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 5, 2006

ForCES Protocol Specification
draft-ietf-forces-protocol-07.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 6, 2006.

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 (ETRI), 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, RFC 2119 [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 3764 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. ForCES FE model [FE-MODEL] presents the capabilities, state and configuration of FEs within the context of the ForCES protocol, so that CEs can accordingly control the FEs in a standardizded way and by means of 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. Information exchanged between FEs and CEs makes extensive use of TLVs. The protocol includes commands for transport of LFB configuration information, association setup, status and event notifications, etc.

This specification does not define a transport mechanism for protocol messages, but does include a discussion of service primitives that must be provided by the underlying transport interface.

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.

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 several Protocol Scenarios and includes message exchange descriptions.

Section 9 describes a mechanism in the protocol to support high availability mechanisms including redundancy and fail over.
Section 10 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.

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 a CE via the ForCES protocol.

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.

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.

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.

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.

ForCES Network Element (NE) - An entity composed of one or more CEs and one or more FEs. To entities outside a NE, the NE represents a single point of management. Similarly, a 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.

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

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.

LFB (Logical Function Block) and LFB Instance -- LFBs are categorized by LFB Classes(or Types). 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 specify 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 or/and 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.

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).

Inter-FE Topology -- See FE Topology.

Intra-FE Topology -- See LFB Topology.

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.

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) 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 layer and the TML layer.

This is depicted in Figure 3 below.

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

Figure 3: ForCES Interface

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

On transmit, the PL layer delivers its messages to the TML layer. The TML layer delivers the message to the destination TML layer(s). On receive, the TML delivers the message to its destination PL layer(s).

4.1.1. The PL layer

The PL is common to all implementations of ForCES and is standardized

by the IETF as defined in this document. The PL layer 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 layer to transmit various subscribed-to events to the CE PL layer 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 layer. 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 layer

The TML layer transports the PL layer messages. The TML is where the issues of how to achieve transport level reliability, congestion control, multicast, ordering, etc. are handled. It is expected 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 layers, are out of scope of the ForCES protocol. The best way to think of them are 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 read a static configuration file. RFC 3746 has a more detailed descriptions 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 of 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. Issuing the ID for the FE or CE would also be issued during the pre-association phase.

c. Security parameterization such as keys etc.

d. Connection association parameters

Example of this might be:

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

o or more 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 start            CE configures
         -------+    +--->---->---->---->------->----+
                |    |                               Y
                Y    |                               |
                |    |                               Y
              +------+--+                        +---------+
              | FE      |                        | FE      |
              | DOWN    |                        | UP      |
              | State   |                        | State   |
              |         |                        |         |
              +---------+                        +---------+
                   ^                                  Y
                   |                                  |
                   +-<---<------<-----<------<----<---+
                    CE configures or FE loses association
```

Figure 4: The FE State Machine

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.

CE configures FE states transitions by means of a so-called FEObject LFB, which is defined in [FE-MODEL] and also explained in Section 4.3.3 of this document. In FEObject 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 FEObject LFB itself is active.

On start up the FE is in the DOWN state unless 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 FE transitions from the UP state to the DOWN state when it receives a FEObject Admin Down action or when it loses its association with the CE. 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.

Note: in the case of loss of association, the FE can also be configured to not go to the DOWN state.

For the FE to properly complete the transition to the DOWN state it must stop packet forwarding and that this may affect 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. 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 8.

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

4.2.2.2. Association 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 continues until a termination occurs because of loss of connectivity or is initiated by either the CE or the FE.

Refer to section on protocol scenarios, Section 8, 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 physically initiated by either party for administrative purposes but may also be driven by operational reasons such as loss of connectivity.

It should be noted that loss of connectivity between TMLs is not necessarily indicative of loss of association between respective PL layers unless the programmed FE Protocol Object time limit is exceeded. In other words if the TML repairs the transport loss before then, the association would still be valid.

When an association is lost between a CE and FE, the FE continues to operate as instructed by the CE via the CE failover policy (for further discussion refer to Section 9 and Appendix B).

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 windows.

The EM (Execute Mode) flag, AT (Atomic Transaction) flag, and TP (Transaction Phase) flag as defined in Common Header Section (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 in one protocol message. The EM flag defined in Common Header Section (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, independent operations in a message targeted at one or more LFB selectors will all be executed if no failure occurs for any of the operations. If there is any failure for any of the operations then none of the operations will be executed, i.e there is roll back for this mode of operation.

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 roll back 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.

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 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). The last message is indicated by by EOT.

Any failure notified by the FE causes the CE to execute an Abort Transaction (ABT) to all FEs involved in the transaction, rolling back all previously executed operations in the transaction.

The transaction commitment phase is signaled from the CE to the FE by an End of Transaction (EOT) configuration message. 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.

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 it 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 9

and Section 7.2).

4.3.2. Scalability

It is desirable that the PL layer 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 layer 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 window.

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 windowing 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:

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

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

These LFBs are detailed in Section 7.2.

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. 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, on config basis, the PL layer may wish to participate in link failover schemes and therefore the TML must support this capability.
   Please refer to Section 9 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 layer 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 the Requirements 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 ports, 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 layer 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 7: 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 Dest IDs are 32 bit IDs which are unique NE-wide and which recognize 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 8: 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 9: 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.

*   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 a 64 bit pointer to a context block. All such implementation specific use of the Correlator is outside the scope of this specification.

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 10: 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 was 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 messages never expect 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. 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.

- 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

```
 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                      | variable TLV Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Value (Data of size TLV length)                    |
~                                                               ~
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 11: TLV Representation

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

TLV Length (16):
: The TLV Length field is two octets, and indicates the length of this TLV including the TLV Type, TLV Length, and the TLV data in octets.

TLV Value (variable):
: The TLV Value field carries the data. For extensibility, the TLV value may in fact be a TLV. TLVs must be 32 bit aligned.

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 an 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. The Length part of the ILV is fixed at 32 bits.

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

Figure 12: 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.

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 RFC 2234. 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 [MAIN-TLV]
MAIN-TLV := [LFBselect-TLV] / [REDIRECT-TLV] /
            [ASResult-TLV] / [ASTreason-TLV]
LFBselect-TLV :=   LFBCLASSID LFBInstance OPER-TLV
OPER-TLV := 1*PATH-DATA-TLV
PATH-DATA-TLV := PATH  [DATA]
PATH := flags IDcount IDs [SELECTOR]
SELECTOR :=  KEYINFO-TLV
DATA := FULLDATA-TLV / SPARSEDATA-TLV / RESULT-TLV /
       1*PATH-DATA-TLV
KEYINFO-TLV := KEYID FULLDATA-TLV
SPARSEDATA-TLV := encoded data that may have optionally
                  appearing elements
FULLDATA-TLV := encoded data element which may nest
                further FULLDATA-TLVs
RESULT-TLV := Holds result code and optional FULLDATA-TLV
```

Figure 13: 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.

- o MAIN-TLV is one of several TLVs that could follow the Mainheader. The appearance of these TLVs is message type specific.

- o LFBCLASSID is a 32 bit unique identifier per LFB class defined at class Definition time.

- o LFBInstance is a 32 bit unique instance identifier of an LFB class

- o OPER-TLV uses the Type field in the TLV to uniquely identify the type of operation i.e one of {SET, GET, DEL,etc.} depending on the message type.

- 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 requests and GET response is terminated by encoded data or response in the form of either FULLDATA-TLV or SPARSEDATA-TLV or RESULT-TLV.

- o 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.

o 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.

o 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.

o 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 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.

o RESULT contains the indication of whether the individual SET succeeded. If there is an indication for verbose response, then SET-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 + InstanceId combination targeted in a message (batch)

o There can one or more operations on an addressed LFB classid+ instanceid 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 classid+instance targeting. To target multiple instances within the same class, multiple LFBselect are needed.

7.1.1.1. Discussion on Grammar

In the case of FULLDATA encoding, data is packed in such a way that a receiver of such data with knowledge of the path can correlate what it means by inferring in the LFB definition. This is an optimization that helps reducing the amount of description for the data in the protocol.

In other words:

It is assumed that the type of the data can be inferred by the context in which data is used. Hence, data will not include its type information. The basis for the inference is typically the LFB class id and the path.

It is expected that a substantial number of operations in ForCES will need to reference optional data within larger structures. For this reason, the SPARSEDATA encoding is introduced to make it easier to encapsulate optionally appearing data elements.

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 D and Appendix C.

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. OTOH, 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.

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. LFB select TLV

The LFB select 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 = LFBselect       |               Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           LFB Class ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         LFB Instance ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Operation TLV                         |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                             ...                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Operation TLV                         |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14: 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.

Operation TLV:
It describes an operation nested in the LFB select TLV. Note that usually there SHOULD be at least one Operation TLV present for an LFB select TLV, but for the Association Setup Message defined in Section 7.4.1. the Operation TLV is optional. In this case there might not be an Operation TLV followed in the LFB select TLV.

7.1.1.1.6. Operation TLV

The Operation TLV is an instance of TLV defined in Section 6.2. It is assumed that specific operations are identified by the Type code of the TLV. Definitions for individual Types of operation TLVs are in corresponding message description sections followed.

SET and GET Requests do not have result information (they are

requests). SET and GET Responses have result information. SET and GET Responses use SET-RESPONSE and GET-RESPONSE operation TLVs.

For a GET response, individual GETs which succeed will have FULLDATA TLVs added to the leaf paths to carry the requested data. For GET elements that fail, instead of the FULLDATA TLV there will be a RESULT TLV.

For a SET response, each FULLDATA or or SPARSEDATA TLV in the original request will be replaced with a RESULT TLV in the response. If the request was for Ack-fail, then only those items which failed will appear in the response. If the request was for ack-all, then all elements of the request will appear in the response with RESULT TLVs.

Note that if a SET 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 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 whatever error it likes.

A SET 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 15: Result TLV

The defined Result Values are

0 = success

1 = no such object

2 = permission denied (e.g., trying to configure an attribute that is read- only)

3 = invalid value (the encoded data could not validly be stored in the field)

4 = invalid array creation (when the subscript in an array create is not allowed)

255 = unspecified error (for when the FE can not decide what went wrong)

others = Reserved

7.1.1.1.8. DATA TLV

A FULLDATA TLV has "T"= FULLDATA, and a 16bit Length followed by the data value/contents. Likewise, a SPARSEDATA TLV has "T" = SPARSEDATA, a 16bit 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. Rules:

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

2. FULLDATA TLV may be used at a particular path only if every element at that path level is present. This requirement holds whether the fields are fixed or variable length, mandatory or optional.

   * If a FULLDATA TLV is used, the encoder MUST layout data for each element in the same order in which the data was defined in the LFB specification. This ensures the decoder is guaranteed to retrieve the data.

   * In the case of a SPARSEDATA, it does not need to be ordered since the "I" in the ILV uniquely identifies the element.

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 SPARSE 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 bit and a T is 16 bit.

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-REPLACE successfully by merely changing the T in the operational TLV.

- There are exceptions to this rule:

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

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

7.1.2. Protocol 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-CREATE
        |        |   |
        |        |   +--  // 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-REPLACE
        |        |
        |        |
        |        + -- T= DEL
        |        |
        |        + -- T= SET-REPLACE
        |
        |
        +--- T = LFBselect
                |
                +-- LFBCLASSID
                |
                +-- LFBInstance
                .
                .
                .
```

Figure 16: PL PDU logical layout

The figure below shows an example 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-CREATE
     |  |
     |  +- 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-REPLACE
     |  |
     |  +- 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 17: Sample operation layout

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 LFB can be found in Appendix B.

The FE Protocol LFB consists of the following elements:

o FE Protocol capabilities (read-only):

   * Supported ForCES protocol version(s) by the FE

   * Any TML capability description(s)

o FE Protocol attributes (can be read and set):

   * Current version of the ForCES protocol

   * FE unicast ID

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

   * 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 for

details. The CE HDI attribute as described below is tied to this policy. If the FE has not received any PL messages within a CE HDI period it declares the connectivity lost. The CE independently chooses the time interval for sending the Heartbeat messages to FE(s) - care must be exercised to ensure the CE->FE HB interval is smaller than the assigned CE HDI.

CE HDI SHOULD be at least 3 times as long as the HB interval. Shorter rates MAY be appropriate in implementations working across a reliable internal interface.

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.

* 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.

* 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.

  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.

  Others - Reserved.

* 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.

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

* Backup CEs - The list of backup CEs an FE is associated with. Refer to Section 9 for details.

* 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:

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

  1 - restart the FE from an intermediate state. In this case, the FE decides from which state it restarts. For example, if the FE is able to retain enough information of pre-association phase after some failure, it then has the ability to start from the post-association phase in this case.

  Others - Reserved

* CE failover policy - This specifies the behavior of the FE during a CE failure and restart time interval, or when the FE loses the CE association. It should be noted that this policy in the case of HA only takes effect after total failure to connect to a new CE. A timeout parameter, the CE Timeout Interval (CE TI) is associated with this attribute. Values of this policy are defined as below:

  0(default) - 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. Note that if the CE still has not been restarted or hasn't been associated back to the FE, after the CE TI has expired, the FE will go operationally down.

  1 - FE should go down to stop functioning immediately.

  2 - FE should go inactive to temporarily stop functioning. If the CE still has not been restarted after a time interval of specified by the CE TI, the FE will go down completely.

  Others - Reserved

* CE Timeout Interval (CE TI) - 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 CE TI

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.

7.2.2. 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 1 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 protocol 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->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 CE correctly. The Src ID (FE ID) may be set to 0 in the header which means that the FE would like the CE to assign an FE ID for the FE in the setup response message.

Message body:
The association setup message body optionally consists of one or more LFB select TLV as described in Section 7.1.1.1.5. The association setup message only operates toward the FE Object and FE Protocol LFBs, therefore, the LFB class ID in the LFB select TLV only points to these two kinds of LFBs.

The Operation TLV in the LFB select 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 like the Heart Beat Interval parameter, etc. The Operation TLV for event notification is is defined below.

Operation TLV for Association Setup:

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

Figure 18: Operation TLV

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 (eg type =  Association setup)
|
|
+--- T = LFBselect
|        |
|        +-- LFBCLASSID = FE object
|        |
|        |
|        +-- LFBInstance = 0x1
|        |
+--- T = LFBselect
|
+-- LFBCLASSID = FE Protocol object
|
|
+-- LFBInstance = 0x1
|
+-- Path-data to one or more attributes
including suggested HB parameters
```

Figure 19: PDU Format

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 Correlator field in the header MUST be the same as that of the corresponding association setup message, so that the association setup message sender can correlate the response correctly. The Dst ID in the header will be set to some FE ID value assigned by the CE if the FE had requested that in the setup message (by SrcID = 0).

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 20: Message Body

Type (16 bits):
The type of the TLV is "ASRresult".

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 = too many associations

3 = 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 21: 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 (the flag is set to 'NoACK' or 'AlwaysACK'), or to indicate under which conditions a response is generated (the flag is set to 'SuccessACK' or 'FailureACK'). The default behavior for the ACK flag is set to always expect a full response from FE. This happens when the ACK flag is not set to any defined value. The correlator field in the message header MUST be set if a response is expected, so that CE can correlate the response correctly. The correlator field can be ignored if no response is expected.

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

Operation TLV for Config:

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

Figure 22: Operation TLV for Config

Type:
The operation type for config message. two types of operations for the config message are defined:

Type = "SET" --- this operation is to set LFB attributes

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

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 operation is, 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 (eg 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 23: PDU Format

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). The Correlator field in the header MUST keep the same as that of the config message to be responded, so that the config message sender can correlate the response with the original message correctly.

Message body:

The config message body MUST consist of at least one LFB select TLV as described in Section 7.1.1.1.5. The Operation TLV in the LFB select TLV is defined below.

Operation TLV for Config Response:

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

Figure 24: Operation TLV for Config Response

Type:

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

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

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

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 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 CE can locate the response back from FE correctly.

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

Operation TLV for Query:

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

Figure 25: TLV for Query

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

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

PATH-DATA-TLV for GET:
 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 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 26: 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 consists 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 anymore. The Correlator field in the header MUST be the same as that of the associated query, so that the query message sender can keep track of the response.

Message body:
The query response message body MUST consist of at least one LFB select TLV as described in Section 7.1.1.1.5. The Operation TLV in the LFB select TLV is defined below.

Operation TLV for Query Response:

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

Figure 27: 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.

PATH-DATA-TLV for GET-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 CE. A config message is used by CE to subscribe/unsubscribe for an event in FE. To subscribe to an event is usually by specifying to the path of such an event as described by FE-Model and defined by LFB library.

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. The Correlator field in the header is also ignored because the response is not expected.

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

Operation TLV for Event Notification:

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

Figure 28: 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 29: PDU Format

7.8. Packet Redirect Message

Packet redirect message is used to transfer data packets between CE and FE. Usually these data packets are IP packets, though they may sometimes be associated with some metadata generated by other LFBs in the model. They may also occasionally be other protocol packets, which usually happens when CE and FE are jointly implementing some high-touch operations. Packets redirected from FE to CE are the data packets that come from forwarding plane, and usually are the data packets that need high-touch operations in CE,or packets for which the IP destination address is the NE. Packets redirected from CE to FE are the data packets that come from the CE and that the CE decides to put into forwarding plane, i.e. an FE.

Supplying such a redirect path between CE and FE actually leads to a possibility of this path being DoS attacked. Attackers may maliciously try to send huge spurious packets that will be redirected by FE to CE, resulting in the redirect path becoming congested. ForCES protocol and the TML layer will jointly supply approaches to prevent such DoS attack. To define a specific 'Packet Redirect Message' makes TML and CE able to distinguish the redirect messages from other ForCES protocol messages.

By properly configuring related LFBs in FE, a packet can also be mirrored to CE instead of purely redirected to CE, i.e., the packet

is duplicated and one is redirected to CE and the other continues its way in the LFB topology.

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'. The ACK flags in the header MUST be ignored, and no response is expected by this message. The correlator field is also ignored because no response is expected.

Message Body:
Consists of (at least) one or more than one TLV that describes packet redirection. 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          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          LFB Class ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        LFB Instance ID                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Meta Data TLV                          |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Redirect Data TLV                      |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 30: Redirect_Data TLV

LFB class ID:
There are only two possible LFB classes here, the 'RedirectSink' LFB or the 'RedirectSource' LFB[FE-MODEL]. If the message is from FE to CE, the LFB class should be 'RedirectSink'. If the message is from CE to FE, the LFB class should be 'RedirectSource'.

Instance ID:
Instance ID for the 'RedirectSink' LFB or 'RedirectSource' LFB.

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

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

Figure 31: Redirected_Data 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 32: Meta Data ILV

Where, Meta Data ID is an identifier for the meta data, which is statically assigned by the LFB definition. This actually implies a Meta Data ID transcoding mechanism may be necessary if a metadata traverses several LFBs while these LFBs define the metadata with different Meta Data IDs.

Usually there are two meta data that are necessary for CE-FE redirect operation. One is the redirected data type (e.g., IP packet, TCP packet, or UDP Packet). For an FE->CE redirect operation, redirected packet type meta data is usually a meta data specified by a Classifier LFB that filter out redirected packets from packet stream and sends the packets to Redirect Sink LFB. For an CE->FE redirect operation, the redirected packet type meta data is usually directly generated by CE.

Another meta data that should be associated with redirected data is the port number in a redirect LFB. For a RedirectSink LFB, the port number meta data tells CE from which port in the lFB the redirected data come. For a RedirectSource LFB, via the meta data, CE tells FE which port in the LFB the redirected data should go out.

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 33: Redirect Data TLV

Redirected Data:

This field presents the whole packet that is to be redirected. The packet should be 32bits aligned.

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.

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 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. 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 a 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 CEID 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.

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.

7.10. Operation Summary

The following table summarizes the TLVs that compose messages, and the applicabiity of operation TLVs to the messages.

Messages	TLVs	Operations
Association Setup	LFBselect	REPORT
Association Setup Response	ASRresult	None
Association Teardown	ASTreason	None
Config	LFBselect	SET, DEL
Config Response	LFBselect	SET-RESPONSE, DEL-RESPONSE
Query	LFBselect	GET
Query Response	LFBselect	GET-RESPONSE
Event Notification	LFBselect	REPORT
Packet Redirect	Redirect	None
Heartbeat	None	None

The following table summarises the applicability of the FULL/SPARSE DATA TLV and the RESULT TLV to the Operation TLVs.

Operations	FULLDATA TLV	SPARSEDATA TLV	RESULT TLV
SET	MAY	MAY	MUST NOT
SET-RESPONSE	MAY	MUST NOT	MUST
DEL	MAY	MAY	MUST NOT
DEL-RESPONSE	MAY	MUST NOT	MUST
GET	MUST NOT	MUST NOT	MUST NOT
GET-RESPONSE	MUST	MUST NOT	MAY
REPORT	MAY	MUST NOT	MUST NOT

8. Protocol Scenarios

8.1. 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 repsonds 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 below.

```
         FE PL                  CE PL

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

Figure 34: Message exchange between CE and FE to establish an NE association

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

8.2. 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 or synchronous heartbeat messages. This continues until a termination (or deactivation) is initiated by either the CE or FE. The figure 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 35: 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 messages 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 which would take place during failover. That is described in the HA section 8.

9. 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 draft. 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.

Two HA modes are defined in the ForCES protocol, Report Primary Mode and Report All Mode. The Report Primary Mode is the default mode of the protocol, in which the FEs only associate with one CE (primary) at a time. The Report All mode is for future study and not part of the current protocol version. In this mode, the FE would establish association with multiple CEs (primary and secondary) and report events, packets, Heart Beats to all the CEs. However, only the primary CE would configure/control the FE in this mode as well. This would help with keeping state between CEs synchronized, although it would not guarantee synchronization.

The HA Modes are configured during Association setup phase, though currently only Report Primary Mode can be configured. A CE-to-CE synchronization protocol would be needed to support fast failover as well as 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.

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. In the Report Primary mode, as there should be no other existing CE-FE associations, the FE PL MUST at this point establish association with the secondary CE. Once the process has started, if the original primary CE comes alive and starts sending commands message to the FE, the FE MUST ignore those messages. If the original CE begins a new association phase with the FE then the FE MUST send an Association Setup Response message with Result = 2 indicating that there are too many associations. It will be up to CE-CE communications, out of scope for this specification, to determine what what, if any changes should be made to FE

configuration following the recovery process.

An explicit message (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 CEID, i.e. they are 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 CEID that was lost, then the FE will need to form a new association using a different CEID. If the capability exists, the FE MAY first attempt to form a new association with original primary CE using a different non multicast CEID.

These two scenarios, Report Primary (default), Report Primary (currently unsupported), are illustrated in the Figure 36 and Figure 37 below.

```
         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 36: CE Failover for Report Primary Mode

```
      FE                   CE Primary         CE Secondary
       |                       |                     |
       | Asso Estb,Caps exchg  |                     |
     1 |<--------------------->|                     |
       |                       |                     |
       |         Asso Estb,Caps|exchange             |
     2 |<----------------------|-------------------->|
       |                       |                     |
       |     All msgs          |                     |
     3 |<--------------------->|                     |
       |                       |                     |
       |    packet redirection,|events, HBs          |
     4 |-----------------------|-------------------->|
       |                       |                     |
       |                    FAILURE                  |
       |                                             |
       |             Event Report (pri CE down)      |
     5 |-------------------------------------------->|
       |                                             |
       |                  All Msgs                   |
     6 |<------------------------------------------->|
```

Figure 37: CE Failover for Report All mode

9.1. Responsibilities for HA

TML level - Transport 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:
For all other functionality including configuring the HA behavior during setup, the CEIDs are used to identify primary, secondary CEs, protocol Messages used to report CE failure (Event Report), Heartbeat messages used to detect association failure, messages to change primary CE (config - move), 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 will take the appropriate actions described before.

10. Security Considerations

ForCES architecture identifies several levels of security in [RFC3746]. ForCES PL uses security services provided by the ForCES TML layer. TML layer provides security services such as endpoint authentication service, message authentication service and confidentiality service. Endpoint authentication service is invoked at the time of pre-association connection establishment phase and message authentication is performed whenever FE or CE receives a packet from its peer.

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

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

- o Operator should configure the same security policies for both primary and backup FE's and CE's (if available). This will ensure uniform operations, and to avoid unnecessary complexity in policy configuration.

- o ForCES PL endpoints SHOULD pre-established connections with both primary and backup CE's. This will reduce the security messages and enable rapid switchover operations for HA.

10.1. 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 layer. Both these mechanism are weak and does not involve cryptographic operation. Operator can choose "No security" level when the ForCES protocol endpoints are within a single box.

In order to have interoperable and uniform implementation across various security levels, each CE and FE endpoint MUST implement this level. The operations that are being performed for "No security" level is required even if lower TML security services are being used.

10.1.1. Endpoint Authentication

Each CE and FE PL layer maintains set of associations list as part of configuration. This is done via CEM and FEM interfaces. FE MUST connect to only those CE's that are configured via FEM similarly, a CE should accept the connection and establish associations for the

FE's which are configured via CEM. CE should validate the FE identifier before accepting the connection during the pre-association phase.

10.1.2. Message authentication

When CE or FE generates 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 if the underlying TLM layer security services.

10.2. ForCES PL and TML security service

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

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

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

10.2.1. Endpoint authentication service

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

It is recommended that an FE, after establishing the connection with the primary CE, should establish the security association with the backup CE (if available). During the switchover operation CE's security state associated with each SA's are not transferred. SA between primary CE and FE and backup CE and FE are treated as two separate SA's.

10.2.2. Message authentication service

This is TML specific operation and is transparent to ForCES PL layer.

For details refer to Section 5.

10.2.3. Confidentiality service

This is TML specific operation and is transparent to ForCES PL layer. For details refer to Section 5.

11. 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.

12. References

12.1. 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.

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

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

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

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

12.2. 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.

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 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
- o Configuration Request: Operation Result Section 7.5.1

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 references.

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 Type

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

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

0x0000	Reserved
0x0001	SET
0x0002	SET-RESPONSE
0x0003	DEL
0x0004	DEL-RESPONSE
0x0005	GET
0x0006	GET-RESPONSE
0x0007	REPORT

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

Operation Type 0x8000-0xFFFF
Operation Types in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the Operation 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	MAIN_TLV
0x0002	REDIRECT-TLV
0x0010	ASResult-TLV
0x0011	ASTreason-TLV
0x1000	LFBselect-TLV
0x0101	OPER-TLV
0x0110	PATH-DATA-TLV
0x0111	KEYINFO-TLV
0x0112	FULLDATA-TLV
0x0113	SPARSEDATA-TLV
0x0114	RESULT-TLV

TLV Type 0x0200-0x7FFF
TLV Types using this range must be documented in an RFC or other permanent and readily available references. [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. LFB Class Id Name Space

The LFB Class ID name space is 32 bits long. The following is the guideline for managing the TLV Result Name Space.

LFB Class ID 0x00000000-0x0000FFFF
LFB Class IDs in this range are allocated through an IETF consensus process. [RFC2434].
Values assigned by this specification:

0x00000000	Reserved
0x00000001	FE Protocol LFB
0x00000002	FE Object LFB

LFB Class ID 0x00010000-0x7FFFFFFF
LFB Class IDs in this range are Specification Required [RFC2434] LFB Class ID using this range must be documented in an RFC or other permanent and readily available references. [RFC2434].

LFB Class Id 0x80000000-0xFFFFFFFFF
LFB Class IDs in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the LFB Class ID Space is unnecessary.

A.6. Association Setup Response

The Association Setup Response name space is 16 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   Too many associations
0x0003   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 references. [RFC2434].

Association Setup Response 0x80000000-0xFFFFFFFFF
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 TLV Result 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  - Out of Memory
0x00000002        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].

LFB Class Id 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.

A.8. Configuration Request Result

The Configuration Request name space is 32 bits long. The following is the guideline for managing the Configuration Request Name Space.

Configuration Request 0x0000-0x00FF
Configuration Requests in this range are allocated through an IETF consensus process. [RFC2434].
Values assigned by this specification:

```
0x0000      Success
0x0001      FE ID Invalid
0x0003      Permission Denied
```

Configuration Request 0x0100-0x7FFF
Configuration Requests in this range are Specification Required [RFC2434] Configuration Requests using this range must be documented in an RFC or other permanent and readily available references. [RFC2434].

0x8000-0xFFFF
Configuration Requests in this range are reserved for vendor private extensions and are the responsibility of individual vendors. IANA management of this range of the Configuration Request 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]

```
<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
     file:/home/hadi/xmlj1/lfbmodel.xsd" 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, refer to
                    <xref target="FPL_sum" /> for details
                </synopsis>
                </specialValue>
              <specialValue value="1">
                 <name>CEHBPolicy1</name>
                 <synopsis>
                     The CE heartbeat policy 1, refer to
                     <xref target="FPL_sum" /> for details
                 </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, refer to
                    <xref target="FPL_sum" /> for details
                  </synopsis>
                </specialValue>
                <specialValue value="1">
                   <name>FEHBPolicy1</name>
                   <synopsis>
                      The FE heartbeat policy 1, refer to
                      <xref target="FPL_sum" /> for details
                   </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, refer to
                      <xref target="FPL_sum" /> for details
                   </synopsis>
                   </specialValue>
                   <specialValue value="1">
                     <name>FERestartPolicy1</name>
                     <synopsis>
                      The FE restart policy 1, refer to
                      <xref target="FPL_sum" /> for details
                     </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, refer to
                      <xref target="FPL_sum" /> for details
                   </synopsis>
                 </specialValue>
               <specialValue value="1">
                  <name>CEFailoverPolicy1</name>
                  <synopsis>
                      The CE failover policy 1, refer to
                      <xref target="FPL_sum" /> for details
                  </synopsis>
                </specialValue>
                <specialValue value="2">
                   <name>CEFailoverPolicy2</name>
                   <synopsis>
                      The CE failover policy 2, refer to
                      <xref target="FPL_sum" /> for details
                   </synopsis>
                  </specialValue>
               </specialValues>
               </atomic>
         </dataTypeDef>
     </dataTypeDefs>

     <LFBClassDefs>
       <LFBClassDef LFBClassID="2">
         <name>FEPO</name>
         <id>1</id>
         <synopsis>
            The FE Protocol Object
         </synopsis>
         <version>1.0</version>
         <derivedFrom>baseclass</derivedFrom>

     <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>FERestartPolicy</name>
             <synopsis>
                The FE Restart Policy
             </synopsis>
             <typeRef>FERestartPolicyValues</typeRef>
           </attribute>

           <attribute elementID="11" access="read-write">
             <name>CEFailoverPolicy</name>
             <synopsis>
               The CE Failover Policy
             </synopsis>
     <typeRef>CEFailoverPolicyValues</typeRef>
           </attribute>

           <attribute elementID="12" access="read-write">
             <name>CETI</name>
             <synopsis>
               The CE Timeout Interval in millisecs
             </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>
         </capabilities>
       </LFBClassDef>
     </LFBClassDefs>
   </LFBLibrary>
```

B.1. Capabilities

At the moment only the SupportableVersions capability is owned by this LFB.

Supportable Versions enumerates all ForCES versions that an FE supports.

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.

```
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 show an attribute suffixed with "v" or "val" to

indicate the value of the referenced attribute. example for attribute foo2, foo1v or foo1value will indicate the value of foo1. 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 =  foo1v
```

2. To set foo2 to 10

```
OPER = SET-REPLACE-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, j1value,j2value  entries
                        representing the entire table
```

Note: One should be able to take a GET-RESPONSE-TLV and convert it to a SET-REPLACE-TLV. If the result in the above example is sent back in a SET-REPLACE-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-CREATE-TLV
        Path-data-TLV:
                flags = 0 , IDCount = 1, IDs=4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 0
                    FULLDATA-TLV containing j1, j2 value for entry 0
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    FULLDATA-TLV containing j1, j2 value for entry 1
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    FULLDATA-TLV containing j1, j2 value for entry 2
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    FULLDATA-TLV containing j1, j2 value for entry 3
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                    FULLDATA-TLV containing j1, j2 value for entry 4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5
                    FULLDATA-TLV containing j1, j2 value for 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-REPLACE-TLV
           Path-data TLV:
                   flags =  0 , IDCount = 1, IDs=4
                   PATH-DATA-TLV
                       flags = 0, IDCount = 1, IDs = 0
                       FULLDATA-TLV containing j1, j2 value for entry 0
                   PATH-DATA-TLV
                       flags = 0, IDCount = 1, IDs = 2
                       FULLDATA-TLV containing j1, j2 value for entry 2

   Result:
   OPER = SET-REPLACE-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, Length = XXX, V =
                        j1value,j2value entry
```

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 j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    FULLDATA-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    FULLDATA-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    FULLDATA-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                    FULLDATA-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5
                    FULLDATA-TLV containing j1value j2value
```

8. Create a row in table2, index 5.

```
OPER = SET-CREATE-TLV
        Path-data-TLV:
                flags = 0, IDCount = 2, IDs=4.5
                FULLDATA TLV, Length = XXX
                        j1value,j2value

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-CREATE-TLV
        Path-data -TLV:
                flags = FIND-EMPTY, IDCount = 1, IDs=4
                FULLDATA TLV, Length = XXX
                        j1value,j2value

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, Length = XXX j1value,j2value
```

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, t1value, t2value
                        index, t1value, t2value
                        .
```

```
                           .
                           .
```

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, Length = XXXX
                        j1value,j2value, j3value, j4value
```

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 = namev
                index, someidv, TLV: T=FULLDATA, L = 4+strlen(namev),
                       V = namev
                index, someidv, TLV: T=FULLDATA, L = 4+strlen(namev),
                       V = namev
                index, someidv, TLV: T=FULLDATA, L = 4+strlen(namev),
                       V = namev
                .
                .
                .
```

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-CREATE-TLV
        Path-TLV:
                flags = 0, IDCount = 2,  IDs=4.20
                FULLDATA TLV, V= j1value,j2value

// Operation 2: Update table1 entry which
// was pointing with t2 = 10 to now point to 20
OPER = SET-REPLACE-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 = j1value,j2value
// second opertion 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
// second opertion SET
OPER = SET-RESPONSE-TLV
        Path-data TLV
                flags = 0 IDCount = 1, IDs=3
                KEYINFO = KEYID=1 KEY_DATA=10
                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-REPLACE-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 = {x1 value}
```

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 = {x2 value}
```

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-REPLACE-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-REPLACE-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-REPLACE-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@acm.org

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

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.