ForCES Working Group
Internet-Draft
Expires: April 25, 2004

J. Hadi Salim
Znyx Networks
R. Haas
IBM Research
S. Blake
Ericsson

October 26, 2003

Netlink2 as ForCES Protocol
draft-jhsrha-forces-netlink2-02.txt

Status of this Memo

Copyright Notice

Abstract

This document describes Netlink2, which is an extension of Linux Netlink [RFC3549]. This document is intended as a proposal for the ForCES IETF working group protocol.

ForCES attempts to define a clear separation between the two entities of the NE in order to have them evolve separately as opposed to the current monolithic evolution.

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].

Table of Contents

1. Introduction

The concept of IP control and forwarding separation was first introduced in the early 1980s by the BSD 4.4 routing sockets [Stevens]. The focus at that time was to provide a simple IP(v4) forwarding service and allow the control plane, either via a command line configuration tool or a dynamic route daemon, to control forwarding tables for that IPv4 forwarding service.

The IP world has evolved considerably since then. Linux Netlink [RFC3549], when observed from a service provisioning and management point of view, takes routing sockets one step further by breaking the narrow focus on IPv4 forwarding. Since the Linux 2.1 kernel, Netlink has been providing the IP service abstraction for a few additional services other than classical RFC 1812 IPv4 forwarding.

Netlink was designed with a goal of solving the forwarding and control separation. This means that many of the main issues have been thought through and resolved over the years. In other words Netlink is proven as a protocol addressing separation of forwarding and control. Netlink is also network-ready because it uses packet formating techniques and concepts (e.g., multicast addressing). This, and the availability of publicly running and tested code which is widely deployed, form a major motivator to base Netlink2 on Netlink.

Netlink2 extends Linux Netlink to meet the requirements of the ForCES working group charter for a protocol. Netlink is extended to have a distributed addressing and transport scheme, and missing mechanisms are added to make Netlink2 meet the ForCES protocol requirements [ForCES_REQ].

Netlink2 operates in a mode where knowledge of the NE, its topology, and LFB modeling MAY have already been discovered, or is discovered within the Netlink2 protocol. Netlink2 can operate over a variety of link, network, and transport media. The transport and media includes but is not limited by:

- L2 such as Ethernet, ATM, FR, etc,
- over bus and I/O interfaces such as PCI, HT, PCI-express, etc
- L3 IPV4, IPv6, IPX etc.
- L4 and above such as TCP, UDP, SCTP, DCCP

In the cases where required mechanisms are missing from the underlying media, they are compensanted for by Netlink2 extensions (refer to Section 8.1)

2. Definitions

We use the definitions provided in [ForCES REQ], as well as the following:

Logical Functional Block (LFB): same as Forwarding Engine Components as defined in [RFC3549]. This is a forwarding datapath component in the FE driven by the ForCES protocol in order to achieve a certain service.

Control Element Component (CPC): same as defined in Control Plane Component in [RFC3549]. This is a component in the CE that drives LFB(s) in order to achieve a certain service.

3. Netlink2 Overview

A datapath packet processing service accomplished by an FE is represented as a logical functional block (LFB) in the FE. CE components (CPC) in the CE interact with LFBs over Netlink2 wires and bundles (described in Section 6.2) to configure and manage a certain service. The interactions between LFBs and CPCs are specific to each service and are defined using templates as presented in [RFC3549].

The Netlink2 message is used to communicate between the FE and CPC for configuration of LFBs, LFB events to the CPCs, and statistics or config querying/gathering (typically by a CPC). Other activities include transfer of control packets between FE and CPC.

Netlink2 messages travel between the CPC and LFB over Netlink2 wires which are part of Netlink2 bundles. Netlink2 wires are abstractions similar to GSMP links [RFC3292], albeit without the limitation to ATM VP:VC, Ethernet link, or TCP connection only.

For instance, the IPv4 Forwarding service (called NETLINK_ROUTE) defines a message template for handling IP routes and the message types to insert, remove, or query a route. The routing CPC(s) and the IPv4 Forwarding LFB(s) interact using these message templates and message types over the Netlink2 bundle to execute the IPv4 Forwarding service.

The message types in Netlink2 messages allow the FE to demultiplex messages to the appropriate LFB.

Messages of a certain service destined to a LFB can travel on different Netlink2 wires within the same bundle

Netlink2 by itself constitutes a base ForCES protocol with a set of mechanisms that can be utilized depending on service requirements. For example, for certain messages between the FE and CE, reliability can be enforced at the transaction level by setting the appropriate flags in the Netlink2 message. However, by default, Netlink2 transactions are not acknowledged.

4. Summary of Netlink2 Modifications to Netlink

To conform to the ForCES requirements [ForCES_REQ], the Netlink protocol [RFC3549] is extended in the following respects:

1. Base header modifications, and feature expandability extensions by means of optional header TLVs to accommodate current generic ForCES requirements and to make it possible to add more in the future. This facilitates adding such features as authentication, checksumming, etc., when required.

2. IP and Transport encapsulations to carry Netlink messages.

With these complementary changes to the existing Netlink functionality, Netlink2 fulfills the requirements to become the ForCES protocol.

4.1 Header Modifications

1. PID field redefinition and addition.

   In Netlink, PID 0 referred to the equivalent of the FE (kernel). The equivalent of the CE (user process) was referred by its OS process id.

   In Netlink2, the PID has additional semantics which give it group identity, unicast capability, etc (discussed later in Section 6.3).

   A PID of the unicastPID type is assigned to each FE and CE in the pre-association phase. In this way the CE uniquely identifies the FE and avoids any collision. We maintain the name PID for historical purposes.

   * Destination PID: the PID field is redefined as the Destination PID field. This field identifies the parties on the wire that must process the message.

   * Source PID: this field is introduced in the header to identify the source of the message.

   Different types of PIDs are discussed in Section 6.3.

2. The Length field has been reduced to 16 bits, with length 0 being reserved. The rest of the old 32-bit Length field is now split between a new version field and a new extended flags field.

3. A Version field is introduced in the Netlink2 header. This 8-bit

field is 4 bits major number and 4 bits minor number in the form of major:minor. For Netlink2, this becomes: 0x20.

4. A new Extended Flags field is introduced to take over the remaining 8 bits from the 16-bits taken from the original 32-bit Length field in Netlink. Turning different bits on enables additional new features such as proclaiming the presence of extended TLVs, etc.

5. Netlink2-extension TLVs follow directly after the Netlink2 base header. They are optional and their purpose is to extend the Netlink2 header. Typical use of Netlink2-specific TLVs is to compensate for capabilities lacking in a underlying transport. For example, in an IP network not deployed with IPSEC, the Netlink2-specific authentication TLV could be used to emulate the features provided by IPSEC-AH.

6. There could be more than one IP service configuration template within a Netlink2 message (as opposed to a single service template per netlink message). Implementation experience Section 6.3 has shown embedding multiple service templates improves performance of FE configuration.

Other than these changes, all mechanisms provided by Netlink are sufficient to meet the requirements for ForCES. The reader is encouraged to refer to [RFC3549] as a companion to this one.

4.2 Addressing and Transport Extensions

1. Support for UDP/TCP/SCTP/DCCP transport over unicast/multicast IP (Section 6.1).

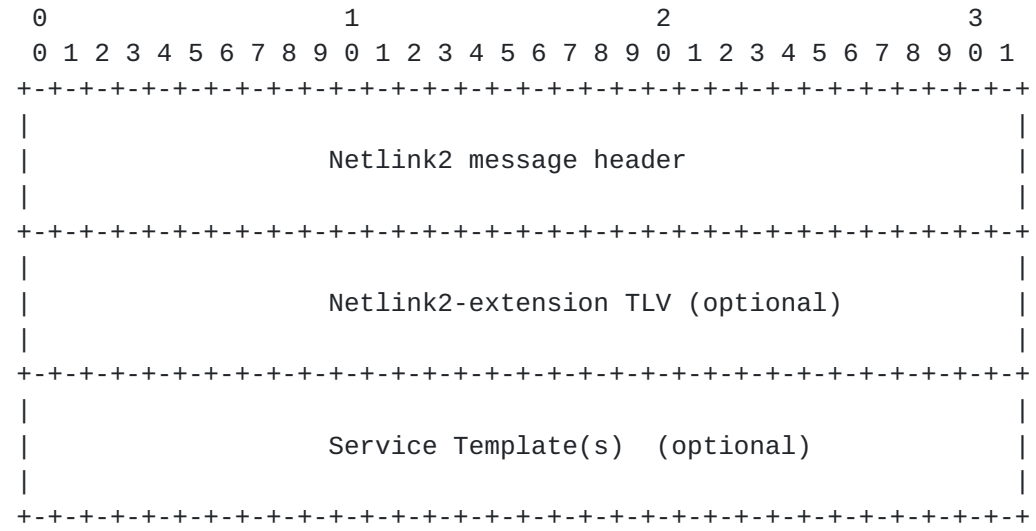
2. Support for bundles (Section 6.2).

3. Message recipient scoping using the Destination PID (Section 6.3).

4. Support for both local scope and global scope addressing (Section 6.4 and Section 6.5).

5. Netlink2 Message Format

There are three levels to a Netlink2 message: The general Netlink2 message header which is mandatory, the Netlink2-extension TLV and service Template(s) which are optional.



Implementation studies [Goutaudier] have shown the above data layout to provide easier parsing while allowing for extensibility (via the optional Netlink2-extension TLV) and scalability (allowing for multiple Service templates).

The Netlink2 message header is generic for all services and contains the command that describes the rest of the message.

The optional Netlink2-extension TLV acts to extend any general missing functionality from the Netlink2 message header. Typically, this would be to allow for compensating for missing underlying transport functionality.

The Service template is specific to a service. As mentioned earlier there could be more than one template per Netlink2 message. Each Service template carries configuration parameters or query requests (CPC->LFB direction) or query responses (LFB->CPC direction). In the case of multiple Service templates, then all the templates MUST be used to execute the same command as defined in the Netlink2 message header. In some special cases the Service template is not used. For example in the case of a Netlink2 SYN, FIN or NOOP command.

5.1 Netlink2 Message Header

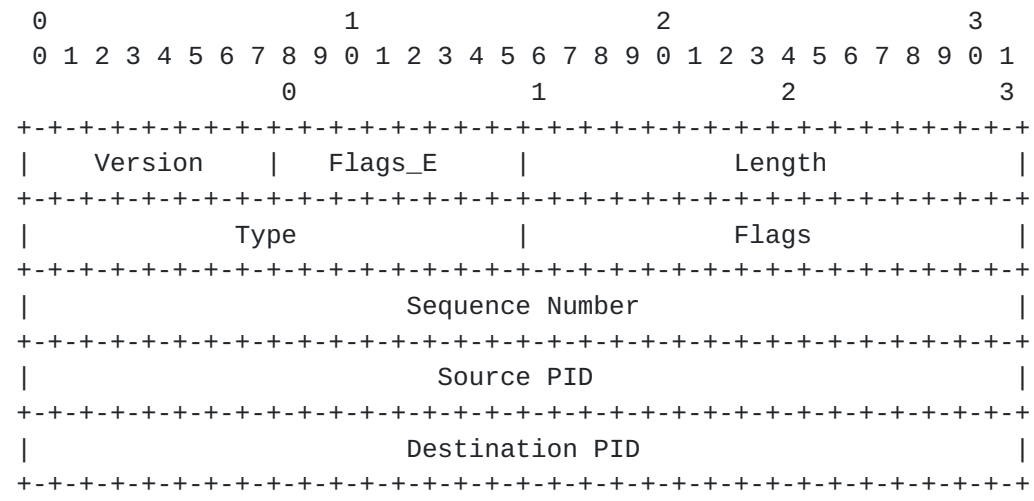
Each Netlink2 message contains a byte stream with a Netlink2 header followed by its associated payload.

A single PDU may contain more than one Netlink2 message. This is referred to as batching. Netlink batching is reused in Netlink2 and allows for messages with different commands (such as adding routes and deleting a QoS policy) to be carried in the same batch PDU.

A Netlink2 message may be split across multiple PDUs if it does not fit into the PDU. This is referred to as a multipart Netlink2 message and is also inherited from Netlink.

For multipart messages, the first and all following headers have the NLM_F_MULTI Netlink header flag set, except for the last header, which has the Netlink header type NLMSG_DONE.

The Netlink2 message header is shown below.



The fields in the header are:

Version: 8 bits

The version field is split into major:minor (4:4 bits) sub-fields. The value for Netlink2 is 0x20.

Flags_E: 8 bits

These are extended flags:

NLM_F_PRIO: Message priority: 1 for high and 0 for low. Additional QoS level set in QoS TLV.

NLM_F_ASTR: Set the ACK strategy: 1 for partial ACKs and 0 for full ACKs

NLM_F_MS: Multiple Service templates are present when this flag is set to 1

NLM_F_EXT: If this flag is set, it implies presence of the extended optional TLVs

Length: 16 bits

The length of the Netlink2 message in bytes including the header.

Type: 16 bits

This field describes the message content. It can be one of the standard message types:

NLMSG_NOOP: message is not executed on LFnot executed on LFB

NLMSG_ERROR the message signals an error and the payload contains a nlmsgerr structure. This can be looked at as a NACK and typically it is from LFB to CPC.

NLMSG_DONE: message terminates a multipart message

NLMSG_SYN: Sent on the first message. Interpreted as a boot message of the sender.

NLMSG_FIN: Sent on the last message. Interpreted as a shutdown message of the sender.

Typically, services specify more message types centered around transactional operations of adding, deleting or querying a command. For example, the NETLINK_ROUTE Service specifies several types for manipulating IPv4 or IPv6 routes such as RTM_NEWROUTE, RTM_DELROUTE, etc.

Flags: 16 bits

The standard flag bits used in Netlink are:

NLM_F_REQUEST: Must be set on all request messages (typically from CE to FE)

NLM_F_MULTI: Indicates the message is part of a multipart message terminated by NLMSG_DONE

NLM_F_ACK: Request for an acknowledgment on success. Typical direction of request is from CPC to LFB.

NLM_F_ECHO: Echo this request. Typical direction of request is from CPC to LFB.

Additional flag bits for GET requests on config information in the LFB:

NLM_F_ROOT: Return the complete table instead of a single entry.

NLM_F_MATCH: Return all matching criteria passed in message content

NLM_F_ATOMIC: This is an atomic or part of an atomic operation (such as two-phase commit).

Convenience macros for flag bits:

NLM_F_DUMP: This is NLM_F_ROOT or'ed with NLM_F_MATCH

Additional flag bits for NEW requests:

NLM_F_REPLACE: Replace existing matching config object with this request.

NLM_F_EXCL: Do not replace the config object if it already exists.

NLM_F_CREATE: Create config object if it does not already exist.

NLM_F_APPEND: Add to the end of the object list.

For readers familiar with BSDish use of such operations in route sockets, the equivalent translations are:

* BSD ADD operation equates to NLM_F_CREATE or-ed with NLM_F_EXCL

* BSD CHANGE operation equates NLM_F_REPLACE

* BSD Check operation equates NLM_F_EXCL

* BSD APPEND equivalent is actually mapped to NLM_F_CREATE

Sequence Number: 32 bits

The sequence number of the message.

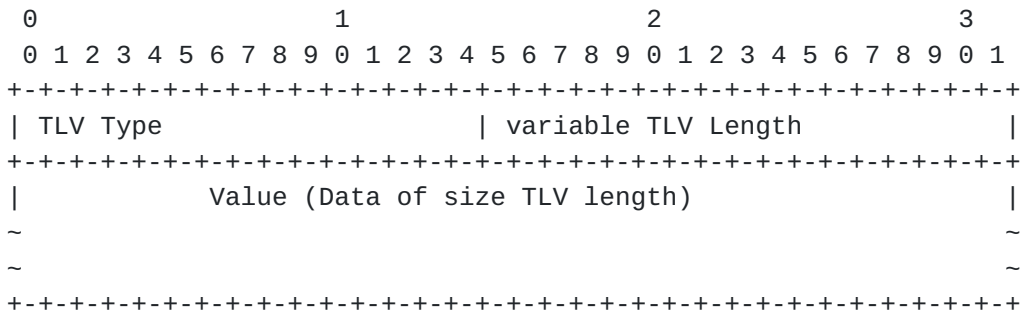
Source PID: 32 bits

The PID of the sender of the message (unicast or logical PID).

Destination PID: 32 bits

The PID of the destination of the message (unicast, logical, or broadcast PID).

5.2 Type Length Value



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.

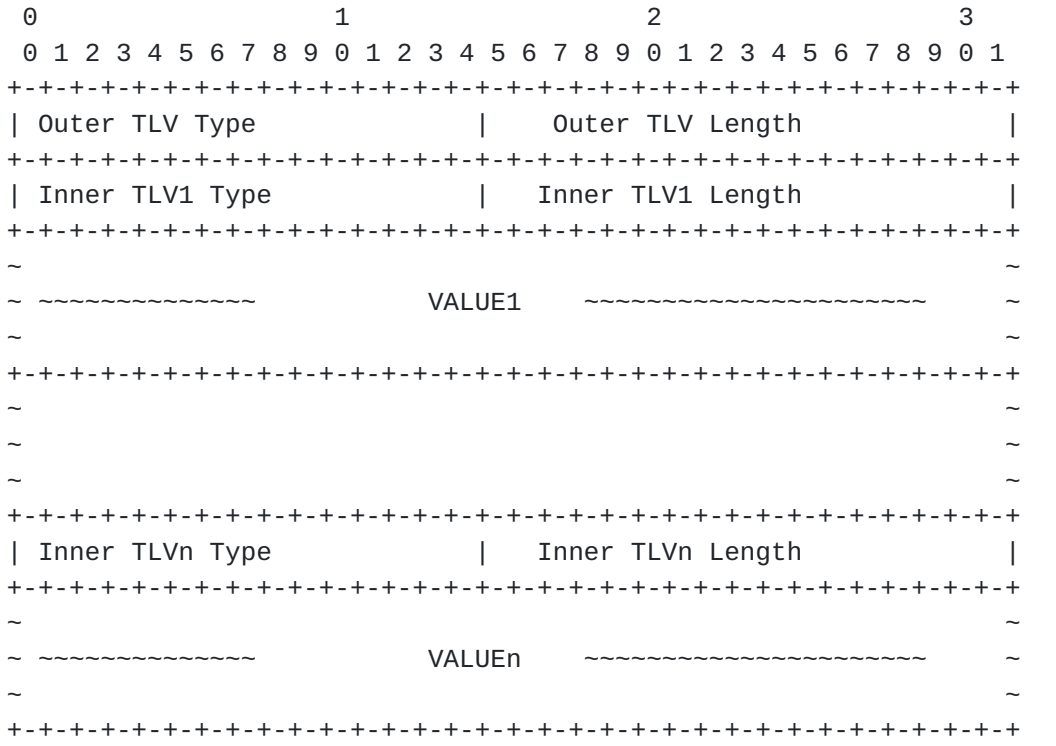
5.3 Encapsulated TLVs

TLV values can be other TLVs. This gives the flexibility of being able to add new attributes when needed. This is important for a protocol such as ForCES for which attributes are expected to vary over a wide range of configurable blocks (CEs, FES, LFBs, etc).

Note that Encapsulated TLVs could be viewed as abstractions that represent dynamic lists of attributes

5.4 Netlink2-extension TLVs

The Netlink2-Extension and Service TLVs are Encapsulated TLVs. They contain their respective TLVs as appropriate in the message being sent.



Outer TLV Type:

This is set to NL2_OPTIONS(0) to indicate the TLV is the Netlink2-Extension TLV. The rest of the possible value types are reserved for future use.

Outer TLV Length:

The Outer TLV Length is the length of everything within the TLV including the Outer TLV Type field , Outer TLV Length, and all the encapsulated TLVs which are treated as the the Outer TLV Value.

Outer TLV Value:

The Outer TLV Value is all the inner TLVs. The figure above shows an outer TLV with n inner TLVs.

Inner TLV type, Length, Value:

These are all just normal TLVs. No assumption is made about their data contents.

6. Addressing and Transport Extensions

We extend Netlink to make it distributed. The focus is on making Netlink2 have a strong local scope view of the world while fitting well into a global scope when the hop distance between the FE and CE increases.

If the network interconnecting the FE(s) and CE(s) is completely hidden from the outside (black-box view), for instance an internal Ethernet segment or a switching fabric in which CE(s) and FE(s) are connected within physical proximity, then communications between FE and CE are assumed to be of a local scope. On the other hand, if communications between FE and CE cross several hops of the network then the scope is considered global

6.1 Transport Methods

The ideal environment for Netlink2 is considered to be a multicast-capable medium with IP above it and with UDP/TCP/SCTP/DCCP running over IP.

On the other hand, Netlink2 is also capable of running directly over L2 (Ethernet for example).

In the case of non-IP, non-multicast-capable environment, extra processing and messaging by the ForCES layer to compensate for services that IP already offers would be needed (eg security, quality of service, fragmentation, etc if underlying transport does not have it).

6.1.1 Why Multicast?

Multicast is considered important to facilitate one-to-many/some communication. For example, a single command from a CE can be multicast to multiple FEs, which eases the scalability requirements mentioned in [ForCES_REQ]. This is discussed in later sections.

When running Netlink2 over non-multicast-capable media, it is expected that mechanisms similar to those used in OSPF NBMA [RFC2328] networks will be put in place.

6.1.2 Why IP?

IP runs on virtually every link layer. Leveraging this fact alone helps deploying the protocol wider and faster.

IP also provides numerous services such as fragmentation and reassembly, prioritization, and security, which are inherent

requirements for the ForCES protocol. This means that to successfully run an alternative to IP requires that similar services be provided by whatever is underneath in order to meet the requirements.

Netlink2-specific optional TLVs can be used to compensate for lacking functionality if running on a network transport other than IP or directly on the link layer.

Netlink already allows the definition of multipart messages with IP segmenting/reassembling when the path MTU is exceeded. When running on top of non-IP media, the Netlink2 message can be limited to not exceed the MTU; the multipart messages facility can be then be used to provide framing for segmenting/reassembling.

The Netlink2-specific Authentication TLV can be used to carry authentication signatures over a transport that does not have this capability.

The Netlink2-specific Checksum TLV can be used to carry checksums over a medium that does not have this capability.

The Netlink2-specific Message Priority TLV can be used to carry prioritization if transports are not capable of making priorities in their headers.

6.1.3 Why UDP/TCP/SCTP/DCCP?

On a local scope, it is assumed that multicast UDP over IP is the preferred mode of operation.

On a global scope it is expected that TCP or SCTP would be used for enhanced reliability and Internet congestion friendliness.

All mentioned protocols provide 16-bit ports, which are further address-demultiplexing points. Also, all three protocols provide checksum capability to enhance integrity of the Netlink2 message. In the case of UDP, the checksum is optional (which fits the model that the local scope is less error-prone than global scope and hence the integrity check could be turned on only when needed).

6.2 The Netlink2 wire and bundle

A Netlink2 wire displays the same behavior as a Netlink wire. It interconnects FEs and CEs in order to support services they jointly offer.

The only conceptual difference between a Netlink2 wire and a Netlink

wire is that whereas the Netlink wire is localized, the Netlink2 wire is distributed.

We also introduce the concept of a Netlink2 bundle. A Netlink2 bundle interconnects a set of FE(s) and/or CE(s) by means of one or more Netlink2 wires. Note that a Netlink2 bundle does not necessarily mean a full-mesh interconnection (see examples later on).

Parties (FEs and CEs) on a Netlink2 bundle share a common configuration, provisioning and event-notification end goals.

A Netlink2 wire MAY be constructed using a multicast connection or a unicast connection or a multiple number of multicast and unicast connections. A wire MUST belong to only one bundle. A bundle may have only a single wire (unicast or multicast). In most cases we believe there will only be one multicast address for a bundle, although scalability issues could require the use of unicast connections in addition.

When a multicast IP address is used, a Netlink2 wire MUST run over UDP - a UDP port is used to uniquely identify the wire. There MAY be multiple wires using the same multicast address as long as they run over different UDP ports.

When a unicast IP address is used, the description of how to connect to an endpoint (CE/FE) is subject to the agreement between the CE and FE. The connection could be directly over IP (Note: need an IP protocol number) or via transport-layer ports (TCP/UDP/SCTP/DCCP).

In both unicast and multicast wires, the necessary parameters (such as IP address and port numbers) can be discovered by the involvement of the FE and CE Managers.

6.2.1 What wires go in a bundle?

Netlink2 provides flexibility to have a bundle of purely unicast wires or multicast wires or a hybrid of both. The decision of what goes into a bundle can be made in the pre-association phase.

A good analogy is to think of a multicast wire as a broadcast link (as is done in Netlink) in which CE(s) and FE(s) are parties attached to that broadcast link.

Depending on the number of FEs and CEs on an NE, a choice of a single multicast wire in the bundle may be sufficient. Multicast allows one-to-some messagging. A single message sent by an originator is seen by all parties on the wire. This simplifies synchronization in an HA environment as well as implementation of the protocol.

The fact that multicast messages are seen by all parties could cause scalability issues as the number of nodes grows. Parties need to filter out messages not destined to them. This can take compute or table resources if filtering is done in hardware. The extra messages also consume unnecessary bandwidth for FE(s) and CE(s) not interested in seeing these messages.

Unicast wires could be used to create point-to-point connections between the parties; when every party is connected to every other party, then this becomes a full mesh.

A full unicast mesh topology removes the need to filter the unnecessary messages but introduces scalability concerns as the number of connections required grows quadratically with the number of parties (FEs and CEs) present. This requires a lot more compute and state information to be maintained at each party. A pure mesh topology also complicates HA because more state must be maintained (for instance, the IP addresses of the CEs and FEs that are active and what their backups are) and therefore needs to perform extra processing to achieve failover. This becomes transparent if multicast is used among all parties.

Netlink2 allows a bundle to have a hybrid of unicast and multicast connections. Note this is a model used by other protocols such as OSPF over broadcast links where the Hello protocol is multicast but responses to LSA updates are unicasted.

We present some examples of Netlink2 bundles:

1. A trivial case is a Netlink2 bundle consisting of a single unicast wire between the CE and FE it interconnects.

2. Multiple FEs and a CE could be interconnected with a Netlink2 bundle using a single multicast connection.

3. In the same example as 2) above, the unicast address of the CE could in addition also be used, for instance, to deliver acknowledgments or notifications from the FEs to the CE, and not be seen by all other FEs. The unicast addresses of the FEs could also be used, for instance, to deliver certain messages only to a specific FE, such as a retransmission of a message in a two-phase commit only to an FE that did not respond.

4. Multiple FEs and CEs could use a wire with two multicast connections: one for all FEs, the other for all CEs, so that messages only relevant to FEs are not seen by CEs and vice-versa.

6.3 Redefining the Netlink PID Semantics

We maintain the name PID for historical purposes and introduce a Destination PID and a Source PID as mentioned earlier.

For every message received by each party on the wire, the destination PID field indicates the recipient of the message. The addressed party could be either a FE or a CE, respectively a LFB or a CPC.

In addition to Netlink2 wires (unicast or multicast) defining the destination of a particular message delivered, the PID types provide further control, namely to define which entity actually has to process the message. So if the bundle uses only a single multicast wire, messages will be heard by all parties on the wire, but only those with a matching PID will actually process these messages. We introduce special- purpose PIDs addressed to specific listeners on the wire.

The following types of PIDs are defined and can be used in the Netlink2 messages. The actual values for the PID of a FE or CE must be the same across all wires of the same bundle and must be established during the pre-association phase.

Default values are given. PIDs must be unique within a Netlink2 wire. They may also be unique within the NE. PIDs are subdivided into two 16-bit subfields named wire and party in the form wire:party.

1. unicastPID: allows one to uniquely address a FE or CE. Each FE/CE must have such a unicast PID. Only the FE or CE assigned to this PID must process an incoming message with such a Destination PID. Other parties MAY silently discard the message. The wire subfield is a unique identifier of the FE or CE. The party subfield acts as a port number: it can for instance be used to further demultiplex a message to the appropriate process in a CE (CPC) or the appropriate LFB in an FE.

   Default value: none.

2. logicalPID: in addition to unicastPID, a FE/CE MAY have zero or more logical PIDs assigned to it. A logicalPID can be used for active-backup pairs of FEs: for instance, the active and the backup FE have the same logical PID or at least the same wire subfield. The wire subfield is an identifier of the group of FEs and/or CEs participating in the group. Pre-association configuration ensures that the same party identifier is not assigned twice to different CPCs or LFBs on the same wire.

   Default value: none.

3. broadcastPID: all parties on all wires must process an incoming message with such a Destination PID. An example of a message that might be broadcast is when a CE is brought down for maintenance.

   Default value: 0xffffffff

4. FEbroadcastPID: all FEs on all wires must process an incoming message with such a Destination PID. Typically a route update from the CE to all FEs. Other parties (CEs) can silently discard the message.

   Default value: 0xffffefff

5. CEbroadcastPID: all CEs on all wires must process an incoming message with such a Destination PID. Other parties (FEs) can silently discard the message.

   Default value: 0xffffdfff

A Netlink2 message must have as Destination PID one of the PIDs types defined above. The Source PID of a Netlink message must be of the unicastPID or logicalPID type. In addition, if the NLM_F_ACK flag is set, then every party processing the message MUST reply with an acknowledgment after processing the message, unless the NLM_F_ASTR flag is used to prevent ACK implosion.

Pre-configured translation tables can be used to map a given PID into the underlying wire in a bundle, i.e., an IP unicast or multicast address.

6.4 Local Scope Addressing and Encapsulation

At a local scope, the preferred addressing used for a wire is a UDP port on top of a multicast IP address.

Multiple wires can run on one multicast address with further demultiplex level based on the UDP port.

The wire addressing parameters MAY be discovered during the pre-association phase.

6.5 Global Scope Addressing and Encapsulation

When addressing a non-local scope the Netlink2 message is encapsulated over a transport header and shuttled to the remote end where it is decapsulated and run as if originating from the local scope of that remote end. The global scope addressing could use any

transport protocol configured (SCTP, UDP, TCP or DCCP) as agreed upon in the pre-association phase.

This can be viewed as extensions of the local scope wires.

7. Protocol Architecture

7.1 Protocol Phases

ForCES in relation to NEs involves three phases: the Pre-Association phase, the association phase where the ForCES protocol operates, and a termination phase where a party in the relationship leaves a bundle.

7.1.1 The Pre-Association Phase

In a simple setup, this phase is static. All the parameters for the association phase are well known (example multicast groups for each Netlink2 wire in a bundle, etc.).

Vendors may use their own proprietary service discovery protocol. As minimum, we assume a static configuration. In fact, although ForCES mandates a minimal set of capability discovery, Netlink2 will also operate in a mode where such capability discovery is done in pre-association phase. In that case, the FE Manager and the CE Manager agree on all the parameters and clearly articulate topology and other information to each other in the pre-association phase.

On completion of the Service Discovery phase, the FEM will have established contact with the appropriate CEM component. Initialization and Authentication will be complete at this point. Both the FE and CE know how to connect to each other for configuration, accounting, identification and authentication purposes. Both sides are also knowledgeable of all necessary protocol parameters such as timers, etc. All capabilities may also have been discovered at this point.

7.1.2 The Association Phase

In this phase, the FE and CE components cooperate to deliver the IP service. The CE component might be registered (in the pre-association phase) to receive FE-specific services (such as link events). Essentially, in this phase, the service is provisioned and executing. The FE component might continuously get updates from the control plane component on how to operate the service (for example, the IPv4 forwarding route additions or deletions).

The association phase is where Netlink2 operates as the ForCES protocol.

On startup, the FE connects to the bundle(s) to which the CE is connected, using procedure defined in Section 8.3.1. The controlling CE will either admit the FE into the NE or reject it.

Once granted access into the NE, the FE is continously updated or queried. The FE may also send async event notifications to the CE. This continues until a termination is initiated by either the CE or FE.

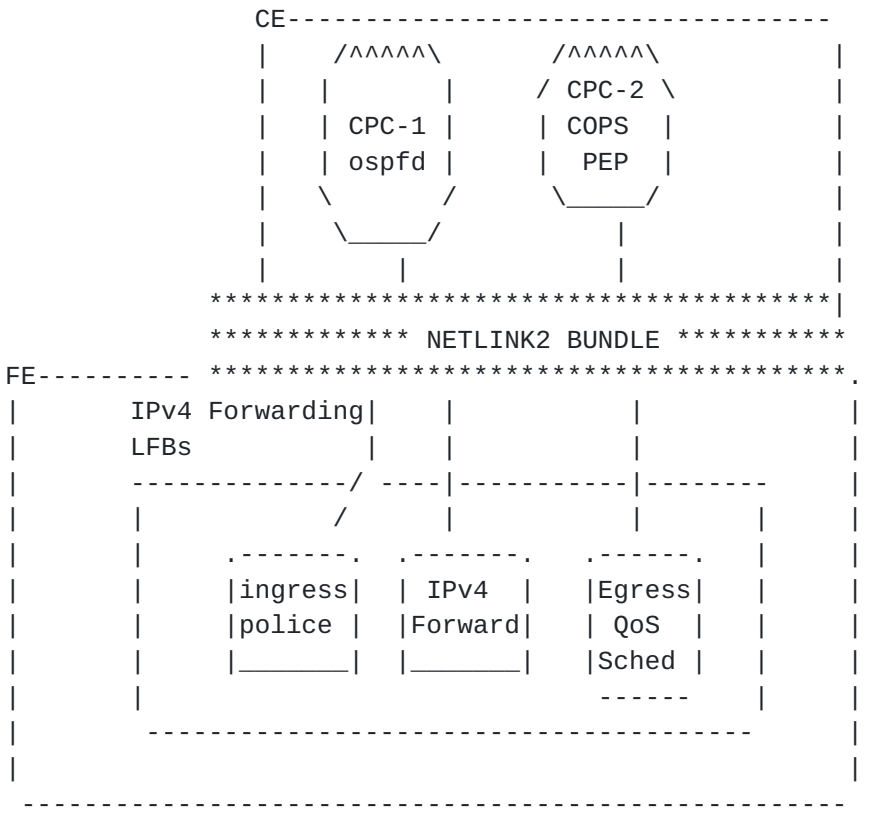
7.1.3 Service Termination

Service termination could be issued by either component of the service abstraction.

FE or the CE initiating the termination will issue a FIN command

7.2 Protocol Logical Model

In the diagram below we show a simple LFB-CPC logical relationship. We use the IPv4 Forwarding LFB as an example.



Netlink2 logically models LFBs and CPCs in the form of service blocks interconnected to each other via a Netlink2 bundle.

Acknowledgements and responses to messages do not have to be sent

onto the same wire from which the triggering messages came from but MUST be sent on the same bundle to the same originating PID. For instance, a wire interconnecting a CE with multiple FEs using a multicast address could be used to send route updates from the CE. On the other hand, independent unicast wires from each FE to the CE could be used to send back route events or acknowledgments. Note that sequencing is done per wire and Source PID, and ACKs can travel back on any wire of a bundle.

The Netlink2 wire can be shared or be specific to a service. There can be multiple Netlink2 wires bundled in a bundle carrying messages of the same service. In order to reduce (for example to avoid extra processing) or restrict the messaging accessible for partitioning or security reasons, additional Netlink2 wires can be used. A possible partitioning is a Netlink2 bundle per service. In the example above the IPv4 Forwarding LFB would be considered a service.

Assuming capabilities have been discovered during the pre-association phase (between the FEM and CEM), blocks (CPCs or LFBs as illustrated above) connect to the agreed wires on the Netlink2 bundle, and listen to receive specific messages. CPCs may connect to multiple Netlink2 wires if it helps them to control the service better. All blocks (CPCs and LFBs) dump packets on the Netlink2 wires.

LFBs or CPCs join Netlink2 wires and listen to messages of interest for processing or monitoring purposes.

All messages addressed to the LFB (for example the IPv4 forwarding LFB illustrated above) will have the FE PID agreed upon by both the CE and the FE at the pre-association phase.

LFBs (as well as CPCs) also process messages with the broadcast PIDs. They may also process messages destined to other LFBs (as well as CPCs) for availability synchronization purposes.

A further demultiplexing point is the command type in the Netlink2 message. Each of the LFBs (e.g., the ingress police LFB above) knows how to respond to a specific command-set as defined by the Netlink2 message type.

7.3 Service Addressing

Connecting to a service is achieved by connecting to a defined Netlink2 bundle by both the CPC and LFB. This Netlink2 bundle is derived in the pre-association phase.

A service would typically be constrained to a specific Netlink2 bundle.

Connecting to a service is followed (at any point during the lifetime of the connection) by either issuing a service-specific command mostly for configuration purposes (from the CPC to the LFB) or for statistics collection. The LFB could also send event announcements to the CPC or respond or queries issued by the CPC.

7.4 Service Templates

LFBs throw events and are configured and queried by using service templates.

Refer to the Netlink document [RFC3549] as well as Section 8.4 for the different templates used for different LFBs that fit within the current scope of the ForCES charter.

7.5 Mechanisms for Creating Protocols

Mechanisms for reliable or non-reliable protocols creation are provided. In addition, mechanisms for facilitating availability are embedded in Netlink2.

7.5.1 Building Reliable Protocols

By default the Netlink2 header flags NLM_F_PRIO and NLM_F_ACK are not set so that Netlink2 messages are sent with a lower priority and do not require acknowledgements.

One could create a reliable protocol between an LFB and a CPC by using the combination of sequence numbers, ACKs and retransmit timers. Both sequence numbers and ACKs are provided by Netlink2. Timers are provided by the operating system or hardware.

Prioritization is an orthogonal mechanism to reliability. When a node runs out of resources, a message sent with a higher priority will get preferential treatment. For instance, if a FE has only enough memory to allocate one message in response to a message from the CE and it has to choose between one of two messages to respond to, then it will use that memory for the request which was sent with the higher priority. This also applies to other resources such as computing cycles and bandwidth. In other words, the NLM_F_PRIO is more than only the classical bandwidth prioritization of packets on a link.

Another orthogonal mechanism provided by Netlink2 is the ACK strategy which is selected by the NLM_F_ASTR flag.

We define two types of acknowledgement strategies:

1. partial ACKs (using multicast ACK slotting and damping techniques [XTP]): receivers multicast an ACK after a random time if they have not yet seen an ACK sent by another receiver. This limits the number of ACKs returned to the source of the message and improves performance. For messages which a CE sends to a group of FEs partial ACKs imply that anyone of the FEs generating an ACK back is sufficient to deem the message was delivered.

2. full ACKs: each receiver sends an ACK back to the source. This allows the source to immediately detect problems with receivers. In two-phase commits it is important that all FEs respond so that the full ACKs strategy should be used.

7.5.2 Building Availability

A protocol component or an application could passively listen to Netlink2 commands and events within one or several Netlink2 wires. Doing so allows a very simple way of building complex applications which are aware of all service components that affect them for HA reasons.

To ensure transparent CE or FE redundancy for certain services, it is sufficient to ensure that the backup CPC/LFB is always attached to the same wires to which the active CPC/LFB is attached, so that the backup CPC/LFB receives all messages destined to the active CPC/LFB (whatever PID they are sent to) as well as all messages originating from the active CPC/LFB.

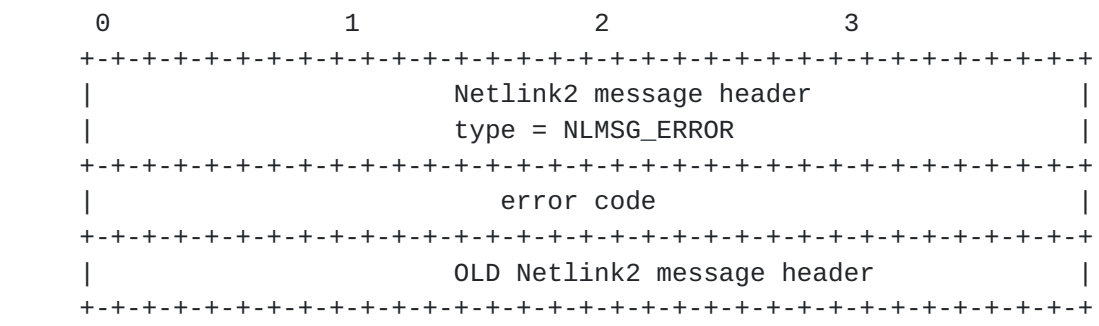
One could create a heartbeat protocol between the LFB and CPC by using the ECHO flags and the NLMSG_NOOP message(Section 8.3.5). The heartbeat, in addition to listening to FE or CE events, could be used to facilitate takeover.

This topic is beyond the scope of ForCES and will not be discussed further here. Note, however, that Netlink2 has the mechanisms required to enable this when required.

7.5.3 The ACK Netlink2 Message

This message is actually used to denote both an ACK and a NACK. Typically the direction is from LFB to CPC (in response to an ACK request message). However, CPC should be able to send ACKs back to LFB when requested. The semantics for this are IP service specific.

```
  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
```



Error code: integer (typically 32 bits)

An error code of zero indicates that the message is an ACK response. An ACK response message contains the original Netlink2 message header that can be used to compare against (sent sequence numbers, etc).

A non-zero error code message is equivalent to a Negative ACK (NACK). In such a situation, the Netlink2 data that was sent down to the kernel is returned appended to the original Netlink2 message header.

7.5.4 Batching

As mentioned earlier (repeated here for clarity) Standard Netlink multi-message batching looks as follows:

NLMSG:NLMSG:NLMSG....

where NLMSG is a Netlink2 header and its associated payload.

This has the advantage of allowing inter-mixing of multiple commands (example adds/deletes) generally in a request from CE->FE. It is also useful for batching multiple events from the FE->CE.

Additionally, studies from [Goutaudier] have motivated batching of Service Templates within a single Netlink2 messages. Recall, a Netlink2 message looks like:

NLMSGHDR:OET:ST

where NLMSGHDR is a Netlink2 header, OET is the optional extension TLVs and ST is the service template.

The template extension now looks like:

NLMSGHDR:OET:ST:ST:ST.....

In other words there are multiple service templates that can fit

within the same message. There are caveats with such a batching scheme since only one ACK may be sent for a whole batch, it implies that it is difficult to know which service configuration failed. In a close proximity, low error rate link batching in this mode should allow for high throughputs for configurations while reducing the number of ACKs back.

7.5.5 Atomicity and Ordering of Transactions

In a two-phase commit messages are bound into a relationship. The first and all following headers have the NLM_F_MULTI Netlink2 header flag set, except for the last header, which has the Netlink2 header type NLMSG_DONE. Typically, in netlink, the NLMSG_DONE shows up in separate PDUs to define a commit.

Atomicity of a transaction including that of a batch is achieved by using the NLM_F_ATOMIC flag. Use of the NLM_F_ATOMIC is expensive because it may necessitate the locking of access to tables (depending on the implementation.

8. Putting together the base protocol for WG charter

The design approach taken for Netlink2 protocol is to avoid over featuring the protocol and focus on the requirements under the current WG charter. Although Netlink2 could be used for CE-CE or FE-FE communication this is not discussed in this document to avoid complexity. Additionaly although Netlink2 provides the minimal required attribute discovery, it will work with existing proprietary or open protocols which exist to discover such attributes.

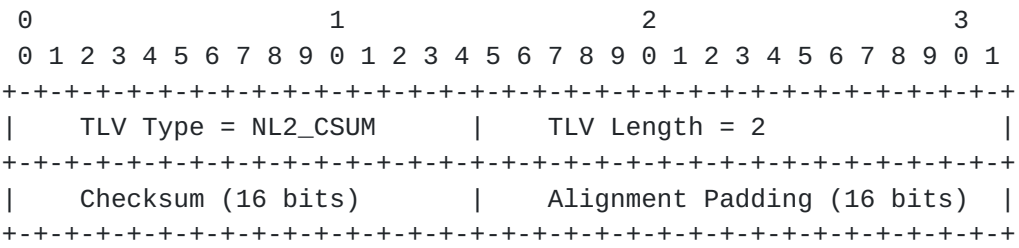
8.1 Netlink2-Extension TLVs

Netlink2-Extension TLVs are mostly used to compensate for the underlying transport not having mechanisms needed by Netlink2.

8.1.1 Authentication

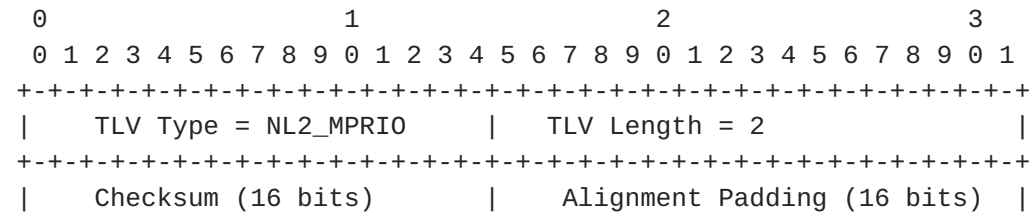
[TBD]

8.1.2 Checksum



This TLV is optional. To compute the correct checksum, an implementation MUST add the optional checksum TLV to the Netlink2 message with the initial checksum value of 0 and compute the checksum over such a Netlink2 message. Refer to [RFC3358] for details on the Checksum TLV.

8.1.3 Message Priority



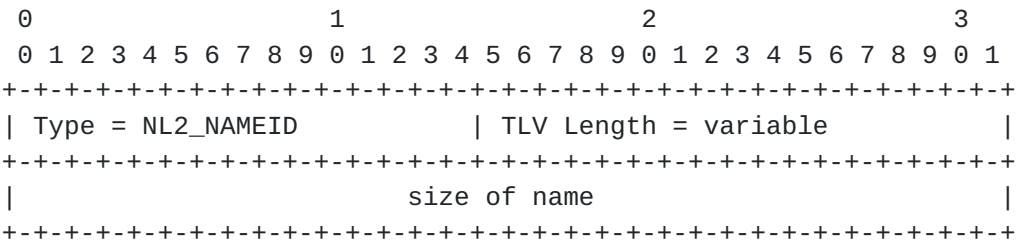
This TLV is optional. It is used if the network does not support prioritization. This field is used to indicate priorities to the remote end.

8.1.4 SYN COOKIE

TBF

TLV_TYPE = NL2_COOKIE.

8.1.5 Name ID



This TLV is optional. It is used to identify a name that a CE or FE wishes to be known as. Typically exchanged with SYN messages.

8.2 LFB and FE Attributes and discovery

In the association phase the CE queries the FE to determine its capabilities. These may include the FE-FE topology, the initial LFB topology for the FE, constraints on how the LFB topology can be modified (if possible), etc. A schema for representing FE and LFB attributes and capabilities is being defined in [ForCES Model]. Appropriate Netlink2 TLVs will be defined to convey the identified parameters as the model work progresses.

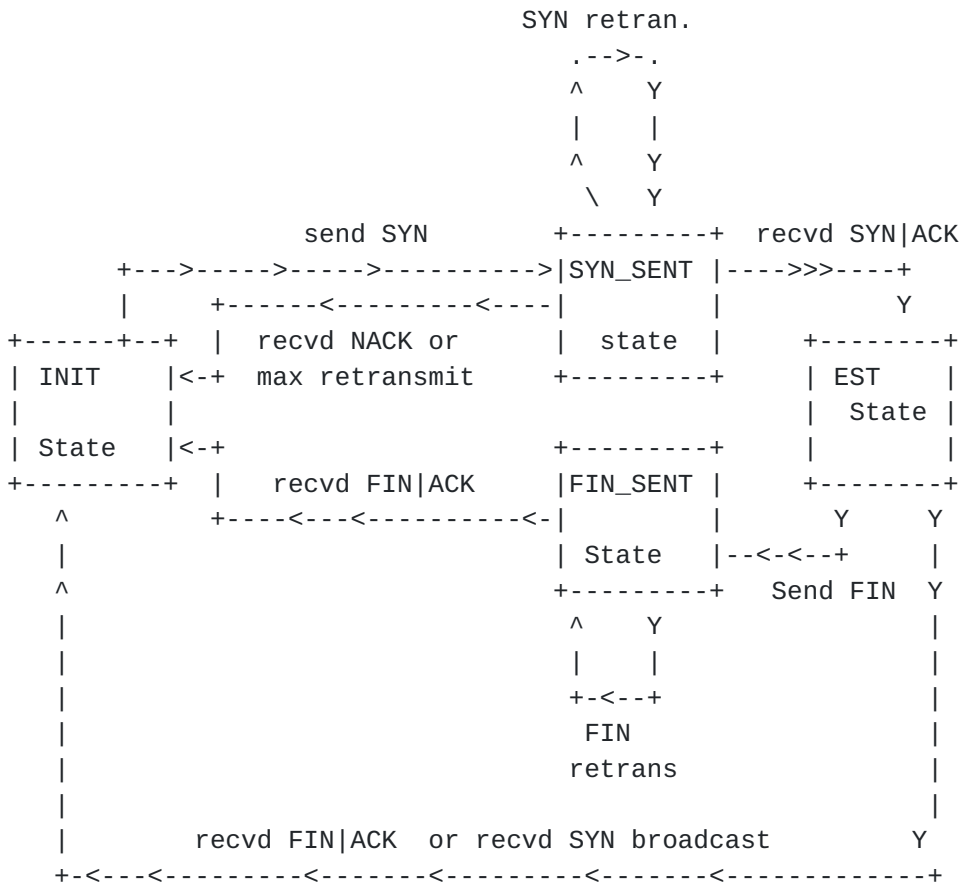
8.3 NE creation

The FE and CE Managers communicate to decide communication parameters and rules that are to be used in the transaction between the CE and FE.

Using the agreed on parameters, the FE attempts to join the NE. The CE may reject the FE or allow it to join. The FE then communicates to

the FEM to inform it of the decision. Note that we do not discuss the FE-FEM or CE-CEM interfaces in this document as it is beyond the scope of ForCES.

8.3.1 FE State transitions



INIT state:

When the FE is started (by FE manager or otherwise) it goes into the INIT state. At this point the FE has been informed by the FE Manager of the following (based on current implementation):

- the bundle to join,
- its PID,
- the PID of the CE,

- the number of retries for the SYN transmission and the SYN timer,
- and the number of retries for the FIN transmission and the FIN timer value.

The FE Manager would also instruct the FE to be either active or passive. Although this is beyond Forces charter, the active/passive setup description is introduced here to describe one way to achieve redundancy. Netlink2 does not mandate how redundancy is achieved. Netlink2 imposes that FE redundancy is the role of the FE plane as such netlink2 is designed so that the CE has no knowledge of FE redundancy. This greatly simplifies the protocol.

After internal initialization, the FE sends a SYN message with the ACK flag on. The message will contain Netlink2-extension TLV of type NL2_NAMEID. The NL2_NAMEID TLV will contain the name the FE wishes to be known as. The FE then enters the SYN_SENT state.

A FE could passively monitor the state of one or more FEs and synchronizes their state and communication data with the CE. The end goal of a passive FE is to act as a backup for the FE whose activities it is monitoring. The monitoring is trivial to achieve if multicast is used. The synchronization may also happen via a FE-FE protocol or via the FE Manager. A passive FE may be called on by the FE manager to take over the functionality of the FE it is monitoring.

SYN_SENT state:

The FE fires the SYN timer and waits for a response from the CE. Two events could happen:

1. The timer expires. If the number of retries has not reached the maximum allowed value, then the SYN is retransmitted and timer restarted. If the maximum number of retries has been reached with the last SYN transmission then the FE notifies the FE manager and goes into INIT state.

2. a packet is received from the CE:

    * A NACK packet to the sent SYN packet. Action: cancel the timer, inform the FE manager on the rejection reasons and go into INIT state.

    * an ACK packet to the sent SYN packet. Action: update the FE manager and go into EST state.

EST state:

This is the established state where normal Forces communication starts.

Several events may force the FE to transition out of the EST state:

1. the FE manager requests it to. In this case the FE will issue a FIN with an ACK request to the CE and transition to the FIN_SENT state.

2. The CE asks it to leave. This is considered a reset of the FE. The FE receives a FIN from the CE to inform it to leave. The FE immediately informs the FE manager, sends a FIN and goes into INIT state.

3. The CE restarts and sends a broadcast SYN. This may be caused by either the CE manager restarting the CE to clear its state or a result of the CE dying and being restarted. Control of restarting of the CE and association to the CE manager is out of scope for ForCES. Upon receiving the broadcast SYN, the FE assumes the CE has no knowledge of any state the FE is in and transits into the INIT state after informing the FE manager.

Additionaly not discussed here are optional heartbeats from the CE to FE. If the CE doesnt see heartbeats after a timeout period then the transition to the INIT state will be made.

FIN_SENT state:

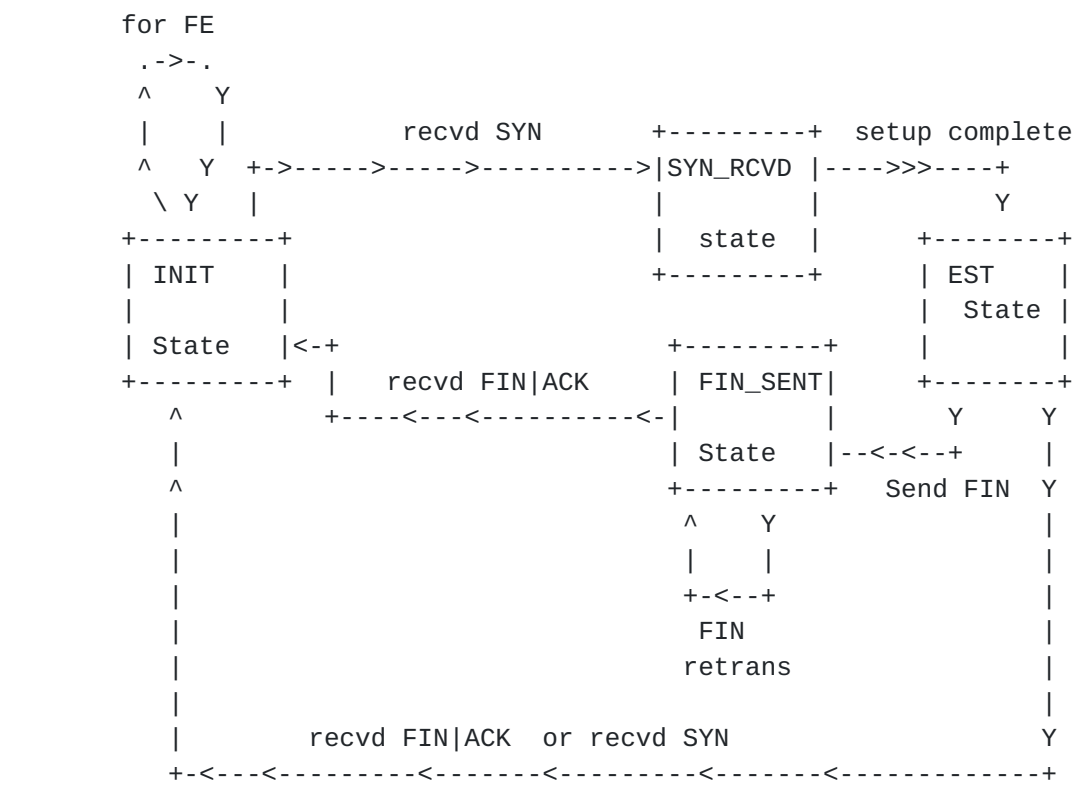
Two events could happen:

1. The timer expires. If the number of retries has not reached the maximum allowed value then the FIN is retransmitted and timer restarted. If the maximum number of retries has been reached with the last FIN transmission then the FE notifies the FE manager and goes into INIT state.

2. a valid FIN|ACK packet is received from the CE. Action: cancel the timer, inform the FE manager and go into INIT state.

8.3.2 CE view of FE State transitions

This is per FE information on the CE side.



INIT state:

When the CE Manager informs the CE of a FE, basic state information is created for the FE and it is placed into the INIT state. At this point the CE has been informed by the CE Manager of the following:

- the bundle the FE will join,
- its PID that the FE is going to use to refer to tthe CE,
- the unicast PID of the FE.
- the number of retries for the SYN transmission and the SYN timer
- the number of retries for the FIN transmission and the FIN timer value.
- the expected timeouts before the FE joins and number of such timeout to wait for the FE.
- whether the FE is interested in restart information if available (refer to the FIN_SENT state)

The CE fires a timer waiting for the FE to join. Two things could

happen:

1. The timer expires. If the number of retries for waiting for the FE to join has not reached the maximum allowed value then the timer is restarted. If the maximum number of retries is reached then the CE deletes the FEs state info and informs the manager.

2. A valid SYN packet is received from the FE. The CE transitions into the SYN_RCVD state.

SYN_RCVD state:

In this state the CE will do any necessary processing to prepare for the FE to be admitted into the NE. The CE issues a SYN|ACK and moves into the EST state.

EST state:

This is the established state where normal Forces communication starts. Several events may force the CE to transition out of the EST state:

1. the CE manager requests it to. In this case the CE will issue a FIN with an ACK request to the FE and transition to the FIN_SENT state.

2. The FE leaves. This is considered a reset of the FE. The FE sends a FIN to the CE to inform it it is leaving. The CE immediately sends a FIN ACK and notifies the CE manager. Transition is made to the INIT state.

Not discussed here is use of hearbeats or other events (eg link down) to transition to the INIT state on discovery that the FE is dead.

FIN_SENT state:

The CE fires the FIN timer and waits for a response from the FE.

Two events could happen:

1. The timer expires. If the number of retries has not reached the maximum allowed value then the FIN is retransmitted and timer restarted. If the maximum number of retries has been reached with the last FIN transmission then the CE notifies the CE manager and goes into INIT state.

2. a valid FIN|ACK packet is received from the FE:

* cancel the timer, inform the CE manager

* transition to the INIT state.

For states that transition to the init state observe that if the FE comes back and joins before the FE expiry time, its LFB state(s) would still be intact and maybe resent to it (The restart policy is agreed on at pre-association time). OTOH, the state will be garbage collected if no SYNs from the FE are seen within the period (or if they are new ones seen but FEM-CEM interface indicates no interest in the restart data).

8.3.3 SYN Message Format

A SYN message contains a base Netlink2 header (refer to Section 5.1) with the appropriate flags followed by the Extension TLV Name ID (refer to Section 8.1.5). The Name ID will have the name the FE wishes to be refered to.

8.3.4 FIN Message Format

A FIN message contains a base Netlink2 header with the appropriate flags (refer to Section 5.1).

8.3.5 NOOP Message Format

A NOOP message contains a base Netlink2 header with the appropriate flags (refer to Section 5.1) set. The NOOP carries no execution message and therefore no operations on LFBs are carried out as a result of receiving it. The flags of the message are still relevant.

A standard use of NOOP message is for heartbeats. A CE may send LFBs keepalive messages using NLMSG_NOOP command. When requesting for replies, the CE sets the NLM_F_ECHO flag on to get the message sent back to it as is (essentially loopback of exact same message sans the ECHO flag).

8.4 LFB and FE Service Templates

In this section we describe Service Templates used to configure FEs and LFBs as well as for async event notification as required by the ForCES WG charter.

Some of these message templates are already described in the Netlink document ([RFC3549]) but are repeated here for clarity.

A feature of Netlink2 is that the same message template is used in configuration, querying or events. In the CE->FE direction configuration commands embedding Service Templates described in this section are used to configure (Add or delete a policy for example). In the FE->CE direction, the templates are used to give back query responses or throw events at the CE (on a per-LFB basis).

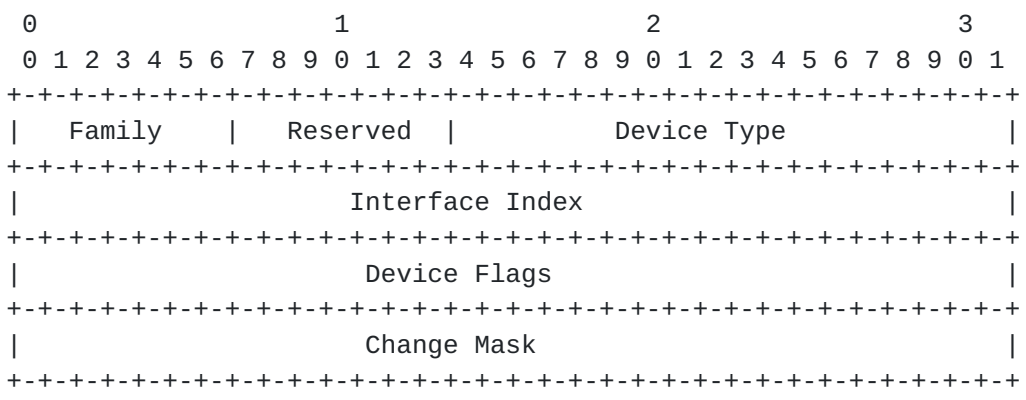
As noted earlier, a single Netlink2 message may carry multiple service templates if the NLM_F_MS flag is set. This is not restricted to the config (CE->FE) only but also extends to responses or events (FE->CE).

8.4.1 Physical Port and Address Functions

[TBF]

8.4.1.1 Interface Service Template

This is very close to what the Port LFB is defined to be in the Model draft. Its expressive semantics are sufficient to define a physical port (regardless of the underlying physical links), virtual interface, etc.



Family: 8 bits This is always set to AF_UNSPEC.

Device Type: 16 bits This defines the type of the link. The link could be Ethernet, PCI, a tunnel, etc.

Interface Index: 32 bits Uniquely identifies interface.

Device Flags: 32 bits

IFF_UP Interface is administratively up.

IFF_BROADCAST	Valid broadcast address set.
IFF_DEBUG	Internal debugging flag.
IFF_LOOPBACK	Interface is a loopback interface.
IFF_POINTOPOINT	Interface is a point-to-point link.
IFF_RUNNING	Interface is operationally up.
IFF_NOARP	No ARP protocol needed for this interface.
IFF_PROMISC	Interface is in promiscuous mode.
IFF_NOTRAILERS	Avoid use of trailers.
IFF_ALLMULTI	Receive all multicast packets.
IFF_MASTER	Master of a load balancing bundle.
IFF_SLAVE	Slave of a load balancing bundle.
IFF_MULTICAST	Supports multicast.
IFF_PORTSEL	Is able to select media type via ifmap.
IFF_AUTOMEDIA	Auto media selection active.
IFF_DYNAMIC	Interface was dynamically created.

Change Mask: 32 bits Reserved for future use. Must be set to 0xFFFFFFFF.

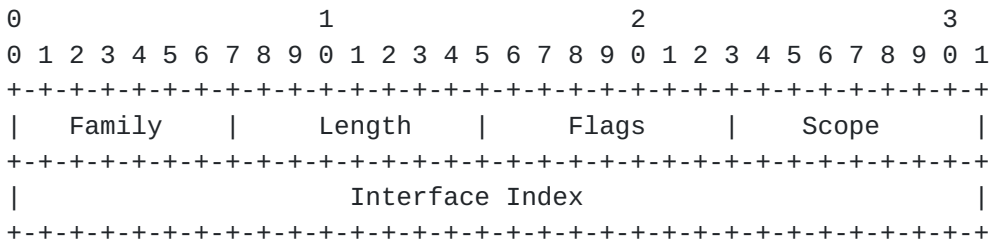
Applicable attributes:

IFLA_UNSPEC	Unspecified.
IFLA_ADDRESS	Hardware address interface L2 address.
IFLA_BROADCAST	Hardware address L2 broadcast address.
IFLA_IFNAME	ASCII string device name.
IFLA_MTU	MTU of the device.
IFLA_LINK	ifindex of link to which this device is bound.
IFLA_QDISC	ASCII string defining egress root queuing discipline.
IFLA_STATS	Interface statistics.

Netlink message types specific to this service:
RTM_NEWLINK, RTM_DELLINK, and RTM_GETLINK

8.4.1.2 Address Service Template

The expressive semantics of this template are sufficient to define addressing for a port LFB (physical or virtual interfaces) including secondary addresses. Although the focus is on IPv4 and IPv6, the template could be used to configure IPX etc. We only focus on IP.



Family: 8 bits
Address Family: AF_INET for IPv4; and AF_INET6 for IPv6.

Length: 8 bits
The length of the address mask.

Flags: 8 bits
IFA_F_SECONDARY For secondary address (alias interface).

IFA_F_PERMANENT For a permanent address set by the user. When this is not set, it means the address was dynamically created (e.g., by stateless autoconfiguration).
IFA_F_DEPRECATED Defines deprecated (IPv4) address.
IFA_F_TENTATIVE Defines tentative (IPv4) address (duplicate address detection is still in progress).
Scope: 8 bits
The address scope in which the address stays valid.
SCOPE_UNIVERSE: Global scope.
SCOPE_SITE (IPv6 only): Only valid within this site.
SCOPE_LINK: Valid only on this device.
SCOPE_HOST: Valid only on this host.

Applicable attributes:

IFA_UNSPEC Unspecified.
IFA_ADDRESS Raw protocol address of interface.

```
IFA_LOCAL       Raw protocol local address.
IFA_LABEL       ASCII string name of the interface.
IFA_BROADCAST   Raw protocol broadcast address.
IFA_ANYCAST     Raw protocol anycast address.
IFA_CACHEINFO   Cache address information.
```

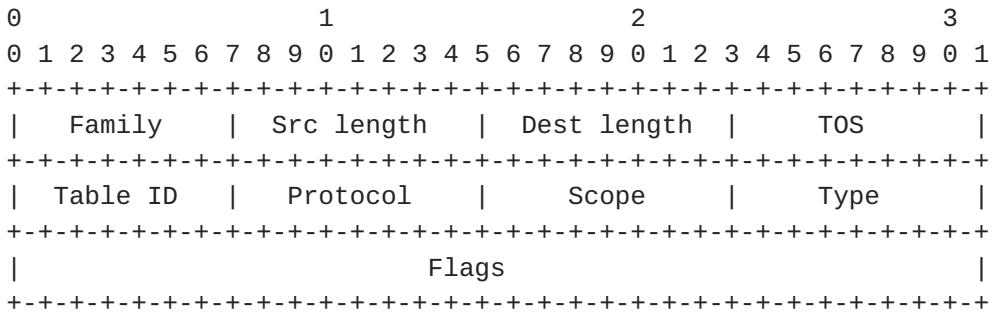
Netlink messages specific to this service: RTM_NEWADDR, RTM_DELADDR, and RTM_GETADDR.

8.4.2 IPv4 and IPv6 L3 Forwarding Functions

In this section we describe two LFB templates necessary for IPv4 and V6 L3 forwarding control.

8.4.2.1 IPv4 and IPv6 Forwarding LFB Template

The expressive semantics of this template are sufficient to describe any IPv4 or IPv6 route configuration including ability to express route entries for virtual routers within a physical router.



Family: 8 bits
Address Family: AF_INET for IPv4; and AF_INET6 for IPv6.

Src length: 8 bits
Prefix length of source IP address.

Dest length: 8 bits
Prefix length of destination IP address.

TOS: 8 bits
The 8-bit TOS (should be deprecated to make room for DSCP).
Table ID: 8 bits
Table identifier. Up to 255 route tables are supported.

```
RT_TABLE_UNSPEC    An unspecified routing table.
RT_TABLE_DEFAULT   The default table.
RT_TABLE_MAIN      The main table.
```

RT_TABLE_LOCAL	The local table.

The user may assign arbitrary values between RT_TABLE_UNSPEC(0) and RT_TABLE_DEFAULT(253).

Protocol: 8 bits
Identifies what/who added the route.

Protocol	Route origin.
RTPROT_UNSPEC	Unknown.
RTPROT_REDIRECT	By an ICMP redirect.
RTPROT_KERNEL	By the kernel.
RTPROT_BOOT	During bootup.
RTPROT_STATIC	By the administrator.

Values larger than RTPROT_STATIC(4) are not interpreted by the kernel, they are just for user information. They may be used to tag the source of a routing information or to distinguish between multiple routing daemons.

Scope: 8 bits
Route scope (valid distance to destination).

RT_SCOPE_UNIVERSE	Global route.
RT_SCOPE_SITE	Interior route in the local autonomous system.
RT_SCOPE_LINK	Route on this link.
RT_SCOPE_HOST	Route on the local host.
RT_SCOPE_NOWHERE	Destination does not exist.

The values between RT_SCOPE_UNIVERSE(0) and RT_SCOPE_SITE(200) are available to the user.

Type: 8 bits
The type of route.

Route type	Description
RTN_UNSPEC	Unknown route.
RTN_UNICAST	A gateway or direct route.
RTN_LOCAL	A local interface route.
RTN_BROADCAST	A local broadcast route (sent as a broadcast).
RTN_ANYCAST	An anycast route.
RTN_MULTICAST	A multicast route.
RTN_BLACKHOLE	A silent packet dropping route.
RTN_UNREACHABLE	An unreachable destination. Packets dropped and host unreachable ICMPs are sent to the

	originator.
RTN_PROHIBIT	A packet rejection route. Packets are dropped and communication prohibited ICMPs are sent to the originator.
RTN_THROW	When used with policy routing, continue routing lookup in another table. Under normal routing, packets are dropped and net unreachable ICMPs are sent to the originator.
RTN_NAT	A network address translation rule.
RTN_XRESOLVE	Refer to an external resolver (not implemented).

Flags: 32 bits
Further qualify the route.

RTM_F_NOTIFY	If the route changes, notify the user.
RTM_F_CLONED	Route is cloned from another route.
RTM_F_EQUALIZE	Allow randomization of next hop path in multi-path routing (currently not implemented).

Attributes applicable to this service:

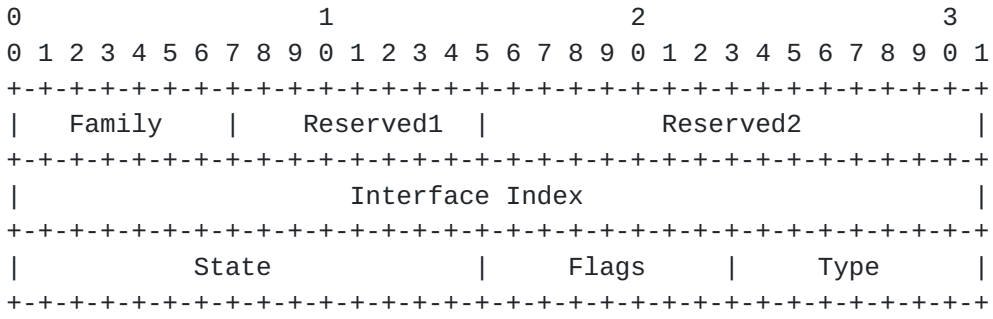
Attribute	Description
RTA_UNSPEC	Ignored.
RTA_DST	Protocol address for route destination address.
RTA_SRC	Protocol address for route source address.
RTA_IIF	Input interface index.
RTA_OIF	Output interface index.
RTA_GATEWAY	Protocol address for the gateway of the route
RTA_PRIORITY	Priority of route.
RTA_PREFSRC	Preferred source address in cases where more than one source address could be used.
RTA_METRICS	Route metrics attributed to route and associated protocols (e.g., RTT, initial TCP window, etc.).
RTA_MULTIPATH	Multipath route next hop's attributes.
RTA_PROTOINFO	Firewall based policy routing attribute.

RTA_FLOW	Route realm.
RTA_CACHEINFO	Cached route information.

Additional Netlink message types applicable to this service: RTM_NEWROUTE, RTM_DELROUTE, and RTM_GETROUTE

8.4.2.2 Neighbor Discovery LFB Template

The expressive semantics for this config are sufficient to describe both IPv4 neighbor resolution via ARP or IPv6 neighbor discovery (RFC2461).



Family: 8 bits
Address Family: AF_INET for IPv4; and AF_INET6 for IPv6.

Interface Index: 32 bits
The unique interface index.

State: 16 bits
A bitmask of the following states:

NUD_INCOMPLETE	Still attempting to resolve.
NUD_REACHABLE	A confirmed working cache entry
NUD_STALE	an expired cache entry.
NUD_DELAY	Neighbor no longer reachable. Traffic sent, waiting for confirmation.
NUD_PROBE	A cache entry that is currently being re-solicited.
NUD_FAILED	An invalid cache entry.
NUD_NOARP	A device which does not do neighbor discovery (ARP).
NUD_PERMANENT	A static entry.

Flags: 8 bits

NTF_PROXY	A proxy ARP entry.
NTF_ROUTER	An IPv6 router.

Attributes applicable to this service:

NDA_UNSPEC	Unknown type.
NDA_DST	A neighbour cache network layer destination address
NDA_LLADDR	A neighbor cache link layer address.
NDA_CACHEINFO	Cache statistics.

Additional Netlink message types applicable to this service: RTM_NEWNEIGH, RTM_DELNEIGH, and RTM_GETNEIGH.

8.4.3 Filtering Functions

TBF

8.4.4 QoS Functions

TBF

8.4.5 IPSEC Functions

TBF

8.4.6 Packet redirection Functions

TBF

8.4.7 Packet Mirroring Functions

TBF

8.4.8 Packet Sampling Functions

TBF

8.5 Security Considerations

CEs may communicate vital and possibly confidential information to FEs via the ForCES protocol. For example, such information can be filtering rules or secret encryption keys. In addition, the ForCES protocol should not open new possibilities for Denial of Service attacks. A single box environment is an interconnect between CEs and FEs that can be physically secured. ForCES messages coming on physical ports not part of the interconnect are dropped. In such an environment, protection is required only against data-packet-based

DoS attacks. A multi-hop environment places more requirements in terms of security. Protection against Netlink2-SYN-flood attack becomes necessary. In addition, some or all of the ForCES messages may have to be authenticated or encrypted.

8.5.1 Denial of Service (DoS) attacks

Preventing DoS attacks resulting from data packets redirected by the FE to the CE can be achieved by shaping according to configurable parameters such as a maximum rate.

A data-packets DoS-resistant FE MUST therefore support the necessary LFBs that permit to place policers that shape traffic redirected to the CE by an FE.

Preventing DoS attacks at the ForCES protocol level (such as Netlink2 SYN flood) may be necessary if the underlying transport protocol is not resistant to such attacks. This can be the case if UDP is used, for instance. In the case of TCP and SCTP, cookie-based mechanisms already exist to prevent SYN flood DoS attacks (refer to the respective RFCs and [TCP-SYN-COOKIES]).

A SYN-flood DoS-resistant FE or CE MUST therefore support a Netlink2-Extension Cookie TLV (TLV_TYPE = NL2_COOKIE). This Cookie TLV is placed in the ACK message that acknowledges a SYN message. This Cookie TLV MUST be returned as is in the SYNACK message. (Note: content and length of the Cookie TLV remain to be standardized, if necessary).

8.5.2 Authentication and Encryption

To perform authentication, the necessary information may be configured statically, such as shared secrets or public and private keys. On the other hand, in a dynamic environment, public keys may have to be distributed using certificates. Such certificates must contain names that are uniquely and permanently assigned to CEs and FEs. Addresses used for routing ForCES messages may change and are not suitable for that purpose. ForCES qualified names (Note: this needs to be defined in a draft of its own) MUST be used similarly to iSCSI qualified names [iSCSI-NAMING].

References

[Diffserv]
"Linux Diffserv", <http://diffserv.sourceforge.net>.

[ForCES_Model]
Yang, L., Halpern, J., Gopal, R., DeKok, A., Haraszti, Z. and S. Blake, "ForCES Forwarding Element Model", October 2003, < <http://www.ietf.org/internet-drafts/draft-ietf-forces-model-01.txt>. >.

[ForCES_REQ]
Khosravi, H. and T. Anderson, "Requirements for Separation of IP Control and Forwarding", October 2003, <http://www.ietf.org/internet-draft/draft-ietf-forces-requirements-07.txt>.

[Goutaudier]
Goutaudier, G., "Enhancements and Prototype Implementation of the ForCES Netlink2 Protocol, IBM Research Report RZ3482", September 2003, <http://domino.watson.ibm.com/library/cyberdig.nsf/papers?SearchView&Query=(goutaudier)>.

[Netfilter]
"Linux Netfilter", <http://www.netfilter.org>.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and C. Davin, "Simple Network Management Protocol (SNMP)", May 1990, <http://www.rfc-editor.org/rfc/rfc1157.txt>.

[RFC1633] Braden, R., Clark, D. and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", June 1994, <http://www.rfc-editor.org/rfc/rfc1633.txt>.

[RFC1812] Baker, F., "Requirements for IP Version 4 Routers", June 1995, <http://www.rfc-editor.org/rfc/rfc1812.txt>.

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

[RFC2328] Moy, J., "OSPF Version 2", April 1998, <http://www.rfc-editor.org/rfc/rfc2328.txt>.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Weiss, W. and Z. Wang, "An Architecture for Differentiated Services", December 1998, <http://www.rfc-editor.org/rfc/rfc2475.txt>.

[RFC2748] Boyle, J., Cohen, R., Durham, D., Herzog, S., Rajan, R. and A. Sastry, "The COPS (Common Open Policy Service) Protocol", January 2000, <http://www.rfc-editor.org/rfc/rfc2748.txt>.

[RFC2844] Przygienda, T., Droz, P. and R. Haas, "OSPF over ATM and Proxy-PAR", May 2000, <http://www.rfc-editor.org/rfc/rfc2844.txt>.

[RFC3036] Andersson, L., Doolan, P., Feldman, N., Fredette, A. and B. Thomas, "LDP Specification", January 2001, <http://www.rfc-editor.org/rfc/rfc3036.txt>.

[RFC3292] Doria, A., "General Switch Management Protocol (GSMP) V3", June 2002, <http://www.rfc-editor.org/rfc/rfc3292.txt>.

[RFC3358] Przygienda, T., "Optional Checksums in Intermediate System to Intermediate System (ISIS)", August 2002, <http://www.rfc-editor.org/rfc/rfc3358.txt>.

[RFC3549] Hadi Salim, J., Khosravi, H., Kleen, A. and A. Kuznetsov, "Linux Netlink as an IP Services Protocol", July 2003, <http://www.rfc-editor.org/rfc/rfc3549.txt>.

[Stevens] Wright, G. and W. Stevens, "TCP/IP Illustrated Volume 2, Chapter 20", June 1995.

[TCP-SYN-COOKIES] Dan, D., "SYN cookies", 1997, <http://cr.yp.to/syncookies.html>.

[XTP] "Xpress Transport Protocol Specification, XTP Revision 4.0", March 1995.

[iSCSI-NAMING] "iSCSI Naming and Discovery, draft-ietf-ips-iscsi-name-disc-10.txt", June 2003, <http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-name-disc-10.txt>.

Authors' Addresses

Jamal Hadi Salim
Znyx Networks
195 Stafford Rd. West
Ottawa, Ontario
Canada

EMail: hadi@znyx.com

Robert Haas
IBM Research
Zurich Research Laboratory
Saeumerstrasse 4
CH-8803 Rueschlikon,
Switzerland

EMail: rha@zurich.ibm.com

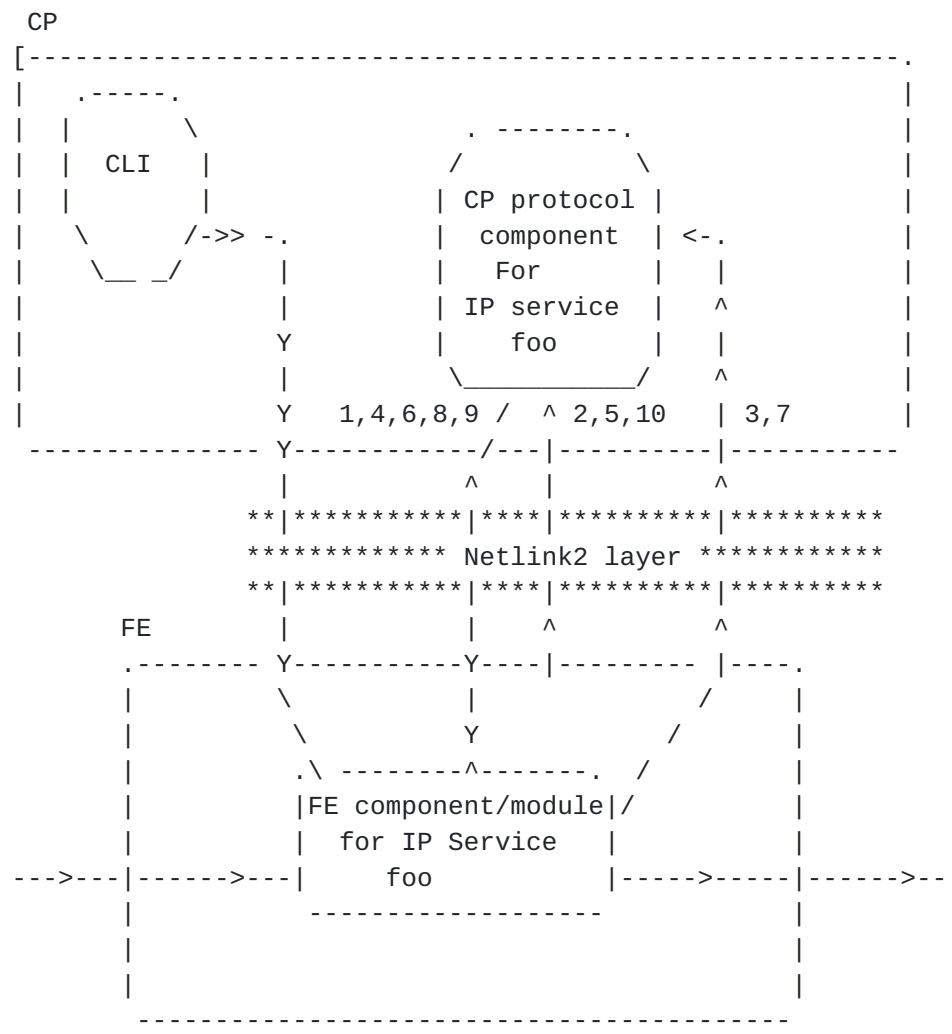
Steven Blake
Ericsson
920 Main Campus Drive, Suite 500
Raleigh, NC 27606
USA

EMail: steven.blake@ericsson.com

Appendix A. Sample Service Hierarchy

In the diagram below we show a simple IP service, foo, and the interaction it has between CP and FE components for the service(labels 1-3).

The diagram is also used to demonstrate CP< - >FE addressing. In this section we illustrate only the addressing semantics. In Appendix 2 , the diagram is referenced again to define the protocol interaction between service foo's CPC and LFB (labels 4-10).



The control plane protocol for IP service foo does the following to connect to its FE counterpart. The steps below are also numbered in the diagram above.

1. Connect to IP service foo through a socket connect. A typical

connection would be via a call to: socket(AF_NETLINK, SOCK_RAW, NETLINK_FOO)

2. Bind to listen to specific async events for service foo

3. Bind to listen to specific async FE events

Note that a wrapper socket can be created on top of the real sockets: depending on the dest PID given, it chooses the most appropriate socket to send the packet onto (if here are two multicast groups, one for all FEs, and one for all FEs and CEs, a packet from the CE to the FEs will use the first multicast group). The wrapper socket basically maps a message to the most appropriate wire in the bundle.

Appendix B. Sample Protocol for the foo IP Service

Our proverbial IP service "foo" is used again to demonstrate how one can deploy a simple IP service control using Netlink2.

These steps are continued from Appendix 1 (hence the numbering).

1. query for current config of FE component

2. receive response to 4) via channel on 3)

3. query for current state of IP service foo

4. receive response to 6) via channel on 2)

5. register the protocol specific packets you would like the FE to forward to you

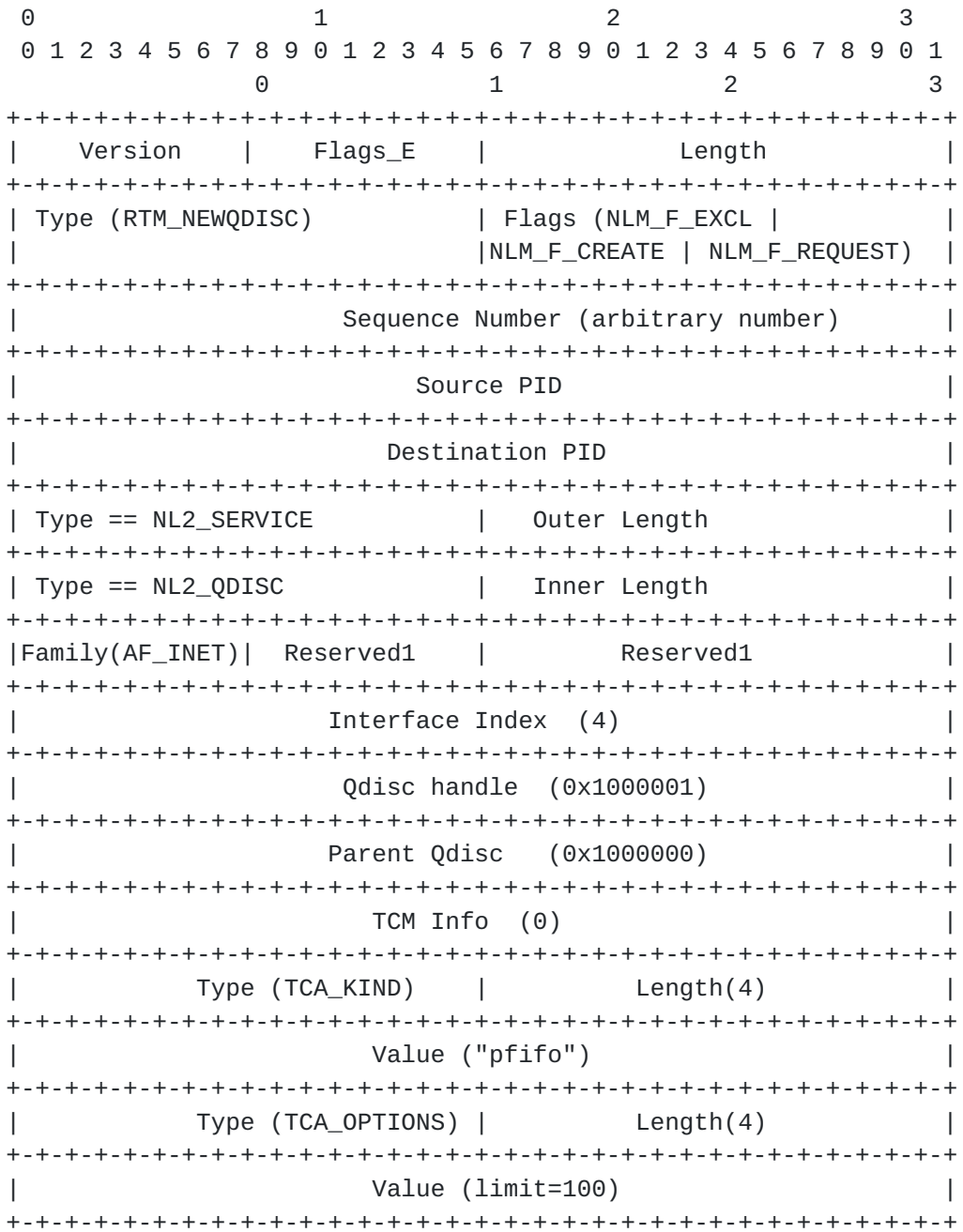
6. send specific service foo commands and receive responses for them if needed

B.1 Interacting with Other IP Services

The diagram in Appendix 1 shows another control component configuring the same service. In this case, it is a proprietary Command Line Interface. The CLI may or may not be using the Netlink protocol to communicate with the foo component. If the CLI should issue commands that will affect the policy of the LFB for service "foo", then the "foo" CPC is notified. It could then make algorithmic decisions based on this input. For example if a FE allowed another service to delete policies installed by a different service and a policy that foo installed was deleted by service bar, there might be a need to propagate this to all the peers of service "foo").

Appendix C. Examples

In this example we show a simple configuration Netlink2 message sent from a TC CPC to an egress TC FIFO queue. This queue algorithm is based on packet counting and drops packets when the limit exceeds the configured limit (100 packets in the example policy below). We assume the queue is in hierarchical setup with a parent 100:0 and a classid of 100:1 and that it is to be installed on device with ifindex of 4.



Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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