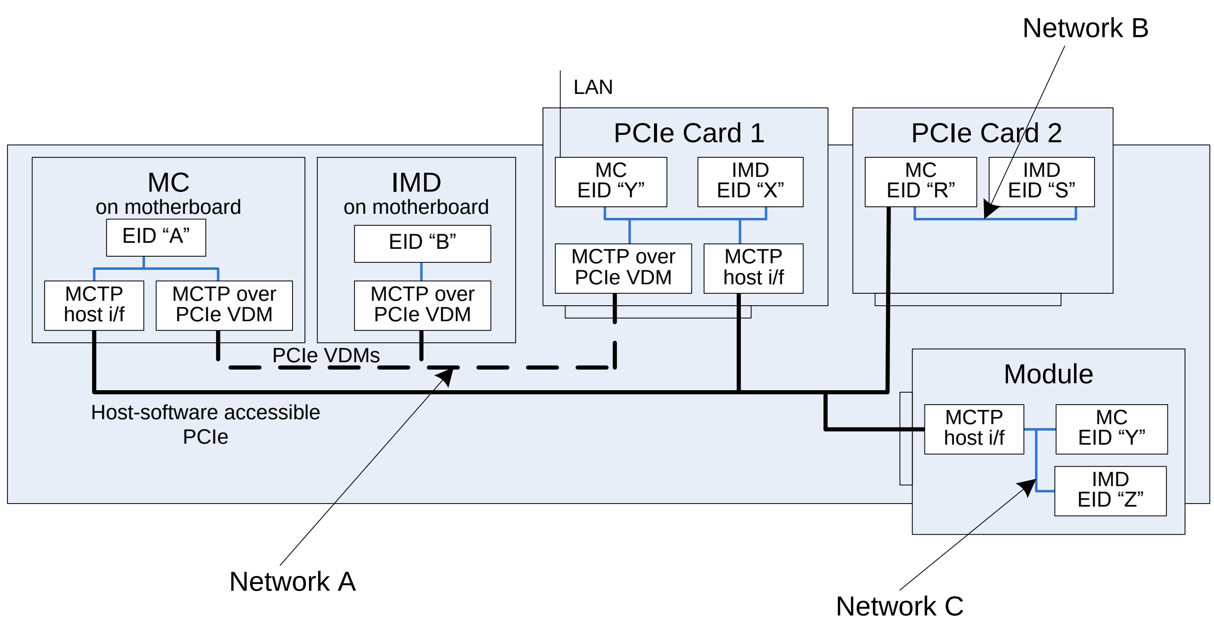


Figure 2 – MCTP Networks

Figure 2 shows the different ways MCTP Networks can exist in a system. In this example, Network A connects a Management Controller (MC) and Intelligent Management Device (IMD) on a motherboard with devices on PCIe Card 1 using MCTP over PCIe Vendor Defined Messages. Note that there are two host interfaces (host i/f) on standard PCIe (host software accessible) that can be used by host software to access this particular network. This network thus requires an MCTP Network ID so that the host software can tell that the two host interfaces connect to the same MCTP Network.

Network B represents a network that is solely used for interconnecting devices within PCIe Card 2. This MCTP Network would typically not require an MCTP Network ID since it is not visible to host software or any other entity that would needs to differentiate Network B from another MCTP Network in the system.

Network C represents an MCTP Network on an add-in module. This network is separate from networks A and B but can accessed by host software through PCIe. Thus, this network requires a Network ID so that host software can differentiate that Network C is a different network than Network A.

MCTP Messages are comprised of one or more MCTP Packets. MCTP defines fields that support the assembly of received MCTP Packets into MCTP Messages and the disassembly of MCTP Messages into packets for transmission.

MCTP is designed to be able to transfer multiple Message Types in an interleaved manner using the same protocol. MCTP Message Types identified using a Message Type number. The use of the message type number is similar to a well-known port number in Internet Protocol. It identifies MCTP Messages that are all associated with a particular specification. This specification defines a Message Type for MCTP Control Messages that are used to initialize and maintain the MCTP Network. The DMTF has also defined Message Types for use by the PMCI (Platform Management Communications Interconnect) specifications, Vendor-specific Messaging over MCTP, and so on. MCTP Message Type number assignments are provided in DSP0239. DSP0239 will be updated as new messages types are defined in the future.

MCTP Control Messages use a request/response protocol. It is important to note that the base transport protocol defined by MCTP just defines a protocol for the transport of MCTP messages. Whether the message content is a request, a response, or something else is part of the particular Message Type definition.

In MCTP, a Bus is defined as a physical medium that shares a single physical address space. MCTP includes the definition of a function called the MCTP Bus Owner. The Bus Owner provides two main functions: It distributes EIDs to Endpoints when the MCTP implementation uses EIDs that are dynamically allocated, and it provides the way for an Endpoint to resolve an EID into the physical address used that is required to deliver a message to the target Endpoint.

Busses can be interconnected within an MCTP Network using MCTP Bridges to forward MCTP packets between busses. Bridges also handle the task of managing the difference in moving packets from one type of physical media to another, such as moving an MCTP packet between SMBus/I2C and PCIe Vendor Defined Messaging.

The following example illustrates how MCTP can be used within a hypothetical platform management subsystem implementation. More complex topologies, with multi-levels of bridges and greater numbers of busses and devices can be readily supported by MCTP as required.

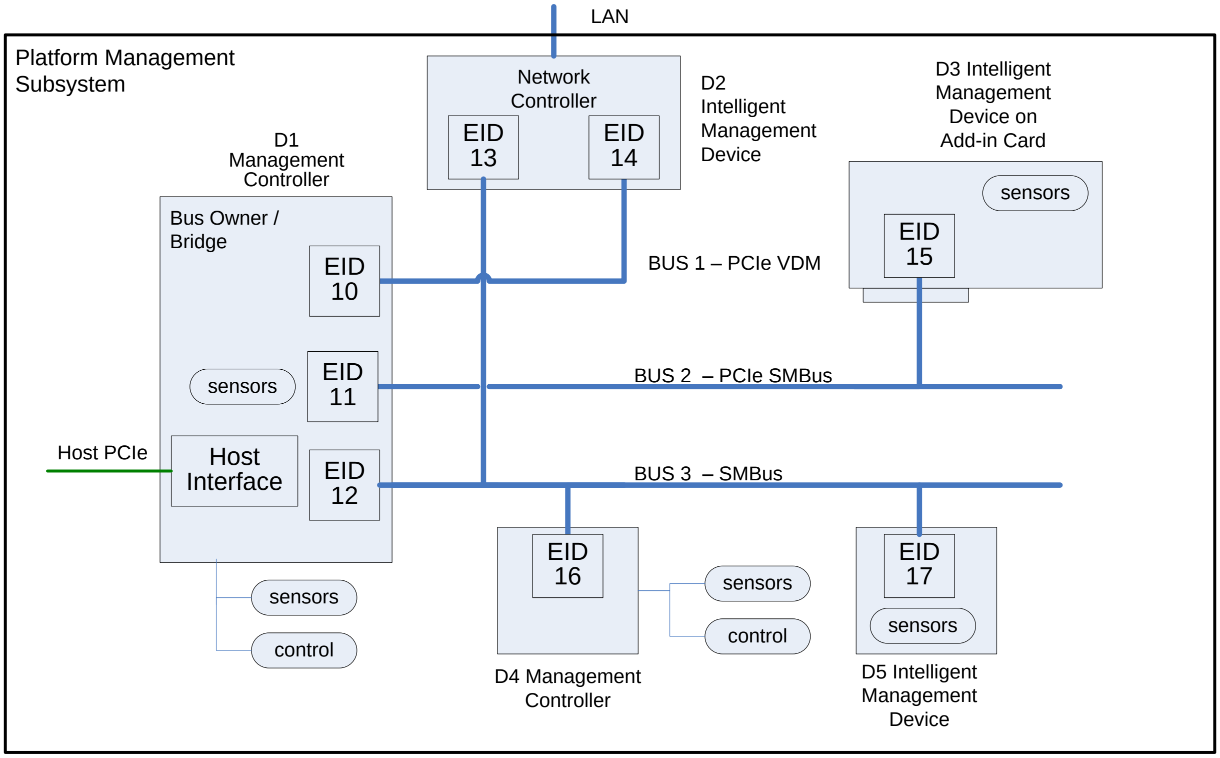


Figure 3 – MCTP Topology

8 MCTP Base Protocol

The MCTP base protocol defines the common fields for MCTP packets and messages and their usage.

Though there are medium-specific packet header fields and trailer fields, the fields for the base protocol are common for all media. These common fields support the routing and transport of messages between MCTP endpoints and the assembly and disassembly of large messages from and into multiple MCTP packets, respectively. The base protocol's common fields include a message type field that identifies what particular higher layer class of message is being carried using the MCTP base protocol.

8.1 MCTP Packet Fields

Figure 4 shows the fields that constitute a generic MCTP packet.

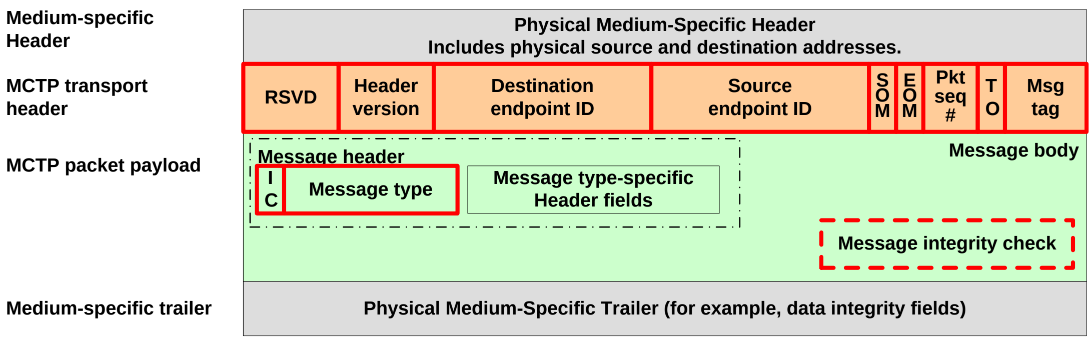


Figure 4 – Generic Message Fields

Table 1 defines the base protocol common fields.

Table 1 – MCTP Base Protocol Common Fields

Field Name	Field Size	Description
Medium-specific header	see description	This field represents the physical addressing and framing information that is used for transferring MCTP packets between devices on a particular physical medium. The size and type of any sub-fields or data within this field are defined by the corresponding transport binding specification for MCTP messaging on a given medium (for example, MCTP over SMBus/I2C, MCTP over PCIe Vendor Defined Messaging, and so on).
Medium-specific trailer	see description	This field represents any additional medium-specific trailer fields (if any) that are required for transferring MCTP packets between devices on a particular physical medium. A typical use of this field would be to hold per-packet data integrity fields (for example CRC, checksum, and so on) that would be specified for the particular medium.
MCTP transport header	32 bits	The MCTP transport header is part of each MCTP packet and provides version and addressing information for the packet as well as flags and a "Message Tag" field that, in conjunction with the source EID, is used to identify packets that constitute an MCTP message. The MCTP transport header fields are common fields that are always present regardless of the physical medium over which MCTP is being used. Note: The positioning of the sub-fields of the MCTP transport header may vary based on the physical medium binding.

Field Name	Field Size	Description
RSVD	4 bits	(Reserved) Reserved for future definition by the MCTP base specification.
Hdr version	4 bits	(Header version) Identifies the format, physical framing, and data integrity mechanism used to transfer the MCTP common fields in messages on a given physical medium. The value is defined in the specifications for the particular medium. Note: The value in this field can vary between different transport bindings.
Destination endpoint ID	8 bits	The EID for the endpoint to receive the MCTP packet. A few EID values are reserved for specific routing. See Table 2 – Special Endpoint IDs.
Source endpoint ID	8 bits	The EID of the originator of the MCTP packet. See Table 2 – Special Endpoint IDs.
SOM	1 bit	(Start Of Message) Set to `1b` if this packet is the first packet of a message.
EOM	1 bit	(End Of Message) Set to `1b` if this packet is the last packet of a message.
Pkt Seq #	2 bits	(Packet sequence number) For messages that span multiple packets, the packet sequence number increments modulo 4 on each successive packet. This allows the receiver to detect up to three successive missing packets between the start and end of a message. Though the packet sequence number can be any value (0-3) if the SOM bit is set, it is recommended that it is an increment modulo 4 from the prior packet with an EOM bit set. After the SOM packet, the packet sequence number must increment modulo 4 for each subsequent packet belonging to a given message up through the packet containing the EOM flag.
TO	1 bit	The TO (Tag Owner) bit identifies whether the message tag was originated by the endpoint that is the source of the message or by the endpoint that is the destination of the message. The Message Tag field is generated and tracked independently for each value of the Tag Owner bit. MCTP message types may overlay this bit with additional meaning, for example using it to differentiate between "request" messages and "response" messages. Set to `1b` to indicate that the source of the message originated the message tag.
Msg tag	3 bits	(Message tag) Field that, along with the Source Endpoint IDs and the Tag Owner (TO) field, identifies a unique message at the MCTP transport level. Whether other elements, such as portions of the MCTP Message Data field, are also used for uniquely identifying instances or tracking retries of a message is dependent on the message type. A source endpoint is allowed to interleave packets from multiple messages to the same destination endpoint concurrently, provided that each of the messages has a unique message tag. For messages that are split up into multiple packets, the Tag Owner (TO) and Message Tag bits remain the same for all packets from the SOM through the EOM.
Message body	See description	The message body represents the payload of an MCTP message. The message body can span multiple MCTP packets.
IC	1 bit	(MCTP integrity check bit) Indicates whether the MCTP message is covered by an overall MCTP message payload integrity check. This field is required to be the most significant bit of the first byte of the message body in the first packet of a message along with the message type bits. `0b` = No MCTP message integrity check `1b` = MCTP message integrity check is present

Field Name	Field Size	Description
Message type	7 bits	Defines the type of payload contained in the message data portion of the MCTP message. This field is required to be contained in the least-significant bits of the first byte of the message body in the first packet of a message. Like the fields in the MCTP transport header, the message type field is one of the common MCTP fields that are present independent of the transport over which MCTP is being used. Unlike the MCTP transport header, however, the message type field is only required to be present in the first packet of a particular MCTP message, whereas the MCTP transport header fields are present in every MCTP packet. See DSP0239 and Table 3 for information on message type values.
Message header	0 to M bytes	Additional header information associated with a particular message type, if any. This will typically only be contained in the first packet of a message, but a given message type definition can define header fields as required for any packet.
Message data	0 to N bytes	Data associated with the particular message type. Defined according to the specifications for the message type.
MCTP packet payload	See description	The packet payload is the portion of the message body that is carried in a given MCTP packet. The packet payload is limited according to the rules governing packet payload and transfer unit sizes. See 8.3, Packet Payload and Transmission Unit Sizes, for more information.
Msg integrity check	Message type-specific	(MCTP message integrity check) This field represents the optional presence of a message type-specific integrity check over the contents of the message body. If present, the Message integrity check field must be carried in the last bytes of the message body. The particular message type definition will specify whether this is required, optional, or not to be used, the field size, and what algorithm is to be used to generate the field. The MCTP base protocol also does not specify whether this field is required on single packet messages (potentially dependent on transmission unit size) or is only required on multiple packet messages. Use of the Msg integrity check field is specific to the particular message type specification.

773 ## 8.2 Special Endpoint IDs

774 Table 2 lists EID values that are reserved or assigned to specific functions for MCTP.

775 **Table 2 – Special Endpoint IDs**

Value	Description
Destination endpoint ID 0	**Null Destination EID**. This value indicates that the destination EID value is to be ignored and that only physical addressing is used to route the message to the destination on the given bus. This enables communication with devices that have not been assigned an EID. Because the physical addresses between buses are not guaranteed to be unique, MCTP does not support bridging messages with a null destination EID between different buses.
Source endpoint ID 0	**Null Source EID**. This value indicates a message is coming from an endpoint that is using physical addressing only. This would typically be used for messages that are delivered from an endpoint that has not been assigned an EID. Because the physical addresses between buses are not guaranteed to be unique, MCTP does not support bridging messages with a null source EID between different buses.
Endpoint IDs 1 through 7	Reserved for future definition.

Value	Description
Endpoint ID `0xFF`	**Broadcast EID**. Reserved for use as a broadcast EID on a given bus. MCTP network-wide broadcasts are not supported. Primarily for use by the MCTP control message type.
All other values	Available for assignment and allocation to endpoints.

8.3 Packet Payload and Transmission Unit Sizes

For MCTP, the size of a transmission unit is defined as the size of the packet payload that is carried in an MCTP packet.

8.3.1 Baseline Transmission Unit

The following are key information points regarding baseline transmission unit:

- The baseline transmission unit (minimum transmission unit) size for MCTP is 64 bytes.
- A message terminus that supports MCTP control messages must always accept valid packets that have a transmission unit equal to or less than the baseline transmission unit. The message terminus is also allowed to support larger transmission units.
- The transmission unit of all packets in a given message must be the same size, except for the transmission unit in the last packet (packet with EOM bit = `1b`). Except for the last packet, this size must be at least the baseline transmission unit size.
- The size of the transmission unit in the last packet must be less than or equal to the transmission unit size used for the other packets (if any).
- If a transmission unit size larger than the baseline transmission unit is negotiated, the transmission unit of all packets must be less than or equal to the negotiated transmission unit size. (The negotiation mechanism for larger transmission units between endpoints is message type-specific and is not addressed in this specification.)
- A given endpoint may negotiate additional restrictions on packet sizes for communication with another endpoint, as long as the requirements of this clause are met.
- All message types must include support for being delivered using packets that have a transmission unit that is no larger than the baseline transmission unit. This is required to support bridging those messages in implementations where there are MCTP bridges that only support the baseline transmission unit.

8.4 Maximum Message Body Sizes

The Message Body can span multiple packets. Limitations on message body sizes are message type-specific and are documented in the specifications for each message type.

8.5 Message Assembly

The following fields (and *only* these fields) are collectively used to identify the packets that belong to a given message for the purpose of message assembly on a particular destination endpoint.

- Msg Tag (Message Tag)
- TO (Tag Owner)
- Source Endpoint ID

As described in 3.2, together these values identify the message terminus on the destination endpoint. For a given message terminus, only one message assembly is allowed to be in process at a time.

8.6 Dropped Packets

Individual packets are dropped (silently discarded) by an endpoint under the following conditions. These packets are discarded before being checked for acceptance or rejection for message assembly. Therefore, these packets will *not* cause a message assembly to be started or terminated.

- **Unexpected "middle" packet or "end" packet**

  A "middle" packet (SOM flag = 0 and EOM flag = 0) or "end" packet (SOM flag = 0 and EOM flag = 1) for a multiple-packet message is received for a given message terminus without first having received a corresponding "start" packet (where the "start" packet has SOM flag = 1 and EOM flag = 0) for the message.

- **Bad packet data integrity or other physical layer error**

  A packet is dropped at the physical data-link layer because a data integrity check on the packet at that layer was invalid. Other possible physical layer errors may include framing errors, byte alignment errors, packet sizes that do not meet the physical layer requirements, and so on.

- **Bad, unexpected, or expired message tag**

  A message with TO bit = 0 was received, indicating that the destination endpoint was the originator of the tag value, but the destination endpoint did not originate that value, or is no longer expecting it. (MCTP bridges do not check message tag or TO bit values for messages that are not addressed to the bridge's EID, or to the bridge's physical address if null-source or destination-EID physical addressing is used.)

- **Unknown destination EID**

  A packet is received at the physical address of the device, but the destination EID does not match the EID for the device or the EID is un-routable.

- **Un-routable EID**

  An MCTP bridge receives an EID that the bridge is not able to route (for example, because the bridge did not have a routing table entry for the given endpoint).

- **Bad header version**

  The MCTP header version (Hdr Version) value is not a value that the endpoint supports.

- **Unsupported transmission unit**

  The transmission unit size is not supported by the endpoint that is receiving the packet.

8.7 Starting Message Assembly

Multiple-packet message assembly begins when the endpoint corresponding to the destination EID in the packet receives a valid "start" packet (packet with SOM = `1b` and EOM = `0b`).

A packet with both SOM = `1b` and EOM = `1b` is considered to be a single-packet message, and is not assembled per se.

Both multiple- and single-packet messages are subject to being terminated or dropped based on conditions listed in the following clause.

8.8 Terminating Message Assembly/Dropped Messages

Message assembly is terminated at the destination endpoint and messages are accepted or dropped under the following conditions:

- **Receipt of the "end" packet for the given message**

  Receiving an "end" packet (packet with EOM = `1b`) for a message that is in the process of being assembled on a given message terminus will cause the message assembly to be completed (provided that the message has not been terminated for any of the reasons listed below). This is normal termination. The message is considered to be accepted at the MCTP base protocol level.

- **Receipt of a new "start" packet**

  Receiving a new "start" packet (packet with SOM = `1b`) for a message to the same message terminus as a message assembly already in progress will cause the message assembly in process to be terminated. All data for the message assembly that was in progress is dropped. The newly received start packet is not dropped, but instead it begins a new message assembly. This is considered an error condition.

- **Timeout waiting for a packet**

  Too much time occurred between packets of a given multiple-packet message. The timeout interval is specific to a particular medium. All data for the message assembly that was in progress are dropped. This is considered an error condition.

- **Out-of-sequence packet sequence number**

  For packets comprising a given multiple-packet message, the packet sequence number for the most recently received packet is not a mod 4 increment of the previously received packet's sequence number. All data for the message assembly that was in progress is dropped. This is considered an error condition.

- **Incorrect transmission unit**

  An implementation may terminate message assembly if it receives a "middle" packet (SOM = `0b` and EOM = `0b`) where the MCTP packet payload size does not match the MCTP packet payload size for the start packet (SOM = `1b` and EOM bit = `0b`). This is considered an error condition.

- **Bad message integrity check**

  For single- or multiple-packet messages that use a message integrity check, a mismatch with the message integrity check value can cause the message assembly to be terminated and the entire message to be dropped, unless it is overridden by the specification for a particular message type.

  NOTE: The message integrity check is considered to be at the message-type level error condition rather than an error at the MCTP base protocol level.

8.9 Dropped Messages

An endpoint may drop a message if the message type is not supported by the endpoint. This can happen in any one of the following ways:

- The endpoint can elect to not start message assembly upon detecting the invalid message type in the first packet.
- The endpoint can elect to terminate message assembly in process.
- The endpoint can elect to drop the message after it has been assembled.

8.10 MCTP Versioning and Message Type Support

There are three types of versioning information that can be retrieved using MCTP control messages:

- MCTP base specification version information
- MCTP packet header version information
- Message type version information

The version of the MCTP base specification that is supported by a given endpoint is obtained through the Get MCTP Version Support command. This command can also be used to discover whether a particular message type is supported on an endpoint, and if so, what versions of that message type are supported.

The Header Version field in MCTP packets identifies the media-specific formatting used for MCTP packets. It can also indicate a level of current and backward compatibility with versions of the base specification, as specified by the header version definition in each medium-specific transport binding specification.

8.10.1 Compatibility with Future Versions of MCTP

An Endpoint may choose to support only certain versions of MCTP. The command structure along with the Get MCTP Version Support command allows endpoints to detect and restrict the versions of MCTP used by other communication endpoints. To support this, all endpoints on a given medium are required to implement MCTP Version 1.0 control commands for initialization and version support discovery.

8.11 MCTP Message Types

Table 3 defines the values for the Message Type field for different message types transported through MCTP. The MCTP control message type is specified within this document. Baseline requirements for the Vendor Defined – PCI and Vendor Defined – IANA message types are also specified within this document. All other message types are specified in the DSP0239 companion document to this specification.

NOTE: A device that supports a given message type may not support that message type equally across all buses that connect to the device.

Table 3 – MCTP Message Types Used in this Specification

Message Type	Message Type Code	Description
MCTP control	`0x00`	Messages used to support initialization and configuration of MCTP communication within an MCTP network. The messages and functions for this message type are defined within this specification.
Vendor Defined – PCI	`0x7E`	Message type used to support VDMs where the vendor is identified using a PCI-based vendor ID. The specification of the initial message header bytes for this message type is provided within this specification. Otherwise, the message body content is specified by the vendor, company, or organization identified by the given vendor ID.
Vendor Defined – IANA	`0x7F`	Message type used to support VDMs where the vendor is identified using an IANA-based vendor ID. (This format uses an "enterprise number" that is assigned and maintained by the Internet Assigned Numbers Authority, www.iana.org, as the means of identifying a particular vendor, company, or organization.) The specification of the initial message header bytes for this message type is provided within this specification. Otherwise, the message body content is specified by the vendor, company, or organization identified by the given vendor ID.

8.12 Security

The basic premise of MCTP is that higher layer protocols will fulfill security requirements (for example, confidentiality and authentication) for communication of management data. This means that the data models carried by MCTP must fulfill the security requirements of a given management transaction. The MCTP protocol itself will not define any additional security mechanisms.

8.13 Limitations

MCTP has been optimized for communications that occur within a single computer system platform. It has not been designed to handle problems that can typically occur in a more generic inter-system networking environment. In particular, compared to networking protocols such as IP and TCP/IP, MCTP has the following limitations:

- MCTP has limited logical addressing. MCTP been optimized for the small number of endpoints that are expected to be utilized within the platform. The 8-bit range of EIDs is limited compared to the ranges available for IP addresses.
- MCTP assumes an MCTP network implementation that does not include loops. There is no mechanism defined in MCTP to detect or reconcile implementations that have connections that form routing loops.

- MCTP assumes a network topology where all packets belonging to a given message will be delivered through the same route (that is, MCTP does not generally support some packets for a message arriving by one route, while other packets for the message arrive by a different route).
- MCTP does not support out-of-order packets for message assembly.
- The MCTP base protocol does not address flow control or congestion control. These behaviors, if required, are specified at the physical transport binding level or at the message type or higher level.
- MCTP is not specified to handle duplicate packets at the base protocol message assembly level. If a duplicate packet is received and passed on to MCTP message assembly, it can cause the entire message assembly to be terminated.

  NOTE: Transport bindings are not precluded from including mechanisms for handling duplicate packets at the physical transport level.

8.14 MCTP Discovery and Addressing

This clause describes how MCTP endpoints and their capabilities are discovered by one another, and how MCTP endpoints are provisioned with the addresses necessary for MCTP communication.

MCTP discovery occurs over the course of several discrete, ordered steps:

1) Bus enumeration
2) Bus address assignment
3) MCTP capability discovery
4) Endpoint ID assignment
5) Distribution and use of routing information

This clause gives an overview of the methods used for accomplishing each of these steps in various operational scenarios. Clause 11 gives details on the messages used to implement these operations.

8.14.1 Bus Enumeration

This step represents existing bus enumeration. (The actions taken in this step are specific to a given medium.) Because enumeration of devices on the physical bus is medium-specific, this information is provided in the transport binding specification for the medium.

8.14.2 Bus Address Assignment

MCTP endpoints require a bus address that is unique to a given bus segment. This step deals with assignment of these addresses. Some bus types (such as PCIe) have built-in mechanisms to effectively deal with this. Others (such as SMBus/I2C) require some additional consideration. Because bus address assignment is medium-specific, this information is provided in the transport binding specification for the medium.

8.14.3 MCTP Capability Discovery

Capability discovery deals with the discovery of the characteristics of individual MCTP endpoints. Capabilities that can be discovered include what message types are supported by an endpoint and what message type versions are supported. See 8.10 for a description of the methods used to accomplish capability discovery.

8.14.4 Endpoint ID Assignment

Endpoint IDs are system-wide unique IDs for identifying a specific MCTP endpoint. They can be dynamically assigned at system startup or hot-plug insertion. See 8.17 for a description of the methods used to accomplish EID assignment.

8.14.5 Distribution and Use of Routing Information

Bridging-capable MCTP endpoints need routing information to identify the next hop to forward a message to its final destination. See Clause 9 for a description of how routing information is conveyed between MCTP endpoints.

8.15 Devices with Multiple Media Interfaces

MCTP fully supports management controllers or management devices that have interfaces on more than one type of bus. For example, a device could have both a PCI Express (PCIe) and an SMBus/I2C interface. In this scenario, the device will typically have a different EID for each interface. (Bridges can include instantiations that have an endpoint shared across multiple interfaces; see 9.1.2 for more information.)

This concept can be useful in different operational scenarios of the managed system. For example, typically a PCIe interface will be used during ACPI "S0" power states (when the system is fully powered up), which will provide significantly higher bandwidths, whereas the SMBus/I2C interface could be used for "S3–S5" low-power sleep states.

The baseline transmission unit is specified to be common across all media, enabling packets to be routed between different media without requiring bridges to do intermediate assembly and disassembly operations to handle differences in packet payload sizes between different media.

Devices that support multiple media interfaces shall meet the command requirements of this specification and the associated transport binding specification for each enabled interface. For a given message type, the device should implement the same message type –specific commands on all MCTP interfaces, regardless of the medium, unless otherwise specified by the message type specification.

8.16 Peer Transactions

Endpoints can intercommunicate in a peer-to-peer manner using the physical addressing on a given bus.

A special value for the EID is used in cases when the physical address is known, but the EID is not known. This capability is used primarily to support device discovery and EID assignment. A device that does not yet have an EID assignment is not addressed using an EID. Rather, the device gets its EID assigned using an MCTP control command, Set Endpoint ID, which uses physical addressing only.

Similarly, depending on the transport binding, a device can also announce its presence by sending an MCTP message to a well-known physical address for the bus owner (for example, for PCIe VDM, this would be the root complex; for SMBus/I2C, the host slave address, and so on).

It is important to note that in cases where two endpoints are on the same bus, they do not need to go through a bridge to communicate with each other. Devices use the Resolve Endpoint ID command to ask the bus owner what physical address should be used to route messages to a given EID. Depending on the bus implementation, the bus owner can either return the physical address of the bridge that the message should be delivered to, or it can return the physical address of the peer on the bus.

8.17 Endpoint ID Assignment and Endpoint ID Pools

MCTP EIDs are the system-wide unique IDs used by the MCTP infrastructure to address endpoints and for routing messages across multiple buses in the system. There is one EID assigned to a given physical address. Most intelligent management devices (IMDs) or management controllers will connect to just a single bus and have a single EID. A non-bridge device that is connected to multiple different buses will have one EID for each bus it is attached to.

Bus owners are MCTP devices that are responsible for issuing EIDs to devices on a bus segment. These EIDs come from a pool of EIDs maintained by the bus owner.

With the exception of the topmost bus owner (see 8.17.1), a given bus owner's pool of EIDs is dynamically allocated at run-time by the bus owner of the bus above it in the hierarchy. Hot-plug devices must have their EID pools dynamically allocated.

Once EIDs are assigned to MCTP endpoints, it is necessary for MCTP devices involved in a transaction to understand something about the route a given message will traverse. Clause 9 describes how this routing information is shared among participants along a message's route.

8.17.1 Topmost Bus Owner

The topmost bus owner is the ultimate source of the EID pool from which all EIDs are drawn for a given MCTP network.

This is illustrated in Figure 5, in which the arrows are used to identify the role of bus ownership. The arrows point outward from the bus owner for the particular bus and inward to a device that is "owned" on the bus.

In Figure 5, device X in diagram A and bridge X in diagram B are examples of topmost bus owners. Diagram A shows a device that connects to a single bus and is the topmost bus owner for the overall MCTP network. Diagram B shows that a bridge can simultaneously be the topmost bus owner, as well as the bus owner for more than one bus. The different colors represent examples of different media.

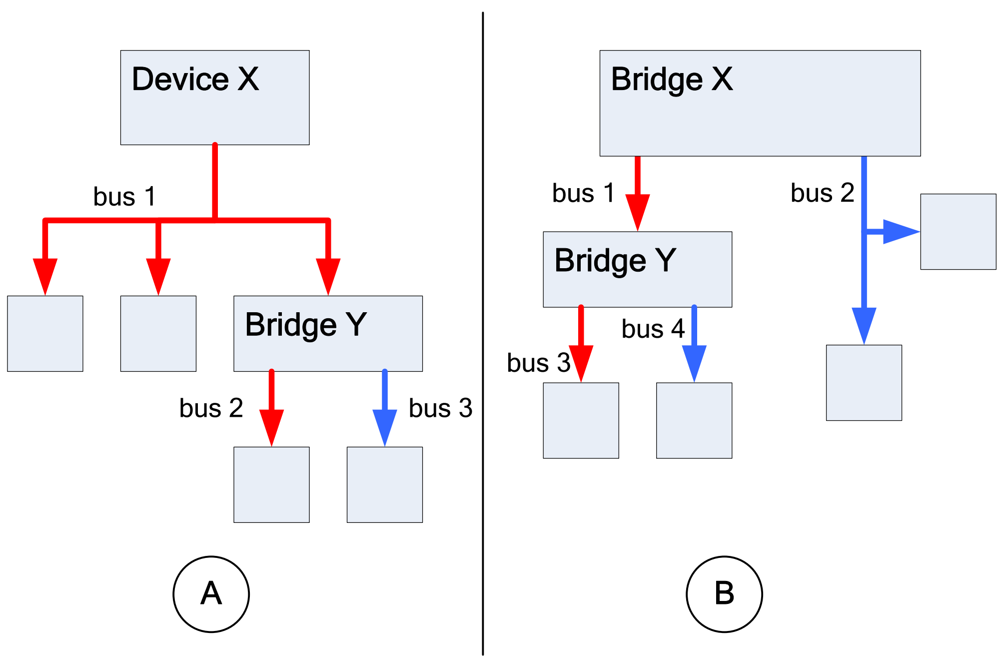


Figure 5 – Topmost Bus Owners

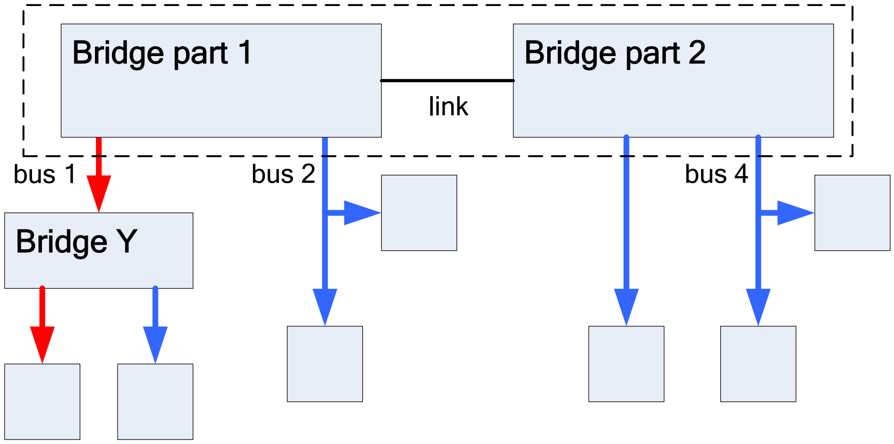


Figure 6 – Split Bridge

An implementation may need to split a bus owner or bridge across two physical devices. Such an implementation must include a mechanism (for example, a link as shown in Figure 6) that enables the two parts to share a common routing table, or have individual copies of the routing table that are kept synchronized. The definition of this mechanism is outside the scope of this specification.

8.17.2 Use of Static EIDs and Static EID Pools

In general, the only device that will require a static (pre-configured) EID assignment will be the topmost bus owner. It needs a static EID because there is no other party to assign it an EID through MCTP. Otherwise, all other devices will have their EIDs assigned to them by a bus owner.

The same principle applies if the device functions as an MCTP bridge. If the device is the highest device in the MCTP bus hierarchy, it will require a static pool of EIDs to be assigned to it as part of the system design. Otherwise, the device will be dynamically allocated a pool of EIDs from a higher bus owner.

An MCTP network implementation is allowed to use static EIDs for devices other than the topmost bus owner. Typically, this would only be done for very simple MCTP networks. Other key EID assignment considerations follow:

- Endpoints that support the option of being configured for one or more static EIDs must also support being configured to be dynamically assigned EIDs.
- No mechanism is defined in the MCTP base specification for a bridge or bus owner to discover and incorporate a static EID into its routing information. Thus, a simple endpoint that is configured with a static EID must also be used with a bus owner that is configured to support the static EIDs for the endpoint.
- All bus owners/bridges in the hierarchy, from the topmost bus owner to the endpoint, must have their routing configurable to support static EID routing information.
- Although an endpoint that uses a static EID must be used with a bus owner that supports static EIDs, the reverse is not true. A bus owner that uses static EIDs does not need to require that the devices on the buses it owns be configured with static EIDs.
- How the configuration of static EIDs occurs is outside the scope of this specification.

- No specified mechanism exists to "force" an override of a bridge’s or bus owner’s routing table entries for static EIDs. That is, commands such as Allocate Endpoint IDs and Routing Information Update only affect entries that are associated with dynamic EIDs.
- MCTP does not define a mechanism for keeping routing tables updated if static EIDs are used with dynamic physical addresses. That is, static EIDs are not supported for use with dynamic physical addresses.
- Bridges can have a mix of both static and dynamic EID pools. That is, the routing table can have both static and dynamic entries and can allocate from static and dynamic EID pools. Only the dynamic EID pool is given to the bridge by the bus owner using the Allocate Endpoint IDs command. There is no specification for how a static EID pool gets configured or how a bridge decides whether to give an endpoint an EID from a static or dynamically obtained EID pool. There is also no MCTP-defined mechanism to read the static EID pool setting from the bridge.
- MCTP bridges and bus owners (except the topmost bus owner) are not required to include support for static EIDs.
- MCTP does not define a mechanism for allocating EID pools that take static EID assignments into account. That is, a bridge cannot request a particular set of EIDs to be allocated to it.
- MCTP bridges/bus owners may be configurable to use only static EIDs.

8.17.3 Use of Static Physical Addresses

In many simple topologies, it is desirable to use devices that have statically configured physical addresses. This can simplify the implementation of the device. For example, an SMBus/I2C device that is not used in a hot-plug application would not need to support the SMBus address assignment (SMBus ARP) protocol. Fixed addresses can also aid in identifying the location and use of an MCTP device in a system. For example, if a system has two otherwise identical MCTP devices, a system vendor will know that the device at address "X" is the one at the front of the motherboard, and the device at address "Y" is at the back, because that is how they assigned the addresses when the system was designed.

Therefore, MCTP transport bindings, such as for SMBus/I2C, are allowed to support devices being at static physical addresses without requiring the binding to define a mechanism that enables the bus owner to discover MCTP devices that are using static addresses.

In this case, the bridge or bus owner must have a-priori knowledge of the addresses of those devices to be able to assign EIDs to those devices and to support routing services for those devices. To support this requirement, the following requirements and recommendations are given to device vendors:

- Devices that act as bus owners or bridges and are intended to support MCTP devices that use static physical addresses should provide a non-volatile configuration option that enables the system integrator to configure which device addresses are being used for devices on each bus that is owned by the bridge/bus owner.
- The mechanism by which this non-volatile configuration occurs is specific to the device vendor. In many cases, the physical address information will be kept in some type of non-volatile storage that is associated with the device and gets loaded when the device is manufactured or when the device is integrated into a system. In other cases, this information may be coded into a firmware build for the device.

8.17.4 Endpoint ID Assignment Process for Bus Owners/Bridges

The bus owner/bridge must get its own EID assignment, and a pool of EIDs, as follows. These steps only apply to bus owner/bridge devices that are not the topmost bus owner.

- Bus owners/bridges must be pre-configured with non-volatile information that identifies which buses they own. (How this configuration is accomplished is device/vendor specific and is outside the scope of this specification.)
- The bus owner/bridge announces its presence on any buses *that it does not own* to get an EID assignment for that bus. The mechanism by which this announcement occurs is dependent on the particular physical transport binding and is defined as part of the binding specification.
- The bus owner/bridge waits until it gets its own EID assignment for one of those buses through the Set Endpoint ID command.
- The bus owner/bridge indicates the size of the EID pool it requires by returning that information in the response to the Set Endpoint ID command.
- For each bus where the bus owner/bridge is itself an "owned" device, the bus owner/bridge will be offered a pool of EIDs by being sent an Allocate Endpoint IDs command from the bus owner.
- The bus owner/bridge accepts allocations only from the bus of the "first" bus owner that gives it the allocation, as described in the Allocate Endpoint IDs command description in 8.10. If it gets allocations from other buses, they are rejected.

The bus owner can now begin to build a routing table for each of the buses that it owns, and accept routing information update information. Refer to Clause 9 for more information.

8.17.5 Endpoint ID Retention

Devices should retain their EID assignments for as long as they are in their normal operating state. Asynchronous conditions, such as device errors, unexpected power loss, power state changes, resets, firmware updates, may cause a device to require a reassignment of its EID. Devices should retain their EID assignments across conditions where they may temporarily stop responding to commands over MCTP, such as during internal resets, error conditions, or configuration updates.

8.17.6 Reclaiming EIDs from Hot-Plug Devices

Bridges will typically have a limited pool of EIDs from which to assign and allocate to devices. (This also applies when a single bus owner supports hot-plug devices.) It is important for bridges to reclaim EIDs so that when a device is removed, the EID can later be re-assigned when a device is plugged in. Otherwise, the EID pool could become depleted as devices are successively removed and added.

EIDs for endpoints that use static addresses are not reclaimed.

No mechanism is specified in the MCTP base protocol for detecting device removal when it occurs. Therefore, the general approach to detecting whether a device has been removed is to re-enumerate the bus when a new device is added and an EID or EID pool is being assigned to that device.

The following approach can be used to detect removed hot-plug devices: The bus owner/bridge can detect a removed device or devices by validating the EIDs that are presently allocated to endpoints that are directly on the bus and identifying which EIDs are missing. It can do this by attempting to access each endpoint that the bridge has listed in its routing table as being a device that is directly on the particular bus. Attempting to access each endpoint can be accomplished by issuing the Get Endpoint ID command to the physical address of each device and comparing the returned result to the existing entry in the routing table. If there is no response to the command, or if there is a mismatch with the existing routing information, the entry should be cleared and the corresponding EID or EID range should be returned to the "pool" for re-assignment. The bus owner/bridge can then go through the normal steps for EID assignment.

This approach should work for all physical transport bindings, because it keeps the "removed EID" detection processing separated from the address assignment process for the bus.

In some cases, a hot-plug endpoint may temporarily go into a state where it does not respond to MCTP control messages. Depending on the medium, it is possible that when the endpoint comes back on line, it does not request a new EID assignment but instead continues using the EID it had originally assigned. If this occurs while the bus owner is validating EIDs to see if any endpoints are no longer accessible, it is possible that the bus owner will assume that the endpoint was removed and reassign its EID to a newly inserted endpoint, unless other steps are taken:

- The bus owner must wait at least $T_{RECLAIM}$ seconds before reassigning a given EID (where $T_{RECLAIM}$ is specified in the physical transport binding specification for the medium used to access the endpoint).
- Reclaimed EIDs must only be reassigned after all unused EIDs in the EID pool have been assigned to endpoints. Optionally, additional robustness can be achieved if the bus owner maintains a short FIFO list of reclaimed EIDs (and their associated physical addresses) and allocates the older EIDs first.
- A bus owner shall confirm that an endpoint has been removed by attempting to access it after $T_{RECLAIM}$ has expired. It can do this by issuing a Get Endpoint ID command to the endpoint to verify that the endpoint is still non-responsive. It is recommended that this be done at least three times, with a delay of at least 1/2 * $T_{RECLAIM}$ between tries if possible. If the endpoint continues to be non-responsive, it can be assumed that it is safe to return its EID to the pool of EIDs available for assignment.

8.17.7 Additional Requirements for Hot-Plug Endpoints

Devices that are hot-plug must support the Get Endpoint UUID command. The purpose of this requirement is to provide a common mechanism for identifying when devices have been changed.

Endpoints that go into states where they temporarily do not respond to MCTP control messages shall re-announce themselves and request a new EID assignment if they are "off line" for more than $T_{RECLAIM}$ seconds, where $T_{RECLAIM}$ is specified in the physical transport binding specification for the medium used to access the endpoint.

8.17.8 Additional Requirements for Devices with Multiple Endpoints

A separate EID is utilized for each MCTP bus that a non-bridge device connects to. In many cases, it is desirable to be able to identify that the same device is accessible through multiple EIDs.

If an endpoint has multiple physical interfaces (ports), the interfaces can be correlated to the device by using the MCTP Get Endpoint UUID command (see 11.5) to retrieve the unique system-wide identifier.

Devices connected to multiple buses must support the Get Endpoint UUID command for each endpoint and return a common UUID value across all the endpoints. This is to enable identifying EIDs as belonging to the same physical device.

8.18 Handling Reassigned EIDs

Though unlikely, it is still possible that during the course of operation of an MCTP network, a particular EID could get reassigned from one endpoint to another. For example, this could occur if a newly hot-swap inserted endpoint device gets assigned an EID that was previously assigned to a device that was subsequently removed.

Under this condition, it is possible that the endpoint could receive a message that was intended for the previously installed device. This is not considered an issue for MCTP control messages because the control messages are typically just used by bus owners and bridges for initializing and maintaining the

MCTP network. The bus owners and bridges are aware of the EIDs they have assigned to endpoints and are thus intrinsically aware of any EID reassignment.

Other endpoints, however, are not explicitly notified of the reassignment of EIDs. Therefore, communication that occurs directly from one endpoint to another is subject to the possibility that the EID could become assigned to a different device in the middle of communication. This must be protected against by protocols specific to the message type being used for the communication.

In general, the approach to protecting against this will be that other message types will require some kind of "session" to be established between the intercommunicating endpoints. By default, devices would not start up with an active session. Thus, if a new device is added and it gets a reassigned EID, it will not have an active session with the other device and the other device will detect this when it tries to communicate.

The act of having a new EID assigned to an existing device should have the same effect. That is, if a device gets a new EID assignment, it would "close" any active sessions for other message types.

The mechanism by which other message types would establish and track communication sessions between devices is not specified in this document. It is up to the specification of the particular message type.

9 MCTP Bridging

One key capability provided by MCTP is its ability to route messages between multiple buses and between buses of different types. This clause describes how routing information is created, maintained, and used by MCTP bridges and MCTP endpoints. Keep the following key points in mind about MCTP bridges:

- An MCTP bridge is responsible for routing MCTP packets between at least two buses.
- An MCTP bridge is typically the bus owner for at least one of those buses.

9.1.1 Routing/Bridging Restrictions

Figure 7 and Figure 8 illustrate some of the supported and unsupported bridging topologies. As shown, it is acceptable for a given topology to have more than one path to get to a given EID. This can occur either because different media are used or because a redundant or failover communication path is desired in an implementation.

A bridge shall not route or forward packets with a broadcast destination ID.

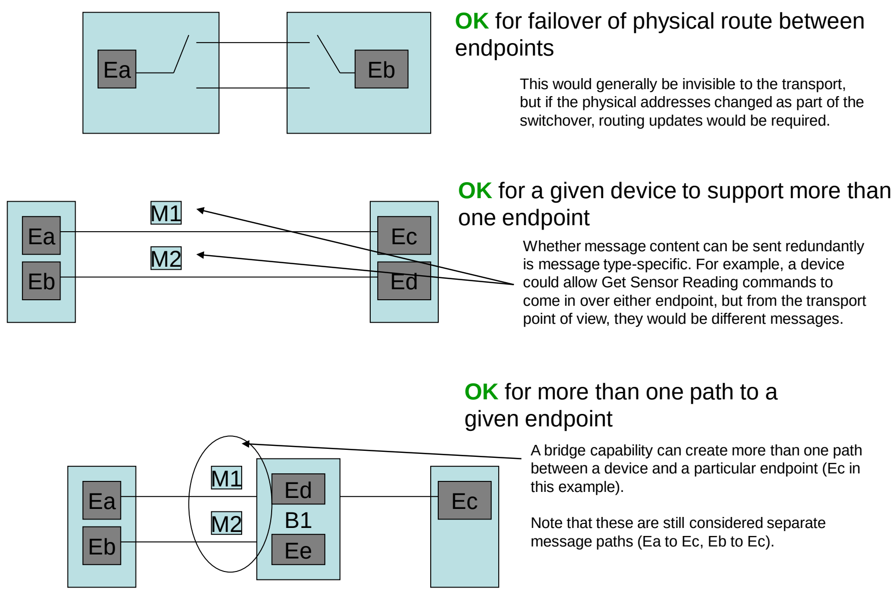


Figure 7 – Acceptable Failover/Redundant Communication Topologies

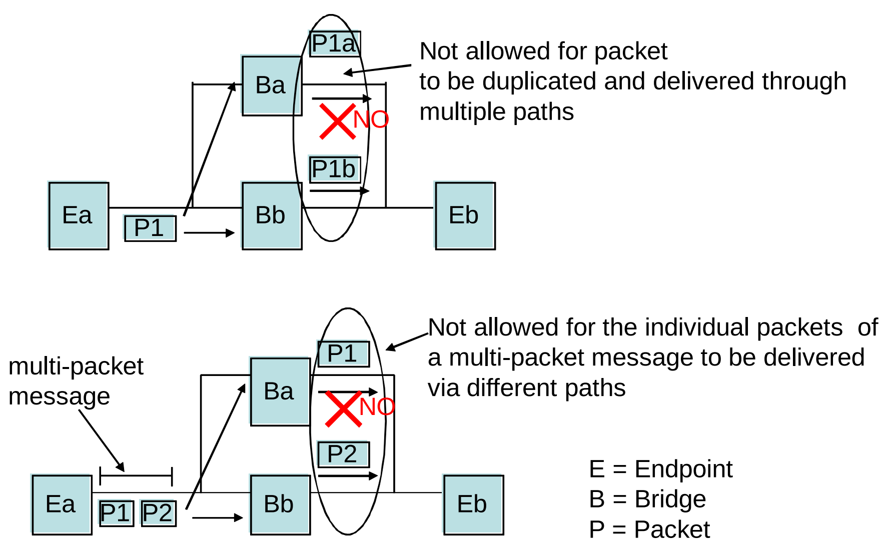


Figure 8 – Routing/Bridging Restrictions

9.1.2 EID Options for MCTP Bridges

An MCTP bridge that connects to multiple buses can have a single EID or multiple EIDs through which the bridge's routing configuration and endpoint functionality can be accessed through MCTP control commands. There are three general options:

- The bridge uses a single MCTP endpoint
- The bridge uses an MCTP endpoint for each bus that connects to a bus owner
- The bridge uses an MCTP endpoint for every bus to which it connects

Examples of these different options are shown in Figure 9, and more detailed information on the options is provided following the figure.

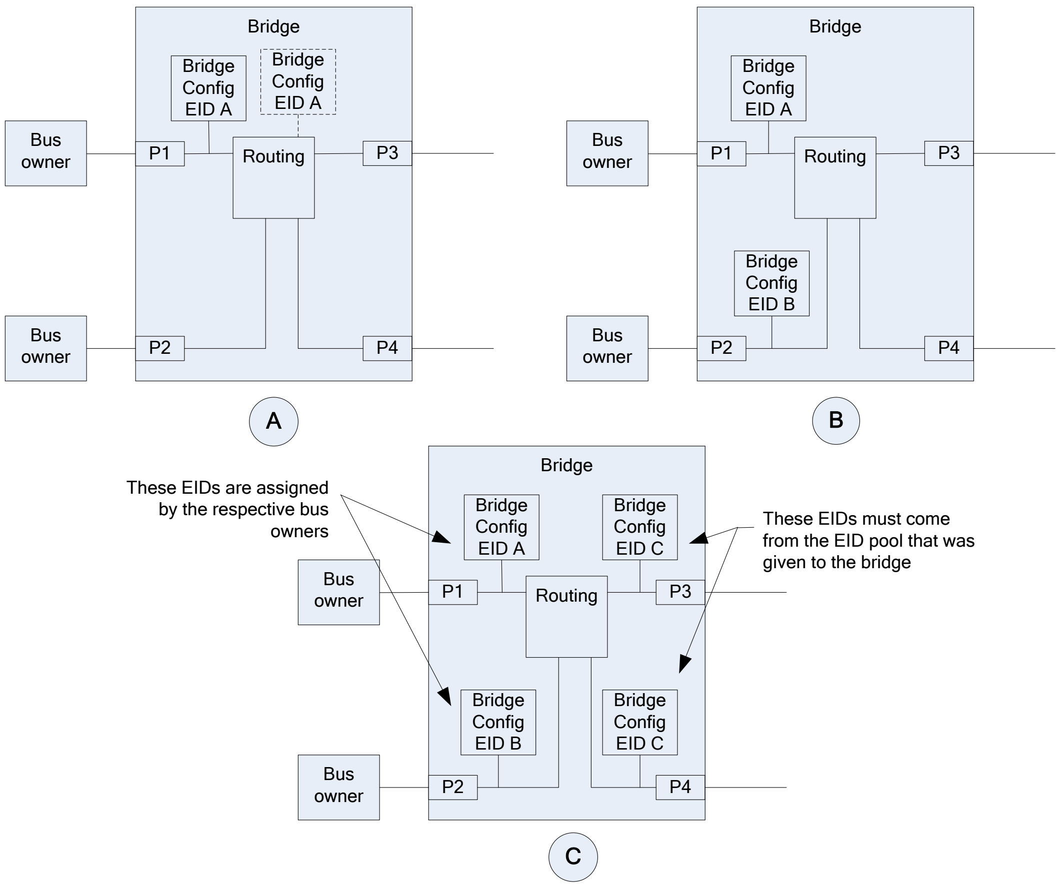


Figure 9 – EID Options for MCTP Bridges

A bridge has only one EID pool. To prevent issues with getting an EID pool allocation from multiple bus owners, a bridge that is accessible through multiple EIDs will only accept EID pool allocation from the first bus that allocation is received from using the Allocate Endpoint IDs command. This behavior is described in more detail in the specification of the Allocate Endpoint IDs command.

If necessary, the Get Endpoint UUID command can be used to correlate that EIDs belong to the same MCTP bridge device. (This correlation is not required for normal initialization and operation of the MCTP network, but it may be useful when debugging.)

The following is a more detailed description of the different EID options for bridges:

- **Single endpoint**

  A single endpoint is used to access the bridge's routing configuration and endpoint functionality. Referring to diagram (A) in Figure 9, an implementation may elect to either have the endpoint functionality be directly associated with a particular bus/port (for example, P1) or the functionality can be located on a "virtual bus" that is behind the routing function. In either case, the routing functionality ensures that the EID can be accessed through any of the buses to which the bridge connects.

Although there is a single endpoint, the bridge shall report the need for EID assignment for that endpoint on each bus that is connected to a bus owner (for example, P1, P2). The multiple announcements provide a level of failover capability in the EID assignment process in case a particular bus owner becomes unavailable. The multiple announcements also help support a consistent EID assignment process across bus owners. To prevent issues with getting conflicting EID assignments from multiple bus owners, the bridge will only accept EID pool allocation from the first bus that an allocation is received from using the Set Endpoint ID command. This behavior is described in more detail in the specification of the Set Endpoint ID command. The bridge shall not report the need for EID assignment on any buses that the bridge itself owns.

- **Endpoint for each bus connection to a bus owner**

  The bridge has one endpoint for each bus connected to a bus owner. This is shown as diagram (B) in Figure 9. There are no explicit endpoints associated with buses that are not connected to a bus owner (for example, the buses connected to ports P3 and P4, respectively.) Because of the way packet routing works, EID A and EID B can be accessed from any of the ports connected to the bridge. Thus, the bridge’s configuration functionality may be accessed through multiple EIDs. Because a separate endpoint communication terminus is associated with each port (P1, P2), the bridge can accept an EID assignment for each bus independently.

  The bridge shall only report the need for EID assignment on buses that connect to a bus owner, and only for the particular MCTP control interface that is associated with the particular bus. For example, the bridge would announce the need for EID assignment for the interface associated with EID A only through P1, and the need for EID assignment for the interface associated with EID B only through P2. The bridge shall not report the need for EID assignment on any buses that the bridge itself owns.

- **Endpoint for every bus connection**

  The bridge has one endpoint for each bus connected to it, as shown as diagram (C) in Figure 6. This includes buses that connect to bus owners (for example, P1, P2) and buses for which the bridge is the bus owner (for example, P3, P4). Because of the way packet routing works, any of these EIDs can be accessed from any of the ports connected to the bridge.

  Because a separate endpoint communication terminus is associated with each owned port (P1, P2), the bridge can accept an EID assignment for the bus owners of each bus independently. The EIDs associated with the buses that the bridge itself owns (for example, P3, P4) must be taken out of the EID pool that is allocated to the bridge.

  The bridge shall only report the need for EID assignment on buses that connect to a bus owner, and only for the particular MCTP control interface that is associated with the particular bus. For example, the bridge would announce the need for EID assignment for the interface associated with EID A only through P1, and the need for EID assignment for the interface associated with EID B only through P2. The bridge shall not report the need for EID assignment on any buses that the bridge itself owns.

9.1.3 Routing Table

An MCTP bridge maintains a routing table where each entry in the table associates either a single EID or a range of EIDs with a single physical address and bus ID for devices that are on buses that are directly connected to the bridge.

If the device is a bridge, there will typically be a range of EIDs that are associated with the physical address of the bridge. There may also be an entry with a single EID for the bridge itself.

9.1.4 Bridging Process Overview

When a bridge receives an MCTP packet, the following process occurs:

1) The bridge checks to see whether the destination EID in the packet matches or falls within the range of EIDs in the table.
2) If the EID is for the bridge itself, the bridge internally consumes the packet.
3) If there is a match with an entry in the routing table, the following steps happen:
   - The bridge changes the physical addresses in the packet and reformats the medium-specific header and trailer fields as needed for the destination bus.
   - The destination physical address from the source bus is replaced with the destination physical address for the destination bus obtained from the entry in the routing table.
   - The bridge replaces the source physical address in the packet it received with the bridge's own physical address on the target bus. This is necessary to enable messages to be routed back to the originator.
   - Packet-specific transport header and data integrity fields are updated as required by the particular transport binding.
4) If there is no match, packets with EID values that are not in the routing table are silently discarded.

9.1.5 Endpoint Operation with Bridging

A bridge does not track the packet transmissions between endpoints. It simply takes packets that it receives and routes them on a per-packet basis based on the destination EID in the packet. It does not pay attention to message assembly or disassembly or message type-specific semantics, such as request/response semantics, for packets that it routes to other endpoints.

Most simple MCTP endpoints will never need to know about bridges. Typically, another endpoint will initiate communication with them. The endpoint can then simply take the physical address and source EID information from the message and use that to send messages back to the message originator.

An endpoint that needs to originate a "connection" to another MCTP endpoint does need to know what physical address should be used for messages to be delivered to that endpoint. To get this information, it needs to query the bus owner for it. An endpoint knows the physical address of the bus owner because it saved that information when it got its EID assignment.

The Resolve Endpoint ID command requests a bus owner to return the physical address that is to be used to route packets to a given EID. (This is essentially the MCTP equivalent of the ARP protocol that is used to translate IP addresses to physical addresses.) The address that is returned in the Resolve Endpoint ID command response will either be the actual physical address for the device implementing the endpoint, or it will be the physical address for the bridge to be used to route packets to the desired endpoint.

Because the physical address format is media-specific, the format of the physical address parameter is documented in the specifications for the particular media-specific physical transport binding for MCTP (for example, MCTP over SMBus/I2C, MCTP over PCIe Vendor Defined Messaging, and so on).

If endpoint A has received a message from another endpoint B, it does not need to issue a Resolve Endpoint ID command. Instead, it can extract the source EID and source physical address from the earlier message from endpoint B, and then use that as the destination EID and destination physical address for the message to Endpoint B.

9.1.6 Routing Table Entries

Each MCTP device that does bridging must maintain a logical routing table. A bus owner must also typically maintain a routing table if more than one MCTP device is connected to the bus that it owns. The routing table is required because the bus owner is also the party responsible for resolving EIDs to physical addresses.

The internal format that a device uses for organizing the routing table is implementation dependent. From a logical point of view, each entry in a routing table will be comprised of at least three elements: An EID range, a bus identifier, and a bus address. This is illustrated in Figure 10.

EID Range	Bus ID	Bus Address

Figure 10 – Basic Routing Table Entry Fields

The *EID range* specifies the set of EIDs that can be reached through a particular bus address on a given bus. Because the bus ID and bus address may correspond to a particular "port" on a bridge, it is possible that there can be multiple non-contiguous ranges (multiple routing table entries) that have the same bus ID/bus address pair route. EIDs and EID ranges can be categorized into three types: downstream, upstream, and local. "Downstream" refers to EIDs that are associated with routing table entries that are for buses that are owned by the bridge that is maintaining the routing table. "Upstream" refers to EIDs that are associated with routing table entries that route to buses that are not owned by the bridge that is maintaining the routing table.

"Local" refers to the EIDs for routing table entries for endpoints that are on buses that are directly connected to the bridge that is maintaining the routing table. A particular characteristic of entries for local EIDs is that the Resolve Endpoint ID command is issued from the same bus that the endpoint is on. The bridge/bus owner delivers the physical address for that endpoint rather than the physical address associated with a routing function. This facilitates allowing endpoints on the same the bus to communicate without having to go through an MCTP routing function.

A routing table entry may not be "local" even if two endpoints are located on the same bus. An implementation may require that different endpoints go through the routing function to intercommunicate even if the endpoints are part of the same bus.

The *bus ID* is an internal identifier that allows the MCTP device to identify the bus that correlates to this route. MCTP does not require particular values to be used for identifying a given physical bus connection on a device. However, this value will typically be a 0-based numeric value.

EXAMPLE: A device that had three buses would typically identify them as buses "0", "1", and "2".

The *bus address* is the physical address of a specific device on the bus through which the EIDs specified in the *EID range* can be reached. This can either be the physical address corresponding to the destination endpoint, or it can be the physical address of the next bridge in the path to the device. The format of this address is specific to the particular physical medium and is defined by the physical medium transport binding.

9.1.7 Routing Table Creation

This clause illustrates the types of routing information that a bridge requires, and where the information comes from. This clause also describes the steps that a bus owner must use to convey that information for a given bus.

Figure 11 helps illustrate the steps that are required to completely establish the routing information required by a bridge (bridge Y). The arrows in Figure 11 point outward from the bus owner and inward to "owned" endpoints on the bus.

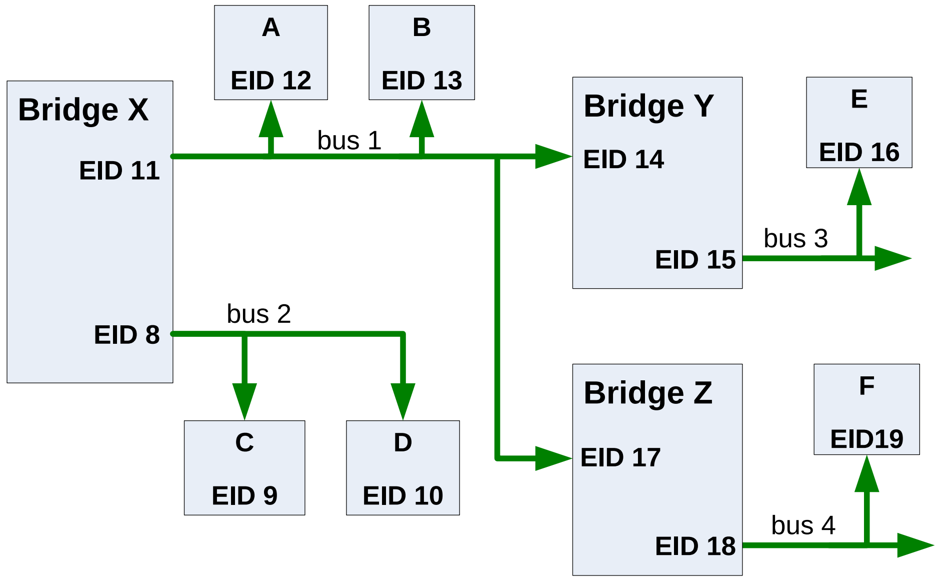


Figure 11 – Routing Table Population

9.1.7.1 Routing Table Population Example

With reference to Figure 11, the following items describe the information that bridge Y will need for routing messages in the example topology shown:

- It needs a set of EIDs allocated to it to use for itself and to allocate to other devices (for example, EIDs 14:16). These are allocated to it by the bus owner (bridge X).
- It needs a routing table that has an entry that maps EID 16 to the physical address for device E on bus 3.
- It needs routing table entries for the local devices on bus 1, which are: bridge X (EID 11), device A (EID 12), device B (EID 13), and bridge Z (EID 17), assuming that devices A and B are to be reached by bridge Y without having to go through bridge X. This information must be given to it by the bus owner (bridge X).
- It needs to know that EIDs 8:10 are accessed through bus owner/bridge X. Therefore, it needs a routing table entry that maps the EID range 9:10 to the physical address for bridge X on bus 1. This information must also be given to it by the bus owner (bridge X).
- It needs to know that EIDs 17:19 are accessed through bridge Z. Therefore, it needs a routing table entry that maps the EID range 17:19 to the physical address for bridge Z on bus 1. Because the bus owner (bridge X) allocated that range of EIDs to bridge Z in the first place, this information is also given to bridge Y by the bus owner (bridge X).

9.1.7.2 Bus Initialization Example

Starting with the description of what bridge Y requires, the following task list shows the steps that bridge X must take to provide routing information for bus 1. Bridge X must:

1) Assign EIDs to devices A, B, C, D, bridge Y, and bridge Z. This is done using the Set Endpoint ID command. The response of the Set Endpoint ID command also indicates whether a device wants an additional pool of EIDs.

2) Allocate EID pools to bridge Y and bridge Z. This is done using the Allocate Endpoint IDs command.

3) Tell bridge Y the physical addresses and EIDs for devices A and B, bridge X (itself), and bridge Z on bus 1. This is done using the Routing Information Update command.

4) Tell bridge Y that EIDs 18:19 are accessed through the physical address for bridge Z on bus 1. This is also done using the Routing Information Update command. (Steps 3 and 4 can be combined and covered with one instance of the command.)

5) Tell bridge Z the physical addresses and EIDs for devices A and B, bridge X (itself), and bridge Y on bus 1. This is also done using the Routing Information Update command.

6) Tell bridge Z that EIDs 15:16 are accessed through the physical address for bridge Y on bus 1. This is also done using the Routing Information Update command. (Steps 5 and 6 can be combined and covered with one instance of the command.)

7) Tell bridge Y and bridge Z that EIDs 8:10 are accessed through bridge X on bus 1. This is also done using the Routing Information Update command. This step could also be combined with steps 3 and 4 for bridge Y and steps 5 and 6 for bridge Z.

9.1.8 Routing Table Updates Responsibility for Bus Owners

After it is initialized for all bridges, routing table information does not typically require updating during operation. However, updating may be required if a bridge is added as a hot-plug device. In this case, when the bridge is added to the system, it will trigger the need for the bus owner to assign it an EID, which will subsequently cause the request for EID pool allocations, and so on. At this time, the bus owner can simply elect to re-run the steps for bus initialization as described in 9.1.7.2.

9.1.9 Consolidating Routing Table Entries

MCTP requires that when an EID pool is allocated to a device, the range of EIDs is contiguous and follows the EID for the bridge itself. Thus, a bridge can elect to consolidate routing table information into one entry when it recognizes that it has received an EID or EID range that is contiguous with an existing entry for the same physical address and bus. (The reason that EID allocation and routing information updates are not done as one range using the same command is because of the possibility that a device may have already received an allocation from a different bus owner.)

9.2 Bridge and Routing Table Examples

The following examples illustrate different bridge and MCTP network configurations and the corresponding information that must be retained by the bridge for MCTP packet routing and to support commands such as Resolve Endpoint ID and Query Hop.

The following clauses (including Table 4 through Table 6) illustrate possible topologies and ways to organize the information that the bridge retains. Implementations may elect to organize and store the same information in different ways. The important aspect of the examples is to show what information is kept for each EID, to show what actions cause an entry to be created, and to show how an EID or EID range can in some cases map to more than one physical address.

The examples show a possible time order in which the entries of the table are created. Note that a given implementation of the same example topology could have the entries populated in a different order. For example, if there are two bus owners connected to a bridge, there is no fixed order that the bus owners would be required to initialize a downstream bridge. Additionally, there is no requirement that bus owners perform EID assignment or EID pool allocation in a particular order. One implementation may elect to allocate EID pools to individual bridges right after it has assigned the bridge its EID. Another implementation may elect to assign all the EIDs to devices first, and then allocate the EID pools to bridges.

9.2.1 Example 1: Bridge D2 with an EID per "Owned" Port

Figure 12 shows the routing table in a bridge (D2), where D2 has an EID associated with each bus connected to a bus owner. In this example, D1 is not implementing any internal bridging between its P1 and P2. Consequently, EID2 cannot be reached by bridging through EID1 and vice versa (see Table 4).

NOTE: If there *was* internal bridging, D1 would need to provide routing information that indicated that EID2 was reachable by going through EID1 and vice versa. In this case, D1 would provide routing information that EID range (EID1…EID2) would be accessed through D1P1a1 on SMBus and D1P1a2 on PCIe.

Key: D = device, P = port, a = physical address

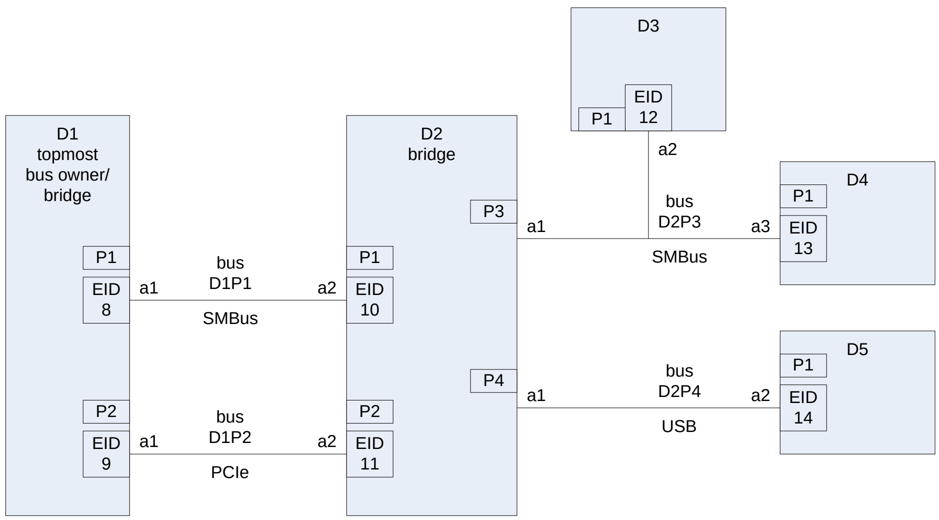


Figure 12 – Example 1 Routing Topology

Table 4 – Example 1 Routing Table for D2

Time	EID	EID Access Port	Medium Type	Access Physical Address	Device/Entry Type	Entry Was Created and Populated By
↓	EID 10	P1	SMBus	D1P1a2	Bridge, Self	Self when EID was assigned by D1
	EID 11	P2	PCIe	D1P2a2	Bridge, Self	Self when EID was assigned by D1
	EID 12	P3	SMBus	D2P3a2	Endpoint	Self after D1 assigned EID pool (typically the entry will not be created until after the bridge D2 assigns EID 12 to D3)
	EID 13	P3	SMBus	D2P3a3	Endpoint	Self after D1 assigned EID pool (typically the entry will not be created until after the bridge D2 assigns EID 13 to D4)
	EID 14	P4	USB	D2P4a2	Endpoint	Self after D1 assigned EID pool (typically the entry will not be created until after the bridge D2 assigns EID 14 to D5)
	EID 8	P1	SMBus	D1P1a1	Bridge	D1 through Routing Information Update command
	EID 9	P2	PCIe	D1P2a1	Bridge	D1 through Routing Information Update command

9.2.2 Example 2: Topmost Bus Owner D1

Figure 13 assumes the following conditions:

- D1 assigns its internal EIDs first.
- The buses are handled in the order D1P1, D1P2, D1P3.
- D1 allocates the EID pool to bridges right after it has assigned the EID to the device.

Similar to Example 1, this example assumes that there is no internal bridging within D1 between P1, P2, and P3. This scenario is reflected in Table 5.

Key: D = device, P = port, a = physical address

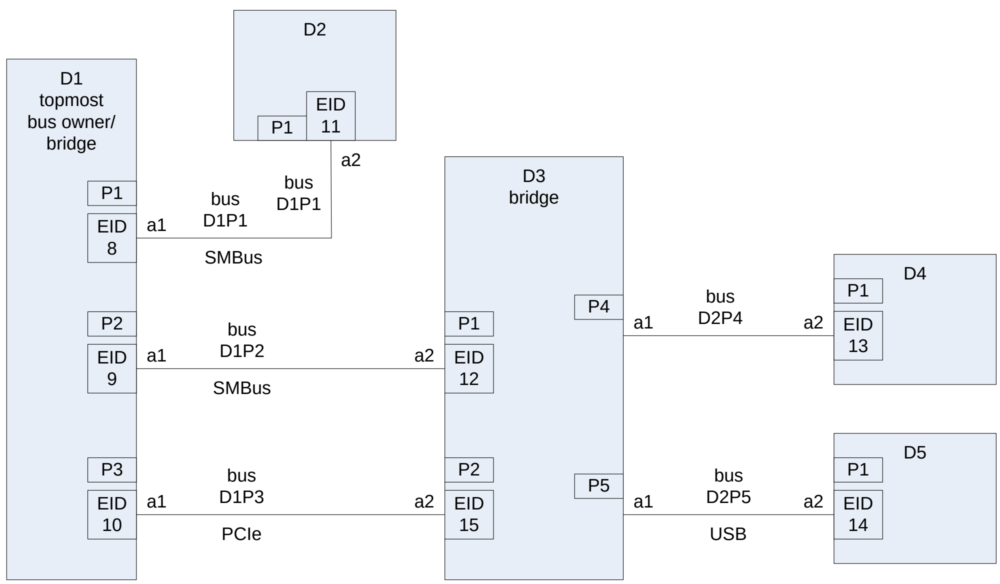


Figure 13 – Example 2 Routing Topology

Table 5 – Example 2 Routing Table for D1

EID	EID Access Port	Medium Type	Access Physical Address	Device/Entry Type	Entry Was Created and Populated By
EID 8	P1	SMBus	D1P1a1	Bridge, self	Self
EID 9	P2	SMBus	D1P2a1	Bridge, self	Self
EID 10	P3	PCIe	D1P3a1	Bridge, self	Self
EID 11	P1	SMBus	D1P1a2	Endpoint	Self upon assigning EID to device D2
EID 12	P2	SMBus	D1P2a2	Bridge	Self upon assigning EID 5 to bridge D3
EID 13:14	P2	SMBus	D1P2a2	Bridge pool	Self upon assigning EID pool to bridge D3
EID 15	P3	PCIe	D1P3a2	Bridge	Self upon assigning EID 8 to bridge D3
EID 13:14	P3	PCIe	D1P3a2	Bridge pool	Self upon issuing an Allocate Endpoint IDs command and finding that bridge D3 already has an assigned pool, D1 creates this entry by extracting the EIDs for this entry from the response to the Allocate Endpoint IDs command

9.2.3 Example 3: Bridge D2 with Single EID

Figure 14 assumes that bridge D2 has a single EID and gets its EID assignment and EID allocation through bus D1P1 first, and that bus D1P2 later gets initialized. This scenario is reflected in Table 6.

Key: D = device, P = port, a = physical address

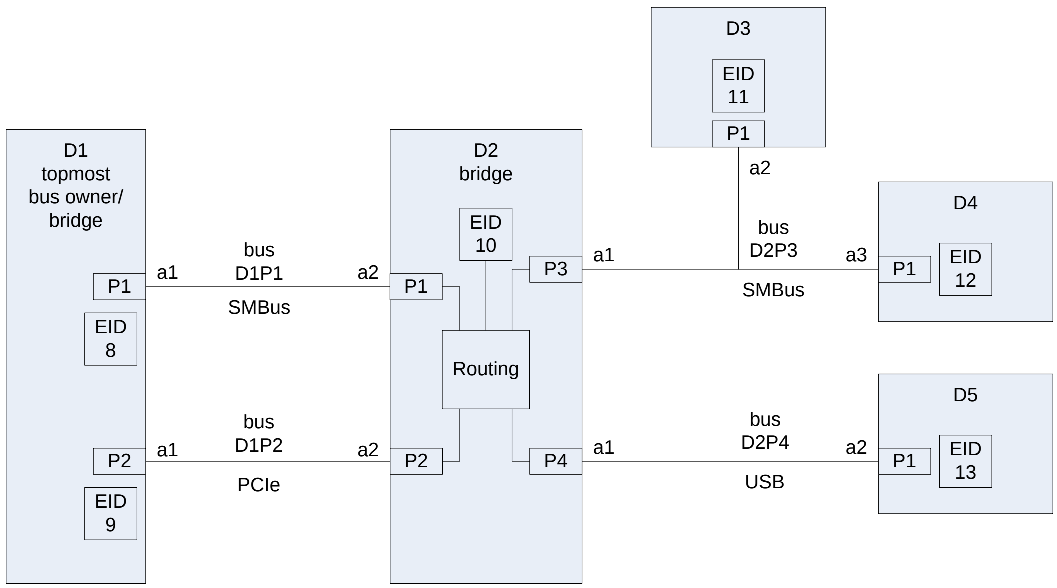


Figure 14 – Example 3 Routing Topology

Table 6 – Example 3 Routing Table for D2

Target EID	Target Endpoint Access Port	Target EID Access Physical Address	Device/Entry Type	Entry Was Created and Populated By
EID 10	P1	D1P1a2	Bridge, self	All four entries created by self (bridge) upon receiving initial EID assignment from D1 through P1
EID 10	P2	D1P2a2	Bridge, self	
EID 10	P3	D2P3a1	Bridge, self	
EID 10	P4	D2P4a1	Bridge, self	
EID 11	P3	D2P3a2	Endpoint	Self after D1 allocated EID pool (typically the entry will not be created until after the bridge D2 assigns EID 11 to D3)
EID 12	P3	D2P3a3	Endpoint	Self after D1 allocated EID pool (typically the entry will not be created until after the bridge D2 assigns EID 12 to D4)
EID 13	P3	D2P4a2	Endpoint	Self after D1 allocated EID pool (typically the entry will not be created until after the bridge D2 assigns EID 13 to D5)

Target EID	Target Endpoint Access Port	Target EID Access Physical Address	Device/Entry Type	Entry Was Created and Populated By
EID 8:9	P1	D1P1a1	Bridge	D1 through Routing Information Update command
EID 8:9	P2	D1P2a1	Bridge	D1 through Routing Information Update command

9.2.4 Additional Information Tracked by Bridges

In addition to the information required to route messages between different ports, a bridge has to track information to handle MCTP control commands related to the configuration and operation of bridging (shown in Table 7).

Table 7 – Additional Information Tracked by Bridges

What	Why
Which buses are connected to a bus owner	This information tells the bridge from which buses it should request EID assignment. This will typically be accomplished as a non-volatile configuration or hardware-strapping option for the bridge.
Which bus the bridge received its EID assignment through the Set Endpoint ID command	If the bridge uses a single EID that is shared across multiple "owned" buses, this information is used to track which bus the request came in on, so that the bridge can reject EID assignment requests from other buses.
Which bus it received the Routing Information Update command from for creating a particular routing table entry	This information is required so that if a future Routing Information Update command is received, the bridge will update only the entries corresponding to that bus.
Which bus it received its EID pool allocation from through the Allocate Endpoint IDs command	This information is used to track which bus the request came in on so that the bridge can reject EID pool allocations from other buses.
The physical medium and physical addressing format used for each port	This information is used to provide the correctly formatted response to commands such as Resolve Endpoint ID and for bridging MCTP packets between the different buses that the bridge supports. Because this is related to the physical ports and hardware of the bridge, this information will typically be "hard coded" into the bridge.

9.3 Endpoint ID Resolution

When a device uses the Resolve Endpoint ID command to request the resolution of a given endpoint to a physical address, the bridge must respond based on which bus the request came in on.

For example, consider Figure 15. If device A wishes to get the physical address needed to send a message to device C, it sends a Resolve Endpoint ID command to bus owner bridge X through address Ax1. Because device A must go through bridge X to get to device C, bridge X responds with its physical address Ax1.

When device B wishes to know the address to use to communicate with device C, it sends a Resolve Endpoint ID request to bridge X through address Ax2. In this case, bridge X can respond by giving device B the direct physical address of device C on bus 2, Ac2.

Thus, the Resolve Endpoint ID command can return a different response based on the bus from which the Resolve Endpoint ID command was received.

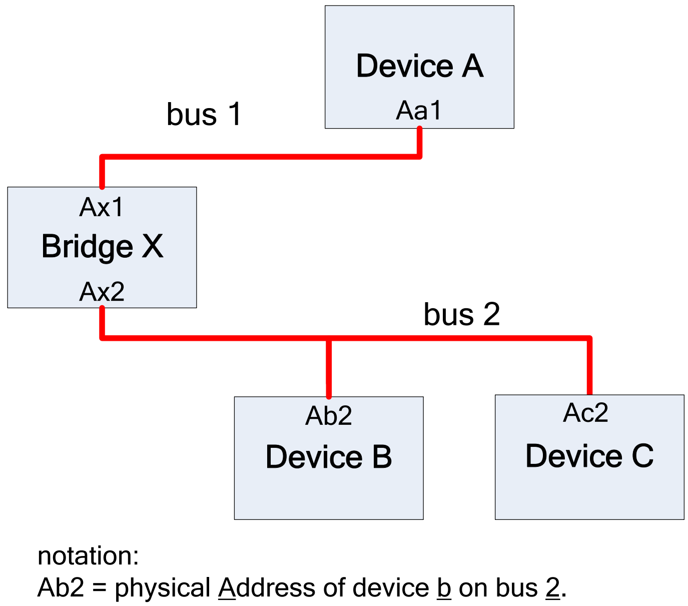


Figure 15 – Endpoint ID Resolution

9.3.1 Resolving Multiple Paths

Cases can occur where there can be more than one possible path to a given EID. A likely scenario is shown in Figure 16. In Figure 16, assume that the system topology supports cards that connect to either SMBus, PCIe, or both. Bridge X is the bus owner for both buses.

NOTE: This is a logical representation of MCTP buses. Physically, the buses may be formed of multiple physical segments, as would be the case if one of the MCTP buses was built using PCIe.

As shown, card C contains a bridge that connects to both buses. Thus, the device with EID 100 can be reached either from bus 1 or bus 2.

If device D wishes to send a message to EID 100, bridge X can choose to route that message either through bus 1 or bus 2. MCTP does not have a requirement on how this is accomplished. The general recommendation is that the bridge preferentially selects the faster available medium. In this example, that would be PCIe.

NOTE: There are possible topologies where that simple rule may not yield the preferred path to a device. However, in most common implementations in PC systems, this approach should be effective. A vendor making a bridge device may consider providing configuration options to enable alternative policies.

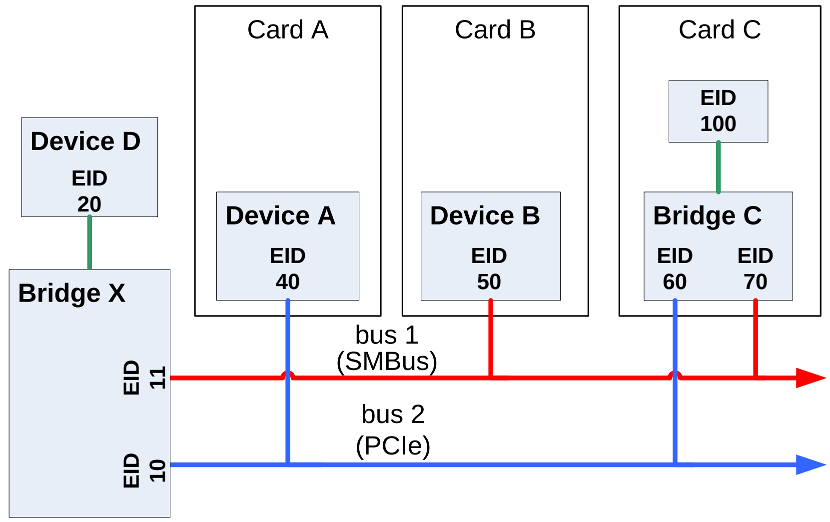


Figure 16 – Resolving Multiple Paths

9.4 Bridge and Bus Owner Implementation Recommendations

This clause provides recommendations on EID pool and routing table sizes for devices that implement bridge and bus owner functionality.

9.4.1 Endpoint ID Pool Recommendations

The system design should seek to minimize the number of devices that need to allocate EID pools to hot-plug devices or add-in cards. If feasible, the system design should have all busses that support hot-plug devices/add-in cards owned by a single device.

If only one device handles the hot-plug devices and add-in cards, it will be simpler for the system integrator to configure devices and allocate EID pools. Because any other bridges in the system that do not handle hot-plug devices only need to handle a fixed number of MCTP devices, it will be known at design time how large an EID pool will be required. The remaining number of EIDs can then simply be allocated to the single device that handles the hot-plug devices and add-in cards.

To support this, it is recommended that devices that operate as bridges include a non-volatile configuration option that enables the system integrator to configure the size of the EID pool they request.

9.4.2 Routing Table Size Recommendations

This clause provides some initial recommendations and approaches on how to determine what target routing table entry support to provide in a device.

- **PCIe slots**

  To provide entries to support devices that plug into PCIe slots, assume that each slot may support both PCIe and SMBus endpoints and provide support for at least two endpoints per bus type.

  This means providing support for at least four directly connected endpoints per card. (Other endpoints may be behind bridges on the card, but this does not affect the routing table size for the bus owner.) This implies at least four routing table entries per PCIe slot. Thus, a device that was designed to support system implementations with eight PCIe slots should have support for 32 routing table entries.

- **Planar PCIe devices**

  In most PC systems, PCIe would be typically implemented as a single MCTP bus owned by a single device as the bus owner. Thus, the number of static devices should be proportional to the number of PCIe devices that are built into the motherboard.

  Typically, this is fewer than eight devices. Thus it is recommended to support at least eight entries for static PCIe devices.

- **Static SMBus/I2C MCTP devices**

  The routing table should also be sized to support an additional number of "static" devices on owned buses. At this time, it is considered unlikely that more than a few MCTP devices would be used on a given SMBus/I2C bus. Most devices would be non-intelligent sensor and I/O devices instead. Conservatively, it is recommended that at least four entries be provided for each SMBus/I2C bus that the device owns.

Example 1: "client" capable device

Four PCIe slots	→	16 routing table entries
Two owned SMBus/I2C busses	→	+8 entries
Static PCIe device support	→	+8 entries
		~32 entries or more

Example 2: volume server capable

Eight PCIe slots	→	32 routing table entries
Four owned SMBus/I2C busses	→	+16 entries
Static PCIe device support	→	+8 entries
		~56 entries or more

9.5 Path and Transmission Unit Discovery

The transmission unit is defined as the size of the MCTP packet payload that is supported for use in MCTP message assembly for a given message. The supported transmission unit sizes are allowed to vary on a per-message type basis.

Intermediate bridges and physical media can limit the transmission unit sizes between endpoints. Therefore, the MCTP control protocol specifies a mechanism for discovering the transmission unit support for the path between endpoints when one or more bridges exist in the path between the endpoints.

The mechanism for path transmission unit discovery also enables the discovery of the bridges and number of "hops" that are used to route an MCTP packet from one endpoint to another.

9.5.1 Path Transmission Unit Negotiation

The MCTP control protocol only specifies how to discover what the path transmission unit size is for the path between endpoints. The MCTP control protocol does not specify a generic mechanism for discovering what transmission unit sizes a particular endpoint supports for a given message type. Discovery and negotiation of transmission unit sizes for endpoints, if supported, is specified by the definition of the particular message type.

9.5.2 Path Transmission Unit Discovery Process Overview

This clause describes the process used for path transmission unit discovery. The discovery process described here is designed to enable one endpoint to discover the path and transmission unit support for accessing a particular "target" endpoint. It does not define a general mechanism for enabling an endpoint to discover the path between any two arbitrary endpoints. For example, referring to Figure 17, the process defines a way for the endpoint at EID 9 to discover the path/transmission unit support on the route to endpoint at EID 14, but this process does not define a process for EID 9 to discover the path/transmission unit support between EID 11 and EID 14.

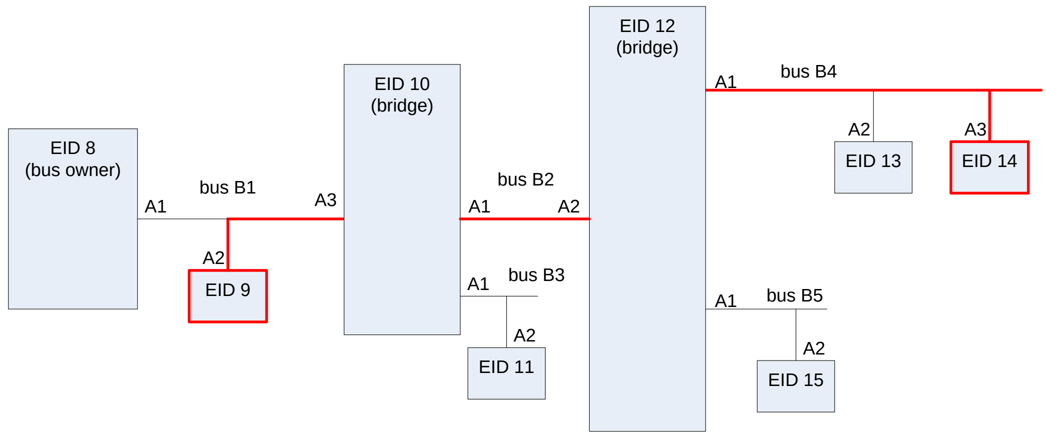


Figure 17 – Example Path Routing Topology

The following example provides an overview of the path/transmission unit discovery process. The example presumes that the MCTP network has already been initialized. Referring to Figure 17, the endpoint with EID 9 wishes to discover the path used to access the endpoint with EID 14. This discovery is accomplished using just two commands, Resolve Endpoint ID and Query Hop, as follows:

1) EID 9 first issues a Resolve Endpoint ID command to the bus owner, EID 8, with EID 14 as the EID to resolve.

2) EID 8 returns the physical address and EID of the bridge, EID 10 in the Resolve Endpoint ID command response.

3) EID 9 queries the bridge, EID 10, using a Query Hop command with EID 14 (the "target" EID) as the request parameter. Note that EID 2 does not need to do another Resolve Endpoint ID command because it already received the physical address of EID 3 from the original Resolve Endpoint ID command.

4) Bridge EID 10 responds to the Query Hop command by returning EID 12, which is the EID of the next bridge required to access EID 14. The bridge EID 10 also returns the transmission unit support that it offers for routing to the target EID.

5) EID 9 then sends a Query Hop command to the bridge at EID 12. Note that EID 9 does not need to do another Resolve Endpoint ID command because it already received the physical address of EID 12 from the original Resolve Endpoint ID command.

6) Bridge EID 12 responds to the Query Hop command by returning EID 14, which, because it is the EID of the target endpoint, tells EID 9 that bridge EID 12 was the last "hop" in the path to EID 6. The bridge EID 5 also returns the transmission unit support that it offers for routing to the target EID.

7) At this point, the bridges in the path to EID 14 have subsequently been discovered and their respective transmission unit support returned. The effective transmission unit support for the path to EID 14 will be the lesser of the transmission unit support values returned by the two bridges.

9.5.3 Path Transmission Unit Discovery Process Flowchart

The following flowchart (Figure 15) shows a generic algorithm for discovering the bridges in the path from one endpoint to a given target endpoint and the path transmission unit support. The flowchart has been intentionally simplified. Note that while the Query Hop command actually supports returning separate transmission unit sizes for the transmit and receive paths, the flowchart is simplified for illustration purposes and just refers to a single transmission unit for both transmit and receive.

Additionally, Figure 18 does not show any explicit steps for error handling nor the process of handling command retries. In general, errors are most likely due to either an invalid EID being sent to the bridge (perhaps due to a programming error at the requester) or the EID not being present in the bridge's routing table. The latter condition could occur under normal operation if the requester did not realize that a routing table update had occurred because of a hot-plug update, for example. This error condition would be indicated by the bridge responding with an `ERROR_INVALID_DATA` completion code.

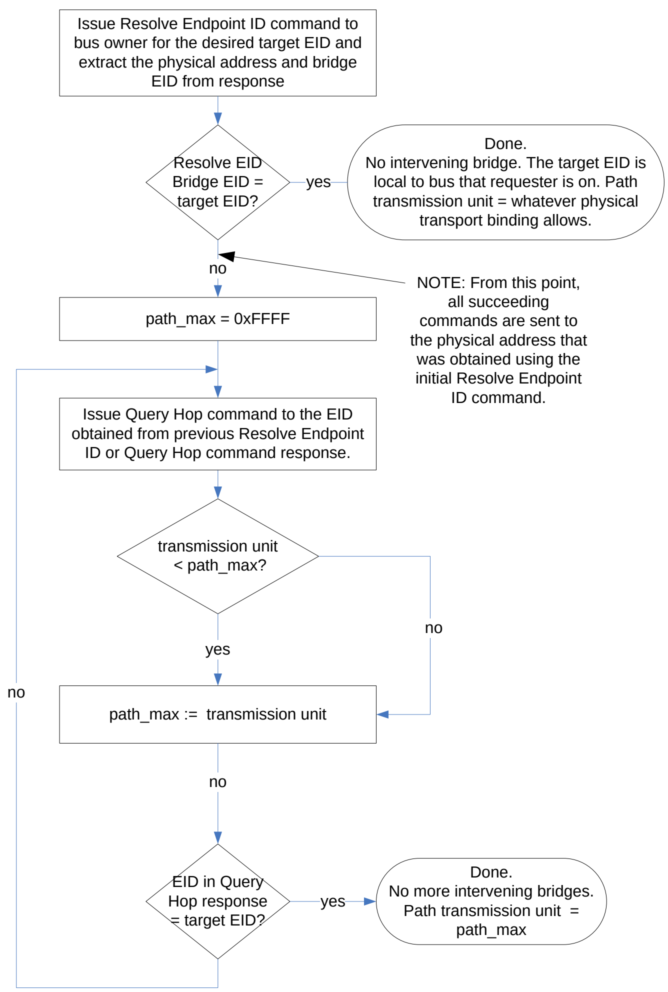


Figure 18 – Path Transmission Unit Discovery Flowchart

9.6 Path Transmission Unit Requirements for Bridges

An MCTP bridge routes packets between different buses, but it does not typically interpret the packet payload contents nor does it do assembly of those packets. Exceptions to this are when the bridge is handling packets addressed to its own EID, receives a Broadcast EID, and if the bridge supports different transmission units based on message type. See Table 32 for more information.

10 MCTP Control Protocol

MCTP control messages are used for the setup and initialization of MCTP communications within an MCTP network. This clause defines the protocol and formatting used for MCTP control messages over MCTP.

10.1 Terminology

The terms shown in Table 8 are used when describing the MCTP control protocol.

Table 8 – MCTP Control Protocol Terminology

Term	Description
Requester	The term "requester" is used to refer to the endpoint that originates an MCTP control Request message.
Responder	The term "responder" is used to refer to the endpoint that originates an MCTP control response message (that is, an endpoint that returns the response to an MCTP control Request message).
Originator or Source	The term "originator" or "source" is used to refer to the endpoint that originates any MCTP control message: Request, Response, or Datagram.
Target or Destination	The term "target" or "destination" is used to refer to the endpoint that is the intended recipient of any MCTP control message: Request, Response, or Datagram.
Asynchronous Notification	The term "asynchronous notification" is used to refer to the condition when an MCTP endpoint issues an un-requested Datagram to another MCTP endpoint.
Broadcast	The term "broadcast" is used when an MCTP control Datagram is sent out onto the bus using the broadcast EID.

10.1.1 Control Message Classes

The different types of messages shown in Table 9 are used under the MCTP control message type.

Table 9 – MCTP Control Message Types

Type	Description
Request	This class of control message requests that an endpoint perform a specific MCTP control operation. All MCTP control Request messages are acknowledged with a corresponding Response message. (Within this specification, the term "command" and "request" are used interchangeably as shorthand to refer to MCTP control Request messages.)
Response	This class of MCTP control message is sent in response to an MCTP control Request message. The message includes a "Completion Code" field that indicates whether the response completed normally. The response can also return additional data dependent on the particular MCTP control Request that was issued.

Type	Description
Datagram	Datagrams are "unacknowledged" messages (that is, Datagrams do not have corresponding Response messages). This class of MCTP control message is used to transfer messages when an MCTP control Response message is neither required nor desirable.
Broadcast Request	A broadcast message is a special type of Request that is targeted to all endpoints on a given bus. All endpoints that receive the message are expected to interpret the Request.
Broadcast Datagram	A Datagram that is broadcast to all endpoints on the bus. Broadcast Datagrams are "unacknowledged" messages (that is, broadcast Datagrams do not have corresponding Response messages).

10.2 MCTP Control Message Format

MCTP control messages use the MCTP control message type (see Table 3). Any message sent with this message type will correspond to the definitions set forth in this clause. The basic format of an MCTP control message is shown in Figure 19. Note that the byte offsets shown in Figure 19 are relative to the start of the MCTP message body rather than the start of the physical packet.

10.2.1 Use of Message Integrity Check

MCTP control messages do not use a Message Integrity Check field. Therefore, the IC bit in MCTP control messages shall always be `0b`.

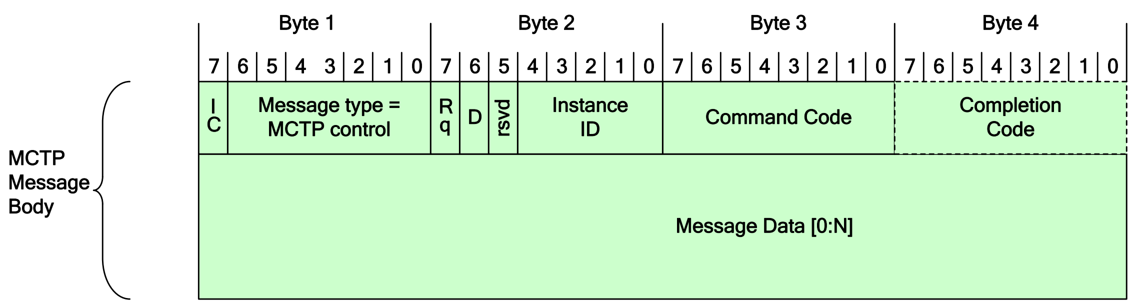


Figure 19 – MCTP Control Message Format

10.3 MCTP Control Message Fields

Table 10 lists the common fields for MCTP control messages.

Table 10 – MCTP Control Message Fields

Field Name	Description
IC*	Message Integrity Check bit = `0b`. MCTP control messages do not include an overall Message Integrity check field.
Message Type*	MCTP control = `0x00` (`000_0000b`). This field identifies the MCTP message as being an MCTP control message.
Rq bit	Request bit. This bit is used to help differentiate between MCTP control Request messages and other message classes. Refer to 10.5.

Field Name	Description
D-bit	Datagram bit. This bit is used to indicate whether the Instance ID field is being used for tracking and matching requests and responses, or is just being used to identify a retransmitted message. Refer to 10.5.
Instance ID	The Instance ID field is used to identify new instances of an MCTP control Request or Datagram to differentiate new requests or datagrams that are sent to a given message terminus from retried messages that are sent to the same message terminus. The Instance ID field is also used to match up a particular instance of an MCTP Response message with the corresponding instance of an MCTP Request message.
Command Code	For Request messages, this field is a command code indicating the type of MCTP operation the packet is requesting. Command code values are defined in Table 12. The format and definition of request and response parameters for the commands is given in Clause 11. The Command Code that is sent in a Request must be returned in the corresponding Response.
Completion Code	This field is only present in Response messages. This field contains a value that indicates whether the response completed normally. If the command did not complete normally, the value can provide additional information regarding the error condition. The values for completion codes are specified in Table 13.
Message Data	Zero or more bytes of parameter data that is specific to the particular Command Code and whether the message is a Request or Datagram, or a Response.
* These fields are MCTP base protocol fields.	

10.4 MCTP Control Message Transmission Unit Size

All MCTP control messages are required to have a packet payload that is no larger than the baseline transmission unit size of 64 bytes.

MCTP control messages are carried in a single MCTP packet. Multiple messages are used if an operation requires more data to be transferred than can be carried in a single message.

10.5 Tag Owner (TO), Request (Rq), and Datagram (D) Bit Usage

For MCTP control messages, the Rq bit shall be set to `1b` if the message is a "command" or Request message and `0b` if the message is a Response message. For Datagram and Broadcast messages, the Rq bit shall always be set to `1b`. MCTP Control messages that have unexpected or incorrect flag bit values shall be silently discarded by the receiver of the message.

For the present specification, Requests and Datagrams are only issued from tag owners (TO bit = `1b`). Provision has been left for the definition of possible future Datagrams that are not issued from tag owners (see Table 11).

Table 11 – Tag Owner (TO), Request (Rq) and Datagram (D) Bit Usage

MCTP Control Message Class	Destination EID Value	Tag Owner (TO) bit	Request (Rq) bit	Datagram (D) bit
Command/Request Responses are expected and tracked by Instance ID at the requester.	Target EID	`1b`	`1b`	`0b`
Response	Target EID	`0b`	`0b`	`0b`

MCTP Control Message Class	Destination EID Value	Tag Owner (TO) bit	Request (Rq) bit	Datagram (D) bit
Broadcast Request Responses are expected and tracked by Instance ID at the requester.	Broadcast EID	1b	1b	0b
Datagram Unacknowledged Request – Responses are neither expected nor tracked by Instance ID at the requester. Duplicate packets are handled the same as retried Command/Request packets.	Target EID	1b	1b	1b
Broadcast Datagram (unacknowledged control command that is broadcast.)	Broadcast EID	1b	1b	1b
Reserved for future definition	all other			

10.6 Concurrent Command Processing

This clause describes the specifications and requirements for handling concurrent overlapping MCTP control requests by endpoints.

10.6.1 Requirements for Responders

An endpoint is not required to process more than one request at a time (that is, it can be "single threaded" and does not have to accept and act on new requests until it has finished responding to any previous request).

A responder that is not ready to accept a new request can either silently discard the request, or it can respond with an `ERROR_NOT_READY` message completion code.

A responder that can accept and process more than one request at a time is not required to return responses in the order that the requests were received.

10.6.2 Requirements for Requesters

An endpoint that issues MCTP control Requests to another endpoint must wait until it gets the response to the particular request, or times out waiting for the response, before issuing a new request, Datagram, or Broadcast Datagram.

An endpoint that issues MCTP control Requests is allowed to have multiple requests outstanding simultaneously to *different* responder endpoints.

An endpoint that issues MCTP control Requests should be prepared to handle responses that may not match the request (that is, it should not automatically assume that a response that it receives is for a particular request). It should check to see that the command code and source EID values in the response match up with a corresponding outstanding command before acting on any parameters returned in the response.

10.6.3 Additional Requirements for Bridges

The packets that are routed *through* a bridge's routing functionality are not interpreted by the bridge and therefore are not considered to constitute concurrent requests.

A bridge must support at least one outstanding MCTP control request for each bus connection (port) through which MCTP control messages can be used to access the bridge's configuration and control functionality.

Bridges must retain temporal ordering of packets forwarded from one message terminus to another.

11 MCTP Control Messages

This clause contains detailed descriptions for each MCTP control message. The byte offsets for the Request and Response parameter information given in the tables for the commands indicates the byte offset for the message data starting with the byte following the Command field.

11.1 MCTP Control Message Command Codes

Table 12 lists the MCTP control messages and their corresponding command code values. The commands and their associated parameters are specified later in this clause. For bridges, the requirements apply equally to all endpoints within the bridge device that are used to configure and control the bridges routing functionality.

Table 12 – MCTP Control Command Numbers

Command Code	Command Name	General Description	OMC		Clause
			E	B	
`0x00`	Reserved	Reserved	–	–	–
`0x01`	Set Endpoint ID	Assigns an EID to the endpoint at the given physical address	Ma Ng	Ca[1] Mg	11.3
`0x02`	Get Endpoint ID	Returns the EID presently assigned to an endpoint. Also returns information about what type the endpoint is and its level of use of static EIDs.	Ma Og	Ma Og	11.4
`0x03`	Get Endpoint UUID	Retrieves a per-device unique UUID associated with the endpoint	Ca[2] Og	Ca[2] Og	11.5
`0x04`	Get MCTP Version Support	Lists which versions of the MCTP control protocol are supported on an endpoint	Ma Og	Ma Og[5]	11.6
`0x05`	Get Message Type Support	Lists the message types that an endpoint supports	Ma Og	Ma Og	11.7
`0x06`	Get Vendor Defined Message Support	Used to discover an MCTP endpoint's vendor-specific MCTP extensions and capabilities	Oa Og	Oa Og	11.8
`0x07`	Resolve Endpoint ID	Used to get the physical address associated with a given EID	Na Og	Ma Og	11.9
`0x08`	Allocate Endpoint IDs	Used by the bus owner to allocate a pool of EIDs to an MCTP bridge	Na Ng	Ma[6] Mg[6]	11.10
`0x09`	Routing Information Update	Used by the bus owner to extend or update the routing information that is maintained by an MCTP bridge	Oa[8] Og[8]	Ma[4] Mg[4]	11.11
`0x0A`	Get Routing Table Entries	Used to request an MCTP bridge to return data corresponding to its present routing table entries	Na Og	Ma Og	11.12
`0x0B`	Prepare for Endpoint Discovery	Used to direct endpoints to clear their "discovered" flags to enable them to respond to the Endpoint Discovery command	Ca[3] Ng	Ca[3] Cg[3]	11.13

Command Code	Command Name	General Description	OMC		Clause
			E	B	
0x0C	Endpoint Discovery	Used to discover MCTP-capable devices on a bus, provided that another discovery mechanism is not defined for the particular physical medium	Ca[3] Cg[3]	Ca[3] Cg[3]	11.14
0x0D	Discovery Notify	Used to notify the bus owner that an MCTP device has become available on the bus	Na Cg[3]	Ca[3] Cg[3]	11.15
0x0E	Get Network ID	Used to get the MCTP network ID	Ca[7]	Ca[7]	11.16
0x0F	Query Hop	Used to discover what bridges, if any, are in the path to a given target endpoint and what transmission unit sizes the bridges will pass for a given message type when routing to the target endpoint	Na Og	Ma Og	11.17
0x10	Resolve UUID	Used by endpoints to find another endpoint matching an endpoint that uses a specific UUID.	Na Og	Oa Og	11.18
0xF0 – 0xFF	Transport Specific	This range of control command numbers is reserved for definition by individual MCTP Transport binding specifications. Transport specific commands are intended to be used as needed for setup and configuration of MCTP on a given media. A particular transport specific command number many have different definitions depending on the binding specification. Transport specific commands shall only be addressed to endpoints on the same medium. A bridge is allowed to block transport specific commands from being bridged to different media. The general format of Transport specific messages is specified in clause 11.17.	-	-	11.19

Key for OMC (optional / mandatory / conditional) column:

E = non-bridge, non-bus owner endpoint (simple endpoint)
B = bridge / bus-owner endpoint
Ma = mandatory (required) to accept. The request must be accepted by the endpoint and a response generated per the following command descriptions.
Mg = mandatory to generate. The endpoint must generate this request as part of its responsibilities for MCTP operation.
Oa = optional to accept
Og = optional to generate
Ca = conditional to accept (see notes)
Cg = conditional to generate (see notes)
Na = not applicable to accept. This command is not applicable to the device type and must not be accepted
Ng = not applicable to generate. This command is used for MCTP configuration and initialization and should not be generated.

1. The topmost bus owner is not required to support the Set Endpoint ID command.2. Hot-plug and add-in devices, and non-bridge devices that connect to multiple busses, are required to support the Get Endpoint UUID command. See 8.17.7 and 8.17.8 for more info.3. Mandatory on a per-bus basis to support endpoint discovery if required by the physical transport binding used for the particular bus type. Refer to the appropriate MCTP physical transport binding specification.
4. The topmost bus owner is not required to accept this command. The command is required to be generated when downstream bridges require dynamic routing information from bus owners that they are connected to. Some implementations may be configured where all routing information has been statically configured into the bridge and no dynamically provided information is required, In this case, it is not required to support the command while the endpoints are configured in that manner.
5. Bridges should use this command to verify that they are initializing devices that are compatible with their MCTP control protocol version.
6. The endpoint is required to accept this command if it indicated support for a dynamic EID pool. The command must be generated by the endpoint if the configuration requires the endpoint to support allocating EID pools to downstream bridges.

Command Code	Command Name	General Description	OMC		Clause
			E	B	

7. See Clause 9 MCTP Network IDs for information for implementation requirements of this command.
8. While it is optional for an endpoint to receive a routing information update, the MCTP Base specification does not specify a bridge or bus owner function that sends such updates to particular endpoints.

11.2 MCTP Control Message Completion Codes

The command/result code field is used to return management operation results for response messages. If a SUCCESS completion code is returned then the specified response parameters (if any) must also be returned in the response. If an error completion code (not SUCCESS) is returned by the responder, unless otherwise specified, the responder shall not return any additional parametric data and the requester shall ignore any additional parameter data provided in the response (if any). See Table 13 for the completion codes.

Table 13 – MCTP Control Message Completion Codes

Value	Name	Description
0x00	SUCCESS	The Request was accepted and completed normally.
0x01	ERROR	This is a generic failure message. (It should not be used when a more specific result code applies.)
0x02	ERROR_INVALID_DATA	The packet payload contained invalid data or an illegal parameter value.
0x03	ERROR_INVALID_LENGTH	The message length was invalid. (The Message body was larger or smaller than expected for the particular request.)
0x04	ERROR_NOT_READY	The Receiver is in a transient state where it is not ready to receive the corresponding message.
0x05	ERROR_UNSUPPORTED_CMD	The command field in the control type of the received message is unspecified or not supported on this endpoint. This completion code must be returned for any unsupported command values received in MCTP control Request messages.
0x80–0xFF	COMMAND_SPECIFIC	This range of completion code values is reserved for values that are specific to a particular MCTP control message. The particular values (if any) and their definition is provided in the specification for the particular command.
all other	Reserved	Reserved

11.3 Set Endpoint ID

The Set Endpoint ID command assigns an EID to an endpoint. This command should only be issued by a bus owner to assign an EID to an endpoint at a particular physical address. Since it is assumed the Endpoint does not already have an EID assigned to it, or because the EID is unknown, the destination EID in the message will typically be set to the special null destination EID value.

The Set Endpoint ID command is also used to provide the Physical Address and EID of the Bus Owner to an Endpoint. An Endpoint that needs to communicate with the Bus Owner may capture the physical address and EID that was used to deliver the Set Endpoint ID message.

Note: Endpoints that are not the Bus Owner should not issue the Set Endpoint ID command because it can cause the receiver of the message to capture incorrect information for the Bus Owner's address.

An MCTP bridge may elect to have a single EID for its functionality, rather than using an EID for each port (bus connection) that is connected to a different bus owner. See 9.1.2 for more information. In this case, the bridge will accept its EID assignment from the "first" bus to deliver the Set Endpoint ID request to the bridge.

It is recognized that different internal processing delays within a bridge can cause the temporal ordering of requests to be switched if overlapping requests are received over more than one bus. Therefore, which request is accepted by an implementation is not necessarily tied to the request that is first received at the bridge, but instead will be based on which request is the first to be processed by the bridge.

If an EID has already been assigned and the Set Endpoint ID command is issued from a different bus without forcing an EID assignment, the command shall return a SUCCESSFUL completion code, but the response parameters shall return an EID assignment status of "EID rejected".

The Set Endpoint ID command functions in the same manner regardless of whether the endpoint uses a static EID. The only difference is that if an endpoint has a static EID, it uses that EID as its initial "default" EID value. The endpoint does not treat this initial EID as if it were assigned to it by a different bus owner. That is, the endpoint shall accept the EID assignment from the first bus that the command is received from, and shall track that bus as the originating bus for the EID for subsequent instances of Set Endpoint ID command. See 8.17.2 for more information. The request and response parameters are specified in Table 14.

Table 14 – Set Endpoint ID Message

	Byte	Description
Request data	1	Operation [7:3] – reserved [1:0] – Operation: 00b Set EID. Submit an EID for assignment. The given EID will be accepted conditional upon which bus the device received the EID from (see preceding text). A device where the endpoint is only reached through one bus shall always accept this operation (provided the EID value is legal). 01b Force EID. Force EID assignment. The given EID will be accepted regardless of whether the EID was already assigned through another bus. Note that if the endpoint is forcing, the EID assignment changes which bus is being tracked as the originator of the Set Endpoint ID command. A device where the endpoint is only reached through one bus shall always accept this operation (provided the EID value is legal), in which case the Set EID and Force EID operations are equivalent. 10b Reset EID (optional). This option only applies to endpoints that support static EIDs. If static EIDs are supported, the endpoint shall restore the EID to the statically configured EID value. The EID value in byte 2 shall be ignored. An ERROR_INVALID_DATA completion code shall be returned if this operation is not supported. 11b Set Discovered Flag. Set Discovered flag to the "discovered" state only. Do not change present EID setting. The EID value in byte 2 shall be ignored. Note that Discovered flag is only used for some physical transport bindings. An ERROR_INVALID_DATA completion code shall be returned if this operation is selected and the particular transport binding does not

	Byte	Description
		support a Discovered flag.
	2	Endpoint ID. 0xFF, 0x00 = illegal. Endpoints are not allowed to be assigned the broadcast or null EIDs. It is recommended that the endpoint return an ERROR_INVALID_DATA completion code if it receives either of these values.
Response data	1	Completion code
	2	[7:6] – reserved [5:4] – EID assignment status: 00b = EID assignment accepted. 01b = EID assignment rejected. EID has already been assigned by another bus owner and assignment was not forced. 10b = reserved. 11b = reserved. [3:2] – reserved. [1:0] – Endpoint ID allocation status (see 11.10 for additional information): 00b = Device does not use an EID pool. 01b = Endpoint requires EID pool allocation. 10b = Endpoint uses an EID pool and has already received an allocation for that pool. 11b = reserved
	3	EID Setting. If the EID setting was accepted, this value will match the EID passed in the request. Otherwise, this value returns the present EID setting.
	4	EID Pool Size. This is the size of the dynamic EID pool that the bridge can use to assign EIDs or EID pools to other endpoints or bridges. It does not include the count of any additional static EIDs that the bridge may maintain. See 8.17.2 for more information. Note that a bridge always returns its pool size regardless of whether it has already received an allocation. 0x00 = no dynamic EID pool.

11.4 Get Endpoint ID

The Get Endpoint ID command returns the EID for an endpoint. This command is typically issued only by a bus owner to retrieve the EID that was assigned to a particular physical address. Thus, the destination EID in the message will typically be set to the special Physical Addressing Only EID value. The request and response parameters are specified in Table 15.

Table 15 – Get Endpoint ID Message

	Byte	Description
Request data	–	–
Response data	1	Completion Code.
	2	Endpoint ID.

	Byte	Description
		`0x00` = EID not yet assigned.
	3	Endpoint Type. [7:6] = reserved [5:4] = Endpoint Type: `00b` = simple endpoint `01b` = bus owner/bridge `10b` = reserved `11b` = reserved [2:0] = reserved [1:0] = Endpoint ID Type: 00b = dynamic EID. The endpoint uses a dynamic EID only. `01b` = static EID supported. The endpoint was configured with a static EID. The EID returned by this command reflects the present setting and may or may not match the static EID value. The following two status return values are optional. If provided, they must be supported as a pair in place of the static EID support status return. It is recommended that this be implemented if the Reset EID option in the Set Endpoint ID command is supported. `10b` = static EID supported. Present EID matches static EID. The endpoint has been configured with a static EID. The present value is the same as the static value. `11b` = static EID supported. Present EID does not match static EID. Endpoint has been configured with a static EID. The present value is different than the static value. See 8.17.2 for more information.
	4	Medium-Specific Information. This byte can hold additional information about optional configuration of the endpoint on the given medium, such as whether certain types of timing or arbitration are supported. This should only be used to report static information. This byte shall be returned as `0x00` unless otherwise specified by the transport binding.

11.5 Get Endpoint UUID

The Get Endpoint UUID command returns a universally unique identifier (UUID), also referred to as a globally unique ID (GUID), for the management controller or management device. The command can be used to correlate a device with one or more EIDs. The format of the ID follows the byte (octet) format specified in RFC4122. RFC4122 specifies four different versions of UUID formats and generation algorithms suitable for use for a device UUID in IPMI. These are version 1 (0001b) "time based", and three "name-based" versions: version 3 (0011b) "MD5 hash", version 4 (0100b) "Pseudo-random", and version 5 "SHA1 hash". The version 1 format is recommended. However, versions 3, 4, or 5 formats are also allowed. A device UUID should never change over the lifetime of the device. The request and response parameters are specified in Table 16.

See 8.17.7 and 8.17.8 for additional requirements on the use of the Get Endpoint UUID command.

Table 16 – Get Endpoint UUID Message Format

	Byte	Description
Request data	–	–
Response data	1	Completion Code
	2:17	UUID bytes 1:16, respectively (see Table 17)

The individual fields within the UUID are stored most-significant byte (MSB) first per the convention described in RFC4122. See Table 17 for an example format.

Table 17 – Example UUID Format

Field	UUID Byte	MSB
time low	1	MSB
	2	
	3	
	4	
time mid	5	MSB
	6	
time high and version	7	MSB
	8	
clock seq and reserved	9	MSB
	10	
node	11	MSB
	12	
	13	
	14	
	15	
	16	

11.6 Get MCTP Version Support

This command can be used to retrieve the MCTP base specification versions that the endpoint supports, and also the message type specification versions supported for each message type. The format of the request and response parameters for this message is given in Table 18.

More than one version number can be returned for a given message type by the Get MCTP Version Support command. This enables the command to be used for reporting different levels of compatibility and backward compatibility with different specification versions. The individual specifications for the given message type define the requirements for which versions number values should be used for that message type. Those documents define which earlier version numbers, if any, must also be listed.

The command returns a completion code that indicates whether the message type number passed in the request is supported or not. This enables the command to also be used to query the endpoint for whether it supports a given message type.

NOTE: Version numbers are listed from oldest to newest. Versioning commands and version formats for vendor-defined message types, 0x7E and 0x7F, are vendor-specific and considered outside the scope of this specification.

Table 18 – Get MCTP Version Support Message

	Byte	Description
Request data	1	Message Type Number The Message Type Number to retrieve version information for: `0xFF` = return MCTP base specification version information. `0x7E, 0x7F` = unspecified. Support of this command for vendor-defined message types is vendor implementation-specific and considered outside the scope of this specification. `0x00` = return MCTP control protocol message version information. other = return version information for a given message type. See MCTP ID for message type numbers.
Response data	1	Completion Code `0x80` = message type number not supported
	2	Version Number Entry count One-based count of 32-bit version numbers being returned in this response. Numerically lower version numbers are returned first.
	3:6	Version Number entry 1: The following descriptions are informational. Refer to DSP4014 for the normative definition of version numbering of DMTF specifications. [31:24] = major version number. This field is used to identify a version of the specification that includes changes that make it incompatible with one or more functions that were defined in versions of the specification that have an older (smaller) major version number. [23:16] = minor version number. This field is used to identify functional additions to the specification that are backward compatible with older (smaller) minor version numbers that share the same major version number. [15:8] = update version number. This field is used for editorial updates to the specification that do not define new functionality nor change existing functionality over the given major.minor release. This field is informational and should be ignored when checking versions for interoperability. [7:0] = "alpha" byte. This value is used for pre-release (work-in-progress) versions of the specification. Pre-release versions of the specification are backward compatible with specification versions that have an older (smaller) minor version numbers that share the same major version number. However, since the alpha value represents a version of the specification that is presently under development, versions that share the same major and minor

	Byte	Description
		version numbers, but have different 'alpha' versions may not be fully interoperable. The encoding of the version number and alpha fields is provided in 11.6.1.
	(7:X)	Version Number Entries 2 through N. Additional 32-bit major/minor version numbers, if any.

11.6.1 Version Field Encoding

The version field is comprised of four bytes referred to as the "major", "minor", "update", and "alpha" bytes. These bytes shall be encoded as follows:

The "major", "minor", and "update" bytes are BCD-encoded, and each byte holds two BCD digits. The "alpha" byte holds an optional alphanumeric character extension that is encoded using one of the alphabetic characters [a-z, A-Z] from the US-ASCII (RFC20) Character Set. The semantics of these fields follows that specified in DSP4014.

The value `0x00` in the alpha field means that the alpha field is not used. Software or utilities that display the version number should not display any characters for this field.

The value `0xF` in the most-significant nibble of a BCD-encoded value indicates that the most-significant nibble should be ignored and the overall field treated as a single-digit value. Software or utilities that display the number should only display a single digit and should not put in a leading "0" when displaying the number.

A value of `0xFF` in the "update" field indicates that the field to be ignored. Software or utilities that display the version number should not display any characters for the field. `0xFF` is not allowed as a value for the "major" or "minor" fields.

EXAMPLES:

Version 1.1.0 → `0xF1F1F000`

Version 3.1 → `0xF3F1FF00`

Version 1.0a → `0xF1F0FF61`

Version 3.7.10a → `0xF3F71061`

Version 10.11.7 → `0x1011F700`

11.6.2 MCTP Base Specification Version Number

MCTP implementations that follow this particular specification shall return the following version information in the response to the Get MCTP Version Support message when the Message Type parameter in the request is set to `0xFF` (return MCTP base specification version information).

The Version Number Entry 1 field shall be used to indicate backward compatibility with Version 1.0 of the base specification as:

1.0 [Major version 1, minor version 0, no update version, no alpha)]

This is reported using the encoding as: `0xF1F0FF00`

The Version Number Entry 2 field shall be used to indicate backward compatibility with Version 1.1 of the base specification as:

1.1.0 [Major version 1, minor version 1, update version 0, no alpha)]

This is reported using the encoding as: `0xF1F1F000`

The version of the MCTP base specification for this specification shall be reported in Version Number Entry 3 as:

1.2.0 [Major version 1, minor version 2, update version 0, no alpha)]

This is reported using the encoding as: `0xF1F2F000`

11.6.3 MCTP Control Protocol Version Information

MCTP implementations that follow this particular specification shall return the following version information in the response to the Get MCTP Version Support message when the Message Type parameter in the request is set to `0x00` (return MCTP control protocol version information).

The Version Number Entry 1 field shall be used to indicate backward compatibility with Version 1.0 of the base specification Control Protocol as:

1.0 [Major version 1, minor version 0, no update version, no alpha)]

This is reported using the encoding as: `0xF1F0FF00`

The Version Number Entry 2 field shall be used to indicate backward compatibility with Version 1.1 of the base specification Control Protocol as:

1.1.0 [Major version 1, minor version 1, update version 0, no alpha)]

This is reported using the encoding as: `0xF1F1F000`

The version of the MCTP base specification Control Protocol for this specification shall be reported in Version Number Entry 3 as:

1.2.0 [Major version 1, minor version 2, update version 0, no alpha)]

This is reported using the encoding as: `0xF1F2F000`

11.7 Get Message Type Support

The Get Message Type Support command enables management controllers to discover the MCTP control protocol capabilities supported by other MCTP endpoints, and get a list of the MCTP message types that are supported by the endpoint. The request and response parameters for this message are listed in Table 19.

The response to this command may be specific according to which bus the request was received over (that is, a device that supports a given message type may not support that message type equally across all buses that connect to the device).

Table 19 – Get Message Type Support Message

	Byte	Description
Request data	–	–
Response data	1	Completion Code.
	2	MCTP Message Type Count. One-based. Number of message types in addition to the MCTP control message type that is supported by this endpoint
	(3:N)	List of Message Type numbers. One byte per number. See Table 3 and MCTP ID.

11.8 Get Vendor Defined Message Support

The Get Vendor Defined Message Support operation enables management controllers to discover whether the endpoint supports vendor-defined messages, and, if so, the vendors or organizations that defined those messages. The format and definition of the request and response parameters for this message is given in Table 20.

Table 20 – Get Vendor Defined Message Support Message

	Byte	Description
Request data	1	Vendor ID Set Selector Indicates the specific capability set requested. Indices start at `0x00` and increase monotonically by 1. If the responding endpoint has one or more capability sets with indices greater than the requested index, it increments the requested index by 1 and returns the resulting value in the response message. The requesting endpoint uses the returned value to request the next capability set.
Response data	1	Completion Code
	2	Vendor ID Set Selector `0xFF` = no more capability sets.
	var	Vendor ID A structured field of variable length that identifies the vendor ID format (presently PCI or IANA) and the ID of the vendor that defined the capability set. The structure of this field is specified in Figure 20 – Structure of Vendor ID Field for Get Vendor Defined Capabilities Message.
	2 bytes	16-bit numeric value or bit field, as specified by the vendor or organization identified by the vendor ID. This value is typically used to identify a particular command set type or major version under the given vendor ID.

11.8.1 Vendor ID Formats

Figure 20 shows the general structure of Vendor ID fields used in this specification. The first byte of the field contains the Vendor ID Format, a numeric value that indicates the definition space and format of the ID. The remainder of the field holds the Vendor ID Data with content and format as specified in Table 21.

The MCTP management controller or management device can pick which format is best suited for the device. In general, if the device does not already have an existing vendor ID that matches one of the specified formats, it is recommended that the IANA enterprise number format be used.

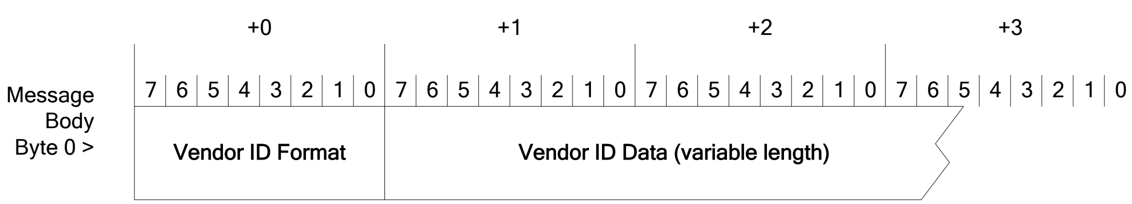


Figure 20 – Structure of Vendor ID Field for Get Vendor Defined Capabilities Message

Table 21 – Vendor ID Formats

Vendor ID Format Name	Vendor ID Format	Vendor ID Data Length	Description
PCI Vendor ID	`0x00`	2	**16-bit Unsigned Integer**. The PCI 2.3 specifications state the following about the PCI vendor ID: "This field identifies the manufacturer of the device. Valid vendor identifiers are allocated by the PCI SIG to ensure uniqueness. `0xFFFF` is an invalid value for the Vendor ID." However, for MCTP this value may be used for identifying aspects other than the manufacturer of the device, such as its use in the Vendor Defined - PCI message type, where it identifies the vendor or organization that defined a particular set of vendor-defined messages. Thus, in some uses, the ID may or may not correspond to the PCI ID for the manufacturer of the device.
IANA Enterprise Number	`0x01`	4	**32-bit Unsigned Integer**. The IANA enterprise number for the organization or vendor expressed as a 32-bit unsigned binary number. For example, the enterprise ID for the DMTF is 412 (decimal) or `0x0000_019C` expressed as a 32-bit hexadecimal number. The enterprise number is assigned and maintained by the Internet Assigned Numbers Authority, www.iana.org, as a means of identifying a particular vendor, company, or organization.

11.9 Resolve Endpoint ID

This command is sent to the bus owner to resolve an EID into the physical address that must be used to deliver MCTP messages to the target endpoint. The command takes an EID as an input parameter in the request and returns the EID and the physical address for routing to that EID (if any) in the response. The response data will also indicate if no mapping was available.

An endpoint knows the physical address of the bus owner by keeping track of which physical address was used when the endpoint received its EID assignment through the Set Endpoint ID command. The endpoint can send this command to the bus owner using the null destination EID value. This eliminates the need for the endpoint to also keep track of the EID of the bus owner. The request and response parameters are specified in Table 22.

Table 22 – Resolve Endpoint ID Message

	Byte	Description
Request data	1	Target Endpoint ID This is the EID that the bus owner is being asked to resolve.
Response data	1	Completion Code
	2	Bridge Endpoint ID This is the EID for the endpoint that is providing the bridging server (if any) that is required to access the target endpoint. If the EID being returned matches the same value as the target EID, it indicates that there is no bridging function that is required to access the target endpoint (that is, the target EID is local to the bus that the Resolve Endpoint ID request was issued over).

	Byte	Description
	3:N	Physical Address. The size of this field is dependent on the particular MCTP physical transport binding used for the bus that this data is being provided for. The size and format of this field is defined as part of the corresponding physical transport binding specification.

11.10 Allocate Endpoint IDs

Bus owners are responsible for allocating pools of EIDs to MCTP bridges that are lower in the bus hierarchy. This is done using the Allocate Endpoint IDs command. The EID for the bridge itself is assigned separately and is *not* part of the pool given with this command.

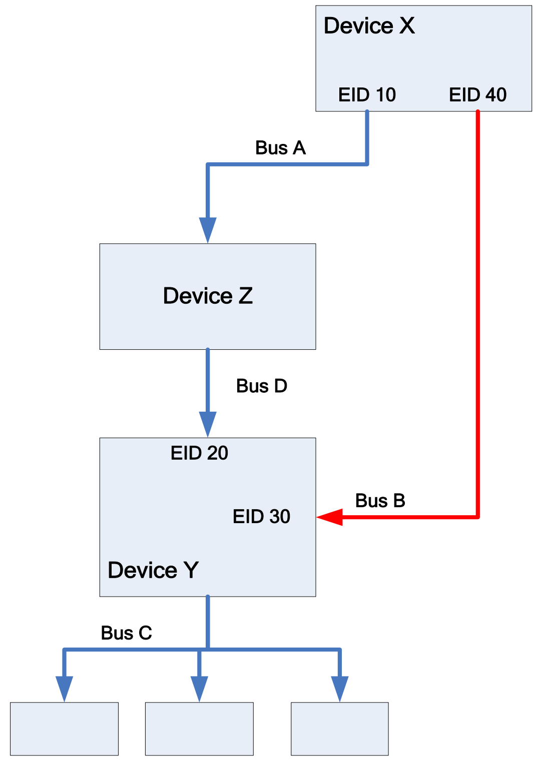
The bus owner will typically use this command as part of the EID assignment process for a bus. When a device has been assigned an EID using the Set Endpoint ID command, the response to that command indicates whether the endpoint supports an EID pool. If the device indicates that it supports an EID pool, the bus owner can then issue the Allocate Endpoint IDs command to supply the pool of EIDs to the device.

NOTE: The Allocate Endpoint IDs command can also cause a bridge to rebuild its routing table. See 11.11.2 for more information.

When an EID or EID pool that was previously allocated becomes unused (for example, due to a hot-swap removal), the bus owner must reclaim the endpoint's EID or EID pool allocation. See 8.17 for additional details.

Referring to Figure 21, there is a potential race condition with handling EID allocation. In the scenario shown in this figure, it is possible that device X and device Z might both be assigning EIDs to device Y at the same time. This also means that, unless steps are taken, device Z could allocate endpoints to device Y only to have this overwritten by a set of endpoints assigned by device X.

To prevent this, the Allocate Endpoint IDs command is only accepted from the "first" bus that provides the EID pool to the device. If another bus owner attempts to deliver an EID pool through another bus, the request will be rejected unless an intentional over-ride is done.

1899 

1900 **Figure 21 – EID Pools from Multiple Bus Owners**

1901 The Allocate Endpoint IDs message fields are described in Table 23.

1902 **Table 23 – Allocate Endpoint IDs Message**

	Byte	Description
Request data	1	Operation Flags: [7:1] – reserved. [1:0] – Operation: `00b` = Allocate EIDs. Submit an EID pool allocation. Do not force allocation. This enables the allocation to be rejected if the bridge has already received its EID pool from another bus. (See additional information in the following clauses.) `01b` = Force allocation. Force bridge to accept this EID pool regardless of whether it has already received its EID pool from another bus. This shall also cause a bridge to rebuild its routing tables, See 11.11.2 for more information. `10b` = Get allocation information Return the response parameters without changing the present allocation. This can be used to query information on the dynamic pool of EIDs presently allocated to the Endpoint, if any. If this operation is selected, the Number of Endpoint IDs and Starting Endpoint ID parameters in the request shall be ignored. `11b` = Reserved
	2	Number of Endpoint IDs (Allocated Pool Size)

	Byte	Description
		Specifies the number of EIDs in the pool being made available to this Endpoint Specifying a count of 0x00 shall be legal. If 0x00 is accepted or forced (and the bridge lacks a static EID pool) no EIDs shall be available for distribution by the particular bridge.
	3	Starting Endpoint ID Specifies the starting EID for the range of EIDs being allocated in the pool. When multiple EIDs are provided, the IDs are sequential starting with this value as the first EID in the range.
Response data	1	Completion Code An error completion code (ERROR_INVALID_DATA should be returned) shall be returned if the number of EIDs being allocated (Number of Endpoint IDs) exceeds the Dynamic Endpoint ID Pool size. (This error condition does not apply to when the number of endpoint IDs passed in the request is `0x00`).
	2	[7:2] – reserved [1:0] – `00b` = Allocation was accepted. In the case that the bridge has a completely static EID pool, the bridge should not track which bus has sourced the command and shall accept the allocation if the Number of Endpoint IDs (Allocated Pool Size) is 0x00. `01b` = Allocation was rejected. The Allocate Endpoint IDs command is accepted only from the "first" bus that provides the EID pool to the device. If another bus owner attempts to deliver an EID pool through another bus, the request will be rejected unless an intentional over-ride is done. (The rationale for this behavior is explained in the text of this clause.) `10b`, `11b` = reserved
	3	Endpoint ID Pool Size (Dynamic) This value is the size of the EID pool used by this endpoint. This is the size of the dynamic EID pool that the bridge can use to assign EIDs or EID pools to other endpoints or bridges. It does not include the count of any additional static EIDs that the bridge may maintain. See 8.17.2 for more information.
	4	First Endpoint ID This field specifies the first EID assigned to the pool for this endpoint. The value is `0x00` if there are no EIDs assigned to the pool.

11.11 Routing Information Update

The Routing Information Update message is used by a bus owner to give routing information to a bridge for the bus on which the message is being received.

Because the physical address format is based on the bus over which the request is delivered, the bus owner must use the medium-specific physical address format for the addresses sent using this command.

An MCTP bridge may be sent more than one instance of this command to transfer the update information. An integral number of routing information update entries must be provided in the command (that is, routing information update entries cannot be split across instances of the command).

11.11.1 Adding and Replacing Entries

The recipient of this command must check to see whether the information in the request corresponds to the EID for an existing entry for the bus over which the command was received. If so, it must replace that entry with the new information. If an entry for a given EID or EID range does not already exist, it must create new entries for the given EIDs. In some cases this may require the bridge to split existing entries into multiple entries.

NOTE: A bus owner is only allowed to update entries that correspond to its bus. For each routing table entry that was created or updated through the Routing Information Update message, the bridge must keep track of which bus it received the Routing Information Update from. This is necessary so that when a Routing Information Update is received from a particular bus, the bridge only updates entries that correspond to entries that were originally given to it from that bus.

11.11.2 Rebuilding Routing Tables

A bridge that receives and accepts the Allocate Endpoint IDs command with the "Force Allocation" bit set (1b) must clear out and rebuild its routing table information. The bridge shall issue commands to reassign EIDs and re-allocate EID pools to all downstream devices. The request and response parameters are specified in Table 24, and format information is provided in Table 25.

Table 24 – Routing Information Update Message

	Byte	Description
Request data	1	Count of update entries (1-based)
	see text	One or more update entries, based on the given count, as illustrated in Table 25
Response data	1	Completion Code `0x80` = Insufficient space to add requested entries to internal routing table

Table 25 – Routing Information Update Entry Format

Byte	Description
1	[7:4] - reserved [3:0] - Entry Type: `00b` = entry corresponds to a single endpoint that is not serving as an MCTP bridge `01b` = entry reflects an EID range for a bridge where the starting EID is the EID of the bridge itself and additional EIDs in the range are routed by the bridge `10b` = entry is for a single endpoint that is serving as an MCTP bridge `11b` = entry is an EID range for a bridge, but does not include the EID of the bridge itself
2	[7:0] Size of EID Range. The count of EIDs in the range.
3	First EID in EID Range. The EID Range is sequential (for example, if the size of the EID Range is 3 and the First EID value given in this parameter is 21, the Entry covers EIDs 21, 22, and 23).

Byte	Description
4:N	Physical Address. The size and format of this field is defined as part of the corresponding physical transport binding specification for the bus that this data is being provided for.

11.12 Get Routing Table Entries

This command can be used to request an MCTP bridge or bus owner to return data corresponding to its present routing table entries. This data is used to enable troubleshooting the configuration of routing tables and to enable software to draw a logical picture of the MCTP network. More than one instance of this command will typically need to be issued to transfer the entire routing table content.

An integral number of routing table entries must be provided in the response to this command (that is, routing table entries cannot be split across instances of the command). The request and response parameters are specified in Table 26, and format information is provided in Table 27.

Table 26 – Get Routing Table Entries Message

	Byte	Description
Request data	1	Entry Handle (`0x00` to access first entries in table)
Response data	1	Completion Code
	2	Next Entry Handle (Use this value to request the next set of entries, if any.) If the routing table data exceeds what can be carried in a single MCTP control response. `0xFF` = No more entries
	3	Number of routing table entries being returned in this response
	4:N	One or more routing table entries, formatted per Table 27. This field will be absent if the number of routing table entries is 0x00.

Table 27 – Routing Table Entry Format

Byte	Description
1	Size of EID range associated with this entry
2	Starting EID
3	Entry Type/Port Number [7:6] – Entry Type: `00b` = entry corresponds to a single endpoint that does not operate as an MCTP bridge `01b` = entry reflects an EID range for a bridge where the starting EID is the EID of the bridge itself and additional EIDs in the range are routed by the bridge `10b` = entry is for a single endpoint that serves as an MCTP bridge `11b` = entry is an EID range for a bridge, but does not include the EID of the bridge itself [5] – Dynamic/Static Entry. Indicates whether the entry was dynamically created or statically configured. Note that statically configured routing information shall not be merged with dynamic information when reporting entry information using this command. While an implementation may internally organize its data that way, dynamic and statically configured routing must be reported as separate entries. Dynamically created entries include entries that were generated from the Routing Information

Byte	Description
	Update command as well as entries that were created as a result of the bridge doing EID assignment and EID pool allocation as a bus owner. `0b` = Entry was dynamically created `1b` = Entry was statically configured [4:0] – Port number This value is chosen by the bridge device vendor and is used to identify a particular bus connection that the physical address for the entry is defined under. In some cases, this number may correspond to an internal "logical" bus that is not directly connected to an external physical bus. Port numbers are required to be static. It is recommended, but not required, that the ports (bus connections) on the bridge be numbered sequentially starting from `0x00`. This specification does not define any requirements or recommendations on how port numbers are assigned to corresponding physical connections on a device.
4	Physical Transport Binding Identifier, according to DSP0239.
5	Physical Media Type Identifier, according to DSP0239. This value is used to indicate what format the following physical address data is given in.
6	Physical Address Size. The size in bytes of the following Physical Address field The size is defined as part of the corresponding physical transport binding specification identified by the physical media type identifier.
7:N	Physical Address. The size and format of this field is defined as part of the corresponding physical transport binding specification. The information given in this field is given MSB first. Any unused bits should be set to `0b`.

11.13 Prepare for Endpoint Discovery

The Endpoint Discovery message is used to determine if devices on a bus communicate MCTP (see Table 28). Whether this message is required depends on the particular medium. Currently, this message may be required only by a particular transport binding, such as PCI Express (PCIe) VDM, because other bindings such as SMBus/I2C may use other mechanisms for determining this information.

Each endpoint (except the bus owner) on the bus maintains an internal flag called the "Discovered" flag.

The Prepare for Endpoint Discovery command is issued as a broadcast Request message on a given bus that causes each endpoint on the bus to set their respective Discovered flag to the "undiscovered" state. The flag is subsequently set to the "discovered" state when the Set Endpoint ID command is received by the endpoint.

An endpoint also sets the flag to the "undiscovered" state at the following times:

- Whenever the physical address associated with the endpoint changes or is assigned
- Whenever an endpoint first appears on the bus and requires an EID assignment
- During operation if an endpoint enters a state that requires its EID to be reassigned
- For hot-plug endpoints: After exiting any temporary state where the hot-plug endpoint was unable to respond to MCTP control requests for more than $T_{RECLAIM}$ seconds (where $T_{RECLAIM}$ is specified in the physical transport binding specification for the medium used to access the endpoint). See 8.17.5 for additional information.

Only endpoints that have their Discovered flag set to "undiscovered" will respond to the Endpoint Discovery message. Endpoints that have the flag set to "discovered" will not respond.

The destination EID for the Prepare for Endpoint Discovery message is set to the Broadcast EID value (see Table 2) in the request message to indicate that this is a broadcast message. The response message sets the destination EID to be the ID of the source of the request message, which is typically the EID of the bus owner. The request and response parameters are specified in Table 28.

The Prepare for Endpoint Discovery message has no effect on existing EID assignments. That is, endpoints shall normally retain their EIDs until they are explicitly changed via the Set Endpoint ID command, and shall not clear them after getting a "Prepare for Endpoint Discovery" command. (Note that endpoints may lose their EIDs under other conditions such as power state changes, etc., as described elsewhere in this specification.)

The Endpoint Discovery and Prepare for Endpoint Discovery commands may only be supported on particular transport bindings (e.g. MCTP over PCIe Vendor Defined Messaging). If the binding does not use this discovery approach (e.g. SMBus/I2C) the endpoint shall return an `ERROR_UNSUPPORTED_CMD` completion status for those commands.

Table 28 – Prepare for Endpoint Discovery Message

	Byte	Description
Request data	–	–
Response data	1	Completion Code

11.14 Endpoint Discovery

This command is used to discover endpoints that have their Discovered flag set to "undiscovered". Only endpoints that have their Discovered flag set to "undiscovered" will respond to this message. Endpoints that have the flag set to "discovered" will not respond.

This message is typically sent as a Broadcast Request message by the bus owner using the Broadcast EID as the destination EID, though for testing purposes endpoints must also accept and handle this command as a non-broadcast Request. Additionally, the request may be sent as a datagram, depending on the transport binding requirements. The request and response (if any) parameters are specified in Table 29.

Table 29 – Endpoint Discovery Message

	Byte	Description
Request data	–	–
Response data	1	Completion Code

11.15 Discovery Notify

This message is available for use as a common message for enabling an endpoint to announce its presence to the bus owner. This will typically be used as part of the endpoint discovery process when an MCTP device is hot-plugged onto or becomes powered up on an MCTP bus.

Whether and how this message is used for endpoint discovery depends on the particular physical transport binding specification. For example, the SMBus/I2C transport binding does not use this message for an endpoint to announce itself because it takes advantage of mechanisms that are already defined for SMBus.

This message should only be sent from endpoints to the bus owner for the bus that the endpoint is on so it can notify the bus owner that the endpoint has come online and may require an EID assignment or update. Additionally, the request may be sent as a datagram, depending on the transport binding requirements. The request and response (if any) parameters are specified in Table 30.

Table 30 – Discovery Notify Message

	Byte	Description
Request data	–	–
Response data	1	Completion Code

11.16 Get Network ID

The Get Network ID command returns a universally unique identifier (UUID), also referred to as a globally unique ID (GUID), for a given MCTP network. Typically this command is sent to the topmost MCTP bus-owner since the topmost bus-owner has this knowledge. A Network ID is required for add-in MCTP networks (For example, an MCTP Network on an add-in card or module). A Network ID is not required for a fixed (not add-in) MCTP network provided there is only one network in the system implementation. A Network ID is required for fixed MCTP networks when more than one fixed network exists in the system implementation and is simultaneously accessible by a common entity such as system software.

The format of the ID follows the byte (octet) format specified in RFC4122. RFC4122 specifies four different versions of UUID formats and generation algorithms suitable for use for a device UUID in IPMI. These are version 1 (0001b) "time based", and three "name-based" versions: version 3 (0011b) "MD5 hash", version 4 (0100b) "Pseudo-random", and version 5 "SHA1 hash". The version 1 format is recommended. However, versions 3, 4, or 5 formats are also allowed. A device UUID should never change over the lifetime of the device. The request and response parameters are specified in Table 16.

Table 31 – Get Network ID Message Format

	Byte	Description
Request data	–	–
Response data	1	Completion Code
	2:17	Network ID bytes 1:16, respectively (see Table 17)

The individual fields within the UUID are stored most-significant byte (MSB) first per the convention described in RFC4122. See Table 17 for an example format.

11.17 Query Hop

This command can be used to query a bridge to find out whether a given EID must be accessed by going through that bridge, and if so, whether yet another bridge must be passed through in the path to the endpoint, or if the endpoint is on a bus that is directly connected to the bridge.

The command also returns the information about the transmission unit information that the bridge supports in routing to the given target endpoint from the bus that the request was received over. See clasue 9.5 for more information.

NOTE: The physical transport binding for MCTP may place additional requirements on the physical packet sizes that can be used to transfer MCTP packet payloads, such as requiring that physical packet sizes be in 32-byte or 64-byte increments, or particular power of 2 increments (for example, 128, 256, 512, and so on).

The request and response parameters are specified in Table 32.

Table 32 – Query Hop Message

	Byte	Description
Request data	1	Target Endpoint ID `0x00`, `0xFF` = reserved. (An `ERROR_INVALID_DATA` completion code shall be returned.)
	2	Message type for which transmission unit information is being requested. Use the MCTP control message type number unless another message type is of interest.
Response data	1	Completion Code An `ERROR_INVALID_DATA` completion code must be returned if the target EID is not covered by any entry in the bridge's routing table.
	2	EID of the next bridge that is used to access the target endpoint, if any Note: This response depends on which bus port the Query Hop request is received over. If this EID is `00h`: The EID is covered by the bridge's routing table, but the target EID does not require access by *going through* this bridge from the port the request was received over. This response will be returned if the target EID is already local to the bus over which the request is being received. This response is also returned when the target EID is an EID for the bridge itself. If this EID is non-zero *and* is different than the target EID passed in request: The EID being provided is the EID of the "next bridge" in the path to the target EID. If this EID is equal to the target EID passed in request: The target EID is accessed by going through this bridge and no additional bridges must be gone through to reach the target.
	3	Message Type. This value either returns the message type that was given in the request, or it returns `0xFF` to indicate that the information is applicable to all message types that are supported by the bridge.
	4:5	Maximum supported incoming transmission unit size in increments of 16 bytes, starting from the baseline transmission unit size (`0x0000` = 64 bytes, `0x0001` = 80 bytes, and so on).

	Byte	Description
	5:6	Maximum supported outgoing transmission unit size in increments of 16 bytes, starting from the baseline transmission unit (0x0000 = 64 bytes, 0x0001 = 80 bytes, and so on). The responder will return whether this transmission unit size is supported for MCTP packets that it transmits for the given message type.

11.18 Resolve UUID

This command is used to get information about an endpoint based on its UUID. This command may be sent from any endpoint to the bus owner. This command takes a UUID as a parameter in the request and returns a list of EIDs and physical addresses that matches this UUID.

A bus owner that supports this command shall keep in the routing table entries the UUID of each of the endpoints. The UUID values can be found using a “Get Endpoint UUID” command. An endpoint knows the physical address of the bus owner by keeping track of which physical address was used when the endpoint received its EID assignment through the Set Endpoint ID command. The endpoint can send this command to the bus owner using the null destination EID value. This eliminates the need for the endpoint to also keep track of the EID of the bus owner. The request and response parameters are specified in Table 12.

Table 33 – Resolve UUID Message

	Byte	Description
Request data	1:16	Requested UUID
	17	Entry Handle (0x00 to access first entries in table)
Response data	1	Completion Code
	2	Next Entry Handle (Use this value to request the next set of entries, if any.) If the EID table data exceeds what can be carried in a single MCTP control response. 0xFF = No more entries
	3	Number of EID entries being returned in this response.
	4:N	One or more routing table entries, formatted per Table 13. This field will be absent if the number of EID entries is 0x00.

Table 34 – Resolve UUID Message Entry Format

Byte	Description
0	EID
1	Physical Transport Binding Type Identifier, according to MCTP ID specification (DSP0239).
2	Physical Media Type Identifier, according to MCTP ID specification (DSP0239). This value is used to indicate what format the following physical address data is given in.
3	Physical Address Size.
4:N	Physical Address.

11.19 Transport Specific

Transport Specific commands are a range of commands that are available for use by transport binding specifications in order to perform additional MCTP Control functions that are defined by a particular transport binding. Transport specific commands shall only be addressed to endpoints on the same medium. A bridge is allowed to block transport specific commands from being bridged to different media.

The request and response parameters are specified in Table 35.

Table 35 – Transport Specific Message

	Byte	Description
Request data	1	MCTP Physical Transport Binding Identifier The ID of the Physical Transport specification that defines the transport specific message. This ID is defined in the MCTP ID companion document to this specification.
	2	MCTP Physical Media Identifier The ID of the physical medium that the message is targeted for. This ID is defined in the MCTP ID companion document to this specification.
	3:N	Transport specific command data. Defined by the transport binding specification identified by the MCTP Physical Transport Binding Identifier given in byte 1. If the Physical Transport Binding Identifier = Vendor Defined: The first four bytes of data shall be the IANA Enterprise ID for the Vendor. MSB first. See 11.8.1 for the information on the IANA Enterprise ID as used in this specification.
Response data	1	Completion Code

12 Vendor Defined – PCI and Vendor Defined – IANA Messages

The Vendor Defined – PCI and Vendor Defined – IANA message types provide a mechanism for providing an MCTP message namespace for vendor-specific messages over MCTP.

The PCI and IANA designations refer to the mechanism that is used to identify the vendor or organization this is specifying the message's functionality and any parametric data or other fields provided in the message body.

Note that this specification only defines the initial bytes in the message body of these messages, and sets the requirement that these messages must follow the requirements set by the MCTP base protocol and any additional requirements necessary to meet the transport of these messages over a particular medium, such as path transmission unit limitations.

Otherwise, any other field definitions and higher level message behavior such as retries, error/completion codes, and so on, is message type-specific and thus is vendor-specific.

12.1 Vendor Defined – PCI Message Format

For these messages, the MCTP message type is set to the value for "Vendor Defined – PCI" as defined in Table 3. The request and response parameters are specified in Table 36.

Table 36 – Vendor Defined – PCI Message Format

	Byte	Description
Request data	1:2	PCI/PCIe Vendor ID. Refer to PCIe. MSB first. This value is formatted per the Vendor Data Field for the PCI Express vendor ID format. See 11.8.1". NOTE: Because the vendor ID format is implied by the command, the Vendor ID Format bytes are not part of this field.
	(3:N)	Vendor-Defined Message Body. 0 to N bytes.
Response data	1:2	PCI/PCIe Vendor ID. Refer to PCIe. MSB first.
	(3:M)	Vendor-Defined Message Body. 0 to M bytes.

12.2 Vendor Defined – IANA Message Format

For these messages, the MCTP message type is set to the value for "Vendor Defined – IANA" as defined in Table 3. The request and response parameters are specified in Table 37.

Table 37 – Vendor Defined – IANA Message Format

	Byte	Description
Request data	1:4	IANA Enterprise ID for Vendor. MSB first. This value is formatted per the Vendor Data Field for the IANA enterprise vendor ID format. See 11.8.1. NOTE: Because the vendor ID format is implied by the command, the Vendor ID Format bytes are not part of this field.
	(5:N)	Vendor-Defined Message Body. 0 to N bytes.
Response data	1:4	IANA Enterprise ID for the Vendor. MSB first.
	(5:M)	Vendor-Defined Message Body. 0 to M bytes.

ANNEX A
(informative)

Notation

A.1 Notations

Examples of notations used in this document are as follows:

- 2:N In field descriptions, this will typically be used to represent a range of byte offsets starting from byte two and continuing to and including byte N. The lowest offset is on the left, and the highest is on the right.
- (6) Parentheses around a single number can be used in message field descriptions to indicate a byte field that may be present or absent.
- (3:6) Parentheses around a field consisting of a range of bytes indicates the entire range may be present or absent. The lowest offset is on the left, and the highest is on the right.
- PCIe Underlined, blue text is typically used to indicate a reference to a document or specification called out in 2, "Normative References", or to items hyperlinked within the document.
- rsvd Abbreviation for Reserved. Case insensitive.
- [4] Square brackets around a number are typically used to indicate a bit offset. Bit offsets are given as zero-based values (that is, the least significant bit [LSb] offset = 0).
- [7:5] A range of bit offsets. The most-significant is on the left, and the least-significant is on the right.
- `1b` The lower case "`b`" following a number consisting of `0`s and `1`s is used to indicate the number is being given in binary format.
- `0x12A` A leading "`0x`" is used to indicate a number given in hexadecimal format.

ANNEX B
(informative)

Change Log

Version	Date	Author	Description
1.0.0	2009-05-21		
1.1.0	2010-02-19	T. Slaight	Updated the glossary and the overview clause, including additions for MCTP host interfaces and descriptions of MCTP networks. Added support for MCTP network IDs and the Get Network ID command. Addressed Mantis issue: 0000417. Added text to Clause 1 (Scope) referencing DSP0238, DSP0239 per WG ballot comments.
1.2.0	2013-01-10	T. Slaight	Added *Resolve UUID* command. Clarified use of Control Protocol Version and versioning for OEM commands, *Prepare for Endpoint Discovery* command, and the *Allocate Endpoint IDs* command. Clarified requirements on MCTP Control message flags and TO bit use. Changed command requirements to allow an Endpoint to optionally accept or generate Routing Information Update commands. Corrected typographic and formatting errors.
1.2.1	2014-12-03	T. Slaight	Corrected error requiring the TD bit to be set to 0b for MCTP VDMs. TD bit should follow PCIe specifications. Corrected misuse of reserved EIDs in figures. Changed document organization to place bridging clauses in a new first level clause "MCTP Bridging". Added clarifications and clause on "Endpoint ID Retention". Added more cross references and clarifications to better identify requirements associated with the *Get Endpoint UUID* command.