Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 5, 2007

A. Doria (Ed.)
ETRI
R. Haas (Ed.)
IBM
J. Hadi Salim (Ed.)
Znyx
H. Khosravi (Ed.)
Intel
W. M. Wang (Ed.)
Zhejiang Gongshang University

March 2007

ForCES Protocol Specification
draft-ietf-forces-protocol-09.txt

Status of this Memo

Copyright Notice

Abstract

This document specifies the Forwarding and Control Element Separation (ForCES) protocol. 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 RFC3654. Besides the ForCES protocol messages, the specification also defines the framework, the mechanisms, and the Transport Mapping Layer (TML) requirements for ForCES protocol.

Authors

The participants in the ForCES Protocol Team, primary co-authors and co-editors, of this protocol specification, are:

Ligang Dong (Zhejiang Gongshang University), Avri Doria (Lulea University of Technology), Ram Gopal (Nokia), Robert Haas (IBM), Jamal Hadi Salim (Znyx), Hormuzd M Khosravi (Intel), and Weiming Wang (Zhejiang Gongshang University).

Table of Contents

1. Terminology and Conventions

The key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in BCP 14, RFC2119 [RFC2119].

2. 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 term "ForCES protocol" and "protocol" as used in this document refers to the protocol used to standardize the information exchange between Control Elements (CEs) and Forwarding Elements (FEs) only.

The ForCES FE model [FE-MODEL] presents a formal way to define FE Logical Function Blocks (LFBs) using XML. LFB configuration attributes, 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 Logical Function Block(LFB) configuration information, association setup, status, and event notifications, etc.

This specification does not define a transport mechanism for protocol messages. A discussion of service primitives that must be provided by the underlying transport interface will be discussed in a future document.

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, descriptions of the Protocol Layer (PL) and a Transport Mapping Layer (TML), as well as of 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.

3. Definitions

This document follows the terminology defined by the ForCES Requirements in [RFC3654] and by the ForCES framework in [RFC3746]. The definitions below 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 which can be reached using an IP address; and on a switch fabric, it is a device which 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.

Datapath - 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.

FE Manager (FEM) - A logical entity responsible for generic FE management tasks. It is used during 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 NAT-PT, 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 ForCES protocol. The LFB may reside at the FE's datapath 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 (Logical Function Block) 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 Metadata - Metadata 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 metadata is identified, produced and consumed by the LFBs. It defines the functionality but not how metadata is encoded within an implementation.

LFB Attribute - Operational parameters of the LFBs that must be visible to the CEs are conceptualized in the FE model as the LFB attributes. The LFB attributes 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 datapath 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 (see below) and a CE Manager (see below) 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 term "ForCES protocol" and "protocol" refer to the Fp reference point in the ForCES Framework in [RFC3746]. This protocol does not apply to CE-to-CE communication, FE-to-FE communication, or to 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 ForCES protocol architecture that defines the ForCES protocol messages, the protocol state transfer scheme, as well as the ForCES protocol architecture itself (including requirements of ForCES TML (see 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 a NE with two CEs and two FEs.

```
                         ---------------------------------------
                         | ForCES Network Element              |
  --------------   Fc    | --------------      --------------  |
  | CE Manager |---------+-|     CE 1   |------|    CE 2    |  |
  --------------         | |            |  Fr  |            |  |
        |                | --------------      --------------  |
        | Fl             |         |  |    Fp       /          |
        |                |       Fp|  |----------| /           |
        |                |         |             |/            |
        |                |         |             |             |
        |                |         |     Fp     /|----|        |
        |                |         |  /--------/      |        |
  --------------     Ff  | --------------      --------------  |
  | FE Manager |---------+-|     FE 1   |  Fi  |     FE 2   |  |
  --------------         | |            |------|            |  |
                         | --------------      --------------  |
                         |   |  |  |  |          |  |  |  |    |
                         ----+--+--+--+----------+--+--+--+-----
                             |  |  |  |          |  |  |  |
                             |  |  |  |          |  |  |  |
                               Fi/f                   Fi/f

       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 Point. 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 interface by example of an MPLS QoS enabled Network Element.

```
      -------------------------------------------------
      |       |       |       |       |       |       |
      |OSPF   |RIP    |BGP    |RSVP   |LDP    |. . .  |
      |       |       |       |       |       |       |
      -------------------------------------------------    CE
      |               ForCES Interface                |
      -------------------------------------------------
                              ^   ^
                              |   |
                      ForCES  |   |data
                      control |   |packets
                      messages|   |(e.g., routing packets)
                              |   |
                              v   v
      -------------------------------------------------
      |               ForCES Interface                |
      -------------------------------------------------    FE
      |       |       |       |       |       |       |
      |LPM Fwd|Meter  |Shaper |MPLS   |Classi-|. . .  |
      |       |       |       |       |fier   |       |
      -------------------------------------------------
```

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.

```
      +-------------------------------------------------
      |               CE PL                            |
      +-------------------------------------------------
      |              CE TML                            |
      +-------------------------------------------------
                                 ^
                                 |
                  ForCES         |   (i.e  ForCES data + control
                  PL             |    packets )
                  messages       |
                  over           |
                  specific       |
                  TML            |
                  encaps         |
                  and            |
                  transport      |
                                 |
                                 v
      +-------------------------------------------------
      |              FE TML                            |
      +-------------------------------------------------
      |               FE PL                            |
      +-------------------------------------------------
```

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 or CE (which would also be issued during the pre-association phase).

c. Security parameterization such as keys etc.

d. Connection association parameters

Example of connection association parameters this might be:

o simple parameters: send up to 3 association messages every 1
   second

o 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 association phase where the ForCES protocol operates to manipulate the parameters of the FEs.

```
         FE assoiated      CE configures transition to UP
        +---->--->---+    +--->---->---->---->------->----+
        |            |    |                               Y
        ^            Y    |                               |
        |            |    |                               Y
    +---+-------+  +------+--+                        +--------+
    |FE Pre-    |  | FE      |                        | FE     |
    |Association|  | DOWN    | CE configures DOWN     | UP     |
    |Phase      |  | State   +<------<-----<------<-- + State  |
    |           |  |         |                        |        |
    +-----------+  +---------+                        +--------+
          ^                                               Y
          |                                               |
          +-<---<------<-----<------<----<---------<------+
                             FE loses association
```

Figure 4: The FE State Machine

In the mandated case, once associated, the FE can only be in one of two states, as indicated above. When the FE is in the DOWN state, it is not forwarding packets. When the FE is in the UP state it may be forwarding packets, depending on the configuration of its specific LFBs. The FE MAY also be in other states when it is capable of graceful restart and high availaibility. The extra transitions are explained in Section 8 and not discussed here so as to allow us to explain the basics with more clarity.

The CE configures FE state transitions by means of the FE Object LFB, which is defined in [FE-MODEL] and also explained in Section 7.2.2. In the FE Object LFB, FE state is defined as an attribute of the LFB, and CE configuration of the FE state equals CE configuration of this attribute. Note that even in the FE DOWN state, the FE is associated.

The FE stays in the DOWN state until it is explicitly configured by the CE to transition to the UP state via an FE Object admin action. This must be done before configuring any other LFBs that affect packet forwarding. The typical setup will bring up the FE to the UP state on association.

The FE transitions from the UP state to the DOWN state when it receives an FEObject Admin Down action. when it 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 DOWN 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 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.

```
   <----Ff ref pt--->             <--Fc ref pt------->
   FE Manager      FE               CE Manager    CE
    |              |                 |             |
    |              |                 |             |
 (security exchange)              (security exchange)
   1|<------------>| authentication 1|<----------->|authentication
    |              |                 |             |
  (FE ID, attributes)             (CE ID, attributes)
   2|<-------------| request        2|<------------|request
    |              |                 |             |
   3|------------->| response       3|------------>|response
   (corresponding CE ID)         (corresponding FE ID)
    |              |                 |             |
    |              |                 |             |
```

Figure 5: Examples of a message exchange over the Ff and Fc reference points

```
<-----------Fl ref pt-------------->            |

FE Manager      FE                CE Manager     CE
 |              |                  |             |
 |              |                  |             |
(security exchange)                |             |
1|<------------------------------->|             |
 |              |                  |             |
(a list of CEs and their attributes)             |
2|<--------------------------------|             |
 |              |                  |             |
(a list of FEs and their attributes)             |
3|-------------------------------->|             |
 |              |                  |             |
 |              |                  |             |
```

Figure 6: An 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. Because the pre-association phase is out of scope, 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:

- o Association Setup Stage

- o Established Stage

o 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 attributes 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 state, 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 FE Protocol Object (FEPO) is defined (refer to Appendix B and Section 7.2.1). FEPO defines various timers which can be used in conjunction with 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 Section 8, Section 7.2.1, Section 4.3.3

and Appendix 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 (Execute 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 commitment 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 7.1.1.1.5) 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. execute-until-failure

c. continue-execute-on-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. I.e., 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. I.e., 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 execute 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.

The AT flag and the TP flag in Common Header (Section 6.1) are provided for 2PC-based transactional operations spanning multiple messages.

The COMMIT operation is specified to be used in the case of a final commit message.

The AT flag, when set, indicates this message belongs to an Atomic Transaction. All messages for a transaction operation must have the AT flag set. If not set, it means 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 this message belongs to. There are four (4) possible phases for an transactional operation known as:

SOT (Start of Transaction)

MOT (Middle of Transaction)

EOT (End of Transaction)

ABT (Abort)

A transaction operation is started with a message the TP flag is set to Start of Transaction (SOT). Multi-part messages, after the first one, are indicated by the Middle of Transaction flag (MOT). 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 only validates that the operation can be executed but does not execute it.

Any failure notified by the FE causes the CE to execute an Abort Transaction (ABT) to all FEs involved in the transaction, rolling back any previously executed operations in the transaction (There must be none if a commit has not been issued).

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. If no response is received from the FE within a specified timeout, the transaction MUST be aborted by the CE.

Note that a transactional operation is generically atomic, therefore it requires that the execute 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 execute 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.

Figure 9 shows an example of how a successful two phase commit between a CE and an FE would look like.

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, for sake of simplicity 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. The decision on what an FE should do after a lost association is dictated by the CE Failover policy (refer to Section 8 and Section 7.2).

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 100Gbps pipe is available and there is sufficient work then the PL should be able to take advantage of this and use all of the 100Gbps 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, amongst 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 which 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.

- o multiple PL PDUs can be aggregated under one TML message

- o multiple LFB classes and instances (as indicated in the LFB selector) can be addressed within one PL PDU

- o 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 (HB) 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 7.2.1 and Section 7.9 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, as well as 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:

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

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

These LFBs are detailed in Section 7.2.

4.4. Protocol Scenarios

This section provides a very high level description of sample message sequences between a CE and 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 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 will take care of establishing any security associations.

This is typically followed by capability query, topology query, etc. When the FE is ready to start forwarding data traffic, it sends an FE UP Event message to the CE. When the CE is ready, it responds by enabling the FE by setting the FEStatus to Adminup (Refer to [FE-MODEL] for details). This indicates to the FE to start forwarding data traffic. At this point the association establishment is complete. These sequences of messages are illustrated in the Figure 7 below.

```
           FE PL                   CE PL

             |                       |
             |   Asso Setup Req      |
             |---------------------->|
             |                       |
             |   Asso Setup Resp     |
             |<----------------------|
             |                       |
             | LFBx Query capability |
             |<----------------------|
             |                       |
             | LFBx Query Resp       |
             |---------------------->|
             |                       |
             | FEO Query (Topology)  |
             |<----------------------|
             |                       |
             | FEO Query Resp        |
             |---------------------->|
             |                       |
             | FEO UP Event          |
             |---------------------->|
             |                       |
             |  Config FEO Adminup   |
             |<----------------------|
             |                       |
             | FEO Config-Resp       |
             |---------------------->|
             |                       |
```

Figure 7: 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 continously 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 8 below, helps illustrate this state.

```
          FE PL                          CE PL

            |                              |
            |    Heart Beat                |
            |<---------------------------->|
            |                              |
            |   Heart Beat                 |
            |----------------------------->|
            |                              |
            | Config-set LFBy (Event sub.) |
            |<-----------------------------|
            |                              |
            |     Config Resp LFBy         |
            |----------------------------->|
            |                              |
            |  Config-set LFBx Attr        |
            |<-----------------------------|
            |                              |
            |     Config Resp  LFBx        |
            |----------------------------->|
            |                              |
            |Config-Query LFBz (Stats)     |
            |<--------------------------- -|
            |                              |
            |    Query Resp LFBz           |
            |----------------------------->|
            |                              |
            |    FE Event Report           |
            |----------------------------->|
            |                              |
            |  Config-Del LFBx Attr        |
            |<-----------------------------|
            |                              |
            |     Config Resp LFBx         |
            |----------------------------->|
            |                              |
            |    Packet Redirect LFBx      |
            |----------------------------->|
            |                              |
            |    Heart Beat                |
            |<-----------------------------|
            .                              .
            .                              .
            |                              |
```

Figure 8: 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) .

4.4.3. Transaction messaging

This section illustrates the message sequence of a successful 2PC between one CE and an FE. The case of the multiple FEs is left as an exercise for the reader

```
      FE PL                                                   CE PL

        |                                                       |
        | (1) Config, SOT,AT, EM=All-or-None, OP= SET/DEL,etc  |
        |<------------------------------------------------------|
        |                                                       |
        | (2) ACKnowledge                                       |
        |------------------------------------------------------>|
        |                                                       |
        | (3) Config, MOT,AT, EM=All-or-None, OP= SET/DEL,etc  |
        |<------------------------------------------------------|
        |                                                       |
        | (4) ACKnowledge                                       |
        |------------------------------------------------------>|
        |                                                       |
        | (5) Config, MOT,AT, EM=All-or-None, OP= SET/DEL,etc  |
        |<------------------------------------------------------|
        |                                                       |
        | (6) ACKnowledge                                       |
        |------------------------------------------------------>|
        .                                                       .
        .                                                       .
        .                                                       .
        .                                                       .
        |                                                       |
        | (N) Config, EOT,AT, EM=All-or-None, OP= COMMIT       |
        |<------------------------------------------------------|
        |                                                       |
        | (N+1)Config-response, ACKnowledge, COMMIT-RESPONSE   |
        |------------------------------------------------------>|
```

Figure 9: Example of a two phase commit

The flow of for a 2PC message sequence is described with more clarity in section Section 4.3.1.2.2. 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 transactional flag are set to indicate a Start of Transaction (SOT), Atomic Transaction (AT), execute-all-or-none.

- The FE validates that it can execute the request successfully and then issues an acknowledgment back to the 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 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 transactional flags are 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 the CE in step #(N+1) which contains a COMMIT-RESPONSE.

5. TML Requirements

The requirements below are expected to be delivered by the TML. This text does not define how such mechanisms are delivered. As an example they could be defined to be delivered via hardware or between 2 or more TML 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.

1. Reliability
   As defined by RFC 3654, section 6 #6.

2. Security
   TML provides security services to the ForCES PL. A TML layer should support the following security services and describe how they are achieved.

   * Endpoint authentication of FE and CE

   * Message authentication

   * Confidentiality service

3. Congestion control
   The congestion control scheme used needs to be defined. The congestion control mechanism defined by the TML should prevent the FE from being overloaded by the CE or the CE from being overwhelmed by traffic from the FE. Additionally, the circumstances under which notification is sent to the PL to notify it of congestion must be defined.

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, the 0-3 could go in one TML priority level, while 4-7 could go in the other.

8. Protection against DoS attacks
   As described in RFC 3654, section 6

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 a 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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|version| rsvd  | Message Type  |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Source ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Destination ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Correlator                          |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Flags                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10: Common Header

The message is 32 bit aligned.

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

rsvd (4 bit):
    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 bit):

Dest ID (32 bit):

* Each of the source and destination IDs are 32 bit IDs which are unique NE-wide and which 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 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|TS |                           sub-ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 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 use of the correlator is outside the scope of this specification.

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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   |     |     |   | |   |                                     |
|ACK| Pri |Rsr  |EM |A|TP |     Reserved                        |
|   |     | vd. |   |T|   |                                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13: Header Flags

- ACK: ACK indicator (2 bit)
  The ACK indicator flag is only used by the CE when sending a Config Message (Section 7.5.1) or a HB message (Section 7.9) 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 below:

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

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

    'FailureACK'(0b10) - to indicate 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 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 protocols 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 from happening.

- 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 stand-alone message or one of multiple messages that belongs 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 which make them a good candidate for encoding the XML definitions of the LFB class model. These are:

- o Providing for binary type-value encoding that is close to the XML string tag-value scheme.

- o Allowing for fast generalized binary-parsing functions.

- o 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.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| TLV Type                      | TLV Length                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Value (Essentially the TLV Data)                  |
~                                                               ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14: TLV Representation

TLV Type (16):
The TLV type field is two octets, and semantically indicates the type of data encapsulated within the TLV.

TLV Length (16):
The TLV length field is two 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 bytes 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

The "Type" values in the TLV are global in scope. This means that wherever TLVs occur in the PDU, a specific Type value refers to the same Type of TLV. This is a design choice that was made to ease debugging of the protocol.

6.3. ILV

A slight variation of the TLV known as the ILV. This sets the type ("T") to be a 32-bit local index that refers to a ForCES element ID. (refer to Section 6.4.1). The Length part of the ILV is fixed at 32 bits.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Identifier                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Length                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Value                                 |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 7.1.1.1.8 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 model draft [FE-MODEL] in order to provide context. The reader is refered to [FE-MODEL] 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, SPARSEDATA, and RESULT TLVs.

6.4.1. Paths

The model draft [FE-MODEL] 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 an "element id". This scheme allows the attribute to be "addressed" in a protocol construct.

These addressable entities could be hierachical (e.g., a table column or a cell within a table row). In order to address hierachical data, the concept of a path is introduced by the model [FE-MODEL]. A path is a series of 32-bit element IDs which are typically presented in a dot-notation (e.g., 1.2.3.4). The protocol grammar (Section 7.1) formally defines how paths are used to reference data that is being encapsulated within a protocol message.

6.4.2. Keys

The model draft [FE-MODEL] 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 which 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 7.1) 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 on 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 elements. 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 7.1.1.1.7 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 is selected first. The LFB Selector is used to select an LFB type and instance being targeted. The protocol grammar (Section 7.1) 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 21).

```
      main hdr (Message type: example "config")
       |
       |
       |
       +- T = LFBselect
              |
              +-- LFBCLASSID (unique per LFB defined)
              |
              |
              +-- LFBInstance  (runtime configuration)
              |
              +-- T = An operation TLV describes what we do to an entity
                  |   //Refer to the OPERSELECT values enumerated below
                  |   //the TLVs that can be used for operations
                  |
                  |
                  +--+-- one or more path encodings to target an entity
                     | // Refer to the discussion on keys and paths
                     |
                     |
                     +-- The associated data, if any, for the entity
                        // Refer to discussion on FULL/SPARSE DATA TLVs
```

Figure 16: Entity Addressing

7. Protocol Construction

7.1. Protocol Grammar

The protocol construction is formally defined using a BNF-like syntax to describe the structure of the PDU layout. This is matched to a precise binary format later in the document.

Since the protocol is very flexible and hierarchical in nature, it is easier at times to see the visualization layout. This is provided in Section 7.1.2

7.1.1. Protocol BNF

The format used is based on [RFC2234]. The terminals of this grammar are flags, IDcount, IDs, KEYID, and encoded data, described after the grammar.

1. A TLV will have the word "-TLV" suffix at the end of its name

2. An ILV will have the word "-ILV" suffix at the end of its name

3. / is used to separate alternatives

4. parenthesized elements are treated as a single item

5. * before an item indicates 0 or more repetitions

6. 1* before an item indicates 1 or more repetitions

7. [] around an item indicates that it is optional (equal to 1*)

The BNF of the PL level PDU is as follows:

```
PL level PDU := MAINHDR MAINSELECT
MAINHDR := The PL PDU header defined in section "Common Header"
MAINSELECT := ASSOCIATION / ASSOCIATION-RESP / ASSOCIATION-TEAR /
           CONFIG / CONFIG-RESP / QUERY / QUERY-RESP /
           EVENT / REDIRECT / HEARTBEAT
ASSOCIATION := LFBselect-TLV
ASSOCIATION-RESP := ASResult-TLV
ASSOCIATION-TEAR := ASTreason-TLV
CONFIG := 1*[LFBselect-TLV]
CONFIG-RESP := 1*[LFBselect-TLV]
QUERY := LFBselect-TLV
QUERY-RESP := LFBselect-TLV
EVENT := LFBselect-TLV
REDIRECT := REDIRECT-TLV
HEARTBEAT := empty message as described in section "Heartbeat Message"
LFBselect-TLV :=   LFBCLASSID LFBInstance 1*OPERSELECT
LFBCLASSID := the LFB Class ID
LFBInstance := the instance of the LFB class
ASResult-TLV := carries the Association Setup result code
ASTreason-TLV := carries the Association Teardown reason
OPERSELECT := 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 element which may nest
                further FULLDATA-TLVs
SPARSEDATA-TLV := encoded data that may have optionally
                  appearing elements
RESULT-TLV := Holds result code and optional FULLDATA-TLV
```

Figure 17: BNF of PL level PDU

- o MAINHDR defines a message type, Target FE/CE ID etc. The MAINHDR also defines the content. As an example the content of a "config" message would be different from an "association" message. The table below illustrates the different message types.

Message Name	Numeric Type	Reference
Association Setup	0x1	Section 7.4.1
Association Setup Response	0x11	Section 7.4.2
Association Teardown	0x02	Section 7.4.3
Config	0x03	Section 7.5.1
Config Response	0x13	Section 7.5.2
Query	0x04	Section 7.6.1
Query Response	0x14	Section 7.6.2
Event Notification	0x05	Section 7.7
Packet Redirect	0x06	Section 7.8
Heartbeat	0x0F	Section 7.9

Table 1

o MAINSELECT is a place holder to select one of several TLVs that could follow the common header. The appearance of these TLVs is message type specific and is demonstrated in the table below.

Message	MAINSELECT	OPERSELECT
Association Setup	LFBselect	REPORT
Association Setup Response	ASRresult	None
Association Teardown	ASTreason	None
Config	LFBselect	SET, DEL, COMMIT, SET-PROPERTY
Config Response	LFBselect	SET-RESPONSE, DEL-RESPONSE, SET-PROPERTY-RESPONSE, COMMIT-RESPONSE
Query	LFBselect	GET, GET-PROPERTY
Query Response	LFBselect	GET-RESPONSE, GET-PROPERTY-RESPONSE
Event Notification	LFBselect	REPORT
Packet Redirect	Redirect	None
Heartbeat	None	None

Table 2

- o When an LFB class is defined, it is assigned a unique value as an identifier. LFBCLASSID contains such an identifier.
- o LFBInstance is the identifier of a particular instance of an LFB class.
- o OPERSELECT is a place holder in the grammar to select the TLV to uniquely identify the type of operation.
- o LFBselect is a TLV that is used by some messages as shown in the grammar above. Table 2 illustrates what each message type could have in terms of MAINSELECT and OPERSELECT restrictions.
- o PATH-DATA-TLV identifies the exact element 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-PROPERTY requests and GET-RESPONSE/GET-PROPERTY-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 bit): count of 32 bit IDs

  * IDs: zero or more 32bit 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 mismatch is a protocol format error.

- A KEYINFO TLV contains information used in content keying.

  * A 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 represent the field or fields which make up the key identified by the KEYID.

- DATA may contain a FULLDATA-TLV, SPARSEDATA-TLV, a RESULT-TLV or 1 or more further PATH-DATA selection. FULLDATA and SPARSEDATA are only allowed on SET or SET-PROPERTY requests, or on responses which 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 and SPARSEDATA contain "the data" whose path has been selected by the PATH. Refer to Section 7.1.1.1 for details.

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

OPERSELECT	FULLDATA TLV	SPARSEDATA TLV	RESULT TLV
SET	MAY	MAY	MUST NOT
SET-PROPERTY	MAY	MAY	MUST NOT
SET-RESPONSE	MAY	MUST NOT	MUST
SET-PROPERTY-RESPONSE	MAY	MAY	MUST NOT
DEL	MAY	MAY	MUST NOT
DEL-RESPONSE	MAY	MUST NOT	MUST
GET	MUST NOT	MUST NOT	MUST NOT
GET-PROPERTY	MUST NOT	MUST NOT	MUST NOT
GET-RESPONSE	MUST	MUST NOT	MAY
GET-PROPERTY-RESPONSE	MUST	MUST NOT	MAY
REPORT	MAY	MUST NOT	MUST NOT
COMMIT	MUST NOT	MUST NOT	MUST NOT
COMMIT-RESPONSE	MUST NOT	MUST NOT	MAY

Table 3

o RESULT contains the indication of whether the individual SET or SET-PROPERTY succeeded. If there is a request for verbose response, then SET-RESPONSE or SET-PROPERTY-RESPONSE will also contain the FULLDATA TLV showing the data that was set. 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 characteristic:

o There can be one or more LFB class ID and instance ID combination targeted in a message (batch)

o There can one or more operations on an addressed LFB class ID/ instance ID combination (batch)

o There can be one or more path targets per operation (batch)

o 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.

7.1.1.1. Discussion on encoding

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

7.1.1.1.1. Data Packing Rules

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

o 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.

o Variable sized data within a FULLDATA TLV will be encapsulated inside another FULLDATA TLV inside the V of the outer TLV. For example of such a setup refer to Appendix C and Appendix D

o In the case of FULLDATA TLVs:

   * When a table is referred to in the PATH (IDs) of a PATH-DATA-TLV, then the FULLDATA's "V" will contain that table's row content prefixed by its 32 bit index/subscript. On the other hand, when PATH flags are 00, the PATH may contain an index pointing to a row in table; in such a case, the FULLDATA's "V" will only contain the content with the index in order to avoid ambiguity.

7.1.1.1.2. Path Flags

The following flags are currently defined:

o SELECTOR Bit: F_SELKEY indicates that a KEY Selector is present following this path information, and should be considered in evaluating the path.

o FIND-EMPTY Bit: This must not be set if the F_SEL_KEY bit is set. This must only be used on a create operation. If set, this indicates that although the path identifies an array, the SET operation should be applied to the first unused element in the array. The result of the operation will not have this flag set, and will have the assigned index in the path.

Example: For a given LFB class, the path 2.5 might select an array in a structure. If one wanted to set element 6 in this array, then the path 2.5.6 would define that element. However if one wanted to create an element in the first empty spot in the array, the CE would then send the TLV with the FIND-EMPTY bit set with the path set to 2.5. Essentially,this is an optimization so as to not require the CE to fully track all the tables.

7.1.1.1.3. 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.

7.1.1.1.4. 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.

7.1.1.1.5. LFBselect-TLV

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

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type = LFBselect       |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           LFB Class ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         LFB Instance ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         OPERSELECT                            |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                            ...                                ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         OPERSELECT                            |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 18: 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.

OPERSELECT:
It describes an operation nested in the LFBselect TLV. Note that usually there SHOULD be at least one OPERSELECT present for an LFB select TLV, but for the Association Setup Message defined in Section 7.4.1. the OPERSELECT is optional.

7.1.1.1.6. OPERSELECT

The OPERSELECT is a place holder in the grammar for TLVs that define operations. The different types are defined in Table 4, below.

OPERSELECT	TLV "Type"	Comments
SET	0x0001	From CE to FE. Used to create or add or update attributes
SET-PROPERTY	0x0002	From CE to FE. Used to create or add or update attributes
SET-RESPONSE	0x0003	From FE to CE. Used to carry response of a SET
SET-PROPERTY-RESPONSE	0x0004	From FE to CE. Used to carry response of a SET-PROPERTY
DEL	0x0005	From CE to FE. Used to delete or remove an attribute
DEL-RESPONSE	0x0006	From FE to CE. Used to carry response of a DEL
GET	0x0007	From CE to FE. Used to retrieve an attribute
GET-PROPERTY	0x0008	From CE to FE. Used to retrieve an attribute property
GET-RESPONSE	0x0009	From FE to CE. Used to carry response of a GET
GET-PROPERTY-RESPONSE	0x000A	From FE to CE. Used to carry response of a GET-PROPERTY
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 an FE to CE. Used to confirm a commit in a 2PC transaction

Table 4

Different messages define OPERSELECT are valid and how they are used

(refer to Table 2 and Table 3).

SET, SET-PROPERTY, and GET/GET-PROPERTY requests are issued by the CE and do not carry RESULT TLVs. On the other hand, SET-RESPONSE, SET-PROPERTY-RESPONSE and GET-RESPONSE/GET-PROPERTY-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-PROPERTY-RESPONSE, each FULLDATA 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 which failed will appear in the response. If the request was for AlwaysACK, then all elements of the request will appear in the response with RESULT TLVs.

Note that if a SET/SET-PROPERTY request with a structure in a FULLDATA 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, the FE can select which error it chooses to return. So if a FULLDATA for a SET/SET-PROPERTY of a structure attempts to write one field which is read only, and attempts to set another field to an invalid value, the FE can return indication of either error.

A SET/SET-PROPERTY operation on a variable length element 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 element or an optional structure element.

7.1.1.1.7. Result TLV

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

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type = RESULT          |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Result Value  |                  Reserved                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 19: 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_ELEMENT_DOES_NOT_EXIST	0x09	The specified path is possible but the element 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 occured while processing a message (no error detected in the message itself)
-	0x18-0xFE	Reserved
E_UNSPECIFIED_ERROR	0xFF	Unspecified error (for when the FE can not decide what went wrong)

Table 5

7.1.1.1.8. DATA TLV

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

Below are the encoding rules for the FULLDATA 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 element at that path level is present. In example 1(c) of Appendix C this concept is illustrated by the presence of all elements 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 element 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, it does not need to be ordered since the "I" in the ILV uniquely identifies the element. Examples 1(a) and 1(b) illustrate the power of SPARSEDATA encoding.

3. Inside a FULLDATA TLV

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

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

4. Inside a SPARSEDATA TLV

   * The values for atomic fields may be given with ILVs (32-bit index, 32-bit length)

5. Any of the FULLDATA TLVs can contain an ILV but an ILV cannot contain a FULLDATA. This is because it is hard to disambiguate ILV since an I is 32 bits and a T is 16 bits.

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

7.1.1.1.9. 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 being 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.

- o 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.

7.1.2. 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 operation.

```
   main hdr (Config in this case)
        |
        |
        +--- T = LFBselect
        |        |
        |        +-- LFBCLASSID
        |        |
        |        |
        |        +-- LFBInstance
        |        |
        |        +-- T = SET
        |        |   |
        |        |   +--  // one or more path targets
        |        |        // with their data here to be added
        |        |
        |        +-- T  = DEL
        |        .   |
        |        .   +--  // one or more path targets to be deleted
        |
        |
        +--- T = LFBselect
        |        |
        |        +-- LFBCLASSID
        |        |
        |        |
        |        +-- LFBInstance
        |        |
        |        + -- T= SET
        |        |    .
        |        |    .
        |        + -- T= DEL
        |        |    .
        |        |    .
        |        |
        |        + -- T= SET
        |        |    .
        |        |    .
        |
        |
        +--- T = LFBselect
                |
                +-- LFBCLASSID
                |
                +-- LFBInstance
                .
                .
                .
```

Figure 20: 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
     |             |    + -- KEY_ID
     |             |    + -- KEY_DATA
     |             |
     |             + -- T = FULLDATA
     |                  + -- data
     |
     |
     T = SET
     |  |
     |  +- T = Path-data
     |  |  |
     |  |  + -- flags
     |  |  + -- IDCount
     |  |  + -- IDs
     |  |  |
     |  |  + -- T = FULLDATA
     |  |          + -- data
     |  +- T = Path-data
     |     |
```

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

Figure 21: Sample operation layout

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

7.2. 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:

o 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 attributes 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.

7.2.1. 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 0x1. 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 attributes 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.

7.2.1.1. FE Protocol capabilities

FE Protocol capabilities are read-only.

7.2.1.1.1. SupportableVersions

ForCES protocol version(s) supported by the FE

7.2.1.1.2. FE Protocol Attributes

FE Protocol attributes (can be read and set).

7.2.1.1.2.1. CurrentRunningVersion

Current running version of the ForCES protocol

7.2.1.1.2.2. FEID

FE unicast ID

7.2.1.1.2.3. MulticastFEIDs

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

7.2.1.1.2.4. 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 below:

- 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 7.9 for details. The CE HDI attribute as described below is tied to this policy.

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

- Others - reserved.

7.2.1.1.2.5. 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 30 s.

7.2.1.1.2.6. 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 CE whenever CE sends such Heartbeat Request Message. Refer to Section 7.9 and Section 4.3.3 for details.

o 1 - This policy specifies that 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 FE to CE during that interval as described in Section 4.3.3.

o Others - Reserved.

7.2.1.1.2.7. 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 FE to CE during that interval as described in Section 4.3.3. The default value for an FE HI is 500ms.

7.2.1.1.2.8. CEID

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

7.2.1.1.2.9. LastCEID

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

7.2.1.1.2.10. BackupCEs

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

7.2.1.1.2.11. 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 7.2.1.1.2.8, Section 7.2.1.1.2.10, Section 7.2.1.1.2.12, and Section 8. When an association is lost, depending on configuration, one of the policies listed below is activated.

o 0 (default) - FE should stop functioning immediately and transition to FE DOWN.

o 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 (Section 7.2.1.1.2.8) or one of possibly several backup CEs (Section 7.2.1.1.2.10), the FE will go operationally down.

o Others - Reserved

7.2.1.1.2.12. CEFTI

CE Failover Timeout Interval (CEFTI) - The time interval associated with the CE failover policy case '0' and '2'. 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.

7.2.1.1.2.13. 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 FE down and then need to restart. The values are defined as below:

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

- o others - Reserved for future use.

7.2.2. FE Object LFB

The FE Object LFB is a logical entity in each FE and contains attributes 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 [FE-MODEL]. 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 [FE-MODEL]. 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 attribute and capability information from this LFB instance when the CE begins controlling the operation of the FE.

7.3. 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 which 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 which resides in the FE.

7.4. Association Messages

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

7.4.1. Association Setup Message

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

Message transfer direction:
FE to CE

Message header:
The Message Type in the header is set 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 the setup is successful or not. 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 7.1.1.1.5. 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 OPERSELECT in the LFBselect TLV is defined as a 'REPORT' operation. More than one attribute may be announced in this message using REPORT operation to let the FE declare its configuration parameters in an unsolicited manner. These may contain attributes suggesting values such as the FE HB Interval, or the FEID. The OPERSELECT used is defined below.

OPERSELECT for Association Setup:

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    Type = REPORT              |               Length          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                     PATH-DATA-TLV for REPORT                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 22: OPERSELECT

Type:
Only one operation type is defined for the association setup message:

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

PATH-DATA-TLV for REPORT:
This is generically a PATH-DATA-TLV format that has been defined in "Protocol Grammar" section (Section 7.1) in the PATH-DATA BNF definition. The PATH-DATA-TLV for 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 7.1.1.1.7 and the FULLDATA-TLV is defined in Section 7.1.1.1.8.

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

```
main hdr (type =  Association Setup)
|
|
+--- T = LFBselect
|        |
|        +-- LFBCLASSID = FE object
|        |
|        |
|        +-- LFBInstance = 0x1
|
+--- T = LFBselect
         |
         +-- LFBCLASSID = FE Protocol object
         |
         |
         +-- LFBInstance = 0x1
             |
             +---OPERSELECT = REPORT
                 |
                 +-- Path-data to one or more attributes
```

Figure 23: PDU Format For Association Setup

7.4.2. 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 the setup is successful or not, i.e., whether an association is established.

Message transfer direction:
CE to FE

Message Header:
The Message Type in the header is set 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.

Message body:
The Association Setup Response message body only consists of one TLV, the Association Result TLV, the format of which is as follows:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type = ASRresult       |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Association Setup Result                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 24: ASResult OPERSELECT

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 msg 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

7.4.3. 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 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.

Message Body:
The association teardown message body only consists of one TLV, the Association Teardown Reason TLV, the format of which is as follows:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Type = ASTreason      |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Teardown Reason                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 25: 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

7.5. 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.

7.5.1. Config Message

This message is sent by the CE to the FE to configure LFB attributes 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 format. Detailed description of the message is as below.

Message transfer direction:
    CE to FE

Message Header:
    The Message Type in the header is set 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 FE is expected by the message.

Message body:
    The config message body MUST consist of at least one LFBselect TLV as described in Section 7.1.1.1.5. The OPERSELECT in the LFBselect TLV is defined below.

OPERSELECT for Config:

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Type               |               Length            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          PATH-DATA-TLV                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 26: OPERSELECT 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 attributes

Type = "SET-PROPERTY" - this operation is to set LFB attribute properties

Type = "DEL" - this operation to delete some LFB attributes

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).

PATH-DATA-TLV:
This is generically a PATH-DATA-TLV format that has been defined in "Protocol Grammar" section (Section 7.1) in the PATH-DATA BNF definition. The restriction on the use of PATH-DATA-TLV for SET/ SET-PROPERTY operation is that it MUST contain either a FULLDATA 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 or SPARSEDATA TLV(s), but MUST NOT contain any RESULT-TLV. The RESULT-TLV is defined in Section 7.1.1.1.7 and FULLDATA and SPARSEDATA TLVs is defined in Section 7.1.1.1.8.

*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 FULL or SPARSEDATA TLV(s)
                   |
                   +-- T = operation { DEL }
                   |   |
                   |   +--  // one or more path targets
```

Figure 27: PDU Format for Config

7.5.2. Config Response Message

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

Message transfer direction:
FE to CE

Message Header:
The Message Type in the header is set 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).

Message body:
The Config message body MUST consist of at least one LFBselect TLV as described in Section 7.1.1.1.5. The OPERSELECT in the LFBselect TLV is defined below.

OPERSELECT for Config Response:

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Type                 |               Length          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        PATH-DATA-TLV                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 28: OPERSELECT 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 SET operation of LFB attributes

Type = "SET-PROPERTY-RESPONSE" - this operation is for the response of SET-PROPERTY operation of LFB attribute properties

Type = "DEL-RESPONSE" - this operation is for the response of the DELETE operation of LFB attributes

Type = "COMMIT-RESPONSE" - this operation is sent to the CE to confirm a commit success in a 2pc transaction. A COMMIT-RESPONSE 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 "Protocol Grammar" section (Section 7.1) in the PATH-DATA 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 7.1.1.1.7.

7.6. Query Messages

The ForCES query messages are used by the CE to query LFBs in the FE for informations like LFB attributes, capabilities, statistics, etc. Query Messages include the Query Message and the Query Response Message.

7.6.1. Query Message

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

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.

Message body:
The query message body MUST consist of at least one LFBselect TLV as described in Section 7.1.1.1.5. The OPERSELECT in the LFBselect TLV is defined below.

OPERSELECT for Query:

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    Type = GET/GET-PROPERTY    |               Length          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    PATH-DATA-TLV for GET/GET-PROPERTY         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 29: TLV for Query

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

Type = "GET" - this operation is to request to get LFB attributes.

Type = "GET-PROPERTY" - this operation is to request to get LFB attributes.

PATH-DATA-TLV for GET/GET-PROPERTY:
This is generically a PATH-DATA-TLV format that has been defined in "Protocol Grammar" section (Section 7.1) in the PATH-DATA BNF definition. The restriction on the use of PATH-DATA-TLV for GET/GET-PROPERTY operation is it MUST NOT contain any SPARSEDATA 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 30: PDU Format

7.6.2. 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 below.

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.

Message body:
The Query Response message body MUST consist of at least one LFBselect TLV as described in Section 7.1.1.1.5. The OPERSELECT in the LFB select TLV is defined below.

OPERSELECT for Query Response:

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |Type = GET-RESPONSE/GET-PROPERTY-RESPONSE|    Length           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |        PATH-DATA-TLV for GET-RESPONSE/GET-PROPERTY-RESPONSE   |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 31: TLV for Query Response

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

Type = "GET-RESPONSE" - this operation is to response to get operation of LFB attributes.

Type = "GET-PROPERTY-RESPONSE" - this operation is to response to GET-PROPERTY operation of LFB attributes.

PATH-DATA-TLV for GET-RESPONSE/GET-PROPERTY-RESPONSE:
This is generically a PATH-DATA-TLV format that has been defined in "Protocol Grammar" section (Section 7.1) in the PATH-DATA BNF definition. The PATH-DATA-TLV for 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 7.1.1.1.7 and the SPARSEDATA and FULLDATA TLVs are defined in Section 7.1.1.1.8.

7.7. Event Notification Message

Event Notification Message is used by FE to asynchronously notify 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 SET-PROPERTY 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 format. Detailed description of the message is as below.

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.

Message Body:
The event notification message body MUST consist of at least one LFBselect TLV as described in Section 7.1.1.1.5. The OPERSELECT in the LFBselect TLV is defined below.

OPERSELECT for Event Notification:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = REPORT              |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     PATH-DATA-TLV for REPORT                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 32: TLV for Event Notification

Type:
Only one operation type is defined for the event notification message:

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

PATH-DATA-TLV for REPORT:
This is generically a PATH-DATA-TLV format that has been defined in "Protocol Grammar" section (Section 7.1) in the PATH-DATA BNF definition. The PATH-DATA-TLV for REPORT operation MAY contain FULLDATA 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 33: PDU Format

7.8. Packet Redirect Message

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

The Packet Redirect message data format is formated as follows:

Message 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 packet redirect message is as below:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type = Redirect        |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Meta Data TLV                         |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Redirect Data TLV                     |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 34: Redirect_Data TLV

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

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = METADATA            |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Meta Data ILV                         |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                            ...                                ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Meta Data ILV                         |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 35: METADARA 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:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Meta Data ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Length                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Meta Data Value                       |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 36: 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:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = REDIRECTDATA        |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Redirected Data                        |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 37: 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 dat for all TLVs. The metadata infers what kind of packet is carried in value field and therefore allows for easy decoding of data encapsulated

7.9. 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 is managed as attributes 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 below.

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 the HB policies defined in Section 7.2.1, 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 a HB message is another HB message, i.e., no specific HB response message is defined. Whenever an FE receives a HB message marked with 'AlwaysACK' from the CE, the FE MUST send a HB message back immediately. The HB message sent by the FE in response to the 'AlwasyACK' 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 'AlwasyACK' 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 7.2.1). 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 38 extends the state machine illustrated in Figure 4 to allow for new states that facilitate connection recovery.

```
                                +-----------------+
          Lost association &&   | Pre-Association |
         CE failover policy = 0 | (Association    |
             +------------>-->--|   in            +<----+
             |                  | progress)       |     |
             |                  +--------+--------+     |
             |                           |              |
             |                           Y              |
             |                           |              |
             |                      Associated          ^
             |                           |              |
             |                           Y              |
             |                           |              |
             |                   +-------+-------+      |
             |        CE issues  |   DOWN FE     |      |
             |        FEO Admin  |  (ForCES      |      ^
             |         UP        |  Active)      |      |
             |         +-------- |               |      |
             |         |         |               |      |
             |         |         +---------------+      ^
             |         Y              ^                 |
             |         |              |                 |CEFTI expired
             |         Y              |CE issues Admin  |    &&
             |         |              | DOWN            |!connected
             |         |              |                 ^
             |         Y              |                 |
           +-+-----------+            |          +------+------+
           | UP          |------------+          |Disconnected |
           |(associated) |                       |             |
           |             |Lost association       |             |
           |             |    &&                 |             |
           |             |--------->------>----->|(Continue    |
           |             |CE failover policy     |Forwarding)  |
           |             |  = 1                  |             |
           +-------------+                       +-------------+
                  ^                                     |
                  | Resynchronize       !CEFTI expired |
                  | complete                   &&       |
                  |                       reconnected   |
                  |           +---------------+         |
                  |           | Resynch state |         |
                  |           | (via          |         |
                  +-----------| graceful      |<--------+
                              | restart)      |
                              +---------------+
```

Figure 38: FE State Machine considering HA

Section 4.2 describes transitions between the UP, DOWN and Pre-association states. In this section we deal with disconnected states.

During a communication failure between the FE and CE (which is caused due to CE or link reasons, i.e. 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 it ever reconnects again, it will re-establish state from scratch.

If the FE's FEPO CE Failover Policy is configured to mode 1, it implies that the FE is capable of HA as well as graceful restart recovery. In such a case, the FE transitions to the disconnected state and the CEFTI timer is started. The FE continues to forward packets during this state. It also recycles through its 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 connect to associate with the new primary CE. If it fails to re-associate with any CE and the CEFTI expires, the FE transitions to the Pre-association state.

If the FE, while in the Disconnected state, manages to reconnect to a new primary CE before CEFTI expires it transitions to the Resynch state. In the Resynch state, the FE tries to recover any state that may have been lost during the Disonnected state. Graceful restart is one such mechanism. How the FE achieves these goals is out of scope for this document.

Figure 39 below illustrates the Forces message sequences that the FE uses to recover the connection.

```
         FE                    CE Primary        CE Secondary
         |                        |                    |
         |  Asso Estb,Caps exchg |                    |
       1 |<---------------------->|                    |
         |                        |                    |
         |       All msgs         |                    |
       2 |<---------------------->|                    |
         |                        |                    |
         |                        |                    |
         |                   FAILURE                   |
         |                                             |
         |         Asso Estb,Caps exchange             |
       3 |<------------------------------------------->|
         |                                             |
         |              Event Report (pri CE down)     |
       4 |-------------------------------------------->|
         |                                             |
         |                   All Msgs                  |
       5 |<------------------------------------------->|
```

Figure 39: 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 attribute in ForCES 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 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 before.

9. Security Considerations

ForCES architecture identifies several levels of security in [RFC3746]. 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.

- o 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.

- o 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.

9.1.1. Endpoint Authentication

Each CE and FE PL maintains a list of associations as part its of 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 recommend even when underlying provides TLM 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.

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 draft 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, Joel M. Halpern (who should probably be listed among the authors), 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 has done a wonderful job of shaping the draft to make it more readable by providing the IESG feedback.

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

[FE-MODEL]
Yang, L., Halpern, J., Gopal, R., DeKok, A., Haraszti, Z., and S. Blake, "ForCES Forwarding Element Model", Feb. 2005.

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

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[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.

11.2. Informational 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.

Appendix A. IANA Considerations

Following the policies outlined in "Guidelines for Writing an IANA Considerations Section in RFCs" (RFC 2434 [RFC2434]), the following name spaces are defined in ForCES.

- o Message Type Name Space Section 7.1.1
- o Operation Type Name Space Section 7.1.1.1.6
- o Header Flags Section 6.1
- o TLV Type Section 7.1.1
- o TLV Result Values Section 7.1.1.1.7
- o LFB Class ID Section 7.1.1.1.5
- o Result: Association Setup Response Section 7.4.2
- o Reason: Association Teardown Message Section 7.4.3

A.1. Message Type Name Space

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

Message Types 0x00 - 0x0F
Message Types in this range are part of the base ForCES Protocol. Message Types in this range are allocated through an IETF consensus action. [RFC2434]
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	AssociationSetupRepsonse
0x12	Reserved
0x13	ConfigRepsonse
0x14	QueryResponse

Message Types 0x20 - 0x7F
Message Types in this range are Specification Required [RFC2434]
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 Name Space is unnecessary.

A.2. Operation Selection

The Operation Selection (OPERSELECT) name space is 16 bits long. The following is the guideline for managing the OPERSELECT Name Space.

OPERSELECT Type 0x0000-0x00FF
OPERSELECT Types in this range are allocated through an IETF consensus process. [RFC2434].
Values assigned by this specification:

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

OPERSELECT Type 0x0100-0x7FFF
OPERSELECT Types using this range must be documented in an RFC or other permanent and readily available reference. [RFC2434].

OPERSELECT Type 0x8000-0xFFFF
OPERSELECT Types in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the OPERSELECT Type Name Space 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 [RFC2434].

A.4. TLV Type Name Space

The TLV Type name space is 16 bits long. The following is the guideline for managing the TLV Type Name Space.

TLV Type 0x0000-0x00FF
TLV Types in this range are allocated through an IETF consensus process. [RFC2434].
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 [RFC2434].

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 Name Space 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_ELEMENT_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 name space is 32 bits long. The following is the guideline for managing the Association Setup Response Name Space.

Association Setup Response 0x0000-0x00FF
Association Setup Responses in this range are allocated through an IETF consensus process [RFC2434].
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 [RFC2434] Values using this range must be documented in an RFC or other permanent and readily available reference [RFC2434].

Association Setup Response 0x1000-0xFFFFFFFF
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 Responses Name Space is unnecessary.

A.7. Association Teardown Message

The Association Teardown Message name space is 32 bits long. The following is the guideline for managing the Association Teardown Message Name Space.

Association Teardown Message 0x00000000-0x0000FFFF
Association Teardown Messages in this range are allocated through an IETF consensus process [RFC2434].
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 [RFC2434] Association Teardown Messages using this range must be documented in an RFC or other permanent and readily available references. [RFC2434].

Association Teardown Message 0x80000000-0xFFFFFFFFF
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 Name Space is unnecessary.

Appendix B. ForCES Protocol LFB schema

The schema described below conforms to the LFB schema described in ForCES Model draft. [FE-MODEL]

Section 7.2.1 describes the details of the different attributes defined in this definition.

```
<LFBLibrary xmlns="http://ietf.org/forces/1.0/lfbmodel"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://ietf.org/forces/1.0/lfbmodel
    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="1">
         <name>FEPO</name>
         <synopsis>
            The FE Protocol Object
         </synopsis>
         <version>1.0</version>

     <attributes>
           <attribute elementID="1" access="read-only">
             <name>CurrentRunningVersion</name>
             <synopsis>Currently running ForCES version</synopsis>
             <typeRef>u8</typeRef>
```

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

```
             <typeRef>uint32</typeRef>
           </attribute>
           <attribute elementID="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>
           </attribute>
           <attribute elementID="10" access="read-write">
             <name>CEFailoverPolicy</name>
             <synopsis>
               The CE Failover Policy
             </synopsis>
             <typeRef>CEFailoverPolicyValues</typeRef>
           </attribute>

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

        <capabilities>
           <capability elementID="30" access="read-only">
              <name>SupportableVersions</name>
              <synopsis>
                 the table of ForCES versions that FE supports
              </synopsis>
```

```
              <array type="variable-size">
               <typeRef>u8</typeRef>
              </array>
           </capability>
           <capability elementID="31" access="read-only">
              <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. Attributes

All Attributes are explained in Section 7.2.1.

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

```
        Path-Data TLV
          Path to an instance of S ...
          SPARSEDATA TLV
            ElementIDof(a), lengthof(a), valueof(a)
            ElementIDof(b), lengthof(b), valueof(b)
            ElementIDof(c), lengthof(c), valueof(c)
```

(b) Describing a subset of fields

```
        Path-Data TLV
          Path to an instance of S ...
          SPARSEDATA TLV
            ElementIDof(a), lengthof(a), valueof(a)
            ElementIDof(c), lengthof(c), valueof(c)
```

Note: Even though there are non-optional elements in structure S, since one can uniquely identify elements, one can selectively send element of structure S (eg 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

```
        Path-Data TLV
          Path to an instance of S ...
          SPARSEDATA TLV
            ElementIDof(a), lengthof(a), valueof(a)
            ElementIDof(b), lengthof(b), valueof(b)
            ElementIDof(c), lengthof(c), valueof(c)
```

(b) Describing a subset of fields using SPARSEDATA

```
        Path-Data TLV
          Path to an instance of S ...
          SPARSEDATA TLV
            ElementIDof(a), lengthof(a), valueof(a)
            ElementIDof(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

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

(b) Describing a subset of fields using SPARSEDATA

```
Path to an instance of U ...
SPARSEDATA TLV
  ElementIDof(a), lengthof(a), valueof(a)
  ElementIDof(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 element 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, with two instances of z[], also described with SPARSEDATA, assuming only the 10th and 15th subscript of z[] are encoded.

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

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

Appendix D. Use Cases

Assume LFB with following attributes for the following use cases.

```
foo1, type u32, ID = 1

foo2, type u32, ID = 2

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

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

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

table4: type array, ID = 6
        elements 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
        elements are:
        p1, type u32, ID = 1
        p2, type array, ID = 2, array elements 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 attribute 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 for data encoding; although SPARSEDATA 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 and SPARSEDATA.

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. An example of "create and give me an index" Assuming one asked for verbose response back in the main message header.

```
OPER = SET-TLV
        Path-data -TLV:
                flags = FIND-EMPTY, IDCount = 1, IDs=4
                FULLDATA-TLV containing valueof(j1), valueof(j2)

Result
If 7 were the first unused entry in the table:
OPER = SET-RESPONSE
        Path-data TLV:
                flags = 0, IDCount = 2, IDs=4.7
                RESULT-TLV indicating success, and
                FULLDATA-TLV containing valueof(j1), valueof(j2)
```

10. 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)
                        .
                        .
                        .
```

11. 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)
```

12. Delete row with KEY match (j1=100, j2=200) in table 2. Note that the j1,j2 pair are a defined key for the table 2.

```
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 (with FULLDATA if verbose)
```

13. Dump contents of table3. It should be noted that this table has a column with element name that is variable sized. The purpose of this use case is to show how such an element 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, L = 4+strlen(namev),
                       V = valueof(v)
                index, someidv, TLV: T=FULLDATA, L = 4+strlen(namev),
                       V = valueof(v)
                index, someidv, TLV: T=FULLDATA, L = 4+strlen(namev),
                       V = valueof(v)
                index, someidv, TLV: T=FULLDATA, L = 4+strlen(namev),
                       V = valueof(v)
                .
                .
                .
```

14. 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 both tables are in the same LFB

Note2: Main header has atomic flag set and the request is for verbose/full results back; 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 = 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
                        SET-RESULT-TLV code = success
                                FULLDATA TLV, Length = XXXX v=20
```

15. Selective setting. On table 4 -- 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}
```

Non-verbose response mode shown:

```
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
```

16. 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)
```

17. From table5's row 10 table10, get X2s based on 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)
```

18. Further example of manipulating a table of tables

```
Consider table 6 which is defined as:
table6: type array, ID = 8
        elements are:
        p1, type u32, ID = 1
        p2, type array, ID = 2, array elements of type type-A

type-A:
        a1, type u32, ID 1,
        a2, type array ID2 ,array elements 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
```

19. Get a whole LFB (all its attributes, 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 encoding of the FE Object LFB
```

Authors' Addresses

Ligang Dong
Zhejiang Gongshang University
149 Jiaogong Road
Hangzhou 310035
P.R.China

Phone: +86-571-88071024
Email: donglg@mail.zjgsu.edu.cn

Avri Doria
ETRI
Lulea University of Technology
Lulea
Sweden

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

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

Phone: +1-781-993-3685
Email: ram.gopal@nokia.com

Robert Haas
IBM
Saumerstrasse 4
8803 Ruschlikon
Switzerland

Phone:
Email: rha@zurich.ibm.com

Jamal Hadi Salim
Znyx
Ottawa, Ontario
Canada

Phone:
Email: hadi@znyx.com

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

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

Weiming Wang
Zhejiang Gongshang University
149 Jiaogong Road
Hangzhou 310035
P.R.China

Phone: +86-571-88057712
Email: wmwang@mail.zjgsu.edu.cn

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).