ForCES Working Group
Internet-Draft
Expires: August, 2007

W. M. Wang
Zhejiang Gongshang Univ.
J. Hadi Salim
Znyx Networks
Alex Audu
Garland SoftWorx
February, 2007

ForCES Transport Mapping Layer (TML) Service Primitives

draft-ietf-forces-tmlsp-01.txt

Status of this Memo

Conventions used in this document

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

Abstract

This document specifies Transport Mapping Layer (TML) Service Primitives for Forwarding and Control Element Separation (ForCES). Based on the service primitives, TML services that are provided by TML to ForCES Protocol Layer (PL) are standardized. To define the

primitives, TML properties represented as TML events, TML attributes, and TML capabilities are also specified in the document.

Table of Contents

1. Introduction

ForCES aims to define a set of specifications for routers, firewalls, gateways, etc based on the architecture of separation of Forwarding Elements (FEs) and Control Elements (CEs). RFC3654 has presented the ForCES requirements, and RFC3746 has defined the ForCES framework. The ForCES FE model [ForCES-Model] is specifying the model to represent an FE. The ForCES protocol [ForCES-PL] is specifying the information exchanging protocol between CE and FE.

The ForCES protocol infrastructure consists of two layers:

1. The Protocol Layer (PL), which is responsible for generating ForCES protocol messages, and processing protocol messages that come from peering protocol layer in the same ForCES NE.

2. The Transport Mapping Layer (TML), which is responsible for transportation of ForCES protocol messages over variant transport media, like IP, Ethernet, ATM, etc.

The ForCES protocol [ForCES-PL] document defines the specifications for PL, while TMLs of different transport media types are to be defined by individual IETF documents. A ForCES PL implementation must be portable across all TMLs. It is feasible that the implementers of TML and PL may be from different organizations. As a result, services TML provides to PL must be specified in a standardizing way.

The purpose of this document is to specify the services that various TMLs must provide for ForCES PL layer. The TML services are represented by a set of TML service primitives and associated TML properties (TML attributes, etc).

Note that this document specifies TML services more at a semantic level, i.e., it does not try to specify details on how the defined TML services shall be implemented. Different Operating System platforms that PL and TML may rely on to be developed may have different programming methods, process techniques, data structures, etc for realizing the set of TML services. As a result, TML interface APIs constructed according to this document may vary in some way. In this condition, one PL portable to various TMLs actually means the PL must provide various interface drivers for different TMLs, while keeping the PL kernel the same for the TML operations.

2. Definitions

This document follows the terminology used by RFC3654, RFC3746, the ForCES protocol[ForCES-PL], and the ForCES FE model [ForCES-Model]. For convenience, some definitions are just copied here:

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 [ForCES-PL].

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

3. Overview

3.1. ForCES Protocol Framework

The ForCES protocol document has presented the protocol framework as in Figure 1. The framework shows the relationship between Protocol Layer (PL) and Transport Mapping Layer (TML). According to this framework, TML lies under PL and provides transportation services for protocol messages to PL. CE PL communicates with FE PL via CE TML and FE TML. On transmission, PL delivers its ForCES messages to its TML. The TML further delivers the messages to the destination peering TML(s). On receive, TML delivers ForCES messages it has received to its PL.

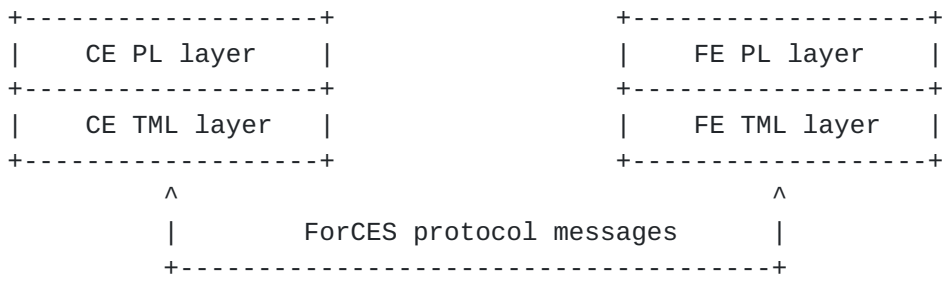

Figure 1. ForCES Protocol Framework

3.2. TML Requirements

The ForCES protocol docuement has also presented TML requirements. We list the requirements as below. Each TML specification must describe how it contributes to achieving the requirements. If, for any reason, a TML does not provide a service listed by the requirements, a justification needs to be provided.

The TML requirements are:

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

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

* Endpoint authentication of FE and CE.

* Message Authentication

* Confidentiality service

3. Congestion Control
The congestion control scheme used needs to be defined. The congestion control mechanism defined by the TML should prevent the FE

from being overloaded by the CE or the CE from being overwhelmed by

traffic from the FE. Additionally, the circumstances under which notification is sent to the PL to notify it of congestion must be defined.

4.Uni/multi/broadcast addressing/delivery if any
If there is any mapping between PL and TML level Uni/Multi/Broadcast addressing it needs to be defined.

5. HA decisions
It is expected that availability of transport links is the TML's responsibility. However, on config basis, the PL layer may wish to participate in link failover schemes and therefore the TML must support this capability.

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

7. Prioritization
It is expected that the TML will be able to handle up to 8 priority levels needed by the PL layer and will provide preferential treatment. TML needs to define how this is achieved. The requirement for supporting up to 8 priority levels does not mean that the underlying TML MUST be capable of handling up to 8 priority levels. In such an event the priority levels should be divided between the available TML priority levels. For example, if the TML only supports 2 priority levels, the 0-3 could go in one TML priority level, while 4-7 could go in the other.

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

4. TML Representation

The document is to define a set of general services for TML that various TMLs and their implementations must fundamentally provide for ForCES PL layer. For this sake, a TML representation is necessary that describes general properties of various TMLs. The following entities are used to represent various TML properties.

TML Events:
When the events happen in TML, PL may be interested to be notified.

TML attributes:
Represent the TML parameters that should be configured by PL when PL asks TML to provide services.

TML capabilities:

TML abilities or capacities that PL is interested to know.

Note that, not all TML properties should be made perceivable and controllable by PL. PL only cares those TML properties that PL should interact with in order for TML to properly provide services to the PL.

4.1. TML events

TML events are triggered by some TML status changes when TML is running. PL layer may be interested to be notified when some TML events occur. TML is responsible to asynchronously notify PL of these events.

How a TML event is asynchronously notified to PL highly depends upon operating system environments PL and/or TML implementations may be based. As an example, some environments may adopt a callback mechanism for notification of events between program processes. In this case and for the PL/TML usage, PL may first construct a callback function to process every event, and then tell TML the callback function handle. Whenever an interested event happens in TML, the TML will notify PL of the event by invoking the callback handle and let PL execute the callback function. In this way, the TML asynchronously passes the event notification to the PL.

However, this document does not try to define specific means for PL/TML notification. Any appropriate means can be adopted under condition that it shall meet the service requirements.

A TML event may appear as a sustained event, i.e., the event will last until the condition triggered the event is changed and the event is then released. For instance, when an error happens in TML, it will last until the error is finally by any means removed. In the sustained event case, PL may be interested in knowing not only when the event takes place, but also when the event is released. To meet this need, an event status parameter should be defined and associated with a sustained event report. The parameter will mark the associated event with either 'occurring' or 'is released' to indicate the two different status of a sustained event. Nevertheless, not all events are sustained events, so that not all event reports need this kind of parameters.

A TML event shall be distinguished as being subscription-free or subscription-requested. A subscription-free TML event will inevitably be notified to PL when it occurs. A subscription-requested TML event is only notified to PL when it has been subscribed for by the PL. A way for PL to subscribe for a subscription-requested TML event will be provided by defining an event handle TML attribute as in 1) of Section 4.2. The TML attribute can also provide more events parameters configuration. See Section 4.2 for more details on the

attribute definition.

An TML event is assigned a TML event id, so that PL can identify the event uniquely.

Followed are descriptions of TML events that must be available for all TML specifications. If, for any reason, an individual TML specification does not provide the events, a justification needs to be provided in the specification.

1) TML error event

This event reports a TML error during TML running to PL. When the event is invoked, something might be wrong in the TML. Failures like TML link failure in TML are also taken as TML errors, which can be understood as fatal errors for some cases.

When the event occurs and an event report is notified to PL, an error code is associated with the event report so as to pass more information about the error to PL layer. The code may expose PL the reason of the error, the type of the error, as such.

TML error event is a subscription-free event. When the event occurs, it will always invoke a report to PL.

TML error event is a sustained event. The error status will last until the error is removed by any means.

As a result, when the event is notified to PL, the following information shall be associated with the event report:

- o TML error code and associated data

- o event status
  - 1 The event is occurring
  - 0 The event is released

Note that it is not restricted that more information may also be associated with the event report, depending upon each TML specification or implementation.

This document defines the following TML errors and associated data. These errors are usually common to all types of TMLs.

TML error code	TML error	Associated Data
1	all local TML link failure	none
2	some local TML link failure	peer TML CE/FE ID(s) the link is connected
3	peer TML unavailable	peer TML CE/FE ID(s)
4	peer TML abnormally left	peer TML CE/FE ID(s)

Each TML specification may define its specific errors. There is also a room for each TML implementation to define specific errors.

TML error event is assigned with a TML event id = 1.

2) Message arrive event

TML shall be able to make it as an event occurrence when it has received a PL ForCES protocol message from peer TML and has made it ready to deliver to local PL. In this way, an asynchronous message receive mode can be realized in PL. In addition to this asynchronous mode, PL can also use a specific TML receive service primitive as defined in Section 5.8 for PL synchronous receiving of ForCES messages.

This event is a subscription-requested event, i.e, unless PL has requested TML to do so, TML will not use an asynchronous event notification way to deliver arrived messages to PL. In this case, PL can still use a TML receive primitive to receive ForCES messages.

When the event is notified to PL, the following information shall be associated:

- the arrived ForCES message length
- the whole arrived ForCES message Protocol Data Unit (PDU)

It is not restricted that other information might also be associated with this event report, depending upon requirements of individual TML specifications or implementations.

Note that the message arrive event is not a sustained event, and there is no need to distinguish its status as occurring or released. When the message is reported to PL, the event is automatically released.

TML message arrive event is assigned with TML event id = 2.

3) TML congestion alert

Although it is expected that TML provide a ForCES message transportation free of congestion, as a general problem for current Internet society, congestion problem is still quite hard to be completely avoided if mechanisms are purely limited in TML resources and without help from PL resources. In many cases, with the help from the resources in PL layer, congestion problem may be much better suppressed. For example, in cases where the OS a TML adopts is capable of detecting an ECN (Early/explicit congestion notification)

where the underlying IP protocol is capable of passing information to

the application, then the TML could pass such information to the PL. The PL could use this information for example to adjust its sending rates or increase or reduce the priority of certain PL messages, etc. More over, even in the case PL could help little for TML congestion, it is still very helpful for PL to know the TML congestion state if it does happen. As a result, in the TML requirement in Section 3.2, it is required that TML must be defined with a method to notify PL of congestion state.

This document specifies that a TML congestion alert event must be supplied with various TMLs and their implementations if the TMLs and implementations are not free from congestion problems due to any reasons. TML notifies PL of congestion state by this TML congestion alert event. It is an alert event because we expect that TML notifies PL of the event when TML is in danger of, rather than in the state of, congestion. An alert event is more helpful, because TML is the only path that an FE is connected to CE, and a complete congestion state in TML may lead to CE lost control of the FE, which is fatal for the FE.

A TML congestion alert event is defined as a subscription-requested event. It means PL will subscribe for it if PL requires the congestion alert. During some usage cases or during some period of usage, PL may not be interested to be notified of such event. For instance, in many cases, congestion problem at CE side are not so serious as that at FE side where FE side often risk DoS attacks from redirect data. In this case, CE side congestion alert event may be turned off so as to save CPU resources, while FE side congestion alert is always open.

A TML congestion alert event is a sustained event. When congestion alerts, it will last until its state is changed back to free of danger for congestion. TML should notify PL twice for the whole congestion report. When there is a congestion alert, TML sends one notification to PL, when it is released, TML sends another notification to PL.

TML congestion alert event is assigned with TML event id = 3.

When the event is notified to PL, the following information will be associated with its report:

- TML congestion alert type
  - 1 congestion alert from control message transmission
  - 2 congestion alert from redriect message transmission
  - 3 alert from redirect DoS attack
- event status

1	The event is happening
0	The event is released

Note that it is not restricted that other information may also be associated with the congestion alert report, depending upon individual TML specifications or implementations. For instance, in several cases, it may be of great help to associate with some extra information as which CE/FE link is the congestion located. However, it may be quite difficult for all TMLs and implementations to provide such information, therefore, as a more suitable choice, it is just left each TML or implementation to decide.

More specific definitions of the three types of TML congestion alerts are presented as below:

1. Congestion alert from control message transmission

We define that ForCES control messages are all kinds of ForCES protocol messages but ForCES redirect messages. ForCES message types are identified by the message type in the ForCES message header. ForCES redirect messages are the messages whose types are marked as 'PacketRedirect'[ForCES-PL]. ForCES redirect messages are used to load redirect data between FE and CE.

Congestion alert from control message transmission indicates that TML is in a state where control message transmission channel is in danger of congestion and control messages is becoming hard to be transmitted to peering TML(s). Individual TML specifications or implementations may specifically define the detailed invoking state for the alert.

Because ForCES control messages are vital for ForCES network elements to properly work, the congestion alert from control message transmission is an important signal for PL to timely take actions to secure the network element.

2. Congestion alert from redirect message transmission

This congestion alert is invoked when the TML comes to a risk that redirect messages are congested during transmission. Each TML specification or implementation may specifically define the detailed invoking state for the alert.

ForCES redirect messages that load redirect data between FE and CE, congestion of which may not be so harmful as that of ForCES control messages, but some redirected data are still vital for ForCES network elements to properly work. For instance routing protocol messages are shipped via ForCES redirect messages. A long time congestion of the messages will severely affect actions of routing protocols. Hence, this congestion alert should be used by PL to avoid redirect message congestion as much as possible to improve performance of whole network elements.

3. Alert from redirect DoS attack

As described, ForCES redirect messages ship redirect data. In FEs, redirect data come via FE interfaces from outer networks. This may leave a hole for malicious attackers [RFC3654, RFC3746]. Attackers may try to start a DoS attack by initiating huge amount of redirect data to some specific ForCES CE. It may make some FE TML abnormally busy transporting redirect data. Because TML channels for ForCES redirect messages and ForCES control messages are often intervened in many TML implementations in the same physical links, it may eventually making control messages transmission blocked by redirect messages transmission, making the network element in a denial of service state.

This alert is used to indicate an alert for redirect DoS attack. Each TML specification or implementation will specifically define the actual invoking state or take a mechanism for the alert to be invoked, or make a justification if the TML is considered free from such DoS attack. If a TML has taken some specific mechanism to make straightforward detection of DoS attacks from redirect data, this alert may just be a result report of the DoS detection.

The TML alert from redirect DoS attack may not be sufficient enough for the PL to assure the DoS attack state. PL may synthesize information from other part of the FE, like information from some FE LFBs, to finally decide it. Whereas, this alert has already been an enough signal for PL to go into some urgent state for the whole system security. Approaches should be taken immediately by PL to try to release the alert state so that the FE is not in a risk of losing control from CE.

4.2.TML attributes

TML attributes usually represent those TML parameters that need to be configured by PL. To represent them as TML attributes, PL can then use TML configuration service primitive as defined in Section 5.5 to make operations to the parameters. PL can then also use TML query service primitive defined in Section 5.6 to retrieve the status of the parameters.

Every TML attribute shall be assigned with a unique id for PL to identify the attribute. The id is called TML attribute id.

Followed are descriptions of basic TML attributes that shall be used by all TMLs. Each TML or implementation may provide more detailed definitions of these TML attributes based on the basic descriptions. Individual TMLs or implementations are also allowed to define more specific TML attributes on their own if necessary.

1) TML event handle

This TML attribute is used for PL to set some parameters for individual TML events. Each TML may individually define its data structure for this attribute, whereas, the data structure may have to appear as a list and every element of the list may have to at least include the following information:

o TML event id
  This id acts as an index for individual TML events.

o subscription flag, if described is a subscription-requested TML event
  This flag is to indicate the state for TML event subscription. To subscribe/unsubscribe an event is to set/reset the flag.

The attribute may also include other information for each TML event implementation. For instance, for an implementation that adopts a callback mechanism for event notifications, the attribute may include a callback handle, which is used for PL to tell TML the callback function handle.

The 'TML event handle' is assigned with a TML attribute id = 1.

2) Multicast list

ForCES protocol requires that TML must support for ForCES message delivery in multicast ways. This multicast is defined at ForCES PL level, i.e., to multicast a ForCES protocol message, a multicast 'Destination ID' at the ForCES message header will be specified (See [ForCES-PL] for more details). To support the PL level multicast, TML must be told the members of the multicast, so that the TML can accordingly deliver messages to all the multicast members. A multicast list is used for this purpose. The multicast list comprises a multicast id, which is exactly the multicast 'Destination ID', and numerous associated members, which are also represented by ForCES 'Destination ID's, represented as below:

  multicast list = {multicast id, member1, member2, ... memberN}

When TML is told this multicast list, it means whenever TML is asked by PL to send a ForCES message whose Destination ID is this multicast id, the TML must deliver the message to all destination CEs or FEs whose ids are individually represented by member1, member2, ... , and memberN. Individual TML specifications should define how such multicast list maps to TML transport level multicast mechanisms. For instance, if TML adopts multiple TCP links for this PL level multicast, every member in the multicast list may be mapped to a specific TCP port and an associated IP address. If TML adopts UDP multicast for this PL level multicast, a UDP multicast group with the

same numbers may be constructed and the multicast is mapped to the UDP multicast group.

The multicast list must be set into TML by PL, hence it is defined as a TML attribute. PL sets up the attribute by use of a TML configuration service primitive.

There might be several multicast lists set to a TML so as to construct multiple multicast paths for PL in this TML. The multicast lists may form a table then in the TML. In this case, a multicast id in every multicast list may act as an index for access of the table.

The 'TML multicast list' is assigned with a TML attribute id = 2.

3) Working TML Type

A TML implementation may be capable of several TML transport ways. For example, a TML with IP transport media may be able to support TML schemes as TCP for control messages transmission and DCCP for redriect message transmission, or TCP for control messages and UDP for redirect messages. In this case, it may be helpful for PL to dynamically specify which TML transport scheme to adopt for current work.

The working TML type is used for above purpose. It is defined as an TML attribute.

It should be noted that, in many cases, PL does not have to manage working TML type. TML may more rely on its own management tool or a CE/FE manager for TML type management. As a result, this TML attribute is defined as an optional TML attribute, i.e., it is allowed that a TML may not provide this TML attribute for PL.

Whereas if the attribute is provided, it should include the following information:

o Working TML Type id
  the TML type represented by an id, which is set to the TML for it works in this type. The TML type id may be assigned with one of the following values:
  1 - an IP TML type with the protocol scheme as TCP+UDP, i.e., TCP for control message transmission and UDP for redirect data transmission.
  2 - an IP TML type with the protocol scheme as TCP+DCCP, i.e., TCP for control message transmission and DCCP for redirect data transmission.
  3 - an IP TML type with the protocol scheme as SCTP, i.e., SCTP for both control and redirect data transmissions.
  4 - an Ethernet TML type

5 an ATM based TML type

The 'Working TML type' is assigned with a TML attribute id = 3.

4) TML media specific attributes

An individual TML specification may require PL to configure some extra TML parameters specific to this TML media. If any, the TML specification shall provide detailed definitions for such attributes.

5) Implementation specific TML attributes

An individual TML implementation may require PL to configure some TML parameters specific to this implementation. If any, the individual implementation will provide detailed definitions for such attributes.

4.3. TML capabilities

TML capabilities represent TML abilities or capacities to provide services to PL. A TML capability can only be read by PL via TML query service primitive.

Note that, the TML query service primitive as described in Section 5.6 is used to query status of TML attributes as well as TML capabilities. A TML attribute id or a TML capability id is simultaneously used for the query operation. As a result, TML attribute id and TML capability id should be kept harmonious and unique to each other.

1) Supported TML type

PL may be interested to know what TML transportation type(s) the associated TML can support. A TML implementation may be capable of only one TML transport type or simultaneously several TML transport types. This TML capability indicates the relative information to PL.

Note that, as mentioned before, in many cases, PL does not have to manage TML types. TML may more rely on its own management tool or the CE/FE managers for TML type management. As a result, this capability is defined as an optional TML property, i.e., it is allowed some TML implementations may decide not to provide this information to PL.

Whereas, if the capability is provided, it should include the following information:

- a list of supported TML Type id(s)
  the TML Type id is as defined before.
- configurable indicator

a flag to indicate if the TML is configurable or not for its working type, i.e., if PL can use a working TML type attribute to set the type to the TML.

The 'Supported TML type' capability is assigned with a TML capability id = 10.

5. TML Service Primitives

5.1.Design Principles

The following principles are applied to the PL-TML service primitives design:

1.PL-TML service primitives should hide implementation details regarding reliability, security, multicast, congestion control, etc from PL.

2.PL-TML service primitives should be decoupled from possible changes of ForCES PL layer such as the update of ForCES protocol and ForCES FE model. More specifically, primitives should be avoided to be coupled with ForCES protocol PDU format.

5.2. TML Open

Format:
Result = TMLopen( )

Result:
the returned result; it shall indicate whether the TML open is succeeded or not. Moreover, if not succeeded, an id called 'TML id' and used to identify the TML should be returned by the primitive. If not succeeded, an error code may be returned to indicate the error type.

Parameters:
none

Service Description:
The primitive is for PL to indicate a TML that the PL is going to associate itself with the TML for services and hence the TML should be ready for use. It highly depends upon each TML specification or individual implementations on what a TML should do when it receives this primitive. For some TMLs, this primitive may just act as an indicator that the PL and the TML has been associated, while every thing for providing services has already been there in the TML. For other TMLs, when received the primitive, they may have to do something to make it ready. For example, for a TML that adopts a connectionless path as one of its transmission path, the path may

always be ready for message transportation without any extra setup;

while for a connection-oriented TML path, a TML open or a TML close (see below) primitive may act as an indicator for the TML path to be set or reset. However, it is also possible that some TML specifications or implementations may choose to have such connection-oriented path always ready for use when the TML has been initially booted.

The 'TML id' returned by this primitive is as an identifier for the PL to recognize the TML. Other primitives as described below will use this id to identify a TML for operations. Having defined the TML id, we imply that the usage scenario as one PL being associated with more than one TML will not excluded by any TML specifications, and may be applied in actual implementations.

An important note is, to better synchronize the operations between peering PLs, if a TML has, for any reason, received any PL messages from peering PL before local PL has formally opened the TML, the TML shall discard all these messages.

5.3. TML close

Format:

```
  Result = TMLclose(
                TML id
                    )
```

Result:
the returned result; it shall indicate whether the TML close operation is succeeded or not. Moreover, if succeeded, an error code may be returned to indicate the error type for the failed TML close.

Parameters:
o TML id (input)
  the id of the TML to be closed.

Service Description:
By this primitive, a PL tears down its association with a TML. It highly depends upon each TML specification or implementation on what a TML should do when received this primitive. For some TMLs, this primitive may just act as an indication that the association of the PL and the TML is terminated and nothing more need to be done. For other TMLs, when received the primitive, they may have to manage to make it terminate the association, e.g., by disconnecting a connection with peering TML. However, it is out of scope of this document to have more details specified.

An important note is, to better synchronize the operations between peer PLs, if a TML has, for any reason, received any PL messages from peer PL after local PL has formally closed the TML, the TML shall

discard all these messages.

5.4.TML Configuration

```
Format:
    Result = TMLconfig(
                TML id,
                operation type,
                TML attribute id,
                TML attribute data
                [,optional parameters]
                       )
```

Result:
the returned result; it shall indicate whether the TML configuration is succeeded or not. Moreover, if not succeeded, an error code may be returned to indicate the error type for the failed configuration.

Parameters:

o TML id (input)
  the id of the TML to be configured.

o operation type (input)
  As an input parameter, it specifies the operation type the TML configuration primitive will do. The following operations must be included:

```
        SET   to set data to an attribute in the TML
        DELETE   to delete data from an attribute in the TML or to
totally remove the attribute from the TML.
```

The following operation may be included:

```
        MODIFY   to modify data for an attribute in the TML
```

Individual TMLs or implementations may define other operations if necessary.

o TML attribute id (input)
  the id inputted to TML; it uniquely specifies the TML attribute the primitive is going to operate on. The id is assigned by individual TML attribute definitions.

o TML attribute data (input)
  a data unit that contains data elements to be configured to a TML attribute. Actual data structure of the data unit will be defined by individual TML implementations.

o optional parameters (input or output):
  Individual TMLs or implementations may allow more parameters for the TML attribute configuration. For e.g., some implementations may choose to take an extra timeout parameter to make the primitive as a non-blocking primitive call. This document does not exclude such usages.

Service Description:
This primitive is used by PL to configure attributes of TML according to service requirements made to the TML. TML attributes are basically described as in Section 4.2. Every attribute to be operated is identified by the TML attribute id. The TML attribute data parameter provides necessary data for the operation. It is up to individual TML implementations to specify data structure for the attribute. It may be organized as an atomic data element or a compound data element. Individual TML implementations should provide detailed description on the data structure used for individual TML attributes.

SET or DELETE operations are two basic operation types to a TML attribute operation. In some cases, more operation types may be required so that management to TML attributes may become more portable to users. This is especially useful for management of TML attributes like TML multicast lists. When PL configures multicast lists to TML, it may require some form of operation variations besides general operations as setting a new multicast list or deleting an existing multicast list. For instance, PL may be interested to add a member to, or delete a member from, an existing multicast list. There may be two approaches for each TML implementation to realize this. One is to define more types of operations. The other is to specifically associate attribute data structure definitions with operation types. Below is an example to show that this is feasible:

- operation = SET, data = {multicast id, member1, member2, ...}
  If the multicast list with the multicast id does not exist in the TML, it is to set a new multicast list, or else, to add the new members as listed to the existing multicast list.

- operation = DELETE, data = {multicast id }
  to delete the whole multicast list with the multicast id.

- operation = DELETE, data = {multicast id, member1, member2, ...}
  to delete the members as listed from an existing multicast list, while keeping the multicast list id.

Note that the TML configuration service primitive is not designed to return any attribute status after configured. To check the TML attribute status, a TML query primitive as defined below should be specifically used.

5.5. TML Query

Format:
Result = TMLquery(

TML id,
TML attribute id or TML capability id,

[,optional parameters]
)

Result:
The result includes a returned result and a queried result;

The returned result shall indicate whether the primitive operation is succeeded or not. Moreover, if not succeeded, an error code may be returned to indicate the error type for the failed query operation.

The queried result shall include the queried data if the query operation is succeeded. Each TML implementation shall define the data structure for the queried result data. Each TML attribute or TML capability may all have its specific data structure.

Note that it is not specified and restricted how the queried result should be implemented in reality. Any techniques may be applied for this purpose under condition that the queried data can be transferred back to PL layer. For instance, some implementations may adopt a return of function call for transferring the queried result data, while some others may just adopt a parameter of function call for the transferring.

Parameters:
o TML id (input)
the id of the TML to be operated.

o TML attribute id or TML capability id (input)
the id that uniquely specifies the TML attribute or TML capability the primitive is going to query.

o optional parameters (input or output):
Besides the mandatory parameters as presented, individual TMLs or implementations may adopt more parameters to customize the query operation. For instance, it may be interested to query a multicast list with a specified multicast id, rather than to query the whole existing multicast lists. This may be realized by defining a parameter composed of a multicast id as an index for the query. The primitive may also include a timeout parameter to make the primitive as a non-blocking operation. This document does not exclude such usages.

Service Description:
This primitive is used by PL to query TML attributes or TML capabilities to know their current status. The TML attribute id or TML capability id is used to specify which attribute the primitive is interested to query. Queried data are included in result of the primitive execution. Note that this primitive definition does not specify any technology on how the queried data shall be transferred

from TML to PL, so as to make the primitive definition independent of any specific OS environments or implementation techniques.

5.6. TML send

Format:

```
  Result = TMLsend(
                TML id,
                message destination id,
                message type,
                message priority,
                message length,
                message PDU
                [, timeout]
                [, more optional parameters]
                  )
```

Result:
a returned code indicating if the TML send primitive is successful or failed. If successful, a success code is returned. If failed, an error code indicating the failure type is returned.

Parameters:

o TML id (input)
  the id of the TML to be operated.

o message destination id (input)
  the id indicating the destination of the ForCES PL message to be sent; equal to the destination ID in the protocol message header.

o message type (input)
  the type of the ForCES protocol message to be sent; equal to the message type in the protocol message header.

o message priority (input)
  the message priority of the protocol message to be sent; equal to the priority bits in the protocol message header.

o message length (input)
  the ForCES protocol message length to be sent, equal to the message length field in the protocol message header, representing the whole protocol message length in DWORDS (4 bytes) units.

o message PDU (input)
  Protocol Data Unit for the whole ForCES protocol message, including the message header and the body. Individual implementations may need to further specify the endian way (big-endian or little-endian, etc).

o timeout (input, optional parameter)

This is an optional parameter to optionally specify the primitive as a non-blocking primitive call. The timeout value specifies how long it may wait before abortion of the primitive call. If not adopted, the primitive is executed in a blocking way.

o other optional parameters
Individual TMLs or implementations may specify more optional parameters if necessary.

Service description:
By this service, PL tries to send a message to one (unicast) or more (multicast) peer PLs via the TML. Note that this primitive has explicitly included all information that are necessary for TML to manage transmission of the PL message, therefore, there is no need for the TML to further retrieve more information by reading the PL message body PDU. In this way, it may be decoupled of changes in ForCES protocol PDU (e.g., by the protocol update) from TML services.

The message destination id is used by the TML to map to TML layer transport addresses for the message transmission. This also includes the mapping of PL layer multicast ids to TML layer multicast addresses. Each TML specification should define the way for such mapping.

The message type is used for the TML to infer the requirements from PL level for the message transmission, regarding its reliability, timeliness, security, and congestion control. With this message type, it is easy to recognize PL redirect messages from PL control messages. Individual TML specifications shall define how the message types are mapped to their individual transportation resources.

The message priority is used for the TML to meet the PL requirement for the message transmission priority; it may also be used for TML to meet the requirements for reliability, timeliness, security, and congestion control. Individual TML specifications may define how the priority is mapped to their available transport mechanisms for prioritized timely transmission. Individual TML specifications may also define how the priority is used for other TML requirements.

By use of an optional timeout parameter, the primitive may be applied either in a blocking way or a non-blocking way.

5.7. TML receive

Format:

```
Result = TMLreceive(
                TML id,
                message length,
                message PDU,
```

timeout,

[, optional parameters]
)

Result:
a returned code indicating if the TML receive primitive is successful or failed. If successful, a success code is returned. If failed, an error code indicating the failure type is returned.

Parameters:
o TML id (input)
the id of the TML to be operated.

o message length (output)
the length of the received ForCES protocol message, representing the whole protocol message length in DWORDS (4 bytes) units. It is a parameter output to PL by TML via this primitive.

o message PDU (output)
Protocol Data Unit for the whole ForCES protocol message received, including the message header and the bogy. Individual implementations may need to specify the endian way (big-endian or little-endian, etc). It is a parameter output to PL by TML via this primitive.

o timeout (input)
This is a mandatory parameter for the TML receive primitive. It mandates that the primitive shall work in a non-blocking way. The timeout value specifies how long it will mostly wait before abortion of this time receiving process.

o optional parameters (input or output)
Individual TMLs or implementations may specify more optional parameters for the primitive if necessary.

Service description:
This service is used for PL to synchronously receive ForCES protocol messages from peering TML via local TML. A received protocol message is returned via the parameters. The primitive specifies that it should be implemented in a non-blocking way. It is because that usually such receiving process may take high priority resources and a blocking way may make the system risk more of fatal errors.

Note that a message arrive event as described before can also be used for PL to receive PL messages from TML. The difference is that this TML receive primitive makes PL to synchronously receive messages, while a message arrive event works in an asynchronous way receiving a message. Usually, an asynchronous method exploits more efficiency in terms of CPU resources.

6. Operation Notes

1) multicast

In a ForCES architecture, PL level multicast may be most commonly for a CE to multicast a ForCES protocol message to multiple FEs. Operation steps for the ForCES system to setup this type of multicast may be presented as below:

a. Before a PL level multicast could be established, usually a PL level unicast mechanism should first be established in the ForCES network element. This means a ForCES message should be able to be delivered in a unicast way between CE and FEs before we setup a multicast path.

b. The CE PL (or its application layer) forms a PL level multicast list as defined in 2) of Section 4.2. Note that, because it represents multicast of a CE to FEs, the multicast list shall include a multicast id, the CE id, and a number of member FE ids.

c. The multicast list should be sent to the CE TML by TML configuration service primitive as described by this document. When CE TML receives this multicast list, the TML is responsible to map the multicast list to its TML multicast mechanism.

d. The multicast list may also need to be sent to all FE members of the multicast by use of ForCES protocol configure messages in order for the FEs to know they belong to this multicast group. Note that a multicast list has been defined in FE as an attribute of the FE Protocol LFB [ForCES-PL]. The FEs further send the PL multicast list to their FE TMLs by means of TML configuration primitive. When the FEs TMLs receive this multicast list, each TML is responsible to map the multicast list to its TML multicast mechanism.

e. When a CE PL generates a message with the multicast id as its destination id and sends it to CE TML, the CE TML will use its TML level multicast mechanism to distribute the messages to individual FEs in the multicast group.

f. At the FEs side, when a CE PL message with the multicast id arrives at the FEs TMLs, each TML use its TML multicast mechanism to accept the message, and further deliver it to the FE PL.

Above steps may vary in some way according to different TML types with their different mechanism supporting for TML level multicast.

2) TBD

7. Security Considerations

The risk of being DoS attacked by redirect data has already been addressed by RFC 3654, RFC 3746, the ForCES protocol specification

[ForCES-PL], etc. Prevention of the DoS attack is one of the key

points to secure the ForCES system. This document specified a TML event notification of alert from redirect DoS attack to specifically support a ForCES system to prevent such attack.

TML congestion problem is a broader point of view that may affect performance of a ForCES system greatly. A TML event notification of TML congestion alert is defined for TML by this document, so that TML primitives defined by this document is more capable of improvement of ForCES system performance.

This document does not define any mechanisms for security services like endpoint authentication of FE and CE, message authentication, and confidentiality service This document just reaffirms the requirement for this security service. This is because that this kind of security requirement are all specific to TML specifications of different TML media or individual implementations. Each TML specification shall provide detailed description on how to meet this requirement.

8. Acknowledgements

The authors would like to thank Joel M. Halpern, Huaiyuan Ma, et al for their invaluable comments during evolution of the document.

9. References

[RFC3654] H. Khosravi, et al., Requirements for Separation of IP Control and Forwarding, RFC 3654, November 2003.

[RFC3746] L. Yang, et al., Forwarding and Control Element Separation (ForCES) Framework, RFC 3746, April 2004.

[ForCES-PL] A. Doria, et al., ForCES protocol specifications, draft-ietf-forces-protocol-08.txt, work-in-progress, Mar. 2006.

[ForCES-Model] J. Halpern, E. Deleganes, ForCES Forwarding Element Model, draft-ietf-forces-model-06.txt. work-in-progress, Oct. 2006.

10. Author's Address

Weiming Wang
Zhejiang Gongshang University
149 Jiaogong Road
Hangzhou 310035
P.R.China
Phone: +86-571-28877721
EMail: wmwang@mail.zjgsu.edu.cn

Jamal Hadi Salim
Znyx Networks

195 Stafford Rd. West

Ottawa, Ontario
Canada
Phone:
Email: hadi@znyx.com

Alex Audu
Garland SoftWorx
Garland, Texas
USA

Phone:
Email: alex.audu@garlandnetworx.com

Copyright Statement