Internet Engineering Task Force (IETF)
Request for Comments: 5810
Category: Standards Track
ISSN: 2070-1721

A. Doria, Ed.
Lulea University of Technology
J. Hadi Salim, Ed.
Znyx
R. Haas, Ed.
IBM
H. Khosravi, Ed.
Intel
W. Wang, Ed.
L. Dong
Zhejiang Gongshang University
R. Gopal
Nokia
J. Halpern

March 2010

Forwarding and Control Element Separation (ForCES) Protocol Specification

Abstract

This document specifies the Forwarding and Control Element Separation (ForCES) protocol. The ForCES protocol is used for communications between Control Elements(CEs) and Forwarding Elements (FEs) in a ForCES Network Element (ForCES NE). This specification is intended to meet the ForCES protocol requirements defined in RFC 3654. Besides the ForCES protocol, this specification also defines the requirements for the Transport Mapping Layer (TML).

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5810.

Table of Contents

1. Introduction

Forwarding and Control Element Separation (ForCES) defines an architectural framework and associated protocols to standardize information exchange between the control plane and the forwarding plane in a ForCES Network Element (ForCES NE). RFC 3654 has defined the ForCES requirements, and RFC 3746 has defined the ForCES framework. While there may be multiple protocols used within the overall ForCES architecture, the terms "ForCES protocol" and "protocol" as used in this document refer to the protocol used to standardize the information exchange between Control Elements (CEs) and Forwarding Elements (FEs) only.

The ForCES FE model [RFC5812] presents a formal way to define FE Logical Function Blocks (LFBs) using XML. LFB configuration components, capabilities, and associated events are defined when the LFB is formally created. The LFBs within the FE are accordingly controlled in a standardized way by the ForCES protocol.

This document defines the ForCES protocol specifications. The ForCES protocol works in a master-slave mode in which FEs are slaves and CEs are masters. The protocol includes commands for transport of LFB configuration information, association setup, status, event notifications, etc.

Section 3 provides a glossary of terminology used in the specification.

Section 4 provides an overview of the protocol, including a discussion on the protocol framework and descriptions of the Protocol Layer (PL), a Transport Mapping Layer (TML), and the ForCES protocol mechanisms. Section 4.4 describes several protocol scenarios and includes message exchange descriptions.

While this document does not define the TML, Section 5 details the services that a TML MUST provide (TML requirements).

The ForCES protocol defines a common header for all protocol messages. The header is defined in Section 6.1, while the protocol messages are defined in Section 7.

Section 8 describes the protocol support for high-availability mechanisms including redundancy and fail over.

Section 9 defines the security mechanisms provided by the PL and TML.

2. Terminology and Conventions

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.2. Other Notation

In Table 1 and Table 2, the following notation is used to indicate multiplicity:

(value)+ .... means "1 or more instances of value"

(value)* .... means "0 or more instances of value"

2.3. Integers

All integers are to be coded as unsigned binary integers of appropriate length.

3. Definitions

This document follows the terminology defined by the ForCES requirements in [RFC3654] and by the ForCES framework in [RFC3746]. The definitions be are repeated below for clarity.

Addressable Entity (AE):

A physical device that is directly addressable given some interconnect technology. For example, on IP networks, it is a device that can be reached using an IP address; and on a switch fabric, it is a device that can be reached using a switch fabric port number.

Control Element (CE):

A logical entity that implements the ForCES protocol and uses it to instruct one or more FEs on how to process packets. CEs handle functionality such as the execution of control and signaling protocols.

CE Manager (CEM):

A logical entity responsible for generic CE management tasks. It is particularly used during the pre-association phase to determine with which FE(s) a CE should communicate. This process is called FE discovery and may involve the CE manager learning the capabilities of available FEs.

Data Path:

A conceptual path taken by packets within the forwarding plane inside an FE.

Forwarding Element (FE):

A logical entity that implements the ForCES protocol. FEs use the underlying hardware to provide per-packet processing and handling as directed/controlled by one or more CEs via the ForCES protocol.

FE Model:

A model that describes the logical processing functions of an FE. The FE model is defined using Logical Function Blocks (LFBs).

FE Manager (FEM):

A logical entity responsible for generic FE management tasks. It is used during the pre-association phase to determine with which CE(s) an FE should communicate. This process is called CE discovery and may involve the FE manager learning the capabilities of available CEs. An FE manager may use anything from a static configuration to a pre-association phase protocol (see below) to determine which CE(s) to use. Being a logical entity, an FE manager might be physically combined with any of the other logical entities such as FEs.

ForCES Network Element (NE):

An entity composed of one or more CEs and one or more FEs. To entities outside an NE, the NE represents a single point of management. Similarly, an NE usually hides its internal organization from external entities.

High Touch Capability:

This term will be used to apply to the capabilities found in some forwarders to take action on the contents or headers of a packet based on content other than what is found in the IP header. Examples of these capabilities include quality of service (QoS) policies, virtual private networks, firewall, and L7 content recognition.

Inter-FE Topology:

See FE Topology.

Intra-FE Topology:

See LFB Topology.

LFB (Logical Function Block):

The basic building block that is operated on by the ForCES protocol. The LFB is a well-defined, logically separable functional block that resides in an FE and is controlled by the CE via the ForCES protocol. The LFB may reside at the FE's data path and process packets or may be purely an FE control or configuration entity that is operated on by the CE. Note that the LFB is a functionally accurate abstraction of the FE's processing capabilities, but not a hardware-accurate representation of the FE implementation.

FE Topology:

A representation of how the multiple FEs within a single NE are interconnected. Sometimes this is called inter-FE topology, to be distinguished from intra-FE topology (i.e., LFB topology).

LFB Class and LFB Instance:

LFBs are categorized by LFB classes. An LFB instance represents an LFB class (or type) existence. There may be multiple instances of the same LFB class (or type) in an FE. An LFB class is represented by an LFB class ID, and an LFB instance is represented by an LFB instance ID. As a result, an LFB class ID associated with an LFB instance ID uniquely specifies an LFB existence.

LFB Meta Data:

Meta data is used to communicate per-packet state from one LFB to another, but is not sent across the network. The FE model defines how such meta data is identified, produced, and consumed by the LFBs. It defines the functionality but not how meta data is encoded within an implementation.

LFB Component:

Operational parameters of the LFBs that must be visible to the CEs are conceptualized in the FE model as the LFB components. The LFB components include, for example, flags, single parameter arguments, complex arguments, and tables that the CE can read and/or write via the ForCES protocol (see below).

LFB Topology:

Representation of how the LFB instances are logically interconnected and placed along the data path within one FE. Sometimes it is also called intra-FE topology, to be distinguished from inter-FE topology.

Pre-association Phase:

The period of time during which an FE manager and a CE manager are determining which FE(s) and CE(s) should be part of the same network element.

Post-association Phase:

The period of time during which an FE knows which CE is to control it and vice versa. This includes the time during which the CE and FE are establishing communication with one another.

ForCES Protocol:

While there may be multiple protocols used within the overall ForCES architecture, the terms "ForCES protocol" and "protocol" refer to the Fp reference points in the ForCES framework in [RFC3746]. This protocol does not apply to CE-to-CE communication, FE-to-FE communication, or communication between FE and CE managers. Basically, the ForCES protocol works in a master-slave mode in which FEs are slaves and CEs are masters. This document defines the specifications for this ForCES protocol.

ForCES Protocol Layer (ForCES PL):

A layer in the ForCES protocol architecture that defines the ForCES protocol messages, the protocol state transfer scheme, and the ForCES protocol architecture itself (including requirements of ForCES TML as shown below). Specifications of ForCES PL are defined by this document.

ForCES Protocol Transport Mapping Layer (ForCES TML):

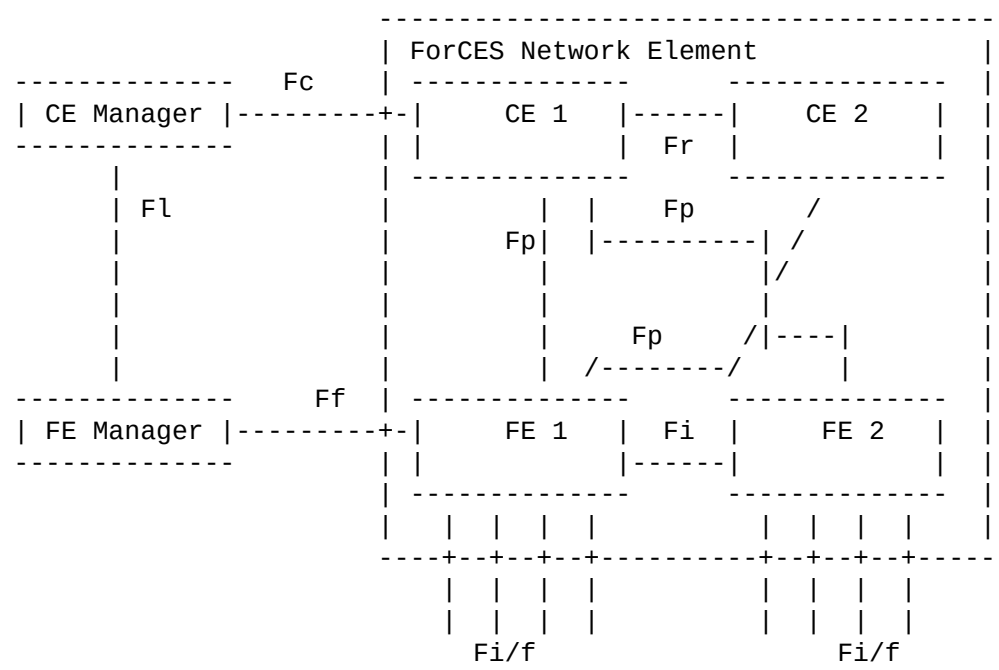
A layer in ForCES protocol architecture that uses the capabilities of existing transport protocols to specifically address protocol message transportation issues, such as how the protocol messages are mapped to different transport media (like TCP, IP, ATM, Ethernet, etc.), and how to achieve and implement reliability, multicast, ordering, etc. The ForCES TML specifications are detailed in separate ForCES documents, one for each TML.

4. Overview

The reader is referred to the framework document [RFC3746], and in particular, Sections 3 and 4, for an architectural overview and an explanation of how the ForCES protocol fits in. There may be some content overlap between the framework document and this section in order to provide clarity. This document is authoritative on the protocol, whereas [RFC3746] is authoritative on the architecture.

4.1. Protocol Framework

Figure 1 below is reproduced from the framework document for clarity. It shows an NE with two CEs and two FEs.



Fp: CE-FE interface
Fi: FE-FE interface
Fr: CE-CE interface
Fc: Interface between the CE manager and a CE
Ff: Interface between the FE manager and an FE
Fl: Interface between the CE manager and the FE manager
Fi/f: FE external interface

Figure 1: ForCES Architectural Diagram

The ForCES protocol domain is found in the Fp reference points. The Protocol Element configuration reference points, Fc and Ff, also play a role in the booting up of the ForCES protocol. The protocol element configuration (indicated by reference points Fc, Ff, and Fl in [RFC3746]) is out of scope of the ForCES protocol but is touched on in this document in discussion of FEM and CEM since it is an integral part of the protocol pre-association phase.

Figure 2 below shows further breakdown of the Fp interfaces by means of the example of an MPLS QoS-enabled Network Element.

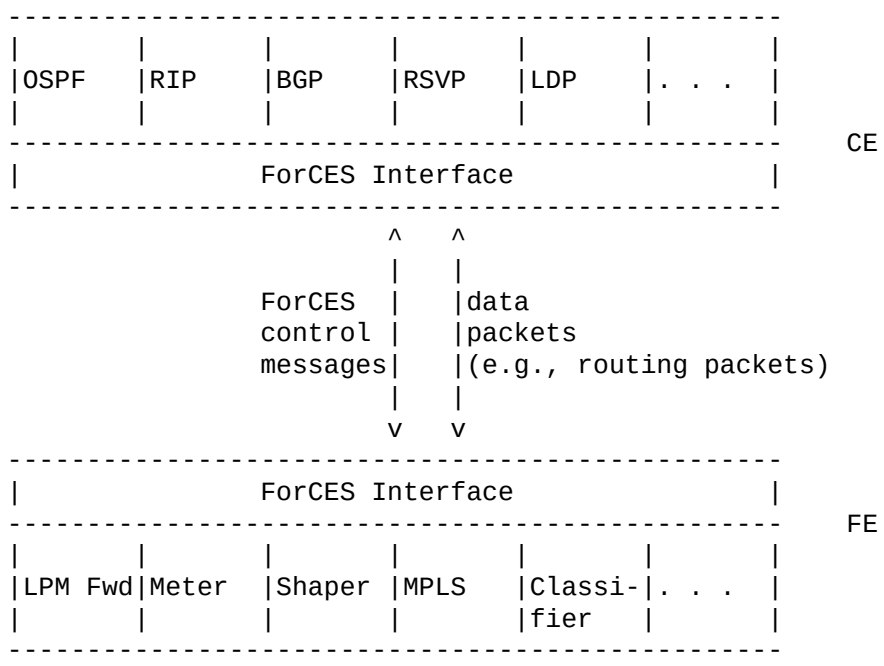


Figure 2: Examples of CE and FE Functions

The ForCES interface shown in Figure 2 constitutes two pieces: the PL and the TML.

This is depicted in Figure 3 below.

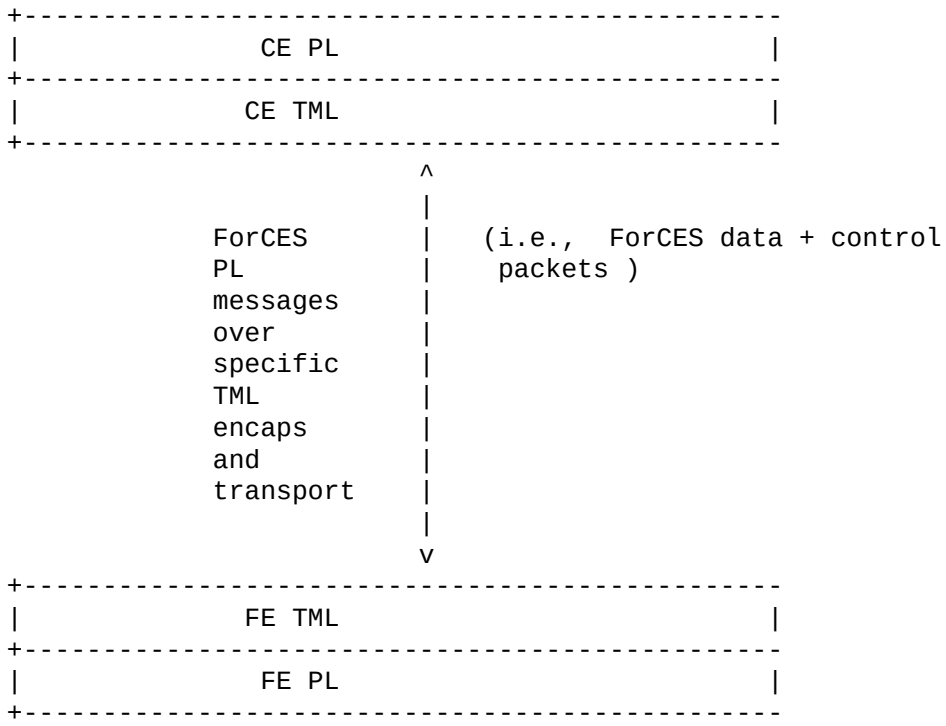


Figure 3: ForCES Interface

The PL is in fact the ForCES protocol. Its semantics and message layout are defined in this document. The TML layer is necessary to connect two ForCES PLs as shown in Figure 3 above. The TML is out of scope for this document but is within scope of ForCES. This document defines requirements the PL needs the TML to meet.

Both the PL and the TML are standardized by the IETF. While only one PL is defined, different TMLs are expected to be standardized. To interoperate, the TML at the CE and FE are expected to conform to the same definition.

On transmit, the PL delivers its messages to the TML. The local TML delivers the message to the destination TML. On receive, the TML delivers the message to its destination PL.

4.1.1. The PL

The PL is common to all implementations of ForCES and is standardized by the IETF as defined in this document. The PL is responsible for associating an FE or CE to an NE. It is also responsible for tearing

down such associations. An FE uses the PL to transmit various subscribed-to events to the CE PL as well as to respond to various status requests issued from the CE PL. The CE configures both the FE and associated LFBs' operational parameters using the PL. In addition, the CE may send various requests to the FE to activate or deactivate it, reconfigure its HA parameterization, subscribe to specific events, etc. More details can be found in Section 7.

4.1.2. The TML

The TML transports the PL messages. The TML is where the issues of how to achieve transport-level reliability, congestion control, multicast, ordering, etc. are handled. It is expected that more than one TML will be standardized. The various possible TMLs could vary their implementations based on the capabilities of underlying media and transport. However, since each TML is standardized, interoperability is guaranteed as long as both endpoints support the same TML. All ForCES protocol layer implementations MUST be portable across all TMLs, because all TMLs MUST have the top-edge semantics defined in this document.

4.1.3. The FEM/CEM Interface

The FEM and CEM components, although valuable in the setup and configurations of both the PL and TML, are out of scope of the ForCES protocol. The best way to think of them is as configurations/ parameterizations for the PL and TML before they become active (or even at runtime based on implementation). In the simplest case, the FE or CE reads a static configuration file. RFC 3746 has a more detailed description on how the FEM and CEM could be used. The pre-association phase, where the CEM and FEM can be used, are described briefly in Section 4.2.1.

An example of typical things the FEM/CEM could configure would be TML-specific parameterizations such as:

a. How the TML connection should happen (for example, what IP addresses to use, transport modes, etc.)

b. The ID for the FE (FEID) or CE (CEID) (which would also be issued during the pre-association phase)

c. Security parameterization such as keys, etc.

d. Connection association parameters

An example of connection association parameters might be:

- simple parameters: send up to 3 association messages every 1 second
- complex parameters: send up to 4 association messages with increasing exponential timeout

4.2. ForCES Protocol Phases

ForCES, in relation to NEs, involves two phases: the pre-association phase where configuration/initialization/bootup of the TML and PL layer happens, and the post-association phase where the ForCES protocol operates to manipulate the parameters of the FEs.

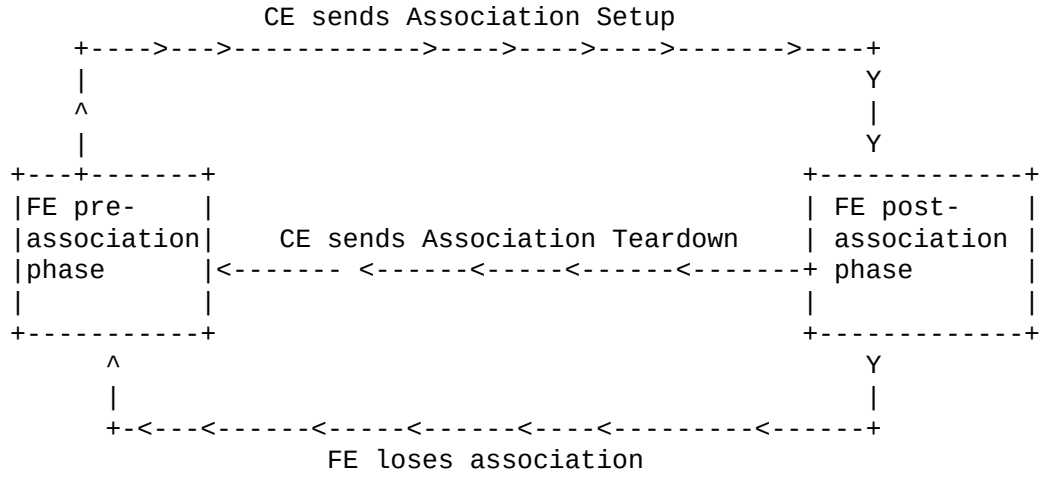


Figure 4: The FE Protocol Phases

In the mandated case, once associated, the FE may forward packets depending on the configuration of its specific LFBs. An FE that is associated to a CE will continue sending packets until it receives an Association Teardown Message or until it loses association. An unassociated FE MAY continue sending packets when it has a high availability capability. The extra details are explained in Section 8 and not discussed here to allow for a clear explanation of the basics.

The FE state transitions are controlled by means of the FE Object LFB FEState component, which is defined in [RFC5812], Section 5.1, and also explained in Section 7.3.2.

The FE initializes in the FEState OperDisable. When the FE is ready to process packets in the data path, it transitions itself to the OperEnable state.

The CE may decide to pause the FE after it already came up as OperEnable. It does this by setting the FEState to AdminDisable. The FE stays in the AdminDisable state until it is explicitly configured by the CE to transition to the OperEnable state.

When the FE loses its association with the CE, it may go into the pre-association phase depending on the programmed policy. For the FE to properly complete the transition to the AdminDisable state, it MUST stop packet forwarding and this may impact multiple LFBS. How this is achieved is outside the scope of this specification.

4.2.1. Pre-association

The ForCES interface is configured during the pre-association phase. In a simple setup, the configuration is static and is typically read from a saved configuration file. All the parameters for the association phase are well known after the pre-association phase is complete. A protocol such as DHCP may be used to retrieve the configuration parameters instead of reading them from a static configuration file. Note, this will still be considered static pre-association. Dynamic configuration may also happen using the Fc, Ff, and Fl reference points (refer to [RFC3746]). Vendors may use their own proprietary service discovery protocol to pass the parameters. Essentially, only guidelines are provided here and the details are left to the implementation.

The following are scenarios reproduced from the framework document to show a pre-association example.

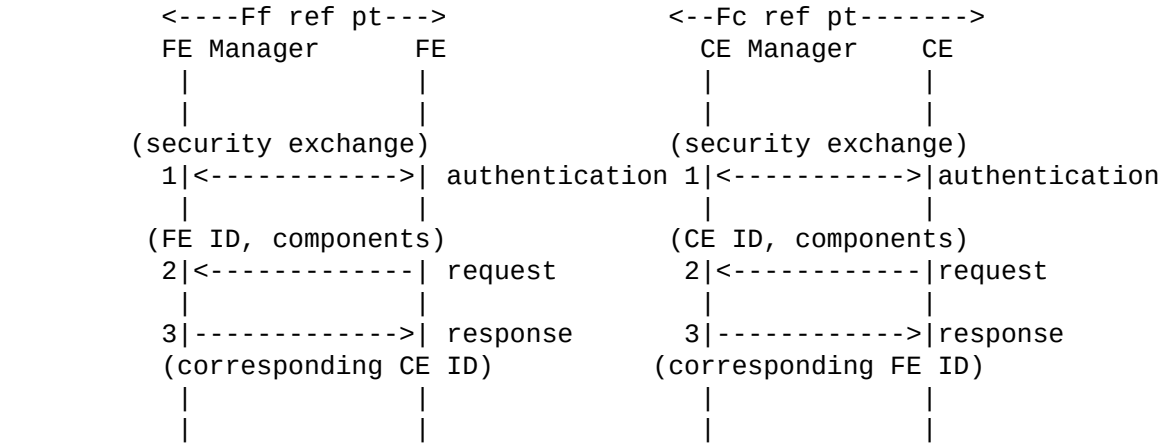


Figure 5: Examples of a Message Exchange over the Ff and Fc Reference Points

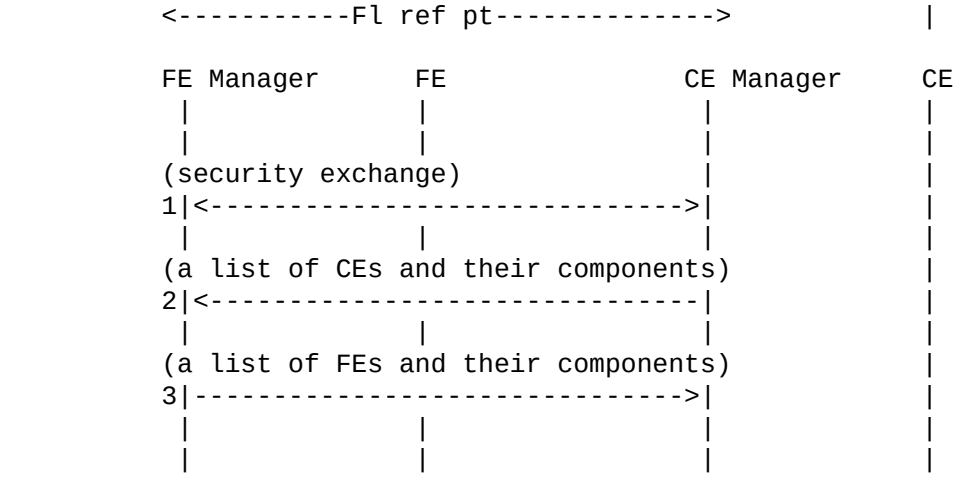


Figure 6: Example of a Message Exchange over the Fl Reference Point

Before the transition to the association phase, the FEM will have established contact with a CEM component. Initialization of the ForCES interface will have completed, and authentication as well as capability discovery may be complete. Both the FE and CE would have the necessary information for connecting to each other for configuration, accounting, identification, and authentication purposes. To summarize, at the completion of this stage both sides have all the necessary protocol parameters such as timers, etc. The Fl reference point may continue to operate during the association phase and may be used to force a disassociation of an FE or CE. The specific interactions of the CEM and the FEM that are part of the

pre-association phase are out of scope; for this reason, these details are not discussed any further in this specification. The reader is referred to the framework document [RFC3746] for a slightly more detailed discussion.

4.2.2. Post-association

In this phase, the FE and CE components communicate with each other using the ForCES protocol (PL over TML) as defined in this document. There are three sub-phases:

- Association Setup Stage
- Established Stage
- Association Lost Stage

4.2.2.1. Association Setup Stage

The FE attempts to join the NE. The FE may be rejected or accepted. Once granted access into the NE, capabilities exchange happens with the CE querying the FE. Once the CE has the FE capability information, the CE can offer an initial configuration (possibly to restore state) and can query certain components within either an LFB or the FE itself.

More details are provided in Section 4.4.

On successful completion of this stage, the FE joins the NE and is moved to the Established Stage.

4.2.2.2. Established Stage

In this stage, the FE is continuously updated or queried. The FE may also send asynchronous event notifications to the CE or synchronous heartbeat notifications if programmed to do so. This stage continues until a termination occurs, either due to loss of connectivity or due to a termination initiated by either the CE or the FE.

Refer to the section on protocol scenarios, Section 4.4, for more details.

4.2.2.3. Association Lost Stage

In this stage, both or either the CE or FE declare the other side is no longer associated. The disconnection could be initiated by either party for administrative purposes but may also be driven by operational reasons such as loss of connectivity.

A core LFB known as the FE Protocol Object (FEPO) is defined (refer to Appendix B and Section 7.3.1). FEPO defines various timers that can be used in conjunction with a traffic-sensitive heartbeat mechanism (Section 4.3.3) to detect loss of connectivity.

The loss of connectivity between TMLs does not indicate a loss of association between respective PL layers. If the TML cannot repair the transport loss before the programmed FEPO timer thresholds associated with the FE is exceeded, then the association between the respective PL layers will be lost.

FEPO defines several policies that can be programmed to define behavior upon a detected loss of association. The FEPO's programmed CE failover policy (refer to Sections 8, 7.3.1, 4.3.3, and B) defines what takes place upon loss of association.

For this version of the protocol (as defined in this document), the FE, upon re-association, MUST discard any state it has as invalid and retrieve new state. This approach is motivated by a desire for simplicity (as opposed to efficiency).

4.3. Protocol Mechanisms

Various semantics are exposed to the protocol users via the PL header including transaction capabilities, atomicity of transactions, two-phase commits, batching/parallelization, high availability, and failover as well as command pipelines.

The EM (Execution Mode) flag, AT (Atomic Transaction) flag, and TP (Transaction Phase) flag as defined in the common header (Section 6.1) are relevant to these mechanisms.

4.3.1. Transactions, Atomicity, Execution, and Responses

In the master-slave relationship, the CE instructs one or more FEs on how to execute operations and how to report the results.

This section details the different modes of execution that a CE can order the FE(s) to perform, as defined in Section 4.3.1.1. It also describes the different modes a CE can ask the FE(s) to use for formatting the responses after processing the operations as requested. These modes relate to the transactional two-phase commit operations.

4.3.1.1. Execution

There are 3 execution modes that can be requested for a batch of operations spanning one or more LFB selectors (refer to Section 7.1.5) in one protocol message. The EM flag defined in the common header (Section 6.1) selects the execution mode for a protocol message, as below:

a. execute-all-or-none

b. continue-execute-on-failure

c. execute-until-failure

4.3.1.1.1. execute-all-or-none

When set to this mode of execution, independent operations in a message MAY be targeted at one or more LFB selectors within an FE. All these operations are executed serially, and the FE MUST have no execution failure for any of the operations. If any operation fails to execute, then all the previous ones that have been executed prior to the failure will need to be undone. That is, there is rollback for this mode of operation.

Refer to Section 4.3.1.2.2 for how this mode is used in cases of transactions. In such a case, no operation is executed unless a commit is issued by the CE.

Care should be taken on how this mode is used because a mis-configuration could result in traffic losses. To add certainty to the success of an operation, one should use this mode in a transactional operation as described in Section 4.3.1.2.2

4.3.1.1.2. continue-execute-on-failure

If several independent operations are targeted at one or more LFB selectors, execution continues for all operations at the FE even if one or more operations fail.

4.3.1.1.3. execute-until-failure

In this mode, all operations are executed on the FE sequentially until the first failure. The rest of the operations are not executed but operations already completed are not undone. That is, there is no rollback in this mode of operation.

4.3.1.2. Transaction and Atomicity

4.3.1.2.1. Transaction Definition

A transaction is defined as a collection of one or more ForCES operations within one or more PL messages that MUST meet the ACIDity properties [ACID], defined as:

Atomicity: In a transaction involving two or more discrete pieces of information, either all of the pieces are committed or none are.

Consistency: A transaction either creates a new and valid state of data or, if any failure occurs, returns all data to the state it was in before the transaction was started.

Isolation: A transaction in process and not yet committed MUST remain isolated from any other transaction.

Durability: Committed data is saved by the system such that, even in the event of a failure and a system restart, the data is available in its correct state.

There are cases where the CE knows exact memory and implementation details of the FE such as in the case of an FE-CE pair from the same vendor where the FE-CE pair is tightly coupled. In such a case, the transactional operations may be simplified further by extra computation at the CE. This view is not discussed further other than to mention that it is not disallowed.

As defined above, a transaction is always atomic and MAY be

a. Within an FE alone
   Example: updating multiple tables that are dependent on each other. If updating one fails, then any that were already updated MUST be undone.

b. Distributed across the NE
   Example: updating table(s) that are inter-dependent across several FEs (such as L3 forwarding-related tables).

4.3.1.2.2. Transaction Protocol

By use of the execution mode, as defined in Section 4.3.1.1, the protocol has provided a mechanism for transactional operations within one stand-alone message. The 'execute-all-or-none' mode can meet the ACID requirements.

For transactional operations of multiple messages within one FE or across FEs, a classical transactional protocol known as two-phase commit (2PC) [2PCREF] is supported by the protocol to achieve the transactional operations utilizing Config messages (Section 7.6.1).

The COMMIT and TRCOMP operations in conjunction with the AT and the TP flags in the common header (Section 6.1) are provided for 2PC-based transactional operations spanning multiple messages.

The AT flag, when set, indicates that this message belongs to an Atomic Transaction. All messages for a transaction operation MUST have the AT flag set. If not set, it means that the message is a stand-alone message and does not participate in any transaction operation that spans multiple messages.

The TP flag indicates the Transaction Phase to which this message belongs. There are 4 possible phases for a transactional operation known as:

SOT (Start of Transaction)

MOT (Middle of Transaction)

EOT (End of Transaction)

ABT (Abort)

The COMMIT operation is used by the CE to signal to the FE(s) to commit a transaction. When used with an ABT TP flag, the COMMIT operation signals the FE(s) to roll back (i.e., un-COMMIT) a previously committed transaction.

The TRCOMP operation is a small addition to the classical 2PC approach. TRCOMP is sent by the CE to signal to the FE(s) that the transaction they have COMMITed is now over. This allows the FE(s) an opportunity to clear state they may have kept around to perform a roll back (if it became necessary).

A transaction operation is started with a message in which the TP flag is set to Start of Transaction (SOT). Multi-part messages, after the first one, are indicated by the Middle of Transaction (MOT) flag. All messages from the CE MUST set the AlwaysACK flag (Section 6) to solicit responses from the FE(s).

Before the CE issues a commit (described further below), the FE MUST only validate that the operation can be executed but not execute it.

Any failure notified by an FE causes the CE to abort the transaction on all FEs involved in the transaction. This is achieved by sending a Config message with an ABT flag and a COMMIT operation.

If there are no failures by any participating FE, the transaction commitment phase is signaled from the CE to the FE by an End of Transaction (EOT) configuration message with a COMMIT operation.

The FE MUST respond to the CE's EOT message. There are two possible failure scenarios in which the CE MUST abort the transaction (as described above):

a. If any participating FE responds with a failure message in relation to the transaction.

b. If no response is received from a participating FE within a specified timeout.

If all participating FEs respond with a success indicator within the expected time, then the CE MUST issue a TRCOMP operation to all participating FEs. An FE MUST NOT respond to a TRCOMP.

Note that a transactional operation is generically atomic; therefore, it requires that the execution modes of all messages in a transaction operation should always be kept the same and be set to 'execute-all-or-none'. If the EM flag is set to other execution modes, it will result in a transaction failure.

As noted above, a transaction may span multiple messages. It is up to the CE to keep track of the different outstanding messages making up a transaction. As an example, the correlator field could be used to mark transactions and a sequence field to label the different messages within the same atomic transaction, but this is out of scope and up to implementations.

4.3.1.2.3. Recovery

Any of the participating FEs or the CE or the associations between them may fail after the EOT Response message has been sent by the FE but before the CE has received all the responses, e.g., if the EOT response never reaches the CE.

In this protocol revision, as indicated in Section 4.2.2.3, an FE losing an association would be required to get entirely new state from the newly associated CE upon a re-association. Although this approach is simplistic and provides likeliness of losing data path

traffic, it is a design choice to avoid the additional complexity of managing graceful restarts. The HA mechanisms (Section 8) are provided to allow for a continuous operation in case of FE failures.

Flexibility is provided on how to react when an FE loses association. This is dictated by the CE failover policy (refer to Section 8 and Section 7.3).

4.3.1.2.4. Transaction Messaging Example

This section illustrates an example of how a successful two-phase commit between a CE and an FE would look in the simple case.

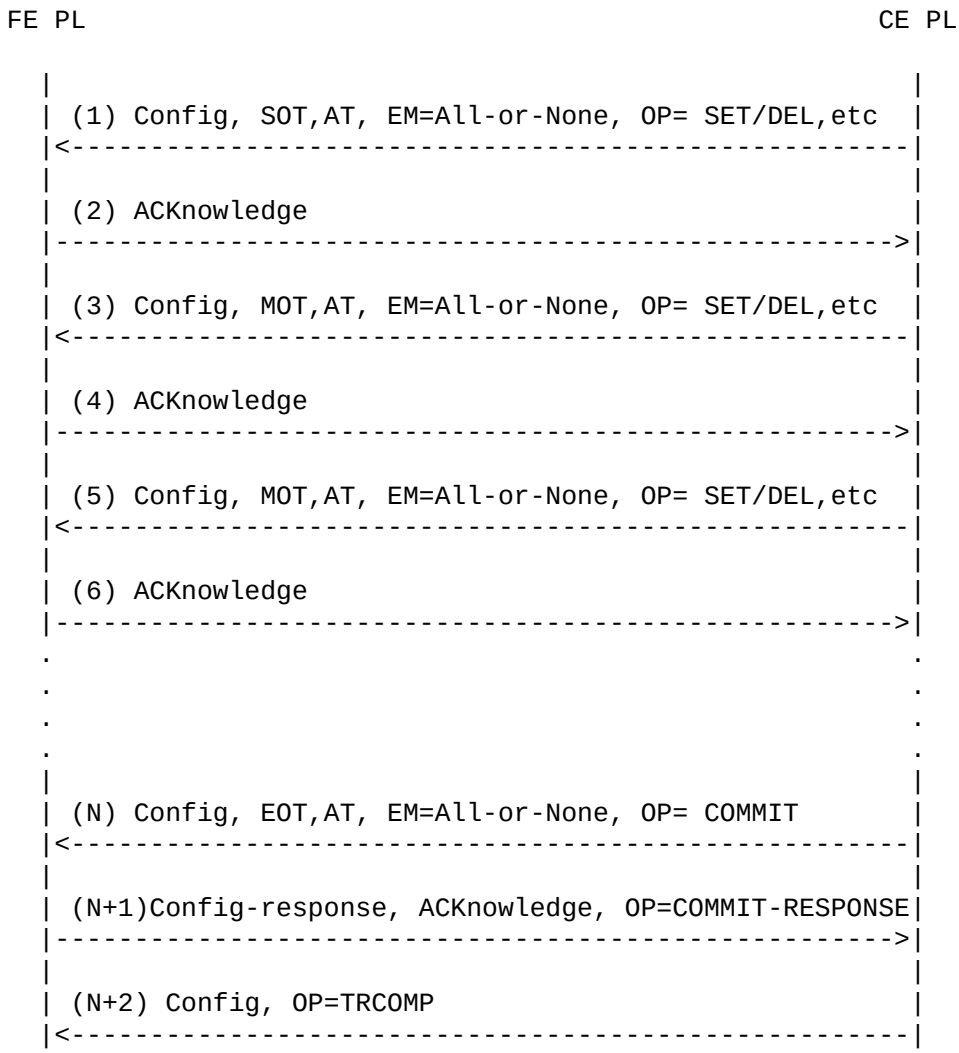


Figure 7: Example of a Two-Phase Commit

For the scenario illustrated above:

- In step 1, the CE issues a Config message with an operation of choice like a DEL or SET. The transaction flags are set to indicate a Start of Transaction (SOT), Atomic Transaction (AT), and execute-all-or-none.

- The FE validates that it can execute the request successfully and then issues an acknowledgment back to the CE in step 2.

- In step 3, the same sort of construct as in step 1 is repeated by the CE with the transaction flag changed to Middle of Transaction (MOT).

- The FE validates that it can execute the request successfully and then issues an acknowledgment back to the CE in step 4.

- The CE-FE exchange continues in the same manner until all the operations and their parameters are transferred to the FE. This happens in step (N-1).

- In step N, the CE issues a commit. A commit is a Config message with an operation of type COMMIT. The transaction flag is set to End of Transaction (EOT). Essentially, this is an "empty" message asking the FE to execute all the operations it has gathered since the beginning of the transaction (message #1).

- The FE at this point executes the full transaction. It then issues an acknowledgment back to the CE in step (N+1) that contains a COMMIT-RESPONSE.

- The CE in this case has the simple task of issuing a TRCOMP operation to the FE in step (N+2).

4.3.2. Scalability

It is desirable that the PL not become the bottleneck when larger bandwidth pipes become available. To pick a hypothetical example in today's terms, if a 100-Gbps pipe is available and there is sufficient work, then the PL should be able to take advantage of this and use all of the 100-Gbps pipe. Two mechanisms have been provided to achieve this. The first one is batching and the second one is a command pipeline.

Batching is the ability to send multiple commands (such as Config) in one Protocol Data Unit (PDU). The size of the batch will be affected by, among other things, the path MTU. The commands may be part of the same transaction or may be part of unrelated transactions that are independent of each other.

Command pipelining allows for pipelining of independent transactions that do not affect each other. Each independent transaction could consist of one or more batches.

4.3.2.1. Batching

There are several batching levels at different protocol hierarchies.

- Multiple PL PDUs can be aggregated under one TML message.
- Multiple LFB classes and instances (as indicated in the LFB selector) can be addressed within one PL PDU.
- Multiple operations can be addressed to a single LFB class and instance.

4.3.2.2. Command Pipelining

The protocol allows any number of messages to be issued by the CE before the corresponding acknowledgments (if requested) have been returned by the FE. Hence, pipelining is inherently supported by the protocol. Matching responses with requests messages can be done using the correlator field in the message header.

4.3.3. Heartbeat Mechanism

Heartbeats (HBs) between FEs and CEs are traffic sensitive. An HB is sent only if no PL traffic is sent between the CE and FE within a configured interval. This has the effect of reducing the amount of HB traffic in the case of busy PL periods.

An HB can be sourced by either the CE or FE. When sourced by the CE, a response can be requested (similar to the ICMP ping protocol). The FE can only generate HBs in the case of being configured to do so by the CE. Refer to Section 7.3.1 and Section 7.10 for details.

4.3.4. FE Object and FE Protocol LFBs

All PL messages operate on LFB constructs, as this provides more flexibility for future enhancements. This means that maintenance and configurability of FEs, NE, and the ForCES protocol itself MUST be expressed in terms of this LFB architecture. For this reason, special LFBs are created to accommodate this need.

In addition, this shows how the ForCES protocol itself can be controlled by the very same type of structures (LFBs) it uses to control functions such as IP forwarding, filtering, etc.

To achieve this, the following specialized LFBs are introduced:

- FE Protocol LFB, which is used to control the ForCES protocol.

- FE Object LFB, which is used to control components relative to the FE itself. Such components include FEState [RFC5812], vendor, etc.

These LFBs are detailed in Section 7.3.

4.4. Protocol Scenarios

This section provides a very high level description of sample message sequences between a CE and an FE. For protocol message encoding refer to Section 6.1, and for the semantics of the protocol refer to Section 4.3.

4.4.1. Association Setup State

The associations among CEs and FEs are initiated via the Association Setup message from the FE. If a Setup Request is granted by the CE, a successful Setup Response message is sent to the FE. If CEs and FEs are operating in an insecure environment, then the security associations have to be established between them before any association messages can be exchanged. The TML MUST take care of establishing any security associations.

This is typically followed by capability query, topology query, etc. When the FE is ready to start processing the data path, it sets the FEO FEState component to OperEnable (refer to [RFC5812] for details) and reports it to the CE as such when it is first queried. Typically, the FE is expected to be ready to process the data path before it associates, but there may be rare cases where it needs time do some pre-processing -- in such a case, the FE will start in the OperDisable state and when it is ready will transition to the OperEnable state. An example of an FE starting in OperDisable then

transitioning to OperEnable is illustrated in Figure 8. The CE could at any time also disable the FE's data path operations by setting the FEState to AdminDisable. The FE MUST NOT process packets during this state unless the CE sets the state back to OperEnable. These sequences of messages are illustrated in Figure 8 below.

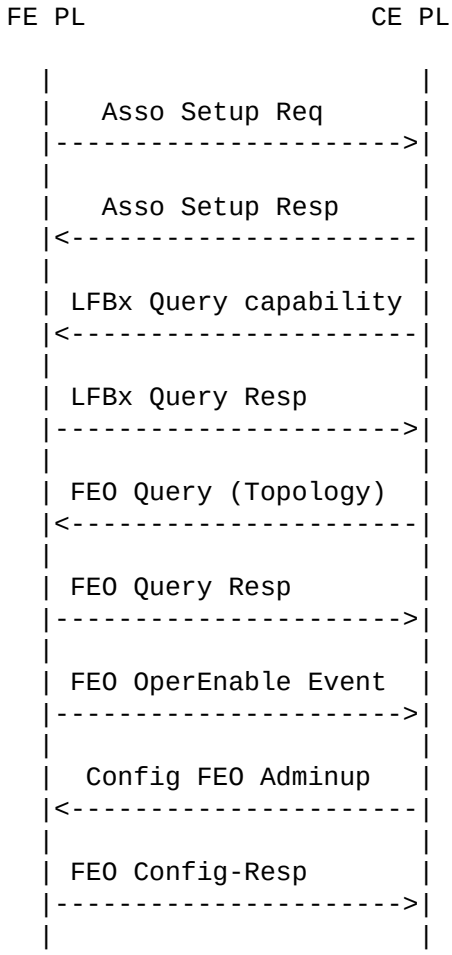


Figure 8: Message Exchange between CE and FE to Establish an NE Association

On successful completion of this state, the FE joins the NE.

4.4.2. Association Established State or Steady State

In this state, the FE is continuously updated or queried. The FE may also send asynchronous event notifications to the CE, synchronous Heartbeat messages, or Packet Redirect message to the CE. This continues until a termination (or deactivation) is initiated by either the CE or FE. Figure 9 below, helps illustrate this state.

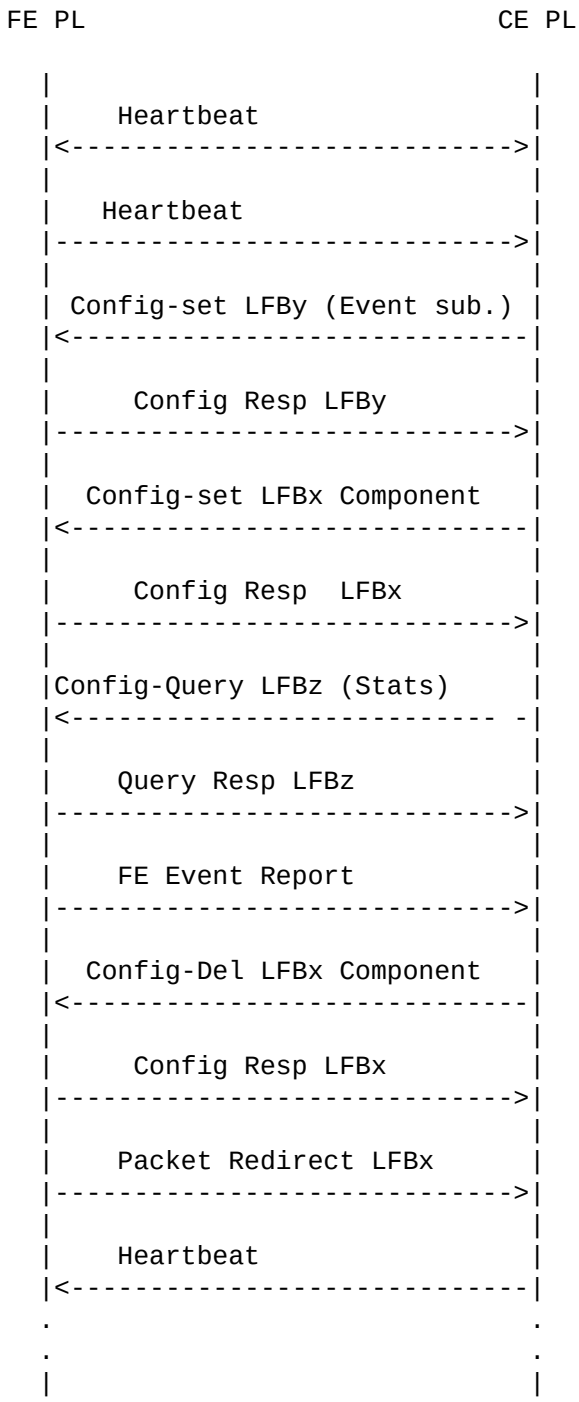


Figure 9: Message Exchange between CE and FE during Steady-State Communication

Note that the sequence of messages shown in the figure serve only as examples and the message exchange sequences could be different from what is shown in the figure. Also, note that the protocol scenarios described in this section do not include all the different message exchanges that would take place during failover. That is described in the HA section (Section 8).

5. TML Requirements

The requirements below are expected to be met by the TML. This text does not define how such mechanisms are delivered. As an example, the mechanisms to meet the requirements could be defined to be delivered via hardware or between 2 or more TML software processes on different CEs or FEs in protocol-level schemes.

Each TML MUST describe how it contributes to achieving the listed ForCES requirements. If for any reason a TML does not provide a service listed below, a justification needs to be provided.

Implementations that support the ForCES protocol specification MUST implement [RFC5811]. Note that additional TMLs might be specified in the future, and if a new TML defined in the future that meets the requirements listed here proves to be better, then the "MUST implement TML" may be redefined.

1. Reliability

Various ForCES messages will require varying degrees of reliable delivery via the TML. It is the TML's responsibility to provide these shades of reliability and describe how the different ForCES messages map to reliability.

The most common level of reliability is what we refer to as strict or robust reliability in which we mean no losses, corruption, or re-ordering of information being transported while ensuring message delivery in a timely fashion.

Payloads such as configuration from a CE and its response from an FE are mission critical and must be delivered in a robust reliable fashion. Thus, for information of this sort, the TML MUST either provide built-in protocol mechanisms or use a reliable transport protocol for achieving robust/strict reliability.

Some information or payloads, such as redirected packets or packet sampling, may not require robust reliability (can tolerate some degree of losses). For information of this sort, the TML could define to use a mechanism that is not strictly reliable (while conforming to other TML requirements such as congestion control).

Some information or payloads, such as heartbeat packets, may prefer timeliness over reliable delivery. For information of this sort, the TML could define to use a mechanism that is not strictly reliable (while conforming to other TML requirements such as congestion control).

2. Security

TML provides security services to the ForCES PL. Because a ForCES PL is used to operate an NE, attacks designed to confuse, disable, or take information from a ForCES-based NE may be seen as a prime objective during a network attack.

An attacker in a position to inject false messages into a PL stream can affect either the FE's treatment of the data path (for example, by falsifying control data reported as coming from the CE) or the CE itself (by modifying events or responses reported as coming from the FE). For this reason, CE and FE node authentication and TML message authentication are important.

The PL messages may also contain information of value to an attacker, including information about the configuration of the network, encryption keys, and other sensitive control data, so care must be taken to confine their visibility to authorized users.

* The TML MUST provide a mechanism to authenticate ForCES CEs and FEs, in order to prevent the participation of unauthorized CEs and unauthorized FEs in the control and data path processing of a ForCES NE.

* The TML SHOULD provide a mechanism to ensure message authentication of PL data transferred from the CE to FE (and vice versa), in order to prevent the injection of incorrect data into PL messages.

* The TML SHOULD provide a mechanism to ensure the confidentiality of data transferred from the ForCES PL, in order to prevent disclosure of PL-level information transported via the TML.

The TML SHOULD provide these services by employing TLS or IPsec.

3. Congestion control

   The transport congestion control scheme used by the TML needs to be defined. The congestion control mechanism defined by the TML MUST prevent transport congestive collapse [RFC2914] on either the FE or CE side.

4. Uni/multi/broadcast addressing/delivery, if any

   If there is any mapping between PL- and TML-level uni/multi/broadcast addressing, it needs to be defined.

5. HA decisions

   It is expected that availability of transport links is the TML's responsibility. However, based upon its configuration, the PL may wish to participate in link failover schemes and therefore the TML MUST support this capability.

   Please refer to Section 8 for details.

6. Encapsulations used

   Different types of TMLs will encapsulate the PL messages on different types of headers. The TML needs to specify the encapsulation used.

7. Prioritization

   It is expected that the TML will be able to handle up to 8 priority levels needed by the PL and will provide preferential treatment.

   While the TML needs to define how this is achieved, it should be noted that the requirement for supporting up to 8 priority levels does not mean that the underlying TML MUST be capable of providing up to 8 actual priority levels. In the event that the underlying TML layer does not have support for 8 priority levels, the supported priority levels should be divided between the available TML priority levels. For example, if the TML only supports 2 priority levels, 0-3 could go in one TML priority level, while 4-7 could go in the other.

   The TML MUST NOT re-order config packets with the same priority.

8. Node Overload Prevention

The TML MUST define mechanisms it uses to help prevent node overload.

Overload results in starvation of node compute cycles and/or bandwidth resources, which reduces the operational capacity of a ForCES NE. NE node overload could be deliberately instigated by a hostile node to attack a ForCES NE and create a denial of service (DoS). It could also be created by a variety of other reasons such as large control protocol updates (e.g., BGP flaps), which consequently cause a high frequency of CE to FE table updates, HA failovers, or component failures, which migrate an FE or CE load overwhelming the new CE or FE, etc. Although the environments under which SIP and ForCES operate are different, [RFC5390] provides a good guide to generic node requirements one needs to guard for.

A ForCES node CPU may be overwhelmed because the incoming packet rate is higher than it can keep up with -- in such a case, a node's transport queues grow and transport congestion subsequently follows. A ForCES node CPU may also be adversely overloaded with very few packets, i.e., no transport congestion at all (e.g., a in a DoS attack against a table hashing algorithm that overflows the table and/or keeps the CPU busy so it does not process other tasks). The TML node overload solution specified MUST address both types of node overload scenarios.

5.1. TML Parameterization

It is expected that it should be possible to use a configuration reference point, such as the FEM or the CEM, to configure the TML.

Some of the configured parameters may include:

- PL ID
- Connection Type and associated data. For example, if a TML uses IP/TCP/UDP, then parameters such as TCP and UDP port and IP addresses need to be configured.
- Number of transport connections
- Connection capability, such as bandwidth, etc.
- Allowed/supported connection QoS policy (or congestion control policy)

6. Message Encapsulation

All PL PDUs start with a common header Section 6.1 followed by one or more TLVs Section 6.2, which may nest other TLVs Section 6.2.1. All fields are in network byte order.

6.1. Common Header

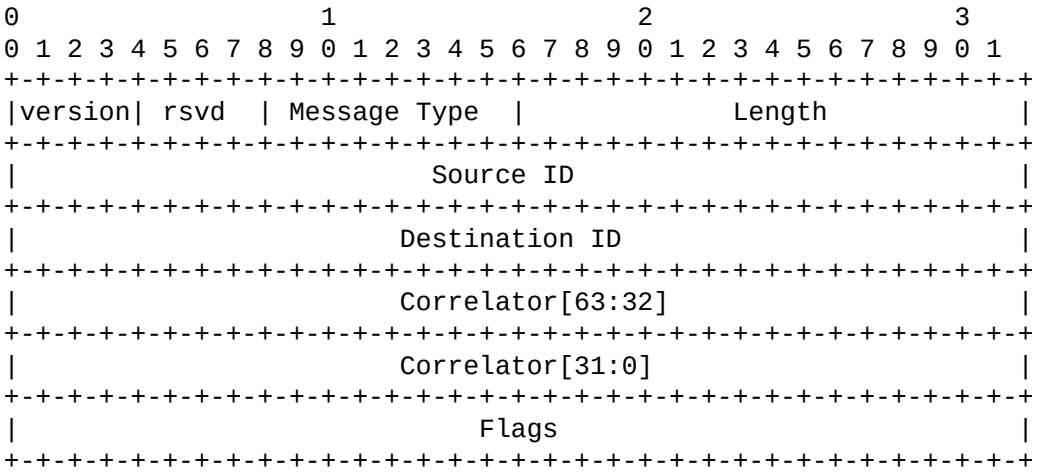


Figure 10: Common Header

The message is 32-bit aligned.

Version (4 bits):
Version number. Current version is 1.

rsvd (4 bits):
Unused at this point. A receiver should not interpret this field. Senders MUST set it to zero and receivers MUST ignore this field.

Message Type (8 bits):
Commands are defined in Section 7.

Length (16 bits):
length of header + the rest of the message in DWORDS (4-byte increments).

Source ID (32 bits):

Dest ID (32 bits):

* Each of the source and destination IDs are 32-bit IDs that are unique NE-wide and that identify the termination points of a ForCES PL message.

* IDs allow multi/broad/unicast addressing with the following approach:

  a. A split address space is used to distinguish FEs from CEs. Even though in a large NE there are typically two or more orders of magnitude of more FEs than CEs, the address space is split uniformly for simplicity.

  b. The address space allows up to 2^30 (over a billion) CEs and the same amount of FEs.

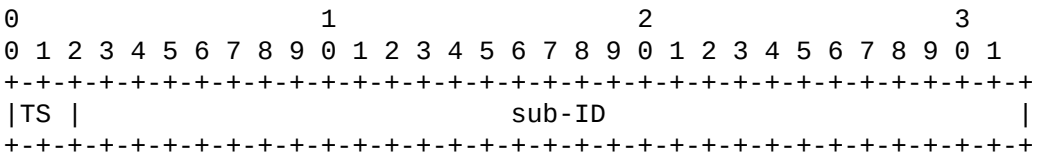


Figure 11: ForCES ID Format

  c. The 2 most significant bits called Type Switch (TS) are used to split the ID space as follows:

TS	Corresponding ID range	Assignment
0b00	0x00000000 to 0x3FFFFFFF	FE IDs (2^30)
0b01	0x40000000 to 0x7FFFFFFF	CE IDs (2^30)
0b10	0x80000000 to 0xBFFFFFFF	reserved
0b11	0xC0000000 to 0xFFFFFFEF	multicast IDs (2^30 - 16)
0b11	0xFFFFFFF0 to 0xFFFFFFFC	reserved
0b11	0xFFFFFFFD	all CEs broadcast
0b11	0xFFFFFFFE	all FEs broadcast
0b11	0xFFFFFFFF	all FEs and CEs (NE) broadcast

Figure 12: Type Switch ID Space

* Multicast or broadcast IDs are used to group endpoints (such as CEs and FEs). As an example, one could group FEs in some functional group, by assigning a multicast ID. Likewise, subgroups of CEs that act, for instance, in a back-up mode may be assigned a multicast ID to hide them from the FE.

+ Multicast IDs can be used for both source or destination IDs.

+ Broadcast IDs can be used only for destination IDs.

* This document does not discuss how a particular multicast ID is associated to a given group though it could be done via configuration process. The list of IDs an FE owns or is part of are listed on the FE Object LFB.

Correlator (64 bits):

This field is set by the CE to correlate ForCES Request messages with the corresponding Response messages from the FE. Essentially, it is a cookie. The correlator is handled transparently by the FE, i.e., for a particular Request message the FE MUST assign the same correlator value in the corresponding Response message. In the case where the message from the CE does not elicit a response, this field may not be useful.

The correlator field could be used in many implementations in specific ways by the CE. For example, the CE could split the correlator into a 32-bit transactional identifier and 32-bit message sequence identifier. Another example is a 64-bit pointer to a context block. All such implementation-specific uses of the correlator are outside the scope of this specification.

It should be noted that the correlator is transmitted on the network as if it were a 64-bit unsigned integer with the leftmost or most significant octet (bits 63-56) transmitted first.

Whenever the correlator field is not relevant, because no message is expected, the correlator field is set to 0.

Flags (32 bits):
Identified so far:

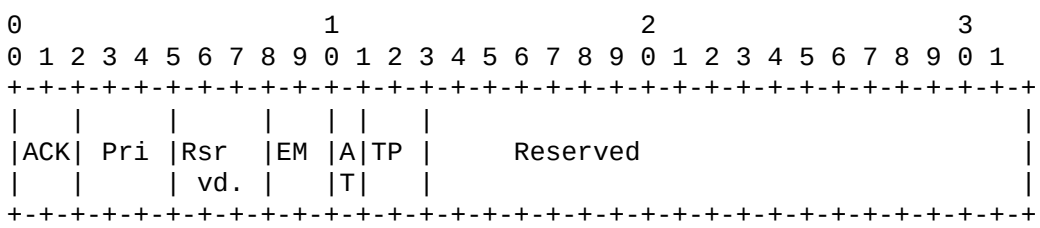


Figure 13: Header Flags

- ACK: ACK indicator (2 bits)

The ACK indicator flag is only used by the CE when sending a Config message (Section 7.6.1) or an HB message (Section 7.10) to indicate to the message receiver whether or not a response is required by the sender. Note that for all other messages than the Config message or the HB message this flag MUST be ignored.

The flag values are defined as follows:

'NoACK' (0b00) - to indicate that the message receiver MUST NOT send any Response message back to this message sender.

'SuccessACK'(0b01) - to indicate that the message receiver MUST send a Response message back only when the message has been successfully processed by the receiver.

'FailureACK'(0b10) - to indicate that the message receiver MUST send a Response message back only when there is failure by the receiver in processing (executing) the message. In other words, if the message can be processed successfully, the sender will not expect any response from the receiver.

'AlwaysACK' (0b11) - to indicate that the message receiver MUST send a Response message.

Note that in above definitions, the term success implies a complete execution without any failure of the message. Anything else than a complete successful execution is defined as a failure for the message processing. As a result, for the execution modes (defined in Section 4.3.1.1) like execute-all-or-none, execute-until-failure, and continue-execute-on-failure, if any single operation among several operations in the same message fails, it will be treated as a failure and result in a response if the ACK indicator has been set to 'FailureACK' or 'AlwaysACK'.

Also note that, other than in Config and HB messages, requirements for responses of messages are all given in a default way rather than by ACK flags. The default requirements of these messages and the expected responses are summarized below. Detailed descriptions can be found in the individual message definitions:

+ Association Setup message always expects a response.

+ Association Teardown Message, and Packet Redirect message, never expect responses.

+ Query message always expects a response.

+ Response message never expects further responses.

- Pri: Priority (3 bits)

ForCES protocol defines 8 different levels of priority (0-7). The priority level can be used to distinguish between different protocol message types as well as between the same message type. The higher the priority value, the more important the PDU is. For example, the REDIRECT packet message could have different priorities to distinguish between routing protocol packets and ARP packets being redirected from FE to CE. The normal priority level is 1. The different priorities imply messages could be re-ordered; however, re-ordering is undesirable when it comes to a set of messages within a transaction and caution should be exercised to avoid this.

- EM: Execution Mode (2 bits)

There are 3 execution modes; refer to Section 4.3.1.1 for details.

Reserved..... (0b00)

'execute-all-or-none' ..... (0b01)

'execute-until-failure' ..... (0b10)

'continue-execute-on-failure' (0b11)

- AT: Atomic Transaction (1 bit)

This flag indicates if the message is a stand-alone message or one of multiple messages that belong to 2PC transaction operations. See Section 4.3.1.2.2 for details.

Stand-alone message ..... (0b0)

2PC transaction message ..... (0b1)

- TP: Transaction Phase (2 bits)

A message from the CE to the FE within a transaction could be indicative of the different phases the transaction is in. Refer to Section 4.3.1.2.2 for details.

SOT (start of transaction) ..... (0b00)

MOT (middle of transaction) .... (0b01)

EOT (end of transaction) .....(0b10)

ABT (abort) .....(0b11)

6.2. Type Length Value (TLV) Structuring

TLVs are extensively used by the ForCES protocol. TLVs have some very nice properties that make them a good candidate for encoding the XML definitions of the LFB class model. These are:

- Providing for binary type-value encoding that is close to the XML string tag-value scheme.
- Allowing for fast generalized binary-parsing functions.
- Allowing for forward and backward tag compatibility. This is equivalent to the XML approach, i.e., old applications can ignore new TLVs and newer applications can ignore older TLVs.

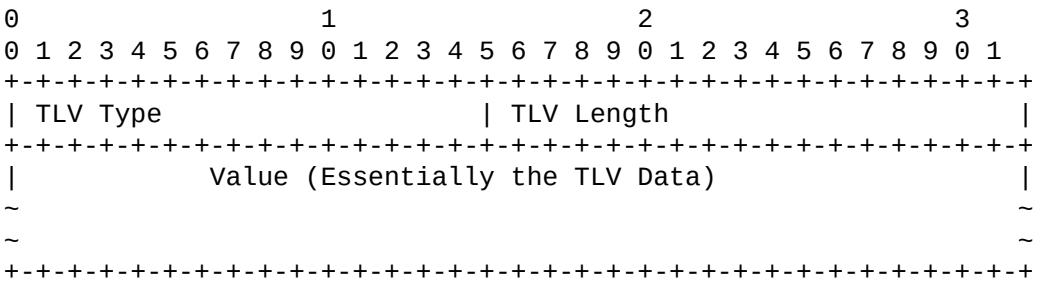


Figure 14: TLV Representation

TLV Type (16):

The TLV type field is 2 octets, and semantically indicates the type of data encapsulated within the TLV.

TLV Length (16):

The TLV length field is 2 octets, and includes the length of the TLV type (2 octets), TLV Length (2 octets), and the length of the TLV data found in the value field, in octets. Note that this length is the actual length of the value, before any padding for alignment is added.

TLV Value (variable):

The TLV value field carries the data. For extensibility, the TLV value may in fact be a TLV. Padding is required when the length is not a multiple of 32 bits, and is the minimum number of octets required to bring the TLV to a multiple of 32 bits. The length of the value before padding is indicated by the TLV Length field.

Note: The value field could be empty, which implies the minimal length a TLV could be is 4 (length of "T" field and length of "L" field).

6.2.1. Nested TLVs

TLV values can be other TLVs. This provides the benefits of protocol flexibility (being able to add new extensions by introducing new TLVs when needed). The nesting feature also allows for a conceptual optimization with the XML LFB definitions to binary PL representation (represented by nested TLVs).

6.2.2. Scope of the T in TLV

There are two global name scopes for the "Type" in the TLV. The first name scope is for OPER-TLVs and is defined in A.4 whereas the second name scope is outside OPER-TLVs and is defined in section A.2.

6.3. ILV

The ILV is a slight variation of the TLV. This sets the type ("T") to be a 32-bit local index that refers to a ForCES component ID (refer to Section 6.4.1).

The ILV length field is a 4-octet integer, and includes the length of the ILV type (4 octets), ILV Length (4 octets), and the length of the ILV data found in the value field, in octets. Note that, as in the case of the TLV, this length is the actual length of the value, before any padding for alignment is added.

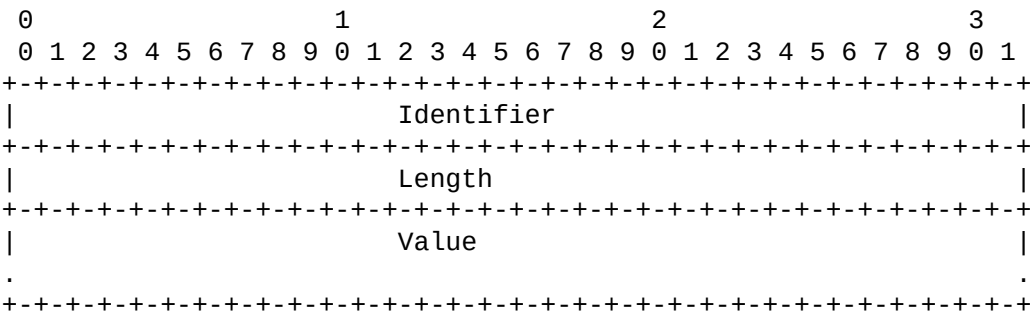


Figure 15: ILV Representation

It should be noted that the "I" values are of local scope and are defined by the data declarations from the LFB definition. Refer to Section 7.1.8 for discussions on usage of ILVs.

6.4. Important Protocol Encapsulations

In this section, we review a few encapsulation concepts that are used by the ForCES protocol for its operations.

We briefly re-introduce two concepts, paths, and keys, from the ForCES model [RFC5812] in order to provide context. The reader is referred to [RFC5812] for a lot of the finer details.

For readability reasons, we introduce the encapsulation schemes that are used to carry content in a protocol message, namely, FULLDATA-TLV, SPARSEDATA-TLV, and RESULT-TLV.

6.4.1. Paths

The ForCES model [RFC5812] defines an XML-based language that allows for a formal definition of LFBs. This is similar to the relationship between ASN.1 and SNMP MIB definition (MIB being analogous to the LFB and ASN.1 being analogous to the XML model language). Any entity that the CE configures on an FE MUST be formally defined in an LFB. These entities could be scalars (e.g., a 32-bit IPv4 address) or vectors (such as a nexthop table). Each entity within the LFB is given a numeric 32-bit identifier known as a "component id". This scheme allows the component to be "addressed" in a protocol construct.

These addressable entities could be hierarchical (e.g., a table column or a cell within a table row). In order to address hierarchical data, the concept of a path is introduced by the model [RFC5812]. A path is a series of 32-bit component IDs that are typically presented in a dot-notation (e.g., 1.2.3.4). Section 7 formally defines how paths are used to reference data that is being encapsulated within a protocol message.

6.4.2. Keys

The ForCES model [RFC5812] defines two ways to address table rows. The standard/common mechanism is to allow table rows to be referenced by a 32-bit index. The secondary mechanism is via keys that allow for content addressing. An example key is a multi-field content key that uses the IP address and prefix length to uniquely reference an IPv4 routing table row. In essence, while the common scheme to address a table row is via its table index, a table row's path could be derived from a key. The KEYINFO-TLV (Section 7) is used to carry the data that is used to do the lookup.

6.4.3. DATA TLVs

Data from or to the FE is carried in two types of TLVs: FULLDATA-TLV and SPARSEDATA-TLV. Responses to operations executed by the FE are carried in RESULT-TLVs.

In FULLDATA-TLV, the data is encoded in such a way that a receiver of such data, by virtue of being armed with knowledge of the path and the LFB definition, can infer or correlate the TLV "Value" contents. This is essentially an optimization that helps reduce the amount of description required for the transported data in the protocol grammar. Refer to Appendix C for an example of FULLDATA-TLVs.

A number of operations in ForCES will need to reference optional data within larger structures. The SPARSEDATA-TLV encoding is provided to make it easier to encapsulate optionally appearing data components. Refer to Appendix C for an example of SPARSEDATA-TLV.

RESULT-TLVs carry responses back from the FE based on a config issued by the CE. Refer to Appendix C for examples of RESULT-TLVs and Section 7.1.7 for layout.

6.4.4. Addressing LFB Entities

Section 6.4.1 and Section 6.4.2 discuss how to target an entity within an LFB. However, the addressing mechanism used requires that an LFB type and instance are selected first. The LFB selector is used to select an LFB type and instance being targeted. Section 7 goes into more details; for our purpose, we illustrate this concept using Figure 16 below. More examples of layouts can be found reading further into the text (example: Figure 22).

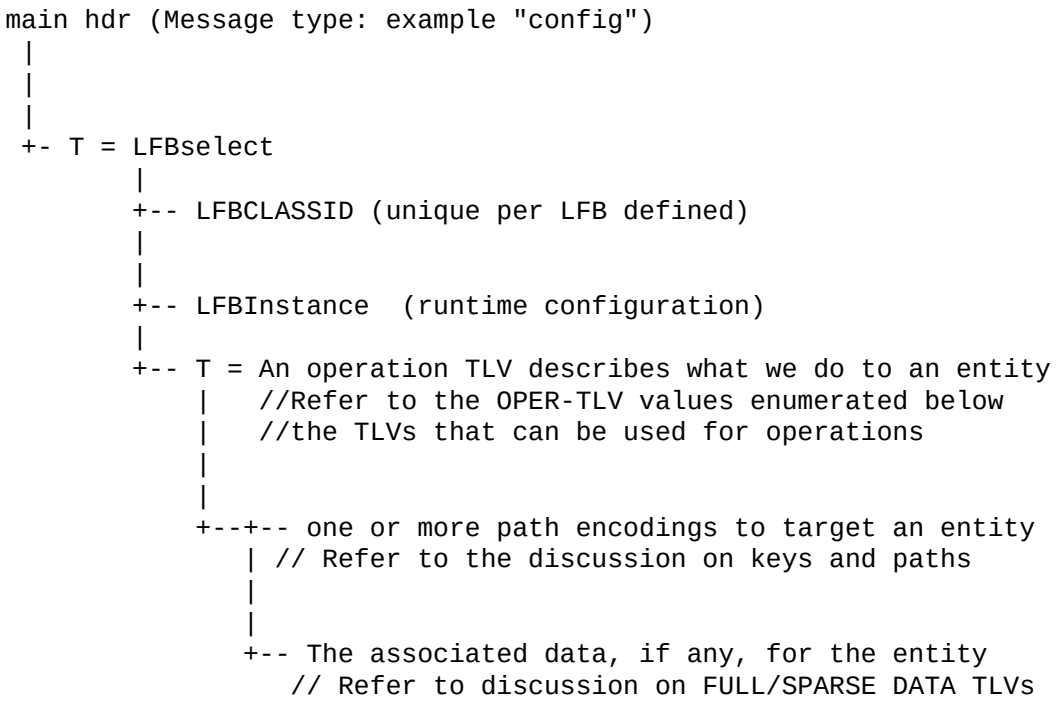


Figure 16: Entity Addressing

7. Protocol Construction

A protocol layer PDU consists of a common header (defined in Section 6.1) and a message body. The common header is followed by a message-type-specific message body. Each message body is formed from one or more top-level TLVs. A top-level TLV may contain one or more sub-TLVs; these sub-TLVs are described in this document as OPER-TLVs, because they describe an operation to be done.

Message Name	Top-Level TLV	OPER-TLV(s)	Reference
Association Setup	(LFBselect)*	REPORT	Section 7.5.1
Association Setup Response	ASRresult-TLV	none	Section 7.5.2
Association Teardown	ASTreason-TLV	none	Section 7.5.3
Config	(LFBselect)+	(SET | SET-PROP | DEL | COMMIT | TRCOMP)+	Section 7.6.1
Config Response	(LFBselect)+	(SET-RESPONSE | SET-PROP-RESPONSE | DEL-RESPONSE | COMMIT-RESPONSE)+	Section 7.6.2
Query	(LFBselect)+	(GET | GET-PROP)+	Section 7.7.1
Query Response	(LFBselect)+	(GET-RESPONSE | GET-PROP-RESPONSE)+	Section 7.7.2
Event Notification	LFBselect	REPORT	Section 7.8
Packet Redirect	REDIRECT-TLV	none	Section 7.9
Heartbeat	none	none	Section 7.10

Table 1

The different messages are illustrated in Table 1. The different message type numerical values are defined in Appendix A.1. All the TLV values are defined in Appendix A.2.

An LFBselect TLV (refer to Section 7.1.5) contains the LFB Classid and LFB instance being referenced as well as the OPER-TLV(s) being applied to that reference.

Each type of OPER-TLV is constrained as to how it describes the paths and selectors of interest. The following BNF describes the basic structure of an OPER-TLV and Table 2 gives the details for each type.

```
OPER-TLV := 1*PATH-DATA-TLV
PATH-DATA-TLV := PATH  [DATA]
PATH := flags IDcount IDs [SELECTOR]
SELECTOR :=  KEYINFO-TLV
DATA := FULLDATA-TLV / SPARSEDATA-TLV / RESULT-TLV /
        1*PATH-DATA-TLV
KEYINFO-TLV := KeyID FULLDATA-TLV
FULLDATA-TLV := encoded data component which may nest
                further FULLDATA-TLVs
SPARSEDATA-TLV := encoded data that may have optionally
                  appearing components
RESULT-TLV := Holds result code and optional FULLDATA-TLV
```

Figure 17: BNF of OPER-TLV

- PATH-DATA-TLV identifies the exact component targeted and may have zero or more paths associated with it. The last PATH-DATA-TLV in the case of nesting of paths via the DATA construct in the case of SET, SET-PROP requests, and GET-RESPONSE/GET-PROP-RESPONSE is terminated by encoded data or response in the form of either FULLDATA-TLV or SPARSEDATA-TLV or RESULT-TLV.

- PATH provides the path to the data being referenced.

  * flags (16 bits) are used to further refine the operation to be applied on the path. More on these later.

  * IDcount (16 bits): count of 32-bit IDs

  * IDs: zero or more 32-bit IDs (whose count is given by IDcount) defining the main path. Depending on the flags, IDs could be field IDs only or a mix of field and dynamic IDs. Zero is used for the special case of using the entirety of the containing context as the result of the path.

- SELECTOR is an optional construct that further defines the PATH. Currently, the only defined selector is the KEYINFO-TLV, used for selecting an array entry by the value of a key field. The presence of a SELECTOR is correct only when the flags also indicate its presence.

- A KEYINFO-TLV contains information used in content keying.

  * A 32-bit KeyID is used in a KEYINFO-TLV. It indicates which key for the current array is being used as the content key for array entry selection.

  * The key's data is the data to look for in the array, in the fields identified by the key field. The information is encoded according to the rules for the contents of a FULLDATA-TLV, and represents the field or fields that make up the key identified by the KeyID.

o DATA may contain a FULLDATA-TLV, SPARSEDATA-TLV, a RESULT-TLV, or 1 or more further PATH-DATA selections. FULLDATA-TLV and SPARSEDATA-TLV are only allowed on SET or SET-PROP requests, or on responses that return content information (GET-RESPONSE, for example). PATH-DATA may be included to extend the path on any request.

  * Note: Nested PATH-DATA-TLVs are supported as an efficiency measure to permit common subexpression extraction.

  * FULLDATA-TLV and SPARSEDATA-TLV contain "the data" whose path has been selected by the PATH. Refer to Section 7.1 for details.

  * The following table summarizes the applicability and restrictions of the FULL/SPARSEDATA-TLVs and the RESULT-TLV to the OPER-TLVs.

OPER-TLV	DATA TLV	RESULT-TLV
SET		none
SET-PROP	(FULLDATA-TLV | SPARSEDATA-TLV)+	none
SET-RESPONSE	none	(RESULT-TLV)+
SET-PROP-RESPONSE	none	(RESULT-TLV)+
DEL		none
DEL-RESPONSE	none	(RESULT-TLV)+
GET	none	none
GET-PROP	none	none
GET-RESPONSE	(FULLDATA-TLV)+	(RESULT-TLV)*
GET-PROP-RESPONSE	(FULLDATA-TLV)+	(RESULT-TLV)*
REPORT	(FULLDATA-TLV)+	none
COMMIT	none	none
COMMIT-RESPONSE	none	(RESULT-TLV)+
TRCOMP	none	none

Table 2

- RESULT-TLV contains the indication of whether the individual SET or SET-PROP succeeded. RESULT-TLV is included on the assumption that individual parts of a SET request can succeed or fail separately.

In summary, this approach has the following characteristics:

- There can be one or more LFB class ID and instance ID combinations targeted in a message (batch).
- There can one or more operations on an addressed LFB class ID/ instance ID combination (batch).
- There can be one or more path targets per operation (batch).
- Paths may have zero or more data values associated (flexibility and operation specific).

It should be noted that the above is optimized for the case of a single LFB class ID and instance ID targeting. To target multiple instances within the same class, multiple LFBselects are needed.

7.1. Discussion on Encoding

Section 6.4.3 discusses the two types of DATA encodings (FULLDATA-TLV and SPARSEDATA-TLV) and the justifications for their existence. In this section, we explain how they are encoded.

7.1.1. Data Packing Rules

The scheme for encoding data used in this document adheres to the following rules:

- The Value ("V" of TLV) of FULLDATA-TLV will contain the data being transported. This data will be as was described in the LFB definition.
- Variable-sized data within a FULLDATA-TLV will be encapsulated inside another FULLDATA-TLV inside the V of the outer TLV. For an example of such a setup, refer to Appendices C and D.
- In the case of FULLDATA-TLVs:
  - When a table is referred to in the PATH (IDs) of a PATH-DATA-TLV, then the FULLDATA-TLV's "V" will contain that table's row content prefixed by its 32-bit index/subscript. On the other

hand, the PATH may contain an index pointing to a row in table; in such a case, the FULLDATA-TLV's "V" will only contain the content with the index in order to avoid ambiguity.

7.1.2. Path Flags

Only bit 0, the SELECTOR Bit, is currently used in the path flags as illustrated in Figure 18.

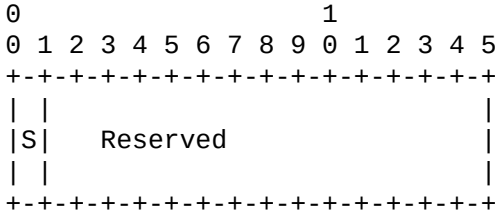


Figure 18: Path Flags

The semantics of the flag are defined as follows:

- SELECTOR Bit: F_SELKEY(set to 1) indicates that a KEY Selector is present following this path information, and should be considered in evaluating the path content.

7.1.3. Relation of Operational Flags with Global Message Flags

Global flags, such as the execution mode and the atomicity indicators defined in the header, apply to all operations in a message. Global flags provide semantics that are orthogonal to those provided by the operational flags, such as the flags defined in path-data. The scope of operational flags is restricted to the operation.

7.1.4. Content Path Selection

The KEYINFO-TLV describes the KEY as well as associated KEY data. KEYs, used for content searches, are restricted and described in the LFB definition.

7.1.5. LFBselect-TLV

The LFBselect TLV is an instance of a TLV as defined in Section 6.2. The definition is as follows:

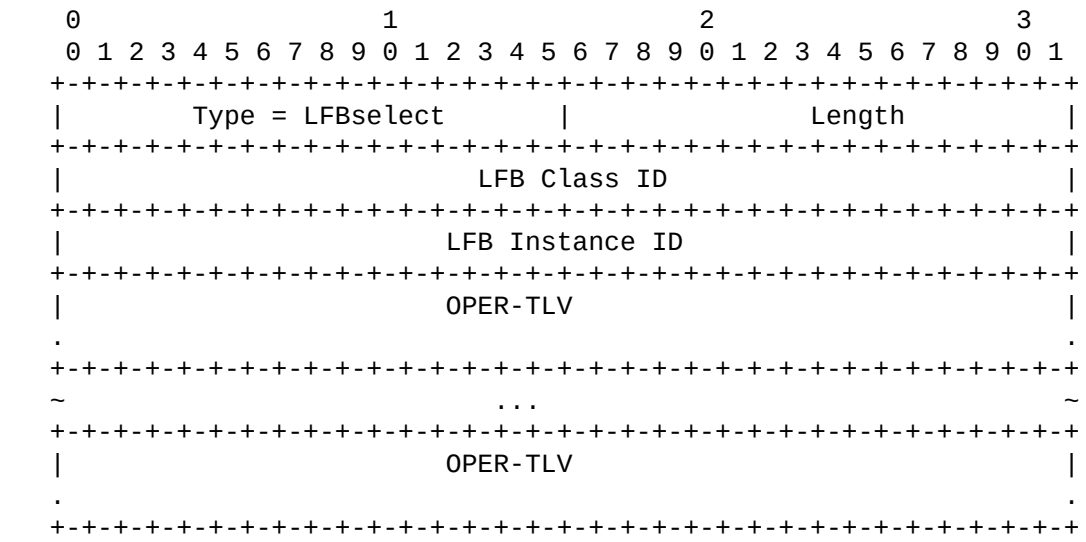


Figure 19: PL PDU Layout

Type:

The type of the TLV is "LFBselect"

Length:

Length of the TLV including the T and L fields, in octets.

LFB Class ID:

This field uniquely recognizes the LFB class/type.

LFB Instance ID:

This field uniquely identifies the LFB instance.

OPER-TLV:

It describes an operation nested in the LFBselect TLV. Note that usually there SHOULD be at least one OPER-TLV present for an LFB select TLV.

7.1.6. OPER-TLV

The OPER-TLV is a place holder in the grammar for TLVs that define operations. The different types are defined in Table 3, below.

OPER-TLV	TLV "Type"	Comments
SET	0x0001	From CE to FE. Used to create or add or update components
SET-PROP	0x0002	From CE to FE. Used to create or add or update component properties
SET-RESPONSE	0x0003	From FE to CE. Used to carry response of a SET
SET-PROP-RESPONSE	0x0004	From FE to CE. Used to carry response of a SET-PROP
DEL	0x0005	From CE to FE. Used to delete or remove an component
DEL-RESPONSE	0x0006	From FE to CE. Used to carry response of a DEL
GET	0x0007	From CE to FE. Used to retrieve an component
GET-PROP	0x0008	From CE to FE. Used to retrieve an component property
GET-RESPONSE	0x0009	From FE to CE. Used to carry response of a GET
GET-PROP-RESPONSE	0x000A	From FE to CE. Used to carry response of a GET-PROP
REPORT	0x000B	From FE to CE. Used to carry an asynchronous event
COMMIT	0x000C	From CE to FE. Used to issue a commit in a 2PC transaction
COMMIT-RESPONSE	0x000D	From FE to CE. Used to confirm a commit in a 2PC transaction
TRCOMP	0x000E	From CE to FE. Used to indicate NE-wide success of 2PC transaction

Table 3

Different messages use OPER-TLV and define how they are used (refer to Table 1 and Table 2).

SET, SET-PROP, and GET/GET-PROP requests are issued by the CE and do not carry RESULT-TLVs. On the other hand, SET-RESPONSE, SET-PROP-RESPONSE, and GET-RESPONSE/GET-PROP-RESPONSE carry RESULT-TLVs.

A GET-RESPONSE in response to a successful GET will have FULLDATA-TLVs added to the leaf paths to carry the requested data. For GET operations that fail, instead of the FULLDATA-TLV there will be a RESULT-TLV.

For a SET-RESPONSE/SET-PROP-RESPONSE, each FULLDATA-TLV or SPARSEDATA-TLV in the original request will be replaced with a RESULT-TLV in the response. If the request set the FailureACK flag, then only those items that failed will appear in the response. If the request was for AlwaysACK, then all components of the request will appear in the response with RESULT-TLVs.

Note that if a SET/SET-PROP request with a structure in a FULLDATA-TLV is issued, and some field in the structure is invalid, the FE will not attempt to indicate which field was invalid, but rather will indicate that the operation failed. Note further that if there are multiple errors in a single leaf PATH-DATA/FULLDATA-TLB, the FE can select which error it chooses to return. So if a FULLDATA-TLV for a SET/SET-PROP of a structure attempts to write one field that is read only, and attempts to set another field to an invalid value, the FE can return indication of either error.

A SET/SET-PROP operation on a variable-length component with a length of 0 for the item is not the same as deleting it. If the CE wishes to delete, then the DEL operation should be used whether the path refers to an array component or an optional structure component.

7.1.7. RESULT TLV

The RESULT-TLV is an instance of TLV defined in Section 6.2. The definition is as follows:

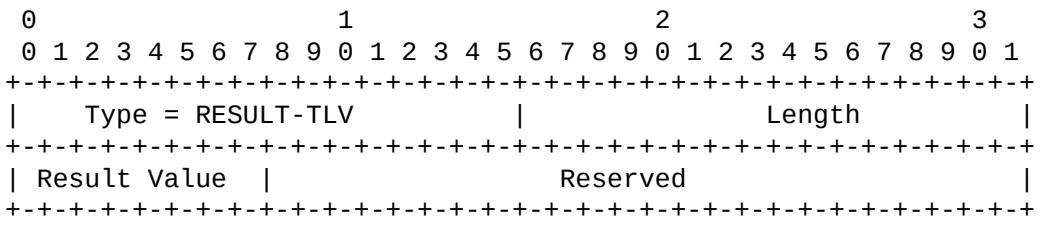


Figure 20: RESULT-TLV

Defined Result Values

Result Value	Value	Definition
E_SUCCESS	0x00	Success
E_INVALID_HEADER	0x01	Unspecified error with header.
E_LENGTH_MISMATCH	0x02	Header length field does not match actual packet length.
E_VERSION_MISMATCH	0x03	Unresolvable mismatch in versions.
E_INVALID_DESTINATION_PID	0x04	Destination PID is invalid for the message receiver.
E_LFB_UNKNOWN	0x05	LFB Class ID is not known by receiver.
E_LFB_NOT_FOUND	0x06	LFB Class ID is known by receiver but not currently in use.
E_LFB_INSTANCE_ID_NOT_FOUND	0x07	LFB Class ID is known but the specified instance of that class does not exist.
E_INVALID_PATH	0x08	The specified path is impossible.
E_COMPONENT_DOES_NOT_EXIST	0x09	The specified path is possible but the component does not exist (e.g., attempt to modify a table row that has not been created).
E_EXISTS	0x0A	The specified object exists but it cannot exist for the operation to succeed (e.g., attempt to add an existing LFB instance or array subscript).
E_NOT_FOUND	0x0B	The specified object does not exist but it MUST exist for the operation to succeed (e.g., attempt to delete a non-existing LFB instance or array subscript).

E_READ_ONLY	0x0C	Attempt to modify a read-only value.
E_INVALID_ARRAY_CREATION	0x0D	Attempt to create an array with an unallowed subscript.
E_VALUE_OUT_OF_RANGE	0x0E	Attempt to set a parameter to a value outside of its allowable range.
E_CONTENTS_TOO_LONG	0x0D	Attempt to write contents larger than the target object space (i.e., exceeding a buffer).
E_INVALID_PARAMETERS	0x10	Any other error with data parameters.
E_INVALID_MESSAGE_TYPE	0x11	Message type is not acceptable.
E_INVALID_FLAGS	0x12	Message flags are not acceptable for the given message type.
E_INVALID_TLV	0x13	A TLV is not acceptable for the given message type.
E_EVENT_ERROR	0x14	Unspecified error while handling an event.
E_NOT_SUPPORTED	0x15	Attempt to perform a valid ForCES operation that is unsupported by the message receiver.
E_MEMORY_ERROR	0x16	A memory error occurred while processing a message (no error detected in the message itself).
E_INTERNAL_ERROR	0x17	An unspecified error occurred while processing a message (no error detected in the message itself).
-	0x18-0xFE	Reserved
E_UNSPECIFIED_ERROR	0xFF	Unspecified error (for when the FE cannot decide what went wrong).

Table 4

7.1.8. DATA TLV

A FULLDATA-TLV has "T"= FULLDATA-TLV and a 16-bit length followed by the data value/contents. Likewise, a SPARSEDATA-TLV has "T" = SPARSEDATA-TLV, a 16-bit length, followed by the data value/contents. In the case of the SPARSEDATA-TLV, each component in the Value part of the TLV will be further encapsulated in an ILV.

Below are the encoding rules for the FULLDATA-TLV and SPARSEDATA-TLVs. Appendix C is very useful in illustrating these rules:

1. Both ILVs and TLVs MUST be 32-bit aligned. Any padding bits used for the alignment MUST be zero on transmission and MUST be ignored upon reception.

2. FULLDATA-TLVs may be used at a particular path only if every component at that path level is present. In example 1(c) of Appendix C, this concept is illustrated by the presence of all components of the structure S in the FULLDATA-TLV encoding. This requirement holds regardless of whether the fields are fixed or variable length, mandatory or optional.

   * If a FULLDATA-TLV is used, the encoder MUST lay out data for each component in the same order in which the data was defined in the LFB specification. This ensures the decoder is able to retrieve the data. To use the example 1 again in Appendix C, this implies the encoder/decoder is assumed to have knowledge of how structure S is laid out in the definition.

   * In the case of a SPARSEDATA-TLV, it does not need to be ordered since the "I" in the ILV uniquely identifies the component. Examples 1(a) and 1(b) in Appendix C illustrate the power of SPARSEDATA-TLV encoding.

3. Inside a FULLDATA-TLV

   * The values for atomic, fixed-length fields are given without any TLV encapsulation.

   * The values for atomic, variable-length fields are given inside FULLDATA-TLVs.

   * The values for arrays are in the form of index/subscript, followed by value as stated in "Data Packing Rules" (Section 7.1.1) and demonstrated by the examples in the appendices.

4. Inside a SPARSEDATA-TLV

   * The values of all fields MUST be given with ILVs (32-bit index, 32-bit length).

5. FULLDATA-TLVs cannot contain an ILV.

6. A FULLDATA-TLV can also contain a FULLDATA-TLV for variable-sized components. The decoding disambiguation is assumed from rule #3 above.

7.1.9. SET and GET Relationship

It is expected that a GET-RESPONSE would satisfy the following:

- It would have exactly the same path definitions as those sent in the GET. The only difference is that a GET-RESPONSE will contain FULLDATA-TLVs.

- It should be possible to take the same GET-RESPONSE and convert it to a SET successfully by merely changing the T in the operational TLV.

- There are exceptions to this rule:

  1. When a KEY selector is used with a path in a GET operation, that selector is not returned in the GET-RESPONSE; instead, the cooked result is returned. Refer to the examples using KEYS to see this.

  2. When dumping a whole table in a GET, the GET-RESPONSE that merely edits the T to be SET will end up overwriting the table.

7.2. Protocol Encoding Visualization

The figure below shows a general layout of the PL PDU. A main header is followed by one or more LFB selections each of which may contain one or more operations.

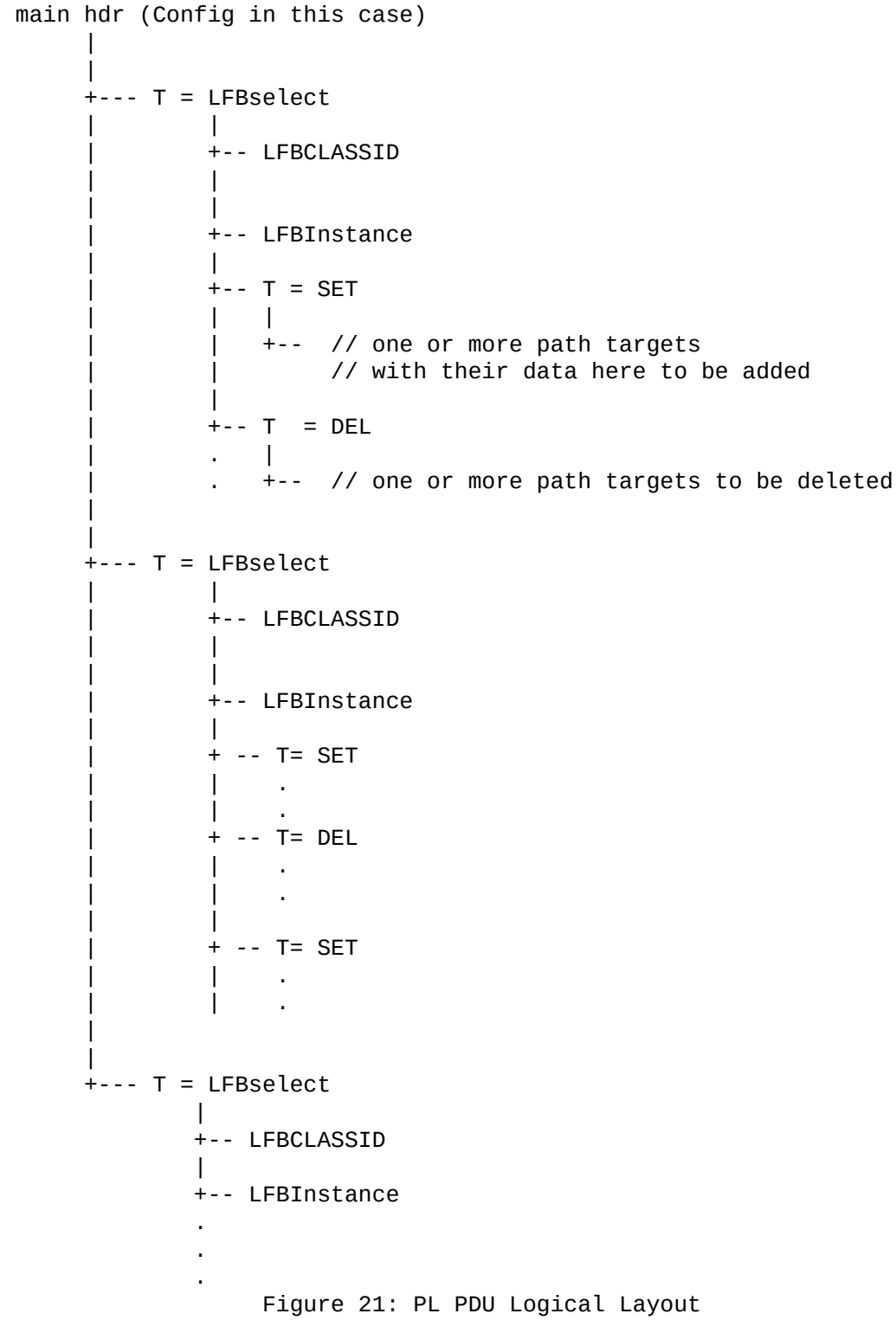


Figure 21: PL PDU Logical Layout

The figure below shows a more detailed example of the general layout of the operation within a targeted LFB selection. The idea is to show the different nesting levels a path could take to get to the target path.

```
T = SET
|  |
|  +- T = Path-data
|       |
|       + -- flags
|       + -- IDCount
|       + -- IDs
|       |
|       +- T = Path-data
|          |
|          + -- flags
|          + -- IDCount
|          + -- IDs
|          |
|          +- T = Path-data
|             |
|             + -- flags
|             + -- IDCount
|             + -- IDs
|             + -- T = KEYINFO-TLV
|             |    + -- KEY_ID
|             |    + -- KEY_DATA
|             |
|             + -- T = FULLDATA-TLV
|                  + -- data
|
|
T = SET
|  |
|  +- T = Path-data
|  |  |
|  |  + -- flags
|  |  + -- IDCount
|  |  + -- IDs
|  |  |
|  |  + -- T = FULLDATA-TLV
|  |          + -- data
|  +- T = Path-data
|     |
```

```
|     + -- flags
|     + -- IDCount
|     + -- IDs
|     |
|     + -- T = FULLDATA-TLV
|              + -- data
T = DEL
   |
   +- T = Path-data
        |
        + -- flags
        + -- IDCount
        + -- IDs
        |
        +- T = Path-data
           |
           + -- flags
           + -- IDCount
           + -- IDs
           |
           +- T = Path-data
              |
              + -- flags
              + -- IDCount
              + -- IDs
              + -- T = KEYINFO-TLV
              |    + -- KEY_ID
              |    + -- KEY_DATA
              +- T = Path-data
                   |
                   + -- flags
                   + -- IDCount
                   + -- IDs
```

Figure 22: Sample Operation Layout

Appendix D shows a more concise set of use cases on how the data encoding is done.

7.3. Core ForCES LFBs

There are two LFBs that are used to control the operation of the ForCES protocol and to interact with FEs and CEs:

- FE Protocol LFB

o FE Object LFB

Although these LFBs have the same form and interface as other LFBs, they are special in many respects. They have fixed well-known LFB Class and Instance IDs. They are statically defined (no dynamic instantiation allowed), and their status cannot be changed by the protocol: any operation to change the state of such LFBs (for instance, in order to disable the LFB) MUST result in an error. Moreover, these LFBs MUST exist before the first ForCES message can be sent or received. All components in these LFBs MUST have pre-defined default values. Finally, these LFBs do not have input or output ports and do not integrate into the intra-FE LFB topology.

7.3.1. FE Protocol LFB

The FE Protocol LFB is a logical entity in each FE that is used to control the ForCES protocol. The FE Protocol LFB Class ID is assigned the value 0x2. The FE Protocol LFB Instance ID is assigned the value 0x1. There MUST be one and only one instance of the FE Protocol LFB in an FE. The values of the components in the FE Protocol LFB have pre-defined default values that are specified here. Unless explicit changes are made to these values using Config messages from the CE, these default values MUST be used for correct operation of the protocol.

The formal definition of the FE Protocol Object LFB can be found in Appendix B.

7.3.1.1. FE Protocol Capabilities

FE Protocol capabilities are read-only.

7.3.1.1.1. SupportableVersions

ForCES protocol version(s) supported by the FE.

7.3.1.1.2. FE Protocol Components

FE Protocol components (can be read and set).

7.3.1.1.2.1. CurrentRunningVersion

Current running version of the ForCES protocol.

7.3.1.1.2.2. FEID

FE unicast ID.

7.3.1.1.2.3. MulticastFEIDs

FE multicast ID(s) list - This is a list of multicast IDs to which the FE belongs. These IDs are configured by the CE.

7.3.1.1.2.4. CEHBPolicy

CE heartbeat policy - This policy, along with the parameter 'CE Heartbeat Dead Interval (CE HDI)' as described below, defines the operating parameters for the FE to check the CE liveness. The policy values with meanings are listed as follows:

- 0 (default) - This policy specifies that the CE will send a Heartbeat message to the FE(s) whenever the CE reaches a time interval within which no other PL messages were sent from the CE to the FE(s); refer to Section 4.3.3 and Section 7.10 for details. The CE HDI component as described below is tied to this policy.

- 1 - The CE will not generate any HB messages. This actually means that the CE does not want the FE to check the CE liveness.

- Others - Reserved.

7.3.1.1.2.5. CEHDI

CE Heartbeat Dead Interval (CE HDI) - The time interval the FE uses to check the CE liveness. If FE has not received any messages from CE within this time interval, FE deduces lost connectivity, which implies that the CE is dead or the association to the CE is lost. Default value is 30 s.

7.3.1.1.2.6. FEHBPolicy

FE heartbeat policy - This policy, along with the parameter 'FE Heartbeat Interval (FE HI)', defines the operating parameters for how the FE should behave so that the CE can deduce its liveness. The policy values and the meanings are:

- 0 (default) - The FE should not generate any Heartbeat messages. In this scenario, the CE is responsible for checking FE liveness by setting the PL header ACK flag of the message it sends to AlwaysACK. The FE responds to the CE whenever the CE sends such Heartbeat Request messages. Refer to Section 7.10 and Section 4.3.3 for details.

- 1 - This policy specifies that the FE MUST actively send a Heartbeat message if it reaches the time interval assigned by the FE HI as long as no other messages were sent from the FE to the CE during that interval as described in Section 4.3.3.

- Others - Reserved.

7.3.1.1.2.7. FEHI

FE Heartbeat Interval (FE HI) - The time interval the FE should use to send HB as long as no other messages were sent from the FE to the CE during that interval as described in Section 4.3.3. The default value for an FE HI is 500 ms.

7.3.1.1.2.8. CEID

Primary CEID - The CEID with which the FE is associated.

7.3.1.1.2.9. LastCEID

Last Primary CEID - The CEID of the last CE with which the FE associated. This CE ID is reported to the new Primary CEID.

7.3.1.1.2.10. BackupCEs

The list of backup CEs an FE can use as backups. Refer to Section 8 for details.

7.3.1.1.2.11. CEFailoverPolicy

CE failover policy - This specifies the behavior of the FE when the association with the CE is lost. There is a very tight relation between CE failover policy and Section 7.3.1.1.2.8, Section 7.3.1.1.2.10, Section 7.3.1.1.2.12, and Section 8. When an association is lost, depending on configuration, one of the policies listed below is activated.

- 0 (default) - The FE should stop functioning immediately and transition to FE OperDisable.

- 1 - The FE should continue running and do what it can even without an associated CE. This basically requires that the FE support CE Graceful restart (and defines such support in its capabilities). If the CEFTI expires before the FE re-associates with either the primary CEID (Section 7.3.1.1.2.8) or one of possibly several backup CEs (Section 7.3.1.1.2.10), the FE will go operationally down.

- Others - Reserved.

7.3.1.1.2.12. CEFTI

CE Failover Timeout Interval (CEFTI) - The time interval associated with the CE failover policy case '0' and '1'. The default value is set to 300 seconds. Note that it is advisable to set the CEFTI value much higher than the CE Heartbeat Dead Interval (CE HDI) since the effect of expiring this parameter is devastating to the operation of the FE.

7.3.1.1.2.13. FERestartPolicy

FE restart policy - This specifies the behavior of the FE during an FE restart. The restart may be from an FE failure or other reasons that have made the FE down and then need to restart. The values are defined as follows:

- 0(default)- Restart the FE from scratch. In this case, the FE should start from the pre-association phase.

- Others - Reserved for future use.

7.3.2. FE Object LFB

The FE Object LFB is a logical entity in each FE and contains components relative to the FE itself, and not to the operation of the ForCES protocol.

The formal definition of the FE Object LFB can be found in [RFC5812]. The model captures the high-level properties of the FE that the CE needs to know to begin working with the FE. The class ID for this LFB class is also assigned in [RFC5812]. The singular instance of this class will always exist, and will always have instance ID 0x1 within its class. It is common, although not mandatory, for a CE to fetch much of the component and capability information from this LFB instance when the CE begins controlling the operation of the FE.

7.4. Semantics of Message Direction

Recall: The PL provides a master(CE)-slave(FE) relationship. The LFBs reside at the FE and are controlled by CE.

When messages go from the CE, the LFB selector (class and instance) refers to the destination LFB selection that resides in the FE.

When messages go from the FE to the CE, the LFB selector (class and instance) refers to the source LFB selection that resides in the FE.

7.5. Association Messages

The ForCES Association messages are used to establish and tear down associations between FEs and CEs.

7.5.1. Association Setup Message

This message is sent by the FE to the CE to set up a ForCES association between them.

Message transfer direction:
FE to CE

Message header:
The Message Type in the header is set to MessageType= 'AssociationSetup'. The ACK flag in the header MUST be ignored, and the Association Setup message always expects to get a response from the message receiver (CE), whether or not the setup is successful. The correlator field in the header is set, so that FE can correlate the response coming back from the CE correctly. The FE may set the source ID to 0 in the header to request that the CE should assign an FE ID for the FE in the Setup Response message.

Message body:
The Association Setup message body optionally consists of zero, one, or two LFBselect TLVs, as described in Section 7.1.5. The Association Setup message only operates on the FE Object and FE Protocol LFBs; therefore, the LFB class ID in the LFBselect TLV only points to these two kinds of LFBs.

The OPER-TLV in the LFBselect TLV is defined as a 'REPORT' operation. More than one component may be announced in this message using the REPORT operation to let the FE declare its configuration parameters in an unsolicited manner. These may contain components suggesting values such as the FE HB Interval or the FEID. The OPER-TLV used is defined below.

OPER-TLV for Association Setup:

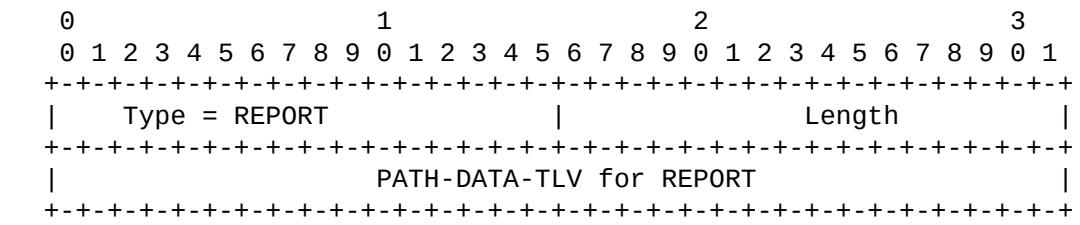


Figure 23: OPER-TLV

Type:
Only one operation type is defined for the Association Setup message:

Type = "REPORT" - This type of operation is for the FE to report something to the CE.

PATH-DATA-TLV for REPORT:
This is generically a PATH-DATA-TLV format that has been defined in Section 7 in the PATH-DATA BNF definition. The PATH-DATA-TLV for the REPORT operation MAY contain FULLDATA-TLV(s) but SHALL NOT contain any RESULT-TLV in the data format. The RESULT-TLV is defined in Section 7.1.7 and the FULLDATA-TLV is defined in Section 7.1.8.

To better illustrate the above PDU format, a tree structure for the format is shown below:

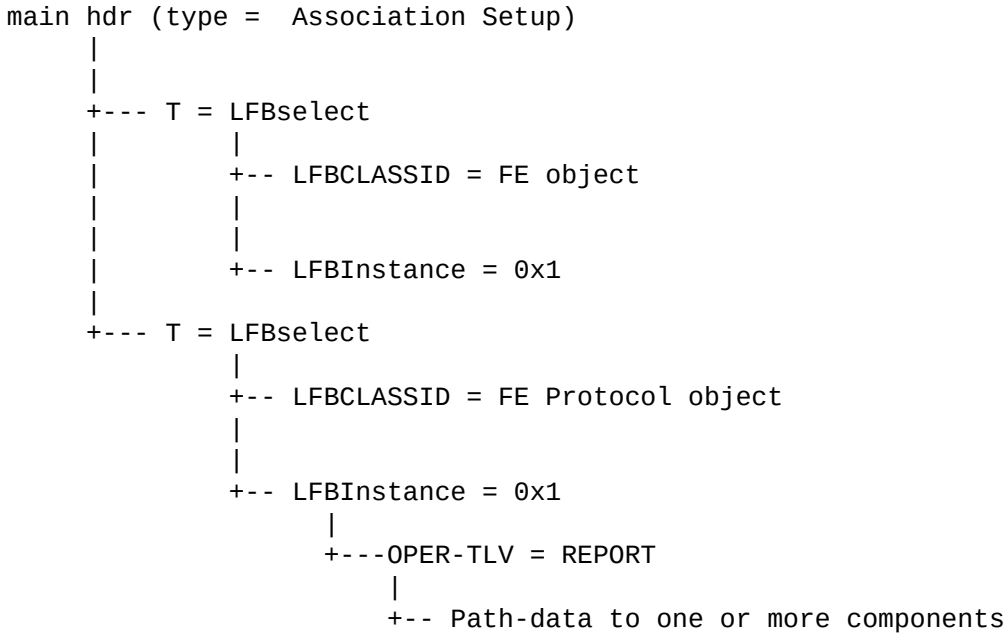


Figure 24: PDU Format for Association Setup Message

7.5.2. Association Setup Response Message

This message is sent by the CE to the FE in response to the Setup message. It indicates to the FE whether or not the setup is successful, i.e., whether an association is established.

Message transfer direction:

CE to FE

Message header:

The Message Type in the header is set to MessageType= 'AssociationSetupResponse'. The ACK flag in the header MUST be ignored, and the Setup Response message never expects to get any more responses from the message receiver (FE). The destination ID in the header will be set to the source ID in the corresponding Association Setup message, unless that source ID was 0. If the corresponding source ID was 0, then the CE will assign an FE ID value and use that value for the destination ID.

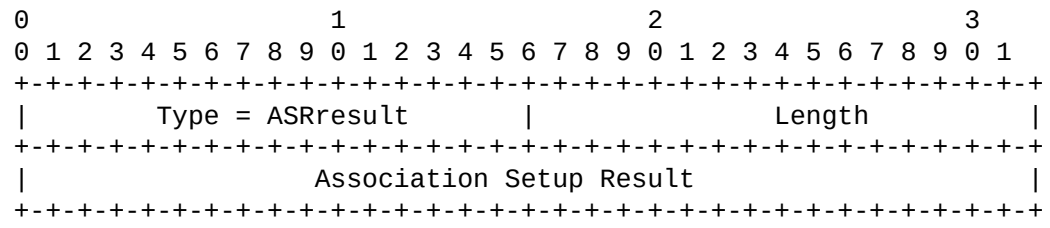


Figure 25: ASResult OPER-TLV

Type (16 bits):

The type of the TLV is "ASResult".

Length (16 bits):

Length of the TLV including the T and L fields, in octets.

Association Setup result (32 bits):

This indicates whether the Setup message was successful or whether the FE request was rejected by the CE. The defined values are:

0 = success

1 = FE ID invalid

2 = permission denied

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
main hdr (type =  Association Setup Response)
 |
 |
 +--- T = ASResult-TLV
```

Figure 26: PDU Format for Association Setup Response Message

7.5.3. Association Teardown Message

This message can be sent by the FE or CE to any ForCES element to end its ForCES association with that element.

Message transfer direction:

CE to FE, or FE to CE (or CE to CE)

Message Header:

The Message Type in the header is set to MessageType= "AssociationTeardown". The ACK flag MUST be ignored. The correlator field in the header MUST be set to zero and MUST be ignored by the receiver.

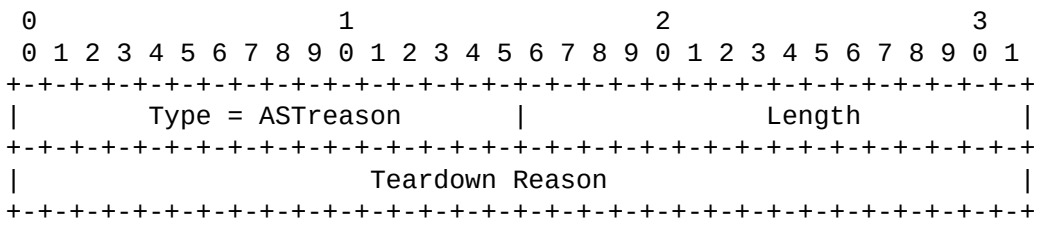


Figure 27: ASTreason-TLV

Type (16 bits):

The type of the TLV is "ASTreason".

Length (16 bits):

Length of the TLV including the T and L fields, in octets.

Teardown reason (32 bits):

This indicates the reason why the association is being terminated. Several reason codes are defined as follows.

0 - normal teardown by administrator

1 - error - loss of heartbeats

2 - error - out of bandwidth

3 - error - out of memory

4 - error - application crash

255 - error - other or unspecified

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
main hdr (type =  Association Teardown)
 |
 |
 +--- T = ASTreason-TLV
```

Figure 28: PDU Format for Association Teardown Message

7.6. Configuration Messages

The ForCES Configuration messages are used by CE to configure the FEs in a ForCES NE and report the results back to the CE.

7.6.1. Config Message

This message is sent by the CE to the FE to configure LFB components in the FE. This message is also used by the CE to subscribe/ unsubscribe to LFB events.

As usual, a Config message is composed of a common header followed by a message body that consists of one or more TLV data formats. Detailed description of the message is as follows:

Message transfer direction:

CE to FE

Message header:

The Message Type in the header is set to MessageType= 'Config'. The ACK flag in the header can be set to any value defined in Section 6.1, to indicate whether or not a response from the FE is expected by the message.

OPER-TLV for Config:

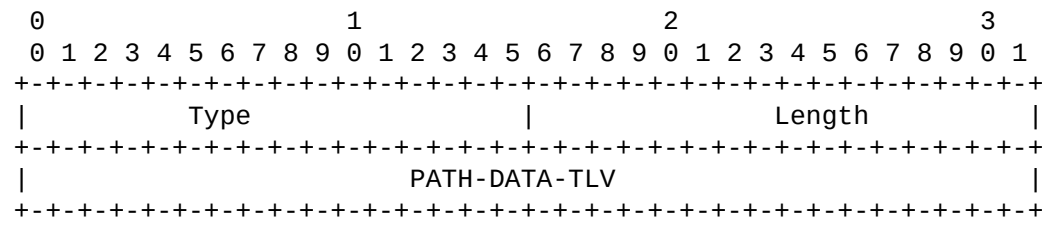


Figure 29: OPER-TLV for Config

Type:

The operation type for Config message. Two types of operations for the Config message are defined:

Type = "SET" - This operation is to set LFB components

Type = "SET-PROP" - This operation is to set LFB component properties.

Type = "DEL" - This operation is to delete some LFB components.

Type = "COMMIT" - This operation is sent to the FE to commit in a 2pc transaction. A COMMIT TLV is an empty TLV, i.e., it has no "V"alue. In other words, there is a length of 4 (which is for the header only).

Type = "TRCOMP" - This operation is sent to the FE to mark the success from an NE perspective of a 2pc transaction. A TRCOMP TLV is an empty TLV, i.e., it has no "V"alue. In other words, there is a length of 4 (which is for the header only).

PATH-DATA-TLV:

This is generically a PATH-DATA-TLV format that has been defined in Section 7 in the PATH-DATA-TLV BNF definition. The restriction on the use of PATH-DATA-TLV for SET/SET-PROP operation is that it MUST contain either FULLDATA-TLV or SPARSEDATA-TLV(s), but MUST NOT contain any RESULT-TLV. The restriction on the use of PATH-DATA-TLV for DEL operation is it MAY contain FULLDATA-TLV or SPARSEDATA-TLV(s), but MUST NOT contain any RESULT-TLV. The RESULT-TLV is defined in Section 7.1.7 and FULLDATA-TLVs and SPARSEDATA-TLVs are defined in Section 7.1.8.

Note: For Event subscription, the events will be defined by the individual LFBs.

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
main hdr (type = Config)
 |
 |
 +--- T = LFBselect
 .        |
 .        +-- LFBCLASSID = target LFB class
 .        |
          |
          +-- LFBInstance = target LFB instance
          |
          |
          +-- T = operation { SET }
          |   |
          |   +--  // one or more path targets
          |      // associated with FULLDATA-TLV or SPARSEDATA-TLV(s)
          |
          +-- T = operation { DEL }
          |   |
          |   +--  // one or more path targets
          |
          +-- T = operation { COMMIT } //A COMMIT TLV is an empty TLV
                    .
                    .
```

Figure 30: PDU Format for Configuration Message

7.6.2. Config Response Message

This message is sent by the FE to the CE in response to the Config message. It indicates whether or not the Config was successful on the FE and also gives a detailed response regarding the configuration result of each component.

Message transfer direction:

FE to CE

Message header:

The Message Type in the header is set to MessageType= 'Config Response'. The ACK flag in the header is always ignored, and the Config Response message never expects to get any further response from the message receiver (CE).

OPER-TLV for Config Response:

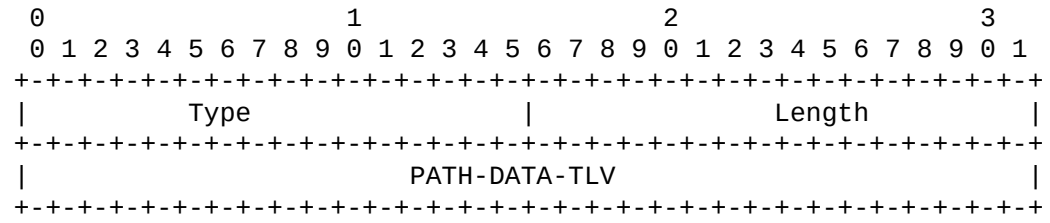


Figure 31: OPER-TLV for Config Response

Type:

The operation type for Config Response message. Two types of operations for the Config Response message are defined:

Type = "SET-RESPONSE" - This operation is for the response of the SET operation of LFB components.

Type = "SET-PROP-RESPONSE" - This operation is for the response of the SET-PROP operation of LFB component properties.

Type = "DEL-RESPONSE" - This operation is for the response of the DELETE operation of LFB components.

Type = "COMMIT-RESPONSE" - This operation is sent to the CE to confirm a commit success in a 2pc transaction. A COMMIT-RESPONSE TLV MUST contain a RESULT-TLV indicating success or failure.

PATH-DATA-TLV:

This is generically a PATH-DATA-TLV format that has been defined in Section 7 in the PATH-DATA-TLV BNF definition. The restriction on the use of PATH-DATA-TLV for SET-RESPONSE operation is that it MUST contain RESULT-TLV(s). The restriction on the use of PATH-DATA-TLV for DEL-RESPONSE operation is it also MUST contain RESULT-TLV(s). The RESULT-TLV is defined in Section 7.1.7.

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
 main hdr (type = ConfigResponse)
  |
  |
  +--- T = LFBselect
  .        |
  .        +-- LFBCLASSID = target LFB class
  .        |
           |
           +-- LFBInstance = target LFB instance
           |
           |
           +-- T = operation { SET-RESPONSE }
           |   |
           |   +--  // one or more path targets
           |        // associated with FULL or SPARSEDATA-TLV(s)
           |
           +-- T = operation { DEL-RESPONSE }
           |   |
           |   +--  // one or more path targets
           |
           +-- T = operation { COMMIT-RESPONSE }
           |           |
           |           +--  RESULT-TLV
```

Figure 32: PDU Format for Config Response Message

7.7. Query Messages

The ForCES Query messages are used by the CE to query LFBs in the FE for information like LFB components, capabilities, statistics, etc. Query messages include the Query message and the Query Response message.

7.7.1. Query Message

A Query message is composed of a common header and a message body that consists of one or more TLV data formats. Detailed description of the message is as follows:

Message transfer direction:

from CE to FE

Message header:

The Message Type in the header is set to MessageType= 'Query'. The ACK flag in the header is always ignored, and a full response for a Query message is always expected. The Correlator field in the header is set, so that the CE can locate the response back from FE correctly.

OPER-TLV for Query:

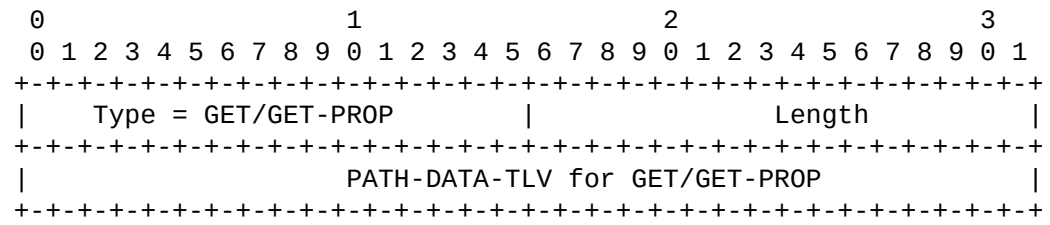


Figure 33: TLV for Query

Type:

The operation type for query. Two operation types are defined:

Type = "GET" - This operation is to request to get LFB components.

Type = "GET-PROP" - This operation is to request to get LFB component properties.

PATH-DATA-TLV for GET/GET-PROP:

This is generically a PATH-DATA-TLV format that has been defined in Section 7 in the PATH-DATA-TLV BNF definition. The restriction on the use of PATH-DATA-TLV for GET/GET-PROP operation is it MUST NOT contain any SPARSEDATA-TLV or FULLDATA- TLV and RESULT-TLV in the data format.

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
main hdr (type = Query)
 |
 |
 +--- T = LFBselect
 .        |
 .        +-- LFBCLASSID = target LFB class
 .        |
          |
          +-- LFBInstance = target LFB instance
          |
          |
          +-- T = operation { GET }
          |   |
          |   +--  // one or more path targets
          |
          +-- T = operation { GET }
          .   |
          .   +--  // one or more path targets
          .
```

Figure 34: PDU Format for Query Message

7.7.2. Query Response Message

When receiving a Query message, the receiver should process the message and come up with a query result. The receiver sends the query result back to the message sender by use of the Query Response message. The query result can be the information being queried if the query operation is successful, or can also be error codes if the query operation fails, indicating the reasons for the failure.

A Query Response message is also composed of a common header and a message body consisting of one or more TLVs describing the query result. Detailed description of the message is as follows:

Message transfer direction:

from FE to CE

Message header:

The Message Type in the header is set to MessageType= 'QueryResponse'. The ACK flag in the header is ignored. As a response itself, the message does not expect a further response.

OPER-TLV for Query Response:

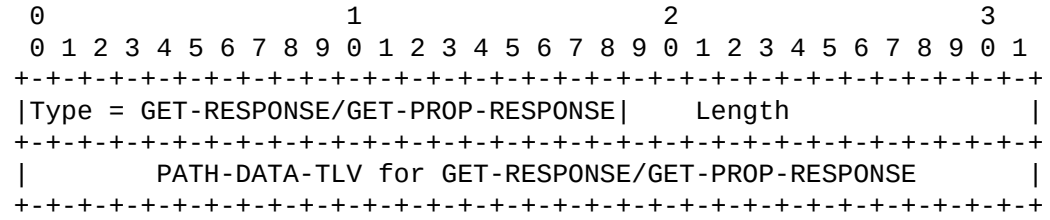


Figure 35: TLV for Query Response

Type:

The operation type for query response. One operation type is defined:

Type = "GET-RESPONSE" - This operation is for the response of the GET operation of LFB components.

Type = "GET-PROP-RESPONSE" - This operation is for the response of the GET-PROP operation of LFB components.

PATH-DATA-TLV for GET-RESPONSE/GET-PROP-RESPONSE:

This is generically a PATH-DATA-TLV format that has been defined in Section 7 in the PATH-DATA-TLV BNF definition. The PATH-DATA- TLV for the GET-RESPONSE operation MAY contain SPARSEDATA-TLV, FULLDATA-TLV, and/or RESULT-TLV(s) in the data encoding. The RESULT-TLV is defined in Section 7.1.7 and the SPARSEDATA-TLVs and FULLDATA-TLVs are defined in Section 7.1.8.

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
main hdr (type = QueryResponse)
  |
  |
  +--- T = LFBselect
  .        |
  .        +-- LFBCLASSID = target LFB class
  .        |
           |
           +-- LFBInstance = target LFB instance
           |
           |
           +-- T = operation { GET-RESPONSE }
           |   |
           |   +--  // one or more path targets
           |
           +-- T = operation { GET-PROP-RESPONSE }
           .   |
           .   +--  // one or more path targets
           .
```

Figure 36: PDU Format for Query Response Message

7.8. Event Notification Message

Event Notification message is used by the FE to asynchronously notify the CE of events that happen in the FE.

All events that can be generated in an FE are subscribable by the CE. The CE can subscribe to an event via a Config message with the SET-PROP operation, where the included path specifies the event, as defined by the LFB Library and described by the FE Model.

As usual, an Event Notification message is composed of a common header and a message body that consists of one or more TLV data formats. Detailed description of the message is as follows:

Message transfer direction:

FE to CE

Message header:

The Message Type in the message header is set to MessageType = 'EventNotification'. The ACK flag in the header MUST be ignored by the CE, and the Event Notification message does not expect any response from the receiver.

OPER-TLV for Event Notification:

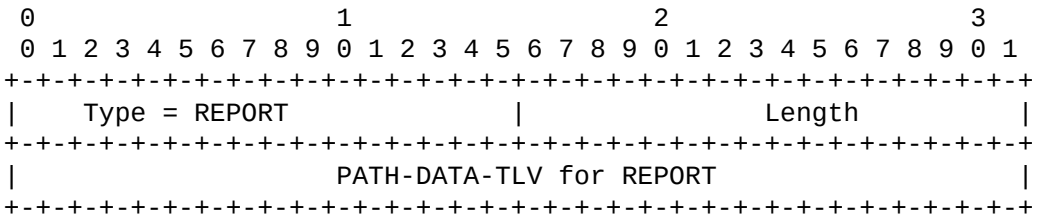


Figure 37: TLV for Event Notification

Type:

Only one operation type is defined for the Event Notification message:

Type = "REPORT" - This type of operation is for the FE to report something to the CE.

PATH-DATA-TLV for REPORT:

This is generically a PATH-DATA-TLV format that has been defined in Section 7 in the PATH-DATA-TLV BNF definition. The PATH-DATA- TLV for the REPORT operation MAY contain FULLDATA-TLV or SPARSEDATA-TLV(s) but MUST NOT contain any RESULT-TLV in the data format.

To better illustrate the above PDU format, a tree structure for the format is shown below:

```
main hdr (type = Event Notification)
  |
  |
  +--- T = LFBselect
             |
             +-- LFBCLASSID = target LFB class
             |
             |
             +-- LFBInstance = target LFB instance
             |
             |
             +-- T = operation { REPORT }
             |   |
             |   +--  // one or more path targets
             |        // associated with FULL/SPARSE DATA TLV(s)
             +-- T = operation { REPORT }
             .   |
             .   +--  // one or more path targets
             .        // associated with FULL/SPARSE DATA TLV(s)
```

Figure 38: PDU Format for Event Notification Message

7.9. Packet Redirect Message

A Packet Redirect message is used to transfer data packets between the CE and FE. Usually, these data packets are control packets, but they may be just data path packets that need further (exception or high-touch) processing. It is also feasible that this message carries no data packets and rather just meta data.

The Packet Redirect message data format is formatted as follows:

Message transfer direction:

CE to FE or FE to CE

Message header:

The Message Type in the header is set to MessageType= 'PacketRedirect'.

Message body:

This consists of one or more TLVs that contain or describe the packet being redirected. The TLV is specifically a Redirect TLV (with the TLV Type="Redirect"). Detailed data format of a Redirect TLV for a Packet Redirect message is as follows:

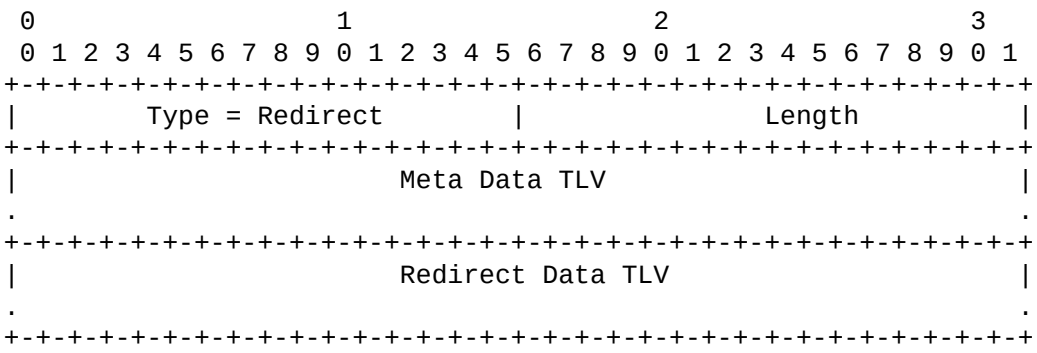


Figure 39: Redirect_Data TLV

Meta Data TLV:

This is a TLV that specifies meta data associated with followed redirected data. The TLV is as follows:

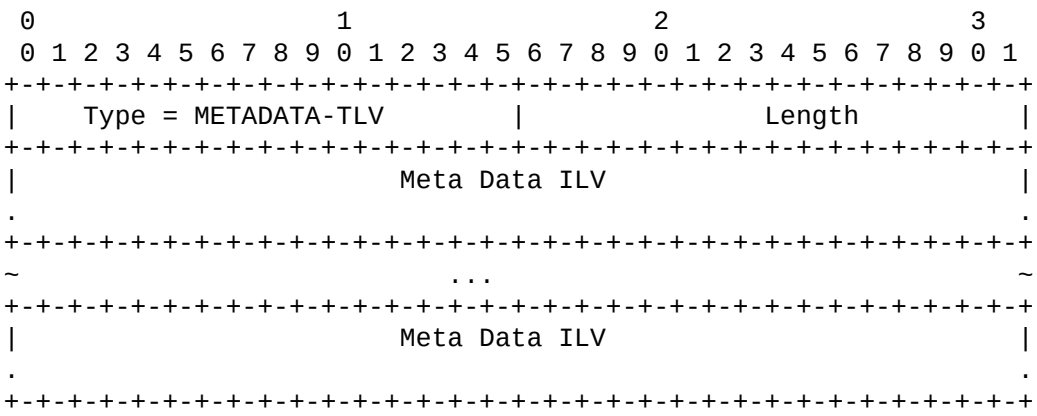


Figure 40: METADATA-TLV

Meta Data ILV:

This is an Identifier-Length-Value format that is used to describe one meta data. The ILV has the format as:

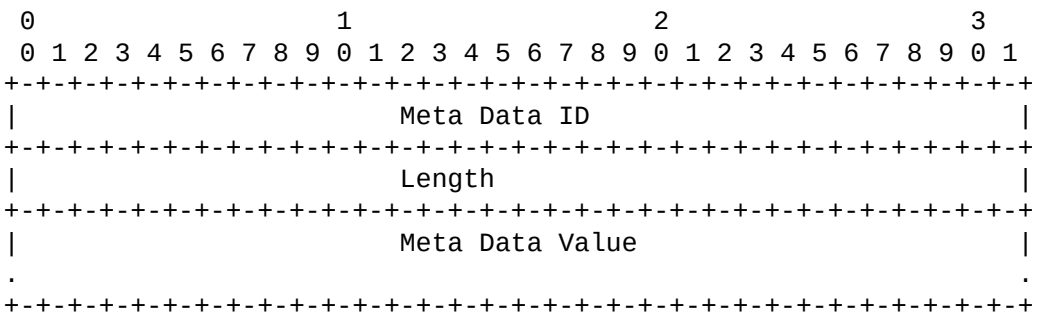


Figure 41: Meta Data ILV

where Meta Data ID is an identifier for the meta data, which is statically assigned by the LFB definition.

Redirect Data TLV:

This is a TLV describing one packet of data to be directed via the redirect operation. The TLV format is as follows:

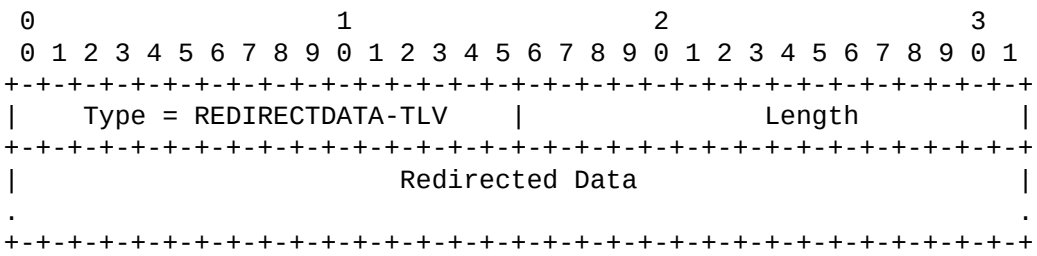


Figure 42: Redirect Data TLV

Redirected Data:

This field contains the packet that is to be redirected in network byte order. The packet should be 32 bits aligned as is the data for all TLVs. The meta data infers what kind of packet is carried in value field and therefore allows for easy decoding of data encapsulated.

To better illustrate the above PDU format, a tree structure for the format is shown below:

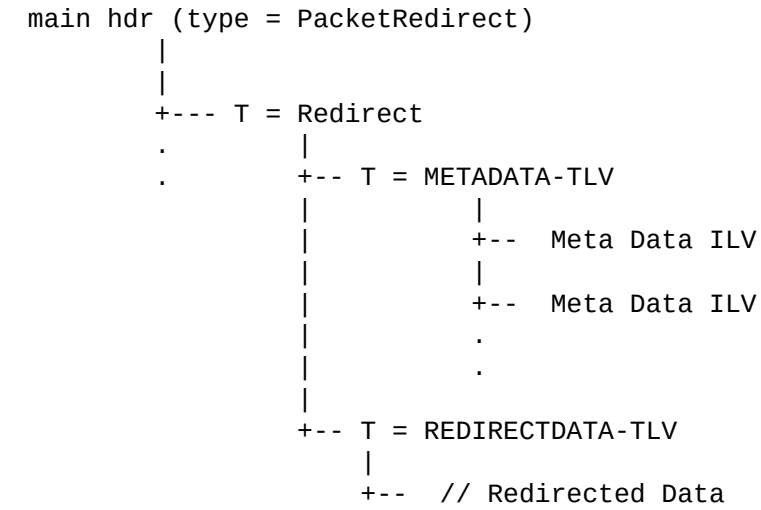


Figure 43: PDU Format for Packet Redirect Message

7.10. Heartbeat Message

The Heartbeat (HB) message is used for one ForCES element (FE or CE) to asynchronously notify one or more other ForCES elements in the same ForCES NE on its liveness. Section 4.3.3 describes the traffic-sensitive approach used.

A Heartbeat message is sent by a ForCES element periodically. The parameterization and policy definition for heartbeats for an FE are managed as components of the FE Protocol Object LFB, and can be set by CE via a Config message. The Heartbeat message is a little different from other protocol messages in that it is only composed of a common header, with the message body left empty. A detailed description of the message is as follows:

Message transfer direction:

FE to CE or CE to FE

Message header:

The Message Type in the message header is set to MessageType = 'Heartbeat'. Section 4.3.3 describes the HB mechanisms used. The ACK flag in the header MUST be set to either 'NoACK' or 'AlwaysACK' when the HB is sent.

* When set to 'NoACK', the HB is not soliciting for a response.

* When set to 'AlwaysACK', the HB Message sender is always expecting a response from its receiver. According to the HB policies defined in Section 7.3.1, only the CE can send such an HB message to query FE liveness. For simplicity and because of the minimal nature of the HB message, the response to an HB message is another HB message, i.e., no specific HB Response message is defined. Whenever an FE receives an HB message marked with 'AlwaysACK' from the CE, the FE MUST send an HB message back immediately. The HB message sent by the FE in response to the 'AlwaysACK' MUST modify the source and destination IDs so that the ID of the FE is the source ID and the CE ID of the sender is the destination ID, and MUST change the ACK information to 'NoACK'. A CE MUST NOT respond to an HB message with 'AlwaysACK' set.

* When set to anything else other than 'NoACK' or 'AlwaysACK', the HB message is treated as if it was a 'NoACK'.

The correlator field in the HB message header SHOULD be set accordingly when a response is expected so that a receiver can correlate the response correctly. The correlator field MAY be ignored if no response is expected.

Message body:

The message body is empty for the Heartbeat message.

8. High Availability Support

The ForCES protocol provides mechanisms for CE redundancy and failover, in order to support High Availability as defined in [RFC3654]. FE redundancy and FE to FE interaction is currently out of scope of this document. There can be multiple redundant CEs and FEs in a ForCES NE. However, at any one time only one primary CE can control the FEs though there can be multiple secondary CEs. The FE and the CE PL are aware of the primary and secondary CEs. This information (primary, secondary CEs) is configured in the FE and in the CE PLs during pre-association by the FEM and the CEM respectively. Only the primary CE sends control messages to the FEs.

8.1. Relation with the FE Protocol

High Availability parameterization in an FE is driven by configuring the FE Protocol Object LFB (refer to Appendix B and Section 7.3.1). The FE Heartbeat Interval, CE Heartbeat Dead Interval, and CE

Heartbeat policy help in detecting connectivity problems between an FE and CE. The CE failover policy defines the reaction on a detected failure.

Figure 44 extends the state machine illustrated in Figure 4 to allow for new states that facilitate connection recovery.

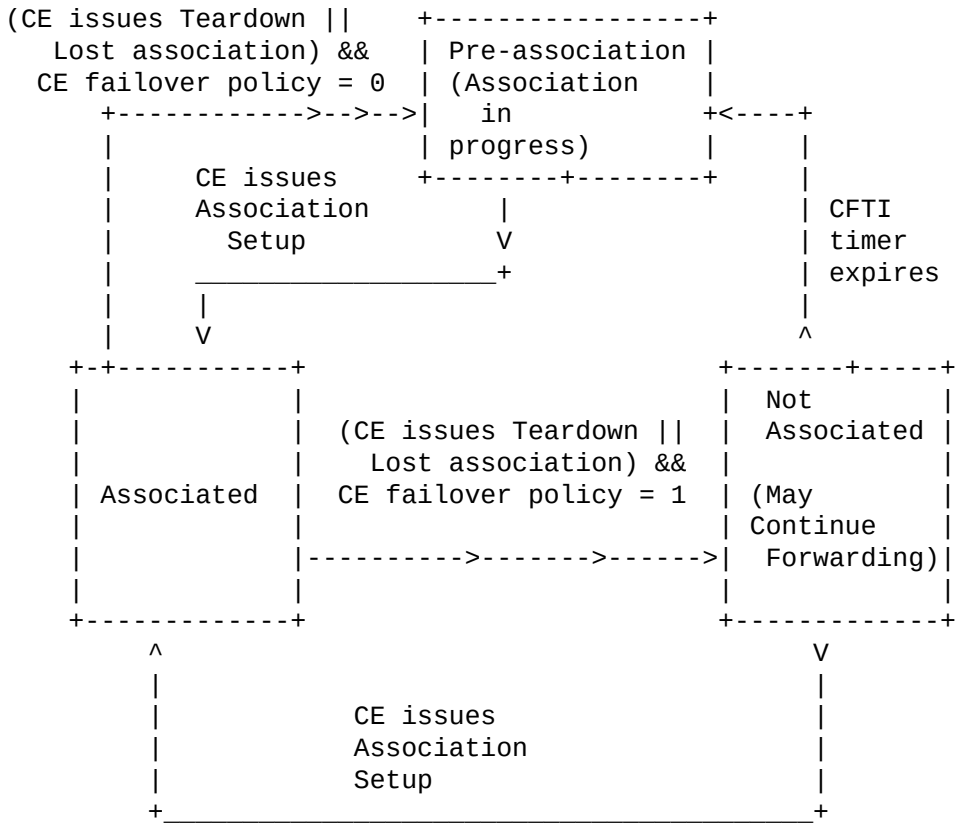


Figure 44: FE State Machine Considering HA

Section 4.2 describes transitions between the pre-association, associated, and not associated states.

When communication fails between the FE and CE (which can be caused by either the CE or link failure but not FE related), either the TML on the FE will trigger the FE PL regarding this failure or it will be detected using the HB messages between FEs and CEs. The communication failure, regardless of how it is detected, MUST be considered as a loss of association between the CE and corresponding FE.

If the FE's FEPO CE failover policy is configured to mode 0 (the default), it will immediately transition to the pre-association phase. This means that if association is again established, all FE state will need to be re-established.

If the FE's FEPO CE failover policy is configured to mode 1, it indicates that the FE is capable of HA restart recovery. In such a case, the FE transitions to the not associated state and the CEFTI timer is started. The FE MAY continue to forward packets during this state. It MAY also recycle through any configured secondary CEs in a round-robin fashion. It first adds its primary CE to the tail of backup CEs and sets its primary CE to be the first secondary. It then attempts to associate with the CE designated as the new primary CE. If it fails to re-associate with any CE and the CEFTI expires, the FE then transitions to the pre-association state.

If the FE, while in the not associated state, manages to reconnect to a new primary CE before CEFTI expires, it transitions to the associated state. Once re-associated, the FE tries to recover any state that may have been lost during the not associated state. How the FE achieves this is out of scope for this document.

Figure 45 below illustrates the ForCES message sequences that the FE uses to recover the connection.

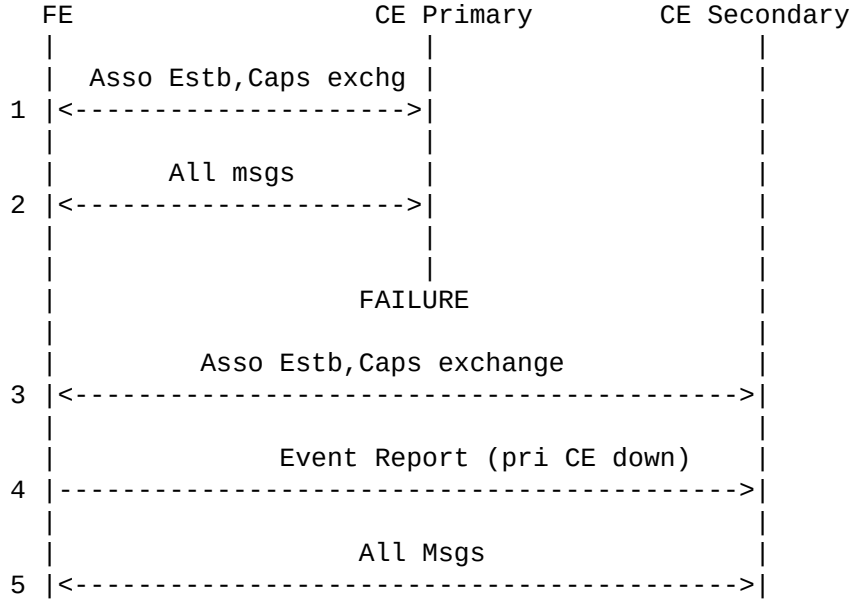


Figure 45: CE Failover for Report Primary Mode

A CE-to-CE synchronization protocol would be needed to support fast failover as well as to address some of the corner cases; however, this will not be defined by the ForCES protocol as it is out of scope for this specification.

An explicit message (a Config message setting primary CE component in the FE Protocol Object) from the primary CE can also be used to change the primary CE for an FE during normal protocol operation.

Also note that the FEs in a ForCES NE could also use a multicast CE ID, i.e., they could be associated with a group of CEs (this assumes the use of a CE-CE synchronization protocol, which is out of scope for this specification). In this case, the loss of association would mean that communication with the entire multicast group of CEs has been lost. The mechanisms described above will apply for this case as well during the loss of association. If, however, the secondary CE was also using the multicast CE ID that was lost, then the FE will need to form a new association using a different CE ID. If the capability exists, the FE MAY first attempt to form a new association with the original primary CE using a different non-multicast CE ID.

8.2. Responsibilities for HA

TML level:

1. The TML controls logical connection availability and failover.

2. The TML also controls peer HA management.

At this level, control of all lower layers, for example, transport level (such as IP addresses, MAC addresses, etc.) and associated links going down are the role of the TML.

PL level:

All other functionality, including configuring the HA behavior during setup, the CE IDs used to identify primary and secondary CEs, protocol messages used to report CE failure (Event Report), Heartbeat messages used to detect association failure, messages to change the primary CE (Config), and other HA-related operations described before, are the PL responsibility.

To put the two together, if a path to a primary CE is down, the TML would take care of failing over to a backup path, if one is available. If the CE is totally unreachable, then the PL would be informed and it would take the appropriate actions described earlier.

9. Security Considerations

The ForCES framework document [RFC3746], Section 8, goes into extensive detail on a variety of security threats, the possible effects of those threats on the protocol, and responses to those threats. This document does not repeat that discussion; the reader is referred to the ForCES framework document [RFC3746] for those details and how the ForCES architecture addresses them.

ForCES PL uses security services provided by the ForCES TML. The TML provides security services such as endpoint authentication service, message authentication service, and confidentiality service. Endpoint authentication service is invoked at the time of the pre-association connection establishment phase and message authentication is performed whenever the FE or CE receives a packet from its peer.

The following are the general security mechanisms that need to be in place for ForCES PL.

- Security mechanisms are session controlled -- that is, once the security is turned on depending upon the chosen security level (No Security, Authentication, Confidentiality), it will be in effect for the entire duration of the session.

- An operator should configure the same security policies for both primary and backup FEs and CEs (if available). This will ensure uniform operations and avoid unnecessary complexity in policy configuration.

9.1. No Security

When "No Security" is chosen for ForCES protocol communication, both endpoint authentication and message authentication service needs to be performed by ForCES PL. Both these mechanism are weak and do not involve cryptographic operation. An operator can choose "No Security" level when the ForCES protocol endpoints are within a single box, for example.

In order to have interoperable and uniform implementation across various security levels, each CE and FE endpoint MUST implement this level.

What is described below (in Section 9.1.1 and Section 9.1.2) are error checks and not security procedures. The reader is referred to Section 9.2 for security procedures.

9.1.1. Endpoint Authentication

Each CE and FE PL maintains a list of associations as part of its configuration. This is done via the CEM and FEM interfaces. An FE MUST connect to only those CEs that are configured via the FEM; similarly, a CE should accept the connection and establish associations for the FEs which are configured via the CEM. The CE should validate the FE identifier before accepting the connections during the pre-association phase.

9.1.2. Message Authentication

When a CE or FE initiates a message, the receiving endpoint MUST validate the initiator of the message by checking the common header CE or FE identifiers. This will ensure proper protocol functioning. This extra processing step is recommended even when the underlying TML layer security services exist.

9.2. ForCES PL and TML Security Service

This section is applicable if an operator wishes to use the TML security services. A ForCES TML MUST support one or more security services such as endpoint authentication service, message authentication service, and confidentiality service, as part of TML security layer functions. It is the responsibility of the operator to select an appropriate security service and configure security policies accordingly. The details of such configuration are outside the scope of the ForCES PL and are dependent on the type of transport protocol and the nature of the connection.

All these configurations should be done prior to starting the CE and FE.

When certificates-based authentication is being used at the TML, the certificate can use a ForCES-specific naming structure as certificate names and, accordingly, the security policies can be configured at the CE and FE.

The reader is asked to refer to specific TML documents for details on the security requirements specific to that TML.

9.2.1. Endpoint Authentication Service

When TML security services are enabled, the ForCES TML performs endpoint authentication. Security association is established between CE and FE and is transparent to the ForCES PL.

9.2.2. Message Authentication Service

This is a TML-specific operation and is transparent to the ForCES PL. For details, refer to Section 5.

9.2.3. Confidentiality Service

This is a TML-specific operation and is transparent to the ForCES PL. For details, refer to Section 5.

10. Acknowledgments

The authors of this document would like to acknowledge and thank the ForCES Working Group and especially the following: Furquan Ansari, Alex Audu, Steven Blake, Shuchi Chawla, Alan DeKok, Ellen M. Deleganes, Xiaoyi Guo, Yunfei Guo, Evangelos Haleplidis, Zsolt Haraszti, Fenggen Jia, John C. Lin, Alistair Munro, Jeff Pickering, T. Sridhlar, Guangming Wang, Chaoping Wu, and Lily L. Yang, for their contributions. We would also like to thank David Putzolu and Patrick Droz for their comments and suggestions on the protocol and for their infinite patience. We would also like to thank Sue Hares and Alia Atlas for extensive reviews of the document.

Alia Atlas did a wonderful job of shaping the document to make it more readable by providing the IESG feedback.

Ross Callon was instrumental in getting us over major humps to getting this document published.

The editors have used the xml2rfc [RFC2629] tools in creating this document and are very grateful for the existence and quality of these tools. The editor is also grateful to Elwyn Davies for his help in correcting the XML source of this document.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5390] Rosenberg, J., "Requirements for Management of Overload in the Session Initiation Protocol", RFC 5390, December 2008.

[RFC5811] Hadi Salim, J. and K. Ogawa, "SCTP-Based Transport Mapping Layer (TML) for the Forwarding and Control Element Separation (ForCES) Protocol", RFC 5811, March 2010.

[RFC5812] Halpern, J. and J. Hadi Salim, "Forwarding and Control Element Separation (ForCES) Forwarding Element Model", RFC 5812, March 2010.

11.2. Informative References

[2PCREF] Gray, J., "Notes on database operating systems", in "Operating Systems: An Advanced Course" Lecture Notes in Computer Science, Vol. 60, pp. 394-481, Springer-Verlag, 1978.

[ACID] Haerder, T. and A. Reuter, "Principles of Transaction-Orientated Database Recovery", 1983.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC3654] Khosravi, H. and T. Anderson, "Requirements for Separation of IP Control and Forwarding", RFC 3654, November 2003.

[RFC3746] Yang, L., Dantu, R., Anderson, T., and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework", RFC 3746, April 2004.

Appendix A. IANA Considerations

Following the policies outlined in "Guidelines for Writing an IANA Considerations Section in RFCs" (RFC 5226 [RFC5226]), the following namespaces are defined in ForCES.

- Message Type Namespace, Section 7
- Operation Type Namespace, Section 7.1.6
- Header Flags, Section 6.1
- TLV Type, Section 7
- TLV Result Values, Section 7.1.7
- LFB Class ID, Section 7.1.5 (resolved by model document, [RFC5812].
- Result: Association Setup Response, Section 7.5.2
- Reason: Association Teardown Message, Section 7.5.3

A.1. Message Type Namespace

The Message Type is an 8-bit value. The following is the guideline for defining the Message Type namespace:

Message Types 0x00 - 0x1F
Message Types in this range are part of the base ForCES protocol. Message Types in this range are allocated through an IETF consensus action [RFC5226].

Values assigned by this specification:

0x00	Reserved
0x01	AssociationSetup
0x02	AssociationTeardown
0x03	Config
0x04	Query
0x05	EventNotification
0x06	PacketRedirect
0x07 - 0x0E	Reserved
0x0F	Hearbeat
0x11	AssociationSetupResponse
0x12	Reserved
0x13	ConfigResponse
0x14	QueryResponse

Message Types 0x20 - 0x7F
Message Types in this range are Specification Required [RFC5226]. Message Types using this range MUST be documented in an RFC or other permanent and readily available reference.

Message Types 0x80 - 0xFF
Message Types in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the Message Type namespace is unnecessary.

A.2. Operation Selection

The Operation Selection (OPER-TLV) namespace is 16 bits long. The following is the guideline for managing the OPER-TLV namespace.

OPER-TLV Type 0x0000-0x0FF
OPER-TLV Types in this range are allocated through an IETF consensus process [RFC5226].

Values assigned by this specification:

0x0000	Reserved
0x0001	SET
0x0002	SET-PROP
0x0003	SET-RESPONSE
0x0004	SET-PROP-RESPONSE
0x0005	DEL
0x0006	DEL-RESPONSE
0x0007	GET
0x0008	GET-PROP
0x0009	GET-RESPONSE
0x000A	GET-PROP-RESPONSE
0x000B	REPORT
0x000C	COMMIT
0x000D	COMMIT-RESPONSE
0x000E	TRCOMP

OPER-TLV Type 0x0100-0x7FFF
OPER-TLV Types using this range MUST be documented in an RFC or other permanent and readily available reference [RFC5226].

OPER-TLV Type 0x8000-0xFFFF
OPER-TLV Types in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the OPER-TLV Type namespace is unnecessary.

A.3. Header Flags

The Header flag field is 32 bits long. Header flags are part of the ForCES base protocol. Header flags are allocated through an IETF consensus action [RFC5226].

A.4. TLV Type Namespace

The TLV Type namespace is 16 bits long. The following is the guideline for managing the TLV Type namespace.

TLV Type 0x0000-0x01FF
TLV Types in this range are allocated through an IETF consensus process [RFC5226].

Values assigned by this specification:

0x0000	Reserved
0x0001	REDIRECT-TLV
0x0010	ASResult-TLV
0x0011	ASTreason-TLV
0x1000	LFBselect-TLV
0x0110	PATH-DATA-TLV
0x0111	KEYINFO-TLV
0x0112	FULLDATA-TLV
0x0113	SPARSEDATA-TLV
0x0114	RESULT-TLV
0x0115	METADATA-TLV
0x0116	REDIRECTDATA-TLV

TLV Type 0x0200-0x7FFF
TLV Types using this range MUST be documented in an RFC or other permanent and readily available reference [RFC5226].

TLV Type 0x8000-0xFFFF
TLV Types in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the TLV Type namespace is unnecessary.

A.5. RESULT-TLV Result Values

The RESULT-TLV RTesult Value is an 8-bit value.

0x00	E_SUCCESS
0x01	E_INVALID_HEADER
0x02	E_LENGTH_MISMATCH
0x03	E_VERSION_MISMATCH
0x04	E_INVALID_DESTINATION_PID
0x05	E_LFB_UNKNOWN
0x06	E_LFB_NOT_FOUND
0x07	E_LFB_INSTANCE_ID_NOT_FOUND
0x08	E_INVALID_PATH
0x09	E_COMPONENT_DOES_NOT_EXIST
0x0A	E_EXISTS
0x0B	E_NOT_FOUND
0x0C	E_READ_ONLY
0x0D	E_INVALID_ARRAY_CREATION
0x0E	E_VALUE_OUT_OF_RANGE
0x0F	E_CONTENTS_TOO_LONG
0x10	E_INVALID_PARAMETERS
0x11	E_INVALID_MESSAGE_TYPE
0x12	E_E_INVALID_FLAGS
0x13	E_INVALID_TLV
0x14	E_EVENT_ERROR
0x15	E_NOT_SUPPORTED
0x16	E_MEMORY_ERROR
0x17	E_INTERNAL_ERROR
0x18-0xFE	Reserved
0xFF	E_UNSPECIFIED_ERROR

All values not assigned in this specification are designated as Assignment by Expert Review.

A.6. Association Setup Response

The Association Setup Response namespace is 32 bits long. The following is the guideline for managing the Association Setup Response namespace.

Association Setup Response 0x0000-0x00FF
: Association Setup Responses in this range are allocated through an IETF consensus process [RFC5226].

Values assigned by this specification:

0x0000	Success
0x0001	FE ID Invalid
0x0002	Permission Denied

Association Setup Response 0x0100-0x0FFF
Association Setup Responses in this range are Specification Required [RFC5226]. Values using this range MUST be documented in an RFC or other permanent and readily available reference [RFC5226].

Association Setup Response 0x1000-0xFFFF
Association Setup Responses in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the Association Setup Response namespace is unnecessary.

A.7. Association Teardown Message

The Association Teardown Message namespace is 32 bits long. The following is the guideline for managing the Association Teardown Message namespace.

Association Teardown Message 0x00000000-0x0000FFFF
Association Teardown Messages in this range are allocated through an IETF consensus process [RFC5226].

Values assigned by this specification:

0x00000000	Normal - teardown by administrator
0x00000001	Error - loss of heartbeats
0x00000002	Error - loss of bandwidth
0x00000003	Error - out of Memory
0x00000004	Error - application crash
0x000000FF	Error - unspecified

Association Teardown Message 0x00010000-0x7FFFFFFF
Association Teardown Messages in this range are Specification Required [RFC5226]. Association Teardown Messages using this range MUST be documented in an RFC or other permanent and readily available references. [RFC5226].

Association Teardown Message 0x80000000-0xFFFFFFFF
Association Teardown Messages in this range are reserved for vendor private extensions and are the responsibility of individual

vendors. IANA management of this range of the Association Teardown Message namespace is unnecessary.

Appendix B. ForCES Protocol LFB Schema

The schema described below conforms to the LFB schema described in the ForCES model [RFC5812].

Section 7.3.1 describes the details of the different components defined in this definition.

```
<LFBLibrary xmlns="urn:ietf:params:xml:ns:forces:lfbmodel:1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    provides="FEPO">
<!-- XXX  -->
  <dataTypeDefs>
     <dataTypeDef>
        <name>CEHBPolicyValues</name>
               <synopsis>
                   The possible values of CE heartbeat policy
               </synopsis>
           <atomic>
           <baseType>uchar</baseType>
           <specialValues>
              <specialValue value="0">
                <name>CEHBPolicy0</name>
                <synopsis>
                     The CE heartbeat policy 0
                </synopsis>
                </specialValue>
              <specialValue value="1">
                 <name>CEHBPolicy1</name>
                 <synopsis>
                      The CE heartbeat policy 1
                 </synopsis>
              </specialValue>
            </specialValues>
            </atomic>
      </dataTypeDef>

      <dataTypeDef>
         <name>FEHBPolicyValues</name>
              <synopsis>
                  The possible values of FE heartbeat policy
             </synopsis>
           <atomic>
           <baseType>uchar</baseType>
           <specialValues>
```

```
              <specialValue value="0">
                <name>FEHBPolicy0</name>
                <synopsis>
                     The FE heartbeat policy 0
                </synopsis>
              </specialValue>
              <specialValue value="1">
                 <name>FEHBPolicy1</name>
                 <synopsis>
                      The FE heartbeat policy 1
                 </synopsis>
                </specialValue>
            </specialValues>
            </atomic>
        </dataTypeDef>

        <dataTypeDef>
        <name>FERestartPolicyValues</name>
            <synopsis>
                The possible values of FE restart policy
            </synopsis>
           <atomic>
           <baseType>uchar</baseType>
           <specialValues>
              <specialValue value="0">
                <name>FERestartPolicy0</name>
                <synopsis>
                     The FE restart policy 0
                </synopsis>
                </specialValue>
            </specialValues>
            </atomic>
        </dataTypeDef>

        <dataTypeDef>
        <name>CEFailoverPolicyValues</name>
            <synopsis>
                The possible values of CE failover policy
            </synopsis>
           <atomic>
           <baseType>uchar</baseType>
           <specialValues>
             <specialValue value="0">
                <name>CEFailoverPolicy0</name>
                <synopsis>
                     The CE failover policy 0
                </synopsis>
              </specialValue>
```

```
              <specialValue value="1">
                 <name>CEFailoverPolicy1</name>
                 <synopsis>
                      The CE failover policy 1
                 </synopsis>
               </specialValue>
              </specialValues>
              </atomic>
       </dataTypeDef>

      <dataTypeDef>
         <name>FEHACapab</name>
                <synopsis>
                    The supported HA features
                </synopsis>
             <atomic>
             <baseType>uchar</baseType>
             <specialValues>
                <specialValue value="0">
                  <name>GracefullRestart</name>
                  <synopsis>
                       The FE supports Graceful Restart
                  </synopsis>
                  </specialValue>
                <specialValue value="1">
                   <name>HA</name>
                   <synopsis>
                        The FE supports HA
                   </synopsis>
                </specialValue>
              </specialValues>
              </atomic>
      </dataTypeDef>
   </dataTypeDefs>

   <LFBClassDefs>
     <LFBClassDef LFBClassID="2">
       <name>FEPO</name>
       <synopsis>
          The FE Protocol Object
       </synopsis>
       <version>1.0</version>

   <components>
         <component componentID="1" access="read-only">
             <name>CurrentRunningVersion</name>
             <synopsis>Currently running ForCES version</synopsis>
             <typeRef>uchar</typeRef>
```

```
  </component>
<component componentID="2" access="read-only">
  <name>FEID</name>
  <synopsis>Unicast FEID</synopsis>
  <typeRef>uint32</typeRef>
</component>
<component componentID="3" access="read-write">
   <name>MulticastFEIDs</name>
   <synopsis>
      the table of all multicast IDs
   </synopsis>
   <array type="variable-size">
    <typeRef>uint32</typeRef>
   </array>
</component>
<component componentID="4" access="read-write">
  <name>CEHBPolicy</name>
  <synopsis>
   The CE Heartbeat Policy
  </synopsis>
  <typeRef>CEHBPolicyValues</typeRef>
</component>
<component componentID="5" access="read-write">
  <name>CEHDI</name>
  <synopsis>
    The CE Heartbeat Dead Interval in millisecs
  </synopsis>
  <typeRef>uint32</typeRef>
</component>
<component componentID="6" access="read-write">
  <name>FEHBPolicy</name>
  <synopsis>
    The FE Heartbeat Policy
  </synopsis>
  <typeRef>FEHBPolicyValues</typeRef>
</component>
<component componentID="7" access="read-write">
  <name>FEHI</name>
  <synopsis>
    The FE Heartbeat Interval in millisecs
  </synopsis>
  <typeRef>uint32</typeRef>
</component>
<component componentID="8" access="read-write">
  <name>CEID</name>
  <synopsis>
     The Primary CE this FE is associated with
  </synopsis>
```

```
            <typeRef>uint32</typeRef>
          </component>

          <component componentID="9" access="read-write">
             <name>BackupCEs</name>
             <synopsis>
                The table of all backup CEs other than the primary
             </synopsis>
             <array type="variable-size">
              <typeRef>uint32</typeRef>
             </array>
          </component>
          <component componentID="10" access="read-write">
            <name>CEFailoverPolicy</name>
            <synopsis>
              The CE Failover Policy
            </synopsis>
            <typeRef>CEFailoverPolicyValues</typeRef>
          </component>

          <component componentID="11" access="read-write">
            <name>CEFTI</name>
            <synopsis>
              The CE Failover Timeout Interval in millisecs
            </synopsis>
            <typeRef>uint32</typeRef>
          </component>
          <component componentID="12" access="read-write">
            <name>FERestartPolicy</name>
            <synopsis>
               The FE Restart Policy
            </synopsis>
            <typeRef>FERestartPolicyValues</typeRef>
          </component>
          <component componentID="13" access="read-write">
            <name>LastCEID</name>
            <synopsis>
               The Primary CE this FE was last associated with
            </synopsis>
            <typeRef>uint32</typeRef>
          </component>
        </components>

       <capabilities>
            <capability componentID="30">
               <name>SupportableVersions</name>
               <synopsis>
                  the table of ForCES versions that FE supports
```

```
               </synopsis>
               <array type="variable-size">
                <typeRef>uchar</typeRef>
               </array>
            </capability>
          <capability componentID="31">
             <name>HACapabilities</name>
             <synopsis>
                the table of HA capabilities the FE supports
             </synopsis>
             <array type="variable-size">
              <typeRef>FEHACapab</typeRef>
             </array>
          </capability>
        </capabilities>

        <events baseID="61">
          <event eventID="1">
            <name>PrimaryCEDown</name>
            <synopsis>
                The pimary CE has changed
            </synopsis>
            <eventTarget>
                <eventField>LastCEID</eventField>
            </eventTarget>
            <eventChanged/>
            <eventReports>
               <eventReport>
                 <eventField>LastCEID</eventField>
               </eventReport>
            </eventReports>
          </event>
        </events>

      </LFBClassDef>
    </LFBClassDefs>
  </LFBLibrary>
```

B.1. Capabilities

Supportable Versions enumerates all ForCES versions that an FE supports.

FEHACapab enumerates the HA capabilities of the FE. If the FE is not capable of graceful restarts or HA, then it will not be able to participate in HA as described in Section 8.1.

B.2. Components

All components are explained in Section 7.3.1.

Appendix C. Data Encoding Examples

In this section a few examples of data encoding are discussed. These example, however, do not show any padding.

==========
Example 1:
==========

Structure with three fixed-lengthof, mandatory fields.

```
struct S {
uint16 a
uint16 b
uint16 c
}
```

(a) Describing all fields using SPARSEDATA-TLV

```
PATH-DATA-TLV
  Path to an instance of S ...
  SPARSEDATA-TLV
    ComponentIDof(a), lengthof(a), valueof(a)
    ComponentIDof(b), lengthof(b), valueof(b)
    ComponentIDof(c), lengthof(c), valueof(c)
```

(b) Describing a subset of fields

```
PATH-DATA-TLV
  Path to an instance of S ...
  SPARSEDATA-TLV
    ComponentIDof(a), lengthof(a), valueof(a)
    ComponentIDof(c), lengthof(c), valueof(c)
```

Note: Even though there are non-optional components in structure S, since one can uniquely identify components, one can selectively send components of structure S (e.g., in the case of an update from CE to FE).

(c) Describing all fields using a FULLDATA-TLV

```
PATH-DATA-TLV
  Path to an instance of S ...
  FULLDATA-TLV
    valueof(a)
    valueof(b)
    valueof(c)
```

```
==========
Example 2:
==========

Structure with three fixed-lengthof fields, one mandatory, two
optional.

        struct T {
        uint16 a
        uint16 b (optional)
        uint16 c (optional)
        }

This example is identical to example 1, as illustrated below.

(a) Describing all fields using SPARSEDATA-TLV

        PATH-DATA-TLV
          Path to an instance of S ...
          SPARSEDATA-TLV
            ComponentIDof(a), lengthof(a), valueof(a)
            ComponentIDof(b), lengthof(b), valueof(b)
            ComponentIDof(c), lengthof(c), valueof(c)

(b) Describing a subset of fields using SPARSEDATA-TLV

        PATH-DATA-TLV
          Path to an instance of S ...
          SPARSEDATA-TLV
            ComponentIDof(a), lengthof(a), valueof(a)
            ComponentIDof(c), lengthof(c), valueof(c)

(c) Describing all fields using a FULLDATA-TLV

        PATH-DATA-TLV
          Path to an instance of S ...
          FULLDATA-TLV
            valueof(a)
            valueof(b)
            valueof(c)

Note: FULLDATA-TLV _cannot_ be used unless all fields are being
described.
```

```
==========
Example 3:
==========

Structure with a mix of fixed-lengthof and variable-lengthof fields,
some mandatory, some optional.  Note in this case, b is variable
sized.

        struct U {
        uint16 a
        string b (optional)
        uint16 c (optional)
        }

(a) Describing all fields using SPARSEDATA-TLV

        Path to an instance of U ...
        SPARSEDATA-TLV
          ComponentIDof(a), lengthof(a), valueof(a)
          ComponentIDof(b), lengthof(b), valueof(b)
          ComponentIDof(c), lengthof(c), valueof(c)

(b) Describing a subset of fields using SPARSEDATA-TLV

        Path to an instance of U ...
        SPARSEDATA-TLV
          ComponentIDof(a), lengthof(a), valueof(a)
          ComponentIDof(c), lengthof(c), valueof(c)

(c) Describing all fields using FULLDATA-TLV

        Path to an instance of U ...
          FULLDATA-TLV
            valueof(a)
            FULLDATA-TLV
              valueof(b)
            valueof(c)

Note: The variable-length field requires the addition of a FULLDATA-
TLV within the outer FULLDATA-TLV as in the case of component b
above.
```

```
==========
Example 4:
==========

Structure containing an array of another structure type.

        struct V {
        uint32 x
        uint32 y
        struct U z[]
        }
```

(a) Encoding using SPARSEDATA-TLV, with two instances of z[], also described with SPARSEDATA-TLV, assuming only the 10th and 15th subscripts of z[] are encoded.

```
     path to instance of V ...
     SPARSEDATA-TLV
      ComponentIDof(x), lengthof(x), valueof(x)
      ComponentIDof(y), lengthof(y), valueof(y)
      ComponentIDof(z), lengthof(all below)
        ComponentID = 10 (i.e index 10 from z[]), lengthof(all below)
            ComponentIDof(a), lengthof(a), valueof(a)
            ComponentIDof(b), lengthof(b), valueof(b)
        ComponentID = 15 (index 15 from z[]), lengthof(all below)
            ComponentIDof(a), lengthof(a), valueof(a)
            ComponentIDof(c), lengthof(c), valueof(c)
```

Note the holes in the components of z (10 followed by 15). Also note the gap in index 15 with only components a and c appearing but not b.

Appendix D. Use Cases

Assume LFB with the following components for the following use cases.

```
foo1, type u32, ID = 1

foo2, type u32, ID = 2

table1: type array, ID = 3
        components are:
        t1, type u32, ID = 1
        t2, type u32, ID = 2  // index into table2
        KEY: nhkey, ID = 1, V = t2

table2: type array, ID = 4
        components are:
        j1, type u32, ID = 1
        j2, type u32, ID = 2
        KEY: akey, ID = 1, V = { j1,j2 }

table3: type array, ID = 5
        components are:
        someid, type u32, ID = 1
        name, type string variable sized, ID = 2

table4: type array, ID = 6
        components are:
        j1, type u32, ID = 1
        j2, type u32, ID = 2
        j3, type u32, ID = 3
        j4, type u32, ID = 4
        KEY: mykey, ID = 1, V = { j1}

table5: type array, ID = 7
        components are:
        p1, type u32, ID = 1
        p2, type array, ID = 2, array components of type-X

Type-X:
        x1, ID 1, type u32
        x2, ID2 , type u32
                KEY: tkey, ID = 1, V = { x1}
```

All examples will use valueof(x) to indicate the value of the referenced component x. In the case where F_SEL** are missing (bits equal to 00) then the flags will not show any selection.

All the examples only show use of FULLDATA-TLV for data encoding; although SPARSEDATA-TLV would make more sense in certain occasions, the emphasis is on showing the message layout. Refer to Appendix C for examples that show usage of both FULLDATA-TLV and SPARSEDATA-TLV.

1. To get foo1

```
OPER = GET-TLV
        PATH-DATA-TLV: IDCount = 1, IDs = 1
Result:
OPER = GET-RESPONSE-TLV
        PATH-DATA-TLV:
                flags=0, IDCount = 1, IDs = 1
                FULLDATA-TLV L = 4+4, V =  valueof(foo1)
```

2. To set foo2 to 10

```
OPER = SET-TLV
        PATH-DATA-TLV:
                flags = 0,  IDCount = 1, IDs = 2
                FULLDATA-TLV: L = 4+4, V=10

Result:
OPER = SET-RESPONSE-TLV
        PATH-DATA-TLV:
                flags = 0,  IDCount = 1, IDs = 2
                RESULT-TLV
```

3. To dump table2

```
   OPER = GET-TLV
        PATH-DATA-TLV:
                IDCount = 1, IDs = 4
   Result:
   OPER = GET-RESPONSE-TLV
        PATH-DATA-TLV:
                flags = 0, IDCount = 1, IDs = 4
                FULLDATA-TLV: L = XXX, V=
                     a series of: index, valueof(j1), valueof(j2)
                     representing the entire table
```

Note: One should be able to take a GET-RESPONSE-TLV and convert it to a SET-TLV. If the result in the above example is sent back in a SET-TLV (instead of a GET-RESPONSE_TLV), then the entire contents of the table will be replaced at that point.

4. Multiple operations example. To create entry 0-5 of table2 (Error conditions are ignored)

```
OPER = SET-TLV
        PATH-DATA-TLV:
                flags = 0 , IDCount = 1, IDs = 4
                PATH-DATA-TLV
                  flags = 0, IDCount = 1, IDs = 0
                  FULLDATA-TLV valueof(j1), valueof(j2) of entry 0
                PATH-DATA-TLV
                  flags = 0, IDCount = 1, IDs = 1
                  FULLDATA-TLV valueof(j1), valueof(j2) of entry 1
                PATH-DATA-TLV
                  flags = 0, IDCount = 1, IDs = 2
                  FULLDATA-TLV valueof(j1), valueof(j2) of entry 2
                PATH-DATA-TLV
                  flags = 0, IDCount = 1, IDs = 3
                  FULLDATA-TLV valueof(j1), valueof(j2) of entry 3
                PATH-DATA-TLV
                  flags = 0, IDCount = 1, IDs = 4
                  FULLDATA-TLV valueof(j1), valueof(j2) of entry 4
                PATH-DATA-TLV
                  flags = 0, IDCount = 1, IDs = 5
                  FULLDATA-TLV valueof(j1), valueof(j2) of entry 5

Result:
OPER = SET-RESPONSE-TLV
        PATH-DATA-TLV:
                flags = 0 , IDCount = 1, IDs = 4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 0
                    RESULT-TLV
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    RESULT-TLV
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    RESULT-TLV
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    RESULT-TLV
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                    RESULT-TLV
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5
                    RESULT-TLV
```

5. Block operations (with holes) example. Replace entry 0,2 of table2.

```
OPER = SET-TLV
        PATH-DATA-TLV:
             flags =  0 , IDCount = 1, IDs = 4
             PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 0
                FULLDATA-TLV containing valueof(j1), valueof(j2) of 0
             PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                FULLDATA-TLV containing valueof(j1), valueof(j2) of 2

Result:
OPER = SET-TLV
        PATH-DATA-TLV:
             flags =  0 , IDCount = 1, IDs = 4
             PATH-DATA-TLV
                 flags = 0, IDCount = 1, IDs = 0
                 RESULT-TLV
             PATH-DATA-TLV
                 flags = 0, IDCount = 1, IDs = 2
                 RESULT-TLV
```

6. Getting rows example. Get first entry of table2.

```
OPER = GET-TLV
        PATH-DATA-TLV:
                IDCount = 2, IDs = 4.0

Result:
OPER = GET-RESPONSE-TLV
        PATH-DATA-TLV:
                IDCount = 2, IDs = 4.0
                 FULLDATA-TLV containing valueof(j1), valueof(j2)
```

7. Get entry 0-5 of table2.

```
OPER = GET-TLV
        PATH-DATA-TLV:
                flags = 0, IDCount = 1, IDs = 4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 0
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5

Result:
OPER = GET-RESPONSE-TLV
        PATH-DATA-TLV:
                flags = 0, IDCount = 1, IDs = 4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 0
                    FULLDATA-TLV containing valueof(j1), valueof(j2)
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    FULLDATA-TLV containing valueof(j1), valueof(j2)
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    FULLDATA-TLV containing valueof(j1), valueof(j2)
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    FULLDATA-TLV containing valueof(j1), valueof(j2)
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                    FULLDATA-TLV containing valueof(j1), valueof(j2)
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5
                    FULLDATA-TLV containing valueof(j1), valueof(j2)
```

```
   8.   Create a row in table2, index 5.

   OPER = SET-TLV
           PATH-DATA-TLV:
                flags = 0, IDCount = 2, IDs = 4.5
                FULLDATA-TLV containing valueof(j1), valueof(j2)

   Result:
   OPER = SET-RESPONSE-TLV
           PATH-DATA-TLV:
                flags = 0, IDCount = 1, IDs = 4.5
                RESULT-TLV

   9.   Dump contents of table1.

   OPER = GET-TLV
           PATH-DATA-TLV:
                   flags = 0, IDCount = 1, IDs = 3

   Result:
   OPER = GET-RESPONSE-TLV
           PATH-DATA-TLV
                   flags = 0, IDCount = 1, IDs = 3
                   FULLDATA-TLV, Length = XXXX
                           (depending on size of table1)
                           index, valueof(t1),valueof(t2)
                           index, valueof(t1),valueof(t2)
                           .
                           .
                           .
```

10. Using keys. Get row entry from table4 where j1=100. Recall, j1 is a defined key for this table and its KeyID is 1.

```
OPER = GET-TLV
        PATH-DATA-TLV:
                flags = F_SELKEY  IDCount = 1, IDs = 6
                KEYINFO-TLV = KeyID=1, KEY_DATA=100

Result:
If j1=100 was at index 10
OPER = GET-RESPONSE-TLV
        PATH-DATA-TLV:
                flags = 0, IDCount = 1, IDs = 6.10
                FULLDATA-TLV containing
                  valueof(j1), valueof(j2),valueof(j3),valueof(j4)
```

11. Delete row with KEY match (j1=100, j2=200) in table2. Note that the j1,j2 pair is a defined key for the table2.

```
OPER = DEL-TLV
        PATH-DATA-TLV:
                flags = F_SELKEY  IDCount = 1, IDs = 4
                KEYINFO-TLV:  {KeyID =1 KEY_DATA=100,200}

Result:
If (j1=100, j2=200) was at entry 15:
OPER = DELETE-RESPONSE-TLV
        PATH-DATA-TLV:
                flags = 0  IDCount = 2, IDs = 4.15
                RESULT-TLV
```

12. Dump contents of table3. It should be noted that this table has a column with a component name that is variable sized. The purpose of this use case is to show how such a component is to be encoded.

```
OPER = GET-TLV
        PATH-DATA-TLV:
             flags = 0 IDCount = 1, IDs = 5

Result:
OPER = GET-RESPONSE-TLV
    PATH-DATA-TLV:
       flags = 0  IDCount = 1, IDs = 5
           FULLDATA-TLV, Length = XXXX
            index, someidv, TLV: T=FULLDATA-TLV, L = 4+strlen(namev),
                   V = valueof(v)
            index, someidv, TLV: T=FULLDATA-TLV, L = 4+strlen(namev),
                   V = valueof(v)
            index, someidv, TLV: T=FULLDATA-TLV, L = 4+strlen(namev),
                   V = valueof(v)
            index, someidv, TLV: T=FULLDATA-TLV, L = 4+strlen(namev),
                   V = valueof(v)
               .
               .
               .
```

13. Multiple atomic operations.

Note 1: This emulates adding a new nexthop entry and then atomically updating the L3 entries pointing to an old NH to point to a new one. The assumption is that both tables are in the same LFB.

Note: Observe the two operations on the LFB instance; both are SET operations.

```
//Operation 1: Add a new entry to table2 index #20.
OPER = SET-TLV
        Path-TLV:
                flags = 0, IDCount = 2,  IDs = 4.20
                FULLDATA-TLV, V= valueof(j1),valueof(j2)

// Operation 2: Update table1 entry which
// was pointing with t2 = 10 to now point to 20
OPER = SET-TLV
        PATH-DATA-TLV:
                flags = F_SELKEY, IDCount = 1, IDs = 3
                KEYINFO-TLV = KeyID=1 KEY_DATA=10
                PATH-DATA-TLV
                        flags = 0  IDCount = 1, IDs = 2
                        FULLDATA-TLV, V= 20

Result:
//first operation, SET
OPER = SET-RESPONSE-TLV
        PATH-DATA-TLV
                flags = 0 IDCount = 3, IDs = 4.20
                RESULT-TLV code = success
                        FULLDATA-TLV, V = valueof(j1),valueof(j2)
// second operation SET - assuming entry 16 was updated
OPER = SET-RESPONSE-TLV
        PATH-DATA-TLV
                flags = 0 IDCount = 2, IDs = 3.16
                PATH-DATA-TLV
                        flags = 0  IDCount = 1, IDs = 2
                        RESULT-TLV code = success
                                FULLDATA-TLV, Length = XXXX v=20
```

14. Selective setting. On table4 -- for indices 1, 3, 5, 7, and 9. Replace j1 to 100, j2 to 200, j3 to 300. Leave j4 as is.

```
PER = SET-TLV
    PATH-DATA-TLV
        flags = 0, IDCount = 1, IDs = 6
        PATH-DATA-TLV
            flags = 0, IDCount = 1, IDs = 1
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 1
                FULLDATA-TLV, Length = XXXX, V = {100}
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                FULLDATA-TLV, Length = XXXX, V = {200}
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                FULLDATA-TLV, Length = XXXX, V = {300}
        PATH-DATA-TLV
            flags = 0, IDCount = 1, IDs = 3
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 1
                FULLDATA-TLV, Length = XXXX, V = {100}
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                FULLDATA-TLV, Length = XXXX, V = {200}
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                FULLDATA-TLV, Length = XXXX, V = {300}
```

```
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 5
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    FULLDATA-TLV, Length = XXXX, V = {100}
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    FULLDATA-TLV, Length = XXXX, V = {200}
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    FULLDATA-TLV, Length = XXXX, V = {300}
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 7
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    FULLDATA-TLV, Length = XXXX, V = {100}
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    FULLDATA-TLV, Length = XXXX, V = {200}
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    FULLDATA-TLV, Length = XXXX, V = {300}
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 9
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    FULLDATA-TLV, Length = XXXX, V = {100}
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    FULLDATA-TLV, Length = XXXX, V = {200}
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    FULLDATA-TLV, Length = XXXX, V = {300}

   response:

   OPER = SET-RESPONSE-TLV
       PATH-DATA-TLV
           flags = 0, IDCount = 1, IDs = 6
           PATH-DATA-TLV
               flags = 0, IDCount = 1, IDs = 1
               PATH-DATA-TLV
                   flags = 0, IDCount = 1, IDs = 1
                   RESULT-TLV
               PATH-DATA-TLV
                   flags = 0, IDCount = 1, IDs = 2
                   RESULT-TLV
```

```
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                RESULT-TLV
        PATH-DATA-TLV
            flags = 0, IDCount = 1, IDs = 3
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 1
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                RESULT-TLV
        PATH-DATA-TLV
            flags = 0, IDCount = 1, IDs = 5
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 1
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                RESULT-TLV
        PATH-DATA-TLV
            flags = 0, IDCount = 1, IDs = 7
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 1
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                RESULT-TLV
        PATH-DATA-TLV
            flags = 0, IDCount = 1, IDs = 9
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 1
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 2
                RESULT-TLV
            PATH-DATA-TLV
                flags = 0, IDCount = 1, IDs = 3
                RESULT-TLV
```

15. Manipulation of table of table examples. Get x1 from table10 row with index 4, inside table5 entry 10.

```
operation = GET-TLV
        PATH-DATA-TLV
                flags = 0  IDCount = 5, IDs=7.10.2.4.1

Results:
operation = GET-RESPONSE-TLV
        PATH-DATA-TLV
                flags = 0  IDCount = 5, IDs=7.10.2.4.1
                FULLDATA-TLV: L=XXXX, V = valueof(x1)
```

16. From table5's row 10 table10, get X2s based on the value of x1 equaling 10 (recall x1 is KeyID 1).

```
operation = GET-TLV
        PATH-DATA-TLV
                flag = F_SELKEY, IDCount=3, IDS = 7.10.2
                KEYINFO-TLV, KeyID = 1, KEYDATA = 10
                PATH-DATA-TLV
                        IDCount = 1, IDS = 2 //select x2

Results:
If x1=10 was at entry 11:
operation = GET-RESPONSE-TLV
        PATH-DATA-TLV
                flag = 0, IDCount=5, IDS = 7.10.2.11
                PATH-DATA-TLV
                        flags = 0  IDCount = 1, IDS = 2
                        FULLDATA-TLV: L=XXXX, V = valueof(x2)
```

17. Further example of manipulating a table of tables

```
Consider table6, which is defined as:
table6: type array, ID = 8
        components are:
        p1, type u32, ID = 1
        p2, type array, ID = 2, array components of type type-A

type-A:
        a1, type u32, ID 1,
        a2, type array ID2 ,array components of type type-B

type-B:
        b1, type u32, ID 1
        b2, type u32, ID 2
```

```
   If for example one wanted to set by replacing:
   table6.10.p1 to 111
   table6.10.p2.20.a1 to 222
   table6.10.p2.20.a2.30.b1 to 333

   in one message and one operation.

   There are two ways to do this:
      a) using nesting
      b) using a flat path data

   A. Method using nesting
      in one message with a single operation

   operation = SET-TLV
           PATH-DATA-TLV
                   flags = 0  IDCount = 2, IDs=6.10
                   PATH-DATA-TLV
                           flags = 0, IDCount = 1, IDs=1
                           FULLDATA-TLV: L=XXXX,
                                   V = {111}
                   PATH-DATA-TLV
                           flags = 0  IDCount = 2, IDs=2.20
                           PATH-DATA-TLV
                                   flags = 0, IDCount = 1, IDs=1
                                   FULLDATA-TLV: L=XXXX,
                                           V = {222}
                           PATH-DATA-TLV :
                                   flags = 0, IDCount = 3, IDs=2.30.1
                                   FULLDATA-TLV: L=XXXX,
                                           V = {333}
```

```
   Result:
   operation = SET-RESPONSE-TLV
           PATH-DATA-TLV
                   flags = 0  IDCount = 2, IDs=6.10
                   PATH-DATA-TLV
                           flags = 0, IDCount = 1, IDs=1
                           RESULT-TLV
                   PATH-DATA-TLV
                           flags = 0  IDCount = 2, IDs=2.20
                           PATH-DATA-TLV
                                   flags = 0, IDCount = 1, IDs=1
                                   RESULT-TLV
                           PATH-DATA-TLV :
                                   flags = 0, IDCount = 3, IDs=2.30.1
                                   RESULT-TLV

   B. Method using a flat path data in
      one message with a single operation

   operation = SET-TLV
           PATH-DATA-TLV :
                   flags = 0, IDCount = 3, IDs=6.10.1
                   FULLDATA-TLV: L=XXXX,
                           V = {111}
           PATH-DATA-TLV :
                   flags = 0, IDCount = 5, IDs=6.10.1.20.1
                   FULLDATA-TLV: L=XXXX,
                           V = {222}
           PATH-DATA-TLV :
                   flags = 0, IDCount = 7, IDs=6.10.1.20.1.30.1
                   FULLDATA-TLV: L=XXXX,
                           V = {333}
   Result:
   operation = SET-TLV
           PATH-DATA-TLV :
                   flags = 0, IDCount = 3, IDs=6.10.1
                   RESULT-TLV
           PATH-DATA-TLV :
                   flags = 0, IDCount = 5, IDs=6.10.1.20.1
                   RESULT-TLV
           PATH-DATA-TLV :
                   flags = 0, IDCount = 7, IDs=6.10.1.20.1.30.1
                   RESULT-TLV
```

18. Get a whole LFB (all its components, etc.).

For example: At startup a CE might well want the entire FE Object LFB. So, in a request targeted at class 1, instance 1, one might find:

```
operation = GET-TLV
        PATH-DATA-TLV
                flags = 0  IDCount = 0

result:
operation = GET-RESPONSE-TLV
        PATH-DATA-TLV
                flags = 0  IDCount = 0
                FULLDATA-TLV encoding of the FE Object LFB
```

Authors' Addresses

Avri Doria (editor)
Lulea University of Technology
Rainbow Way
Lulea SE-971 87
Sweden

Phone: +46 73 277 1788
EMail: avri@ltu.se

Jamal Hadi Salim (editor)
Znyx
Ottawa, Ontario
Canada

Phone:
EMail: hadi@mojatatu.com

Robert Haas (editor)
IBM
Saumerstrasse 4
8803 Ruschlikon
Switzerland

Phone:
EMail: rha@zurich.ibm.com

Hormuzd M Khosravi (editor)
Intel
2111 NE 25th Avenue
Hillsboro, OR 97124
USA

Phone: +1 503 264 0334
EMail: hormuzd.m.khosravi@intel.com

Weiming Wang (editor)
Zhejiang Gongshang University
18, Xuezheng Str., Xiasha University Town
Hangzhou 310018
P.R. China

Phone: +86-571-28877721
EMail: wmwang@zjgsu.edu.cn

Ligang Dong
Zhejiang Gongshang University
18, Xuezheng Str., Xiasha University Town
Hangzhou 310018
P.R. China

Phone: +86-571-28877751
EMail: donglg@zjgsu.edu.cn

Ram Gopal
Nokia
5, Wayside Road
Burlington, MA 310035
USA

Phone: +1-781-993-3685
EMail: ram.gopal@nsn.com

Joel Halpern
P.O. Box 6049
Leesburg, VA 20178
USA

Phone: +1-703-371-3043
EMail: jmh@joelhalpern.com