ForCES Protocol Specification
draft-ietf-forces-protocol-02.txt

Status of this Memo

Copyright Notice

Abstract

This specification documents the Forwarding and Control Element Separation protocol. This protocol is designed to be used between a Control Element and a Forwarding Element in a Routing Network Element.

Authors

The participants in the ForCES Protocol Team, co-authors and co-editors, of this draft, 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. Introduction

This specification provides a draft definition of an IP-based protocol for Control Element control of an Forwarding Element. The protocol is a TLV based protocol that include commands for transport of LFB information as well as TLVs for association, configuration, status, and events.

This specification does not specify a transport mechanism for messages, but does include a discussion of the services that must be provided by the transport interface.

1.1 Sections of this document

Section 2 provides a glossary of terminology used in the specification.

Section 3 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 4 details the services that the TML must provide.

The Forces protocol is defined to have a common header for all other message types. The header is defined in Section 5.1, while the protocol messages are defined in Section 6.

Section 7 describes several Protocol Scenarios and includes message exchange descriptions.

Section 8 describes mechanism in the protocol to support high availability mechanisms including redundancy and fail over.
Section 9 defines the security mechanisms provided by the PL and TML.

2. Definitions

This document follows the terminology defined by the ForCES Requirements in [RFC3654] and by the ForCES framework in [RFC3746]. This document also uses the terminology defined by ForCES FE model in [FE-MODEL]. We copy the definitions of some of the terminology as indicated below:

Addressable Entity (AE) - A physical device that is directly addressable given some interconnect technology. For example, on IP networks, it is a device to which we can communicate using an IP address; and on a switch fabric, it is a device to which we can communicate 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 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 a FE Manager (see below) and a CE Manager (see below) are determining which FE and CE should be part of the same network element.

Post-association Phase - The period of time during which a FE does know which CE is to control it and vice versa, including 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 a FE.

FE Manager (FEM) - A logical entity that operates in the pre-association phase and is responsible for determining to which CE(s) a FE should communicate. This process is called CE discovery and may involve the FE manager learning the capabilities of available CEs. A 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, a FE manager might be physically combined with any of the other logical entities such as FEs.

CE Manager (CEM) - A logical entity that operates in the pre-association phase and is responsible for determining to 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. A CE manager may use anything from a static configuration to a pre-association phase protocol (see below) to determine which FE to use. Being a logical entity, a CE manager might be physically combined with any of the other logical entities such as CEs.

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) type -- A template representing a fine-grained, logically separable and well-defined packet processing operation in the datapath. LFB types are the basic building blocks of the FE model.

LFB (Logical Function Block) Instance -- As a packet flows through an FE along a datapath, it flows through one or multiple LFB instances, with each implementing an instance of a certain LFB type. There may be multiple instances of the same LFB in an FE's datapath. Note that we often refer to LFBs without distinguishing between LFB type and LFB instance when we believe the implied reference is obvious for the given context.

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, 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.

Following terminologies are defined by this document:

ForCES Protocol - While there may be multiple protocols used within the overall ForCES architecture, the term "ForCES protocol" refers only to the protocol used at the Fp reference point in the ForCES Framework in RFC3746 [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 specifically addresses the 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 is specifically addressed in a separate ForCES TML Specification document.

3. 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.

3.1 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 is out of scope of the ForCES protocol but is touched on in this document 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 a MPLS QoS enabled Network Element.

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

Figure 2: Examples of CE and FE functions

The ForCES Interface shown in Figure 2 constitutes two pieces: the PL and 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).

3.1.1 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 throw various subscribed-to events to the CE PL layer as well as respond to various status requests issued from the CE PL. The CE configures both the FE and associated LFBs attributes using the PL layer. In addition the CE may send various requests to the FE to activate or deactivate it, reconfigure its HA parametrization, subscribe to specific events etc. More details in Section 6.

3.1.2 The TML layer

The TML layer is essentially responsible for transport of 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 different TMLs each could implement things differently based on 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 should be portable across all TMLs, because all TMLs have the same top edge semantics as defined in this document.

3.1.3 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 which they use as the FEM/CEM interface. RFC 3746 has a lot more detailed descriptions on how the FEM and CEM could be used. We discuss the pre-association phase where the CEM and FEM play briefly in section Section 3.2.1.

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

a. how the TML connection should happen (example what IP addresses to use, transport modes etc);
b. the ID for the FE or CE would also be issued at this point.
c. Security parameterization such as keys etc.
d. Connection association parameters

Example "send up to 3 association messages each 1 second apart" Vs " send up to 4 association messages with increasing exponential timeout".

3.2 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.

3.2.1 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 config 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 config parameters instead of reading them from a static config 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.

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 4: 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 5: 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 the appropriate CEM component. Initialization of the ForCES interface will be completed, and authentication as well as capability discovery may be complete as well. Both the FE and CE would have the necessary information for connecting to each other for configuration, accounting, identification and authentication purposes. Both sides also would 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 more detailed discussion.

3.2.2 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 state
- o Established State
- o Association teardown state.

3.2.2.1 Association setup state

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 the protocol scenarios section.

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

3.2.2.2 Association Established state

In this state the FE is continuously updated or queried. The FE may also send asynchronous event notifications to the CE or synchronous heartbeat notifications. This continues until a termination is initiated by either the CE or the FE.

Refer to section on protocol scenarios Section 7 for more details.

3.3 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.

3.3.1 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 back the results.

This section details the different modes of execution that a CE can order the FE(s) to perform in Section 3.3.1.1. It also describes the different modes a CE can ask the FE(s) to format the responses back after processing the operations requested in Section 3.3.1.2.

3.3.1.1 Execution

There are two schools of thoughts which are being tossed around at the moment on Forces operations from the CE to the FE. We try to support both and leave the details to implementation (the intelligence is at the CE).

1. The CE knows exact details about the resources at the FE (memory, table sizes, etc). When it says "SET" it knows that would work etc. In this scenario, it is contended by the proponents of this optimization that when a CE decides to send a command to an FE it will work 100%. In other words, remote operations without any transactional properties (given the CE computes the success of

the tranaaction).

2. The classical Two-phase-commit(2PC)[REference to be added here] scheme; undo/rollback when things go wrong. undoing is initiated by the CE after an error response from any one FE being synchronized is received.

There are 3 execution modes that could be requested for operations spanning on one or more LFB selectors:

Transactional all-or-none.
Any-of-N independent operations.
go-to-N loose transaction.

3.3.1.1.1 Requirements for ForCES Transactions

ForCES transactions MUST support ACIDity [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 its state before the transaction was started.
- *Isolation*. A transaction in process and not yet committed must remain isolated from any other transaction.
- *Durability*. Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state.

3.3.1.1.1.1 Transaction definition

We define a transaction as a collection of one or more ForCES operations within one or more messages MUST have ACID properties.

3.3.1.1.1.2 Transaction protocol

A 2PC starts with a START | ATOMIC flag on its first message of a transaction. A transaction may span multiple messages. It is up to the CE to keep track of the different seq #s making up a transaction. This may then be followed by more messages which are part of the same atomic transaction.

Any failure notified by the FE causes the CE to execute an ABORT to all FEs involved in the transaction, rolling back all previously executed operations in the transaction.

The transaction commitment phase is signalled by an empty DONE msg type.

TBD: We may need an ABORT message(used for rollback purposes) or maybe a DONE with an ABORT flag to undo will suffice.

3.3.1.1.1.3 Recovery

Any of the participating FEs, or the CE, or the associations between them, may fail after the DONE message has left the CE and before it has received all the responses, (possibly the DONE never reached the FEs). At this point it is known that none of the operations failed but it is presumed that the data has not yet been made durable by the FEs. The means of detecting such failures may include loss of heartbeat (within the scope of ForCES) or mechanisms outside the scope of ForCES. When the associations are re-established, the CE will discover a transaction in an intermediate state. Some FEs will have made the data durable and closed the transaction; others may have failed while doing so, and may, or may not, still have that data. At this point the transaction enters the recovery phase.

The CE re-issues an empty DONE message to all FEs involved in the transaction. Those that completed the transaction confirm this to the CE. Those that did not, commit the data and confirm this to the CE. An FE that has lost all records of the transaction MUST reply with status UNKNOWN and the actions subsequently taken by the CE are implementation dependent.

This requires knowledge at both the CE and FE; not sure how to signal it. Global flags: ATOMIC, START, ABORT? (used by 2PC) New msg type: DONE, ABORT?(used by 2PC)

3.3.1.1.2 one-of-N independent operations

In which several independent operations are targeted at one or more LFB selectors. Execution continues at the FE when one or more operations fail. This mode is signalled by a missing ATOMIC flag.

3.3.1.1.3 go-to-N loose transaction

In which all operations are executed on FE sequentially until first failure. The rest of the operations are not executed but everything upto failed is not undone unlike #1. flag: GOTON (global)

3.3.1.1.4 Relation to Multipart messages

Multipart flags apply. I.e all messages in a transaction except for the last have a MULTIPART flag on.
There has to be consistency across the multi parts of the messages. In other words the first message starting with mode #1 above, implies the rest do. Any inconsistency implies a cancelled transaction in which all messages are dropped and the sender

NACKED.

3.3.1.2 Response formating

Editorial Note: This is still under discussion and ties into the RESULT TLV discussed in Section 6.1.

3.3.2 FE, CE, and FE protocol LFBs

All PL messages follow the LFB structure as this provides more flexibility for future enhancements. This means maintanance and configurability of FEs, NE, as well as the protocol require a fit in the LFB architecture. For this reason special LFBs are created to accomodate 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 LFBs are used:

- o FE Protocol LFB
- o FE LFB
- o CE LFB

These LFBs are detailed in Section 6.2. A short description is provided here:

- o The FE Protocol LFB is a logical entity in each FE that is used to control the ForCES protocol. The CE operates on this LFB to subscribe or unsubscribe to Heartbeat messages, define the Heartbeat interval, or to discover which ForCES protocol version is supported and which TMLs the FE supports. The FE Protocol LFB also contains the various ForCES ID to be used: unicast IDs a table of the PL multicast IDs the FE must be listening to. [TBD: do we need a CE Protocol LFB?]
- o The FE LFB (referred to as "FE attributes" in the model draft) should not be confused with the FE Protocol Object. The FE 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 between the CE and the FE. Such attributes can be FEState (refer to model draft), vendor, etc. The FE LFB contains in particular a table that maps a virtual LFB Instance ID to one or more Instance IDs of LFBs in the FE.
- o The CE LFB is the counterpart of the FE LFB. The CE LFB is a logical entity in each CE and contains attributes relative to the CE itself, and not to the operation of the ForCES protocol between the CE and the FE. This LFB can be used to convey event notifications from a CE to FEs. Some events may be sent by the CE without prior subscription by the FEs.

3.3.3 Scaling by Concurrency

It is desirable that the PL layer not become the bottleneck when larger bandwidth pipes become available. To pick a mythical 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 are 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 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 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.

3.3.3.1 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 can be addressed within one PL PDU
- o Multiple operations can be addressed to a single LFB class and instance

3.3.3.2 Command Pipelining

TBD

4. 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 inter-TML 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. 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. Timeliness

   Editorial Note: Does the TML allow for obsoleting msgs? If yes, it needs to define how.
6. 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 the HA Section Section 8 for details.
7. Encapsulations used.
   Different types of TMLs will encapsulate the PL messages on different types of headers. The TML needs to specify the encapsulation used.
8. 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.
   TML needs to define how this is achieved.
9. Protection against DoS attacks
   As described in the Requirements RFC 3654, section 6

4.1 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:

- o PL ID
- o 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.
- o Number of transport connections
- o Connection Capability, such as bandwidth, etc.
- o Allowed/Supported Connection QoS policy (or Congestion Control Policy)

5. Message encapsulation

All PL layer PDUs start with a common header [Section 5.1] followed by a one or more TLVs [Section 5.2] which may nest other TLVs [Section 5.2.1].

5.1 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
                 0               1               2             3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|version| rsvd  | Message Type  |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Source ID                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Destination ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Sequence Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Flags                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: 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 SHOULD set it to zero.
Command (8 bits):
    Commands are defined in Section 6.
Source ID (32 bit):
Dest ID (32 bit):

* Each of the source and Dest IDs are 32 bit IDs which recognize the termination points. Ideas discussed so far are desire to recognize if ID belongs to FE or CE by inspection. Suggestions for achieving this involves partitioning of the ID allocation. Another alternative maybe to use flags to indicate direction (this avoids partition).
* IDs will allow multi/broad/unicast
* Addressing
    a. As ForCES may run between multiple CEs and FEs and over different protocols such as IPv4 and IPv6, or directly over Ethernet or other switching-fabric interconnects, it is necessary to create an addressing scheme for ForCES

entities. Mappings to the underlying TML-level addressing can then be defined as appropriate.

b. Fundamentally, unique IDs are assigned to CEs and FEs. A split address space is used to distinguish FEs from CEs. Even though we can assume that 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.

c. Special IDs are reserved for FE broadcast, CE broadcast, and NE broadcast.

d. Subgroups of FEs belonging, for instance, to the same VPN, may be assigned a multicast ID. Likewise, subgroups of CEs that act, for instance, in a back-up mode may be assigned a multicast ID. These FEs and CE multicast IDs are chosen in a distinct portion of the ID address space. Such a multicast ID may comprise FEs, CEs, or a mix of both.

e. As a result, 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
 0               1               2               3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|TS |                           sub-ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: ForCES ID Format

f. The ForCES ID is 32 bits. The 2 most significant bits called Type Switch (TS) are used to split the ID space as follows:

```
A.  TS    Corresponding ID range       Assignment
B.  --    ----------------------       ----------
C.  0b00   0x00000000 to 0x3FFFFFFF     FE IDs (2^30)
D.  0b01   0x40000000 to 0x7FFFFFFF     CE IDs (2^30)
E.  0b10   0x80000000 to 0xBFFFFFFF     reserved
F.  0b11   0xC0000000 to 0xFFFFFFEF     multicast IDs
    (2^30 - 16)
G.  0b11   0xFFFFFFF0 to 0xFFFFFFFC     reserved
H.  0b11   0xFFFFFFFD                   all CEs broadcast
I.  0b11   0xFFFFFFFE                   all FEs broadcast
J.  0b11   0xFFFFFFFF                   all FEs and CEs
    (NE) broadcast
```

g. It is desirable to address multicast and/or broadcast messages to some LFB instances of a given class. For instance, assume FEs FEa and FEb:

- FEa has LFBs LFBaX1 and LFBaX2 of class X

- similarly, FEb has two LFBs LFBbX1 and LFBbX2 of class X.

A broadcast message should be addressable to only LFBs LFBaX1 and LFBbX1 (this can be the case for instance if these two LFBs belong to the same VPN). To achieve this, a VPN ID (3 octets OUI and 4 octets VPN Index) as defined in RFC 2685 should be used within the ForCES message body as a TLV.

As an alternative, a particular multicast ID MAY be associated to a given VPN ID through some configuration means. Messages delivered to such a multicast ID MUST only be applied to LFBs belonging to that VPN ID.

Sequence (32 bits)
Unique to a PDU. [Discussion: There may be impact on the effect of subsequence numbers].

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

Flags(32 bits):
Identified so far:

- ACK indicator(2 bit)
  The description for using the two bits is:
  'NoACK' (00)
  'SuccessACK'(01)
  'UnsuccessACK'(10)
  'ACKAll' (11)
- Priority (3 bits)
  TBD
- Throttle flag
- Batch (2 bits)
- Atomicity (1 or more bits. TBD)

Editorial Note: There are several open issues, listed below, in the header which still need to be settled:

1. Parallelization of PL Windowing/subsequence
   Someone to look into ISCSI
2. events and replies and relation to peer to peer vs master slave
3. We need to discuss whether some of the Flags such as those for Atomicity, Batching are needed in the common header or only belong to the PATH flags.

5.2 Type Length Value

```
 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)                   |
 ~                                                               ~
 ~                                                               ~
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TLV Type:

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

TLV Length:

The TLV Length field is two octets, and indicates the length of this TLV including the TLV Type, TLV Length, and the TLV data.

TLV Value:

The TLV Value field carries the data. For extensibility, the TLV Value may be a TLV. In fact, this is the case with the Netlink2-extension TLV. The Value encapsulated within a TLV is dependent of the attribute being configured and is opaque to Netlink2 and therefore is not restricted to any particular type (example could be ascii strings such as XML, or OIDs etc).

TLVs must be 32 bit aligned.

Figure 8: TLV

5.2.1 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 easy mapping between the XML LFB definitions to binary PL representation.

5.2.2 Scope of the T in TLV

The "Type" value in TLV is of global scope. This means that wherever

in the PDU hierachy a Type has global connotations. This is a design choice to ease debugging of the protocol.

6. Protocol Construction

6.1 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 hierachical in nature, it is easier at times to see the visualization layout. This is provided in Section 6.1.2

6.1.1 Protocol BNF

The format used is based on RFC 2234. The terminals of this gramar are flags, IDcount, IDs, KEYID, KEY_DATA and DATARAW, described after the grammar.

1. A TLV will have the word "TLV" at the end of its name
2. / is used to separate alternatives
3. parenthesised elements are treated as a single item
4. * before an item indicates 0 or more repetitions 1* before an item indicates 1 or more repetitions
5. [] 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 1*LFBselect-TLV
     LFBselec-TLV    :=   LFBCLASSID LFBInstance 1*OPER-TLV
     OPER-TLV : = 1*PATH-DATA-TLV
     PATH-DATA-TLV := PATH  [DATA]
     PATH := flags IDcount IDs [SELECTOR]
     SELECTOR :=  KEYINFO-TLV
     DATA := DATARAW-TLV / RESULT-TLV / 1*PATH-DATA-TLV
     KEYINFO-TLV := KEYID KEY_DATA
     DATARAW-TLV := encoded data which may nest DATARAW TLVs
     RESULT-TLV := not yet defined.  Holding result code and optional
DATARAW
```

- 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 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 OPERATION is one of {ADD,DEL,etc.} depending on the message type [Editorial note: List all known operations here]

- o PATH-DATA-TLV identifies the exact element targeted. It may have zero or more paths associated with it terminated by zero or more data values associated.
  - * Note: SELECTOR is retained in this grammar to allow for future extensibility.
  - * NOTE! NOTE! There is a selector known as BLKINFO that is left out of this doc for now that will be revisted later. BLKINFO is defined as:
  - * SELECTOR := KEYINFO-TLV / BLOCK-SELECTION-TLV BLKINFO TLV := [BLK_RANGE_INDEX] / [BLK_LIST_INDEX] / [BLK_CNT_INDEX] A PathData can only have one selector either a KEYINFO TLV or a BLKINFO TLV (i.e not both).
- 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.
  - * KEY_DATA is the data to look for in the array, in the fields identified by the keyfield. The information is encoded according to the rules for the contents of a DATARAW, and represent the field or fields which make up the key identified by the KEYID.
- o DATA may contain a DATARAW or 1 or more further PATH-DATA selection DATARAW is only allowed on SET requests, or on responses which return content information (GET Response for example.) PATH-DATA may be included to extent the path on any request.
  - * Note: Nested PATH-DATA TLVs are supported as an efficiency measure to permit common subexpression extraction.
  - * DATARAW contains "the data" whose path is selected.
- o RESULT contains the indication of whether the individual SET succeeded. If there is an indication for verbose response, then SETRESULT will also contain the DATARAW 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. Note: This is one of several ways of handling set results. This is still

being discussed.

In summary this approach has the following characteristic:

- o There can be one or more LFB Class + InstanceId combo targeted in a message (batch)
- o There can one or more operations on an addressed LFB classid+instanceid combo(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.

6.1.1.1 Discussion on Grammar

Data is packed in such a way that a receiver of such data with knowledge of the path can correlate what it means by infering 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.

OPEN ISSUE: There is another view of how DATA should be represented posted a while back by Zsolt/Steve. We need to review it and compare against the scheme described here. The criteria is:

- o efficiency of encoding
- o efficiency of parsing and decoding
- o the packaging overhead.

6.1.1.1.1 Data Packing Rules

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

- o The Value of DATARAW TLV will contain the data being transported. This data will be as was described in the LFB definition.
- o By definition in the Forces protocol, all TLVs are 32 bit aligned. Therefore because DATARAW is a TLV, elements not aligned in 32 bit values will be padded.
- o As an example a 16 bit value will have an extra 16 bit pad; however two 16 bits values in a structure will be shipped together with no padding etc.

- o Variable sized data will be encapsulated inside another DATARAW TLV inside the V of the outer TLV. For example of this see Appendix D example 13.
- o When a table is refered in the PATH (ids), then the RAWDATA's V will contain that tables row content prefixed by its 32 bit index/subscript OTOH, when SELECTOR flags are 00, the PATH may contain an index pointing to a row in table; in such a case, the RAWDATA's V will only contain the content sans the index in order to avoid ambiguity.

6.1.1.1.2 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.

6.1.1.1.3 Relation of operational flags with global message flags

Should be noted that other applicable flags such as atomicity indicators as well as verbosity result formaters are in the main headers flags area.

6.1.1.1.4 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.

6.1.1.1.5 Operation TLVs

It is assumed that specific operations are identified by the type code of the TLV. And that response are also identified by specific TLV opcodes

6.1.1.1.6 SET and GET Relationship

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

- o it would have exactly the same path definitions as that was sent in the GET. The only difference being a GET-RESPONSE will contain DATARAW TLVs.

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

o There are exceptions to this rule:

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

   2. When dumping a whole table in a GET, the GET-RESPONSE, merely editing the T to be SET will endup overwritting the table.

6.1.2 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 10: PL PDU layout

The figure below shows an example general layout of the operation within a targetted 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 = DATARAW
        |                  + -- data
```

```
        |
        |
        T = SET-REPLACE
        |  |
        |  +- T = Path-data
        |  |  |
        |  |  + -- flags
        |  |  + -- IDCount
        |  |  + -- IDs
        |  |  |
        |  |  + -- T = DATARAW
        |  |          + -- data
        |  +- T = Path-data
        |     |
        |     + -- flags
        |     + -- IDCount
        |     + -- IDs
        |     |
        |     + -- T = DATARAW
        |             + -- 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 11: Sample operation layout

6.2 Core ForCES LFBs

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

FE protocol LFB
FE LFB
CE 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.

Editorial Note: The CE LFB is under discussion still and may end up being removed.

6.2.1 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 LFB Instance ID is assigned the value 0x1. There must always 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 the operation of the protocol.

The formal definition of the FE Protocol LFB can be found in Appendix C

The FE Protocol LFB consists of the following elements:

- o FE Protocol events that can be subscribed/unsubscribed:
  * FE heartbeat
  * FE TML events (TBD)
- o FE Protocol capabilities (read-only):
  * Supported ForCES protocol version(s) by the FE
  * Supported ForCES FE model(s) by the FE

   * Some TML capability description(s)
- FE Protocol attributes (can be read and set):
   * Current version of the ForCES protocol
   * Current version of the FE model
   * FE unicast ID
   * FE multicast ID(s) (list)
   * Association Expiry Timer
   * Heartbeat Interval
   * Primary CE
   * FE failover and restart policy
   * CE failover and restart policy

     Note: Is there a difference between the CE and FE failover policies?

     TBD: Define default values for each attribute where applicable.

6.2.2 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 FE LFB Class ID is assigned the value 0x2. The FE LFB Instance ID is assigned the value 0x1. There must always be one and only one instance of the FE LFB in an FE.

The formal definition of the FE Object LFB can be found in [FE-MODEL]

The FE LFB consists of the following elements:

FE Events:

- FEAllEvents: subscribing to this corresponds to subscribing to all events below
- FEStatusChange: events that signal FE Status:
  + Up
  + Down
  + Active
  + Inactive
  + Failover
- FE DoS alert
- FE capability change

FE attributes:

- FEStatus: to set the FE mode as:
  + Active
  + Inactive
  + Shutdown

  Note: This replaces the State Maintenance messages
- FELFBInstancelist
- FENeighborList
- MIID table: a list of virtual LFB Instance IDs that map to a list of Instance IDs of LFBs in that FE

* FE Behavior Exp. Timer
* HA Mode
* FE DoS protection policy
* FEPrivateData: Proprietary info such as name, vendor, model.
  Note: The attributes below were previously under Query message.
* Inter-FE topology Intra-FE topology

6.2.3 CE LFB

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

Editorial Note: NOTE: The CE LFB is under discussion still and may end up being removed.

The CE LFB consists of the following elements:

CE Events:

* CEAllEvents: subscribing to this corresponds to subscribing to all events listed below.
  Note: Do we want to allow an FE to explicitly subscribe to CE events?
* CEStatusChange: events that signal CE Up/Down/Active/Inactive/Failover.
  Note: Such events do not necessarily need to be subscribed to, they can fire even without subscription and be sent to the FE

Note: TBD: what else do we need in the CE LFB?

6.3 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.

6.4 Association Messages

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

6.4.1 Association Setup Message

This message is sent by the FE to the CE to setup a ForCES association between them. This message could also be used by CEs to join a ForCES NE, however CE-to-CE communication is not covered by this protocol.

Message transfer direction:
FE to CE
Message Header:
The Message Type in the header is set MessageType= 'Association Setup'. The ACK flag in the header is ignored, because the setup message will always expect to get a response from the message receiver (CE) whether the setup is successful or not. The Src ID (FE ID) may be set to 0 in the header which means that the FE would like the CE to assign a FE ID for the FE in the setup response message.
Message body:
The LFB selection points to the FE Object and more than one FE Object attribute may be announced in this message. The layout looks like:

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

```
    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 = LFB select      |               Length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                 LFB Class ID = FE Object                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        LFB Instance ID                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Type = operation Show  |               Length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~          Attributes path and data                             ~
   ~                                                               ~
   ~                                                               ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Type = LFB select      |               Length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                 LFB Class ID = FE  Protocol Object            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        LFB Instance ID                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Type = operation Show  |               Length          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                                                               ~
   ~          Attributes path and data                             ~
   ~                                                               ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 12

Type (16 bits):
   LFB Select
Length (16 bits):
   Length of the TLV including the T and L fields, in bytes.
FE Object and Protocol LFBs:
   These contains the FE parameters e.g. HBI will be exchanged with
   the CE using the FE Protocol LFB.
   Editorial Note: In certain situations (such as use of multicast
                    IDs), it might not be possible to make use of the
                    procedure described above for the FE to
                    dynamically obtain an ID from the CE. Such
                    situations need to be identified.

this message will still require some small discussion; for now goal is to convert to something using the two core FE LFBs.

6.4.2 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= 'Setup Response'. The ACK flag in the header is always ignored, because the setup response message will never expect to get any more response from the message receiver (FE). 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 setup response message body consists of LFBSelect & Result TLV, the format of which is as follows:

```
 main hdr (eg type =  Association setup response)
       |
       |
       |
       +--- T = LFBselect
       |        |
       |        +-- LFBCLASSID = FE object
       |        |
       |        |
       |        +-- LFBInstance = 0x1
       |        |
       |        +--- T = Operation = SHOW
       |             |
       |             +-- Path-data to one or more attibutes
       |                           including FE NAME
       |                           May contain RESULT TLVs
       +--- T = LFBselect
                |
                +-- LFBCLASSID = FE Protocol object
                |
                |
                +-- LFBInstance = 0x1
                |
                +--- T = Operation = SHOW
```

```
                  |
                  +-- Path-data to one or more attibutes
                      eg HB parameters
                      May contain RESULT TLVs


      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 = LFB select      |               Length          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                 LFB Class ID = FE Object                      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        LFB Instance ID                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |        Type = operation Show  |               Length          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               |
     ~          Attributes path and data                             ~
     ~                                                               ~
     ~                                                               ~
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |        Type = LFB select      |               Length          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                 LFB Class ID = FE  Protocol Object            |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        LFB Instance ID                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |        Type = operation Show  |               Length          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               |
     ~                                                               ~
     ~          Attributes path and data                             ~
     ~                                                               ~
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13

Type (16 bits):
    LFB Select
Length (16 bits):
    Length of the TLV including the T and L fields, in bytes.
FE Object LFB:
    The FE parameters e.g. HBI may be exchanged using this LFB.

Result (16 bits):
This indicates whether the setup msg was successful or whether the FE request was rejected by the CE.

6.4.3 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= "Asso. Teardown". The ACK flag in the header is always ignored, because the teardown message will never expect to get any response from the message receiver.

Message body:
The association teardown message body consists of LFBSelect & FEReason TLV, the format of which is as follows:

Editorial Note: Details of how Reason will be carried in the Teardown message are still unclear. There is no such attribute at the FE Object at the moment. There is also no operation by the name of FEReason.

```
     main hdr (eg type =  Association tear)
          |
          |
          |
          |
          +--- T = LFBselect
                   |
                   +-- LFBCLASSID = FE object
                   |
                   |
                   +-- LFBInstance = 0x1
                   |
                   +--- T = Operation = FEReason
                        |
                        +-- Path-data to string containing reason


           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 = LFB select      |               Length          |
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |                 LFB Class ID = FE Object                      |
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |                        LFB Instance ID                        |
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |         Type = T.reason       |               Length          |
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          |                   Reason                                      |
          +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14

Type (16 bits):
    LFB Select
Length (16 bits):
    Length of the TLV including the T and L fields, in bytes.
T.reason (32 bits):
    This indicates the reason why the association is being
    terminated.

6.5 Configuration Messages

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

6.5.1 Config Message

This message is sent by the CE to the FE to configure FE or LFB attributes. This message is also used by the CE to subscribe/unsubscribe to FE and LFB events.

Message transfer direction:
    CE to FE
Message Header:
    The Message Type in the header is set MessageType= 'Config'. The ACK flag in the header is can be used by the CE to turn off any response from the FE. The default behavior is to turn on the ACK to get the config response from the FE.
Message body:
    The Config message body consists of one or more TLVs, the format of a single (LFB) TLV is as follows:

```
 main hdr (eg type = config)
      |
      |
      +--- T = LFBselect
      |        |
      |        +-- LFBCLASSID = target LFB class
      |        |
      |        |
      |        +-- LFBInstance = target LFB instance
      |        |
      |        |
      |        +-- T = operation { GET, DEL,  etc}
      |        |   |
      |        |   +--  // one or more path targets
      |        |        // discussed later
      |        |
      |        +-- T = operation { GET, DEL,  etc}
      |        |   |
      |        |   +--  // one or more path targets
      |        |        // discussed later
      |        |
      |        +-- T = operation { GET, DEL,  etc}
      |        |   |
      |        |   +--  // one or more path targets
      |        |        // discussed later
      |        |


  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 = LFB select      |               Length          |
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           LFB Class ID                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         LFB Instance ID                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Operation (GET)           |               Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                 |
~                          Config path                            ~
|                                                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Operations (DEL)    |               Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                 |
~                          Config path                            ~
|                                                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 15

Type (16 bits):
    LFB Select.
Length (16 bits):
    Length of the TLV including the T and L fields, in bytes.
LFB Class ID (16 bits):
    This field uniquely recognizes the LFB class/type.
LFB Instance ID (16 bits):
    This field uniquely identifies the LFB instance.
Type (16 bits):
    The operations include, ADD, DEL, UPDATE/REPLACE, DEL ALL, EVENT SUBSCRIBE, EVENT UNSUBSCRIBE, CANCEL(under discussion).
Length (16 bits):
    Length of the TLV including the T and L fields, in bytes.
Config path + Data (variable length):
    This will carry LFB specific data (<path>, presentation preliminary found in this doc but still TBD.). The config data might itself be of the form of a TLV. Should be noted only a CREATE, REPLACE will have data while the rest will only carry path information of what to DELete or GET.
    *Note: FE Activate/Deactivate, Shutdown FE commands for State Maintenance will be sent using Config messages.
    *Note: For Event subscription, the events will be defines by the individual LFBs.

6.5.2 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, because the config response message will never expect to get any more response from the message receiver (CE).
Message body:
    The Config response message body consists of one or more TLVs, the format of a single TLV 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 = LFB select      |               Length          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          LFB Class ID                         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        LFB Instance ID                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    Operations (GET)           |               Length          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Operation Result          |           reserved            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 ~                         Config Result                         ~
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |    Type = Operations (DEL)    |               Length          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Operation Result          |           reserved            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 ~                         Config Result                         ~
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 16

Editorial Note: The operation result etc is still under discussion and will depend on verbosity levels in the main message headers

Type (16 bits):
    LFB Select.
Length (16 bits):
    Length of the TLV including the T and L fields, in bytes.
LFB Class ID (16 bits):
    This field uniquely recognizes the LFB class/type.
LFB Instance ID (16 bits):
    This field uniquely identifies the LFB instance.
Type (16 bits):
    The operations are same as those defined for Config messages.
Length (16 bits):
    Length of the TLV including the T and L fields, in bytes.
Operation Result (16 bits):
    This indicates the overall result of the config operation, whether it was successful or it failed.
Config Result (variable length):
    This will carry LFB specific results (single or Array LFB specific result entries). The config result might itself be of the form of a TLV.

6.6 Query and Query Response Messages

The ForCES query and query response messages are used by ForCES elements (CE or FE) to query LFBs in other ForCES element(s) Current version of ForCES protocol limits the use of the messages only for CE to query information of FE.

6.6.1 Query Message

As usual, 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:
    Current version limits the query message transfer direction only from CE to FE.
Message Header:
    The Message Type in the header is set to MessageType= 'Query'. The ACK flag in the header SHOULD be set 'ACKAll', meaning a full response for a query message is always expected. If the ACK flag is set other values, the meaning of the flag will then be ignored, and a full response will still be returned by message receiver.
Message body:
    The query message body consists of (at least) one or more than one TLVs that describe entries to be queried. The TLV is called LFBselect TLV and the data format 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 17

Editorial Note:

1. Under discussion is whether there is a need for explicit multiple LFB insatance addressing here. One way to realize it is to define a specific Instance select TLV to substitute above 'LFB Instance ID' field. The TLV may have following format:

```
INSselectTLV := Type Length Value
Type := INSselect
Value := InstanceID (RangeMark | Instance ID)+
```

2. An applicable RangeMark is '0xffffffff', the value of which is the same as Instance broadcast ID. Because there will be no broadcast address applied in this place, there will be no worry of ambiguity here.

Operation TLV:

The Operation TLV for the 'Query' message is formatted as:

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

Figure 19

PATH-DATA for GET:
This is generically a PATH-DATA format that has been defined in "Protocol Grammar" section in the PATH-DATA BNF definition, with the limitation specifically for GET operation that the PATH-DATA here will not allow DATARAW-TLV and RESULT-TLV present in the data format, so as to meet the genius of a GET operation.

To better understand the above PDU format, we can show a tree structure for the format as below:

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

Figure 20

6.6.2 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:
Current version limits the query response message transfer direction only from FE to CE.

Message Header:
The Message Type in the header is set to MessageType= 'QueryResponse'. The ACK flag in the header SHOULD be set 'NoACK', meaning no further response for a query response message is expected. If the ACK flag is set other values, the meaning of the flag will then be ignored. The Sequence Number in the header SHOULD keep the same as that of the query message to be responded, so that the query message sender can keep track of the responses.

Message body:
The message body for a query response message consists of (at least) one or more than one TLVs that describe query results for individual queried entries. The TLV is also called LFBselect TLV, and has exactly the same data format as query message, except the Operation TLV content is different. The order of the TLV here matches the TLVs in the corresponding Query message, and the TLV numbers should also keep the same. The Operation TLV here is a 'GET-RESPONSE' TLV and the data is a 'PATH-DATA' format for Query Response Data, as below:

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

Figure 21

PATH-DATA for GET-RESPONSE:
This is generically a PATH-DATA format that has been defined in "Protocol Grammar" section in the PATH-DATA BNF definition. The response data will be included in the DATARAW-TLV and/or RESULT-TLV inside the PATH-DATA format.

6.7 Event Notification and Response Messages

The Event Notification Message is used for one ForCES element to asynchronously notify one or more other ForCES elements in the same ForCES NE on just happened events in it. The Event Notification Response Message is used for the receiver of the Event Notification Message to acknowledge the reception of the event notification.

Events in current ForCES protocol can be categorized into following types:

- o Events happened in CE
- o Events happened in FE

Events can also be categorized into two classes according to whether they need subscription or not. An event in one ForCES element that needs to be subscribed will send notifications to other ForCES elements only when the other elements have subscribed to the element for the event notification. How to subscribe/unsubscribe for an event is described in the Configure Message section. An event that does not need to be subscribed will always send notifications to other ForCES elements when the event happens. Events will be defined in the ForCES FE model XML definitions for LFBs as attributes; i.e they will have a path to them that can be used by the config message to subscribe to.

Editorial Note: There is an argument that it is preferable to have all events subscribable.

6.7.1 Event Notification Message

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, or CE to FE

Message Header:

The Message Type in the message header is set to MessageType = 'EventNotification'. The ACK flag in the header can be set as: ACK flag ='NoACK'|'SuccessAck'|'UnsuccessACK'|'ACKAll'. Note that the 'Success' here only means the receiver of the message has successfully received the message.

Message Body:

The message body for an event notification message consists of (at least) one or more than one TLVs that describe the notified events. The TLV is defined 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 = LFBselect       |               Length          |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                          LFB Class ID                         |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                        LFB Instance ID                        |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                        Operation TLV                          |
        .                                                               .
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        ~                           ...                                 ~
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        |                        Operation TLV                          |
        .                                                               .
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 22

Operation TLV:
This is a TLV that describes the event to be notified, as follows:

```
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |    OPER = REPORT              |               Length          |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                        PATH-DATA for REPORT                   |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 23

PATH-DATA for REPORT:
This is generically a PATH-DATA format that has been defined in "Protocol Grammar" section in the PATH-DATA BNF definition. The report data will be included in the DATARAW-TLV inside the PATH-DATA format.

To better understand the above PDU format, we can show a tree structure for the format as below:

```
   main hdr (type = Event Notification)
        |
        |
        +--- T = LFBselect
        |        |
        |        +-- LFBCLASSID = target LFB class
        |        |
        |        |
        |        +-- LFBInstance = target LFB instance
        |        |
        |        |
        |        +-- T = operation { REPORT }
        |        |   |
        |        |   +--  // one or more path targets
        |        |        // under discussion
        |        +-- T = operation { REPORT }
        |        |   |
        |        |   +--  // one or more path targets
        |        |
```

Figure 24

6.7.2 Event Notification Response Message

After sending out an Event Notification Message, the sender may be interested in ensuring that the message has been received by receivers, especially when the sender thinks the event notification is vital for system management. An Event Notification Response Message is used for this purpose. The ACK flag in the Event Notification Message header are used to signal if such acknowledge is requested or not by the sender.

Detailed description of the message is as below:
Message Transfer Direction:
   >From FE to CE or from CE to FE, just inverse to the direction of the Event Notification Message that it responses.
Message Header:
   The Message Type in the header is set MessageType= 'EventNotificationResponse'. The ACK flag in the header SHOULD be set 'NoACK', meaning no further response for the message is expected. If the ACK flag is set other values, the meaning of the flag will then be ignored. The Sequence Number in the header SHOULD keep the same as that of the message to be responded, so that the event notificatin message sender can keep track of the responses.

Message Body:
The message body for an event notification response message consists of (at least) one or more than one TLVs that describe the notified events. The TLV is also called LFBselect TLV, and has exactly the same data format as Event Notification Message, except the Operation TLV inside is different. The order of the TLV here matches the TLVs in the corresponding Event Message, and the TLV numbers should keep the same. The Operation TLV here is a 'REPORT-RESPONSE' TLV and the data is a 'PATH-DATA' format for event response data, as below:

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

Figure 25

PATH-DATA for REPORT-RESPONSE:
This is generically a PATH-DATA format that has been defined in "Protocol Grammar" section in the PATH-DATA BNF definition. The response data will be included in the RESULT-TLV inside the PATH-DATA format.

Editorial Note: There is a debate on whether the Event Notification Response Message is necessary or not. The pro for it is some event notification senders may be interested in knowing if receivers have had success/unsuccess receptions of the events or not. An alternative to generate such response is for the protocol to define a universal ACK message so that it can act as responses for any types of messages as well as the event notification messages, when the message senders are interested in knowing whether the messages have been successfully received or not (different from the responses for the message processing results).

6.8 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 associated with some metadata generated by other LFBs in the model, or they may occasionally be other protocol packets, which

usually happen 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 are decided by CE to put into forwarding plane in 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, making the redirect path been 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.

Editorial Note: There are also discussions on how LFBs in FE model that are related to packet redirect operations should be defined. Although it is out of the scope of forces protocol, how to define the LFBs affect the Packet Redirect Message described here. Because currently it is still in progress in FE model on how to define such LFBs, we try to post some thoughts on this here for discussion. They will be removed later along with the progress of the FE model work.

Thought 1: To define LFBs called 'RedirectSink' and 'RedirectTap' for packet redirect.
An LFB in FE called 'RedirectSink' is responsible to collect data packets that need to be redirected to CE. From the perspective of the FE LFB topology, the 'RedirectSink' LFB is an LFB with only one input port and without any output port, and the input port can then be connected to any other LFB in FE model by means of a datapath in the forwarding plane.
From the perspective of the ForCES protocol layer, the 'RedirectSink' LFB will generate the Packet Redirect Messages when it receives data packets from forwarding plane.

An LFB in FE called 'RedirectTap' is responsible to receive data packets that are redirected from CE. From the perspective of the FE LFB topology, the 'RedirectTap' LFB is an LFB with only one output port and without any input port, and the output port can then be connected to any other LFB in FE model by means of a datapath in the forwarding plane. From the perspective of ForCES protocol layer, the 'RedirectTap' LFB can receive the Packet Redirect Messages from CE, and un-encapsulate the data packets from the message and put them to datapaths in the forwarding plane. Actually the 'RecirectTap' LFB acts more like a transcoder that transfers the ForCES protocol messages to normal data packets in IP forwarding plane. As a result, if we need to have redirected packets connected to some LFB (say a Scheduler) in FE model, we only need to connect the 'RedirectTap LFB to the Scheduler LFB directly via a datapath as follows:

```
+-----------------+       +-----------+
| RedirectTap LFB |------>|           |
+-----------------+       |           |
                          | Scheduler |
    From other LFB   ---->|    LFB    |
                          |           |
                          +-----------+
```

Figure 26

By use of several 'RedirectSink' LFBs and several 'RedirectTap' LFBs that connect to several different datapaths in FE forwarding plane, multiple packet redirect paths between CE and FE can be constructed.

Thought 2: There might be another way a packet could be redirected: directly by a forwarding path, e.g., by FPGA/ASIC/NP microcode. In such a case we do not need to put in a lot of smartness. Probably a link layer or even network level header is enough. The receiver demuxes it only based on some protocol type in the link layer or network transport layer. The pros for this appraoch is it may provide a fast and cost-effective path for packet redirect. The cons for this is it may more or less confuses the Fp reference point definition in ForCES framework.

We describe the Packet Redirect Message data format in details 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 SHOULD be set 'NoACK', meaning no response is expected by this message. If the ACK flag is set other values, the meanings will be ignored.

Message Body:

   Consists of one or more TLVs, with every TLV having the following data format:

```
     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 27

LFB class ID:

   There are only two possible LFB classes here, the 'RedirectSink' LFB or the 'RedirectTap' LFB. 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 'RedirectTap'.

Instance ID:

   Instance ID for the 'RedirectSink' LFB or 'RedirectTap' LFB.

Operation TLV:

   This is a TLV describing one packet of data to be directed via the specified LFB above. The order of the data number is also the order the data packet arrives the redirector LFB, that is, the Redirected Data #1 should arrive earlier than the Redirected Data #2 in this redirector LFB. The TLV format is as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type = PAYLOAD             |              Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Path(or Sequence Number?)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                        Redirected Data                        ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 28

Path(or Sequence Number?):
   [Under discussion and TBD]
Type:
   [TBD]
Redirected Data:
   This field will make a detailed description of the data to be redirected as well as the data itself. The encoding of the description is based on the ForCES FE model if the redirector LFB is defined by FE model, or based on vendor specifications if the redirector LFB is defined by vendors. The description will usually include the name (or the name ID) of the redirected packet data (such as 'IPv4 Packet', 'IPv6 Packet'), and the packet data itself. It may also include some metadata (metadata name (or name ID) and its value)associated with the redirected data packet.

6.9 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 time interval to send the message is set by the Association Setup Message described in Section 6.1.1. A little different from other protocol messages, a Heartbeat message is only composed of a common header, withe 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'. The ACK flag in the header SHOULD be set to 'NoACK', meaning no response from receiver(s) is expected by the message sender. Other values of the ACK flag will always be ignored by the message receiver.

Message Body:
The message body is empty for the Heartbeat Message.

6.10 Operation Summary

The following tables summarize the operations and their applicabiity to the messages.

No Operations for the following messages:
- Assoc-Setup
- Assoc-Setup-Resp
- Assoc-Teardown
- Heartbeat

Operation	Query	Query-Resp	Config	config-Resp
Set			X	X
Delete			X	X
Update			X	X
Get	X	X		
Event subscribe			X	X
Event unsubscribe			X	X

Operation	Packet-Redir	Event-Notif	Event-Notif-Resp
Payload	X		
Event		X	X

7. Protocol Scenarios

7.1 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 association 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 followed by capability query, topology query. When the FE is ready to start forwarding data traffic, it sends a FE UP Event message to the CE. The CE responds with a FE ACTIVATE State Maintenance message to ask the FE to go active and 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     |
                        |<----------------------|
                        |                       |
                        |    Capability Query   |
                        |<----------------------|
                        |                       |
                        |      Query Resp       |
                        |---------------------->|
                        |                       |
                        |       Topo Query      |
                        |<----------------------|
                        |                       |
                        |   Topo Query Resp     |
                        |---------------------->|
                        |                       |
                        |       FE UP Event     |
                        |---------------------->|
                        |                       |
                        |  Config-Activate FE   |
                        |<----------------------|
                        |                       |
```

Figure 29: Message exchange between CE and FE to establish an NE association

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

7.2 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. Figure below helps illustrate this state.

```
                      FE PL                   CE PL

                        |                       |
```

```
                        |    Heart Beat         |
                        |<----------------------|
                        |                       |
                        |   Heart Beat          |
                        |---------------------->|
                        |                       |
                        |  Config-Subscribe Ev  |
                        |<----------------------|
                        |                       |
                        |     Config Resp       |
                        |---------------------->|
                        |                       |
                        |  Config-Add LFB Attr  |
                        |<----------------------|
                        |                       |
                        |     Config Resp       |
                        |---------------------->|
                        |                       |
                        |   Query LFB Stats     |
                        |<----------------------|
                        |                       |
                        |    Query Resp         |
                        |---------------------->|
                        |                       |
                        |    FE Event Report    |
                        |---------------------->|
                        |                       |
                        |  Config-Del LFB Attr  |
                        |<----------------------|
                        |                       |
                        |     Config Resp       |
                        |---------------------->|
                        |                       |
                        |    Packet Redirect    |
                        |---------------------->|
                        |                       |
                        |    Heart Beat         |
                        |<----------------------|
                        .                       .
                        .                       .
                        |                       |
                        |  Config-Activate FE   |
                        |<----------------------|
                        |                       |
```

Figure 30: 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.

8. High Availability Support

The ForCES protocol provides mechanisms for CE redundancy and failover, in order to support High Availability as defined in [RFC3654]. FE redundancy and FE to FE interaction is currently out of scope of this draft. There can be multiple redundant CEs and FEs in a ForCES NE. However, at any time there can only be one Primary CE controlling the FEs and 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, CE PLs during pre-association by FEM, CEM respectively. Only the primary CE sends Control messages to the FEs. The FE may send its event reports, redirection packets to only the Primary CE (Report Primary Mode) or it may send these to both primary and secondary CEs (Report All Mode). (The latter helps with keeping state between CEs synchronized, although it does not guarantee synchronization.) This behavior or HA Modes are configured during Association setup phase but can be changed by the CE anytime during protocol operation. A CE-to-CE synchronization protocol will be needed in most cases to support fast failover, however this will not be defined by the ForCES protocol.

During a communication failure between the FE and CE (which is caused due to CE or link reasons, i.e. not FE related), the TML on the FE will trigger the FE PL regarding this failure. This can also be detected using the HB messages between FEs and CEs. The FE PL will send a message (Event Report) to the Secondary CEs to indicate this failure or the CE PL will detect this and one of the Secondary CEs takes over as the primary CE for the FE. During this phase, if the original primary CE comes alive and starts sending any commands to the FE, the FE should ignore those messages and send an Event to all CEs indicating its change in Primary CE. Thus the FE only has one primary CE at a time.

An explicit message (Config message- Move command) from the primary CE, can also be used to change the Primary CE for an FE during normal protocol operation. In order to support fast failover, the FE will establish association (setup msg) as well as complete the capability exchange with the Primary as well as all the Secondary CEs (in all scenarios/modes).

These two scenarios (Report All, Report Primary) have been illustrated in the figures below.

```
                 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 31: CE Failover for Report All mode

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

Figure 32: CE Failover for Report Primary Mode

8.1 Responsibilities for HA

TML level - Transport level:

1. The TML controls logical connection availability and failover.
2. The TML also controls peer HA managements.

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

PL Level:
All the 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.

9. Security Considerations

ForCES architecture identified several [Reference Arch] levels of security. 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.

Following are the general security mechanism that needs to be in place for ForCES PL layer.

- o Security mechanism 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.

9.1 No Security

When No security is chosen for ForCES protocol communication, both endpoint authentication and message authentication service needs 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 an 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.

9.1.1 Endpoint Authentication

Each CE and FE PL layer maintain 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 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.

9.1.2 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. We recommend this extra step processing even if the underlying TLM layer security services.

9.2 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.

9.2.1 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.

We recommend that 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.

9.2.2 Message authentication service

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

9.2.3 Confidentiality service

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

10. Acknowledgments

The authors of this draft would like to acknowledge and thank the following: Alex Audu, Steven Blake, Allan DeKok, Ellen M. Deleganes, Yunfei Guo, Joel M. Halpern, Zsolt Haraszti, Jeff Pickering, Guangming Wang, Chaoping Wu, Lily L. Yang, and Alistair Munro for their contributions. We would also like to thank David Putzolu, and Patrick Droz for their comments and suggestions on the protocol.

11. References

11.1 Normative References

[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.

11.2 Informational References

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

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

Author's Address

Avri Doria
ETRI

Phone: +1 401 663 5024
Email: avri@acm.org

Appendix A. Individual Authors/Editors Contact

Ligang Dong
Zhejiang Gongshang University
149 Jiaogong Road
Hangzhou 310035
P.R.China
Phone: +86-571-88071024
EMail: donglg@mail.hzic.edu.cn

Avri Doria
ETRI
EMail: avri@acm.org

Ram Gopal
Nokia
5, Wayside Road
Burlington MA 01803
USA
Phone: 1-781-993-3685
EMail: ram.gopal@nokia.com

Robert Haas
IBM
Saumerstrasse 4
8803 Ruschlikon
Switzerland
EMail: rha@zurich.ibm.com

Jamal Hadi Salim
Znyx
Ottawa, Ontario
Canada
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.hzic.edu.cn

Appendix B. IANA considerations

tbd

Appendix C. Forces Protocol LFB schema

The schema described below conforms to the LFB schema (language?) 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  -->
     <LFBClassDefs>
       <LFBClassDef>
         <name>FEPO</name>
         <id>1</id>
         <synopsis> The FE Protocol Object </synopsis>
         <version>1.0</version>
         <derivedFrom>baseclass</derivedFrom>
         <events>
   <!--
        The CE sets this event attribute to "yes" to kick HBs
        from the FE. By default no HBs are generated
   -->
           <attribute>
             <name>HBstate</name>
             <id>2</id>
             <synopsis>Heartbeat event status(yes/no)</synopsis>
             <typeRef>boolean</typeRef>
           </attribute>
         </events>
   <!--
   -->
         <capabilities>
           <capability>
              <name>SupportableVersions</name>
              <id>1</id>
              <synopsis>the table of ForCES versions that FE supports</
synopsis>
              <array type="variable-size">
               <typeRef>u8</typeRef>
            </array>
           </capability>
         </capabilities>
   <!--
   ADD other attributes related to HA and failover and restart
   policies later
   -->
         <attributes>
           <attribute access="read-write">
```

```
            <name>HBI</name>
            <id>3</id>
```

```
            <synopsis>Heartbeat Interval in millisecs</synopsis>
            <typeRef>uint32</typeRef>
          </attribute>
          <attribute access="read-write">
            <name>HBDI</name>
            <id>4</id>
            <synopsis>Heartbeat Dead Interval in millisecs</synopsis>
            <typeRef>uint32</typeRef>
          </attribute>
          <attribute access="read-only">
            <name>CurrentRunningVersion</name>
            <id>5</id>
            <synopsis>Currently running ForCES version</synopsis>
            <typeRef>u8</typeRef>
          </attribute>
        </attributes>
   <!--
   -->
      </LFBClassDef>
    </LFBClassDefs>
   </LFBLibrary>
```

C.1 Events

At the moment only one event, HBstate, can be subscribed to by the CE.

By subscribing to the HBstate event, the CE infact kicks the FE into motion to start issuing heartbeats.

C.2 Capabilities

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

SupportableVersions enumerates all ForCES versions that an FE supports.

C.3 Attributes

C.3.1 HBI

This attribute carries the Heartbeat Interval of the heartbeat from the FE -> CE in millisecs. The value of this interval is by default set by the FE but could be overwritten in the association setup by the CE.

TBD (this really belongs in the protocol draft but here for capture purposes:

Define it as simply that the CE and FE must hear from each other at the configured interval. The FE on her side generates a heartbeat notification if he has nothing else to say. In otehr words, The lack of any messages from the CE to which the FE responded to after a period of HBI will result in a FE firing a HB message. The lack of any message within DeadInterval will force the FE to ask for an ACK for its HB message (by setting the ACK flag in the header).

Other adaptive heartbeats schemes which could be used: have the CE adjust the FE timers depending on the number of FEs present. Example, its 1 sec for upto 100 FEs and 2 seconds for [101,200] 4 seconds interval for > 200 nodes etc ... Some adaptation of this is used by mmusic mbus protocol.

C.3.2 HBDI

This attribute carries the Heartbeat Dead Interval in millisecs.

TBD:

The original goal for HBDI was for HA purposes - to discover if the CE is still around by sending a heartbeat message to the CE with an ACK flag in the mainheader to request for a response. This hasnt been discussed in details yet; however, the general view at the time was for the FE to associate (failover) to another CE after that deadinterval period of not hearing from the CE - as defined by policy which resides in that same LFB definition. Two such failover methodologies are mentiooned briefly infact in the protocol draft but since the current attributes are unknown, the details are missing from the xml.

C.3.3 CurrentRunningVersion

This attribute describes which version of ForCES is currently running.

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.

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
                DATARAW-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
                DATARAW 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
                DATARAW=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
   (Ignore error conditions for now)

```
OPER = SET-CREATE-TLV
        Path-data-TLV:
                flags = 0 , IDCount = 1, IDs=4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 0
                    DATARAW-TLV containing j1, j2 value for entry 0
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    DATARAW-TLV containing j1, j2 value for entry 1
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    DATARAW-TLV containing j1, j2 value for entry 2
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    DATARAW-TLV containing j1, j2 value for entry 3
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                    DATARAW-TLV containing j1, j2 value for entry 4
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5
                    DATARAW-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
                    DATARAW-TLV containing j1, j2 value for entry 0
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    DATARAW-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
                DATARAW 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
                    DATARAW-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 1
                    DATARAW-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 2
                    DATARAW-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 3
                    DATARAW-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 4
                    DATARAW-TLV containing j1value j2value
                PATH-DATA-TLV
                    flags = 0, IDCount = 1, IDs = 5
                    DATARAW-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
                DATARAW 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 we asked for verbose response back in the main message header.

```
OPER = SET-CREATE-TLV
        Path-data -TLV:
                flags = FIND-EMPTY, IDCount = 1, IDs=4
                DATARAW 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
                        DATARAW-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
                DATARAW 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.

```
NOTE! NOTE!
There is still debate as to whether this must reference only 1 entry.

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

14. Multiple atomic operations.

[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]

Main header has atomic flag set and we are request 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
                   DATARAW 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
                           DATARAW 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
                           DATARAW 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
                                   DATARAW 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
                                   DATARAW TLV, Length = XXXX v=20
```

15. Selective setting (Example posted by Weiming). 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
                   DATARAW TLV, Length = XXXX, V = {100}
               Path-data TLV
                   flags = 0, IDCount = 1, IDs = 2
                   DATARAW TLV, Length = XXXX, V = {200}
               Path-data TLV
                   flags = 0, IDCount = 1, IDs = 3
                   DATARAW TLV, Length = XXXX, V = {300}
           Path-data TLV
               flags = 0, IDCount = 1, IDs = 3
               Path-data TLV
                   flags = 0, IDCount = 1, IDs = 1
                   DATARAW TLV, Length = XXXX, V = {100}
               Path-data TLV
                   flags = 0, IDCount = 1, IDs = 2
```

```
                DATARAW TLV, Length = XXXX, V = {200}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 3
                DATARAW TLV, Length = XXXX, V = {300}
        Path-data TLV
            flags = 0, IDCount = 1, IDs = 5
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 1
                DATARAW TLV, Length = XXXX, V = {100}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 2
                DATARAW TLV, Length = XXXX, V = {200}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 3
                DATARAW TLV, Length = XXXX, V = {300}
        Path-data TLV
            flags = 0, IDCount = 1, IDs = 7
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 1
                DATARAW TLV, Length = XXXX, V = {100}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 2
                DATARAW TLV, Length = XXXX, V = {200}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 3
                DATARAW TLV, Length = XXXX, V = {300}
        Path-data TLV
            flags = 0, IDCount = 1, IDs = 9
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 1
                DATARAW TLV, Length = XXXX, V = {100}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 2
                DATARAW TLV, Length = XXXX, V = {200}
            Path-data TLV
                flags = 0, IDCount = 1, IDs = 3
                DATARAW 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
                DATARAW TLV: L=XXXX, V = {x1 value}
```

17. From table5's row 10 table10, get X2s based on on the value of x1 equlaing 10 (recal 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
                        DATARAW TLV: L=XXXX, V = {x2 value}
```

18. Further example of table of table

Weiming would like to update different items on a hierachy of data. So this example is there to show how that can be done with the current BNF.

```
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
```

So lets say we 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

```
operation = SET-REPLACE-TLV
        Path-data-TLV
                flags = 0  IDCount = 2, IDs=6.10
                Path-data-TLV
                        flags = 0, IDCount = 1, IDs=1
                        DATARAW TLV: L=XXXX,
                                V = {111}
                Path-data-TLV
                        flags = 0  IDCount = 2, IDs=2.20
                        Path-data-TLV
                                flags = 0, IDCount = 1, IDs=1
                                DATARAW TLV: L=XXXX,
                                        V = {222}
                        Path-data TLV :
                                flags = 0, IDCount = 3, IDs=2.30.1
                                DATARAW 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) using a flat path data
   operation = SET-REPLACE-TLV
           Path-data TLV :
                   flags = 0, IDCount = 3, IDs=6.10.1
                   DATARAW TLV: L=XXXX,
                           V = {111}
           Path-data TLV :
                   flags = 0, IDCount = 5, IDs=6.10.1.20.1
                   DATARAW TLV: L=XXXX,
                           V = {222}
           Path-data TLV :
                   flags = 0, IDCount = 7, IDs=6.10.1.20.1.30.1
                   DATARAW 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 targetted 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
                   DATARAW encoding of the FE Object LFB
```

Appendix E. Implementation Notes

E.1 TML considerations

Having separated the PL from the TML layer, it became clear that the TML layer needed to understand the desires of the PL layer to service it. Example: How does the TML layer map prioritization or reliability needs of a PL message? To see the challenge involved, assume that all of the FE TML, FE PL, CE TML and CE PL are implemented by different authors probably belonging to different organizations. Three implementation alternatives were discussed.

As an example, consider a TML which defines that PL messages needing reliability get sent over a TCP connection; then TML-PL interfaces are:

- o PL to call a special API: example send_reliable(msg) which is translated by the TML to mean send via TCP.
- o PL to call a generic API: example send(msg) with explicit msg flags turned to say "reliability needed" and the TML translates this to mean send via TCP.
- o PL sends the Forces Messages such a message is inferred to mean send via TCP by the TML.

in #1 and #2 the msg includes a ForCES msg with metadata flags which are consumed by the TML layer.

#3 is a technique that will be referred as inference-by-TML technique. It simplifies the standardization effort since both #1 and #2 will require standardization of an API. Two ideas discussed for TML inference of PL messages are:

1. Looking at the flags in the header.
2. Looking at the message type.

#1 and #2 can still be used if a single organization implements both (PL and TML) layers. It is also reasonable that one organization implements the TML and provides an abstraction to another organization to implement a PL layer on.

E.1.1 PL Flag inference by TML

1. Reliability
   This could be "signalled" from the PL to the TML via the ACK flag. The message type as well could be used to indicate this.
2. No reliability
   Could be signalled via missing ACK flag. The message type as well could be used to indicate this.
3. Priorities
   A remapping to be defined via the FEM or the CEM interface depending on the number of TML priorities available.

4. Addressing
   This is TML specific. For example a TML that is capable of multicast transport may map a multicast PL ID to a multicast transport address.
5. Event notifications
   The TML must be able to send to the PL notifications.
   1. The TML should be able to send Transport level congestion notifications to the PL.
   2. Link events for HA purposes if configuration requires it
   3. Events that will trigger PL layer events from the TML.
      As an example, an HA event at the TML layer like a failure of CE detected at TML on the FE may belong to this. In this case, a PL event msg will be triggered and sent to CE.
   4. Events that are intrinsic to the same CE or FE a TML is located. These will not trigger any PL msg, instead, they just act as notification to PL core (FE object). The congestion event generated at the transmission source side may belong to this, because it usually only needs to tell the upper PL at the same side rather than the opposite side that congestion has happened along the path. E.g., a congestion event at CE TML layer only need to tell CE PL of this, rather than the opposite FE via a PL msg.

E.1.2 Message type inference to Mapping at the TML

In this case one would define the desires of the different message types and what they expect from the TML. For example:

1. Association Setup, Teardown, Config, Query the PL will expect the following services from TML: Reliable delivery and highest prioritization.
2. Packet Redirect, HB Message Types, and Event Reports the PL will require the following services from TML: Medium Prioritization, and notifications when excessive losses are reached.

Appendix F. Changes between -01 and -02

1. Renamed definitions.xml to Definitions.xml
2. Added Alistair Munro to acks list.
3. path-data additions + full BNF conformant to RFC 2234
4. Appendix C with examples. #3 and #4 are the biggest changes incorporate many many days of discussion.
5. appendix with beginings of FE protocol LFB xml. The FE Object is referenced as being in the Model draft
6. Some cosmetic things like:
    1. For readability, introducing section 'protocol construction' which now encapsulates 'Protocol Messages' (which used to be a top section)
    2. A new subsection "protocol grammar' goes underneath the same section.
    3. added TLV definition subsection
    4. Many new "editorial notes"
7. Closure of all but one outstanding issue from the tracker.
8. Any other cosmetic changes posted (Hormuzd, David, Robert, Avri).
9. Rearranged text a little to introduce new sections to make text more readable
10. Rewrote the atomicity section (still under construction input text on ACID from Robert and Alistair)
11. fixed up the model reference to have all authors and added acid reference
12. Weiming's updates to query and event msgs to add path-data.

Appendix G. Changes between -00 and -01

1. Major Protocol changes
   * Restructured message format to apply operation to LFB as opposed to having operation be the primary organizing principle
   * Worked with model team to bring the draft into harmony with their model approach
2. Document changes
   * Replaced FE protocol Object and FE Object sections with combined section on FE, CE and FE protocol LFBs
   * Removed minor version id
   * Added Header flags
   * Added BNF description of message structure
   * Added tree structure description of PDUs
   * Added section on each type of LFB
   * Added structural description of each message
   * Moved query messages section to come after config message section
   * Replace state maintenance section
   * Added section with tables showing the operations relevant to particular messages
   * Reworked HA section
   * Many spelling and grammatical corrections

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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