ForCES Working Group
Internet Draft

Jamal Hadi Salim
Znyx Networks
Hormuzd Khosravi
Intel
Andi Kleen
Suse
Alexey Kuznetsov
INR/Swsoft
September 2001

Netlink as an IP services protocol
draft-ietf-forces-netlink-00.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 [RFC-2119].

1. Abstract

This document describes Linux Netlink, which is used in Linux both as an inter-kernel messaging system as well as between kernel and

user-space. The purpose of this document is intended as informational in the context of prior art for the ForCES IETF working group. The focus of this document is to describe netlink from a context of a protocol between a Forwording Engine Component (FEC) and a Control Plane Component(CPC) that define an IP service.

The document ignores the ability of netlink as a inter-kernel messaging system, as a an inter-process communication scheme (IPC) or its use in configuring other non-network as well as network but non-IP services (such as decnet etc).

2. Introduction

The concept of IP Service control-forwarding separation was first introduced in the early 1980s by the BSD 4.4 routing sockets[stevens]. The focus at that time was a simple IP(v4) forwarding service and how the CPC, either via a command line configuration tool or a dynamic route daemon, can control forwarding tables for that IPV4 forwarding service.

The IP world has evolved considerably since those days. Linux netlink, when observed from a service provisioning point of view takes routing sockets one step further by breaking the barrier of focus around IPV4 forwarding. Since the 2.1 kernel, netlink has been providing the IP service abstraction to a few services other than the classical IPv4 forwarding.

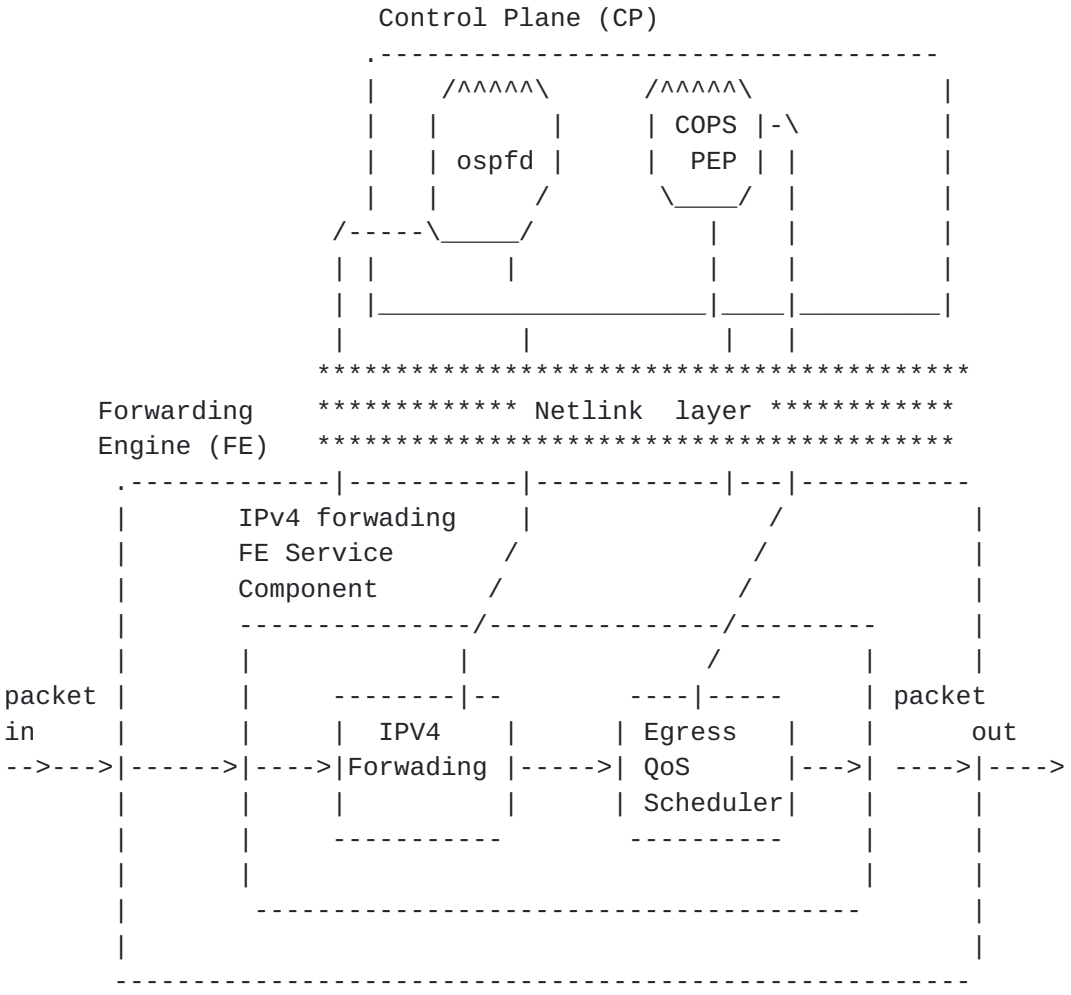
We first give some concept definitions and then describe how netlink fits in.

2.1. Some definitions

A Control plane(CP) is an execution environment that may have several components which we refer to as CPCs. Each CPC provides control for a different IP service being executed by a FE component. This means that there might be several CPCs on a physical CP if it is controlling several IP services. In essence, the cohesion between a CP component and a FE component is the service abstraction.

In the diagram below we show a simple FE<->CP setup to provide an example of the classical IPv4 service with an extension to do some basic QoS egress scheduling and how it fits in this described

model.



2.1.1. Control Plane Components (CPCs)

Control plane components would encompass signalling protocols with diversity ranging from dynamic routing protocols such as OSPF [RFC2328] to tag distribution protocols such as CR-LDP [RFC3036]. Classical Management protocols and activities also fall under this category. These include SNMP [RFC1157], COPS [RFC2748] or proprietary CLI/GUI configuration mechanisms.

The purpose of the control plane is to provide an execution environment for the above mentioned activities with the ultimate goal being to configure and manage the second NE component: the FE. The result of the configuration would define the way packets travesing the FE are treated.

The CP components are traditionaly run in software since they tend to be very rich in syntax and are moving targets requiring ease of modification.

In the above diagram, ospfd and COPS are distinct CPCs.

2.1.2. Forwarding Engine Components

The FE is the entity of the NE that incoming packets (from the network into the NE) first encounter.

The FE's service specific component massages the packet to provide it with a treatment to achieve a IP service as defined by the control plane components for that IP service. Different services will utilize different FEC. Service modules maybe chained to achieve a more complex service (as shown in the diagram). When built for providing a specific service, the FE service component will adhere to a Forwading Model.

In the above diagram, the IPV4 FE component includes both the IPV4 Forwarding service module as well as the Egress Scheduling service module. Another service might may add a policy forwarder between the IPV4 forwader and the QoS egress Scheduler. A simpler classical service would have constituted only the IPV4 forwarder.

2.1.3. IP Services

An IP Service is the treatment of an IP packet within the NE. This treatment is provided by a combination of both the CPC and FEC

The time span of the service is from the moment when the packet arrives at the NE to the moment it departs. In essence an IP service in this context is a Per-Hop Behavior. A service control/signaling protocol/management-application (CP components running on NEs defining the end to end path) unifies the end to end view of the IP service. As noted above, these CP components then define the

behavior of the FE (and therefore the NE) to a described packet.

A simple example of an IP service is the classical IPv4 Forwading. In this case, control components such as routing protocols(OSPF, RIP etc) and proprietary CLI/GUI configurations modify the FE's forwarding tables in order to offer the simple service of forwarding packets to the next hop. Traditionally, NEs offering this simple service are known as routers.

Over the years it has become important to add aditional services to the routers to meet emerging requirements. More complex services extending classical forwarding were added and standardized. These newer services might go beyond the layer 3 contents of the packet header. However, the name "router", although a misnomer, is still used to describe these NEs. Services (which may look beyond the classical L3 headers) here include firewalling, Qos in Diffserv and RSVP, NATs, policy based routing etc. Newer control protocols or management activities are introduced with these new services.

One extreme definition of a IP service is something a service provider would be able to charge for.

3. Netlink Architecture

IP services components control is defined by using templates.

The FEC and CPC participate to deliver the IP service by communicating using these templates. The FEC might continously get updates from the control plane component on how to operate the service (example for V4 forwarding route additions or deletions).

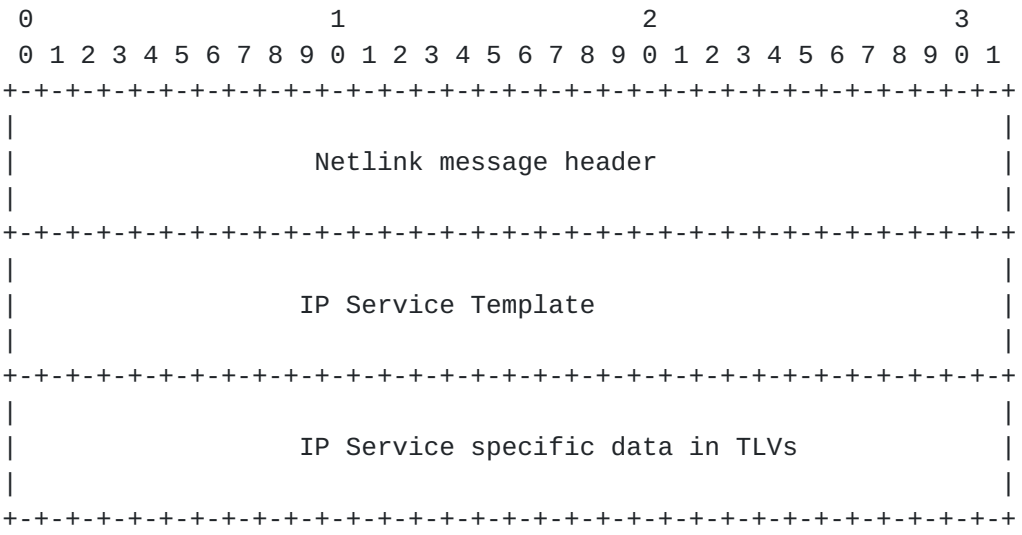
The interaction between the FEC and the CPC, in the netlink context, would define a protocol. Netlink provides the mechanism for the CPC(residing in user space) and FEC(residing in kernel space) to define their own protocol definition. The FEC and CPC, using netlink mechanisms, may choose to define a reliable protocol between each other, for example. By default netlink provides an unreliable communication.

Note that the FEC and CPC can both live in the same memory protection domain and use the connect() system call to create a path to the peer and talk to each other. We will not discuss this further other than to say it is available as a mechanism. Through out this document we will refer interchangbly to the FEC to mean kernel-space and the CPC to mean user-space.

Note: Netlink allows participation in IP services by both service components.

3.1. The message format

There are three levels to a netlink message: The general netlink message header, the IP service specific template, the IP service specific data.



3.2. Wire Model

[In here we describe the pseudo-wire model that netlink uses inside the kernel]

3.3. Protocol Model

This section expands on how netlink provides the mechanism for service oriented FEC and CPC interaction.

3.3.1. Service Addressing

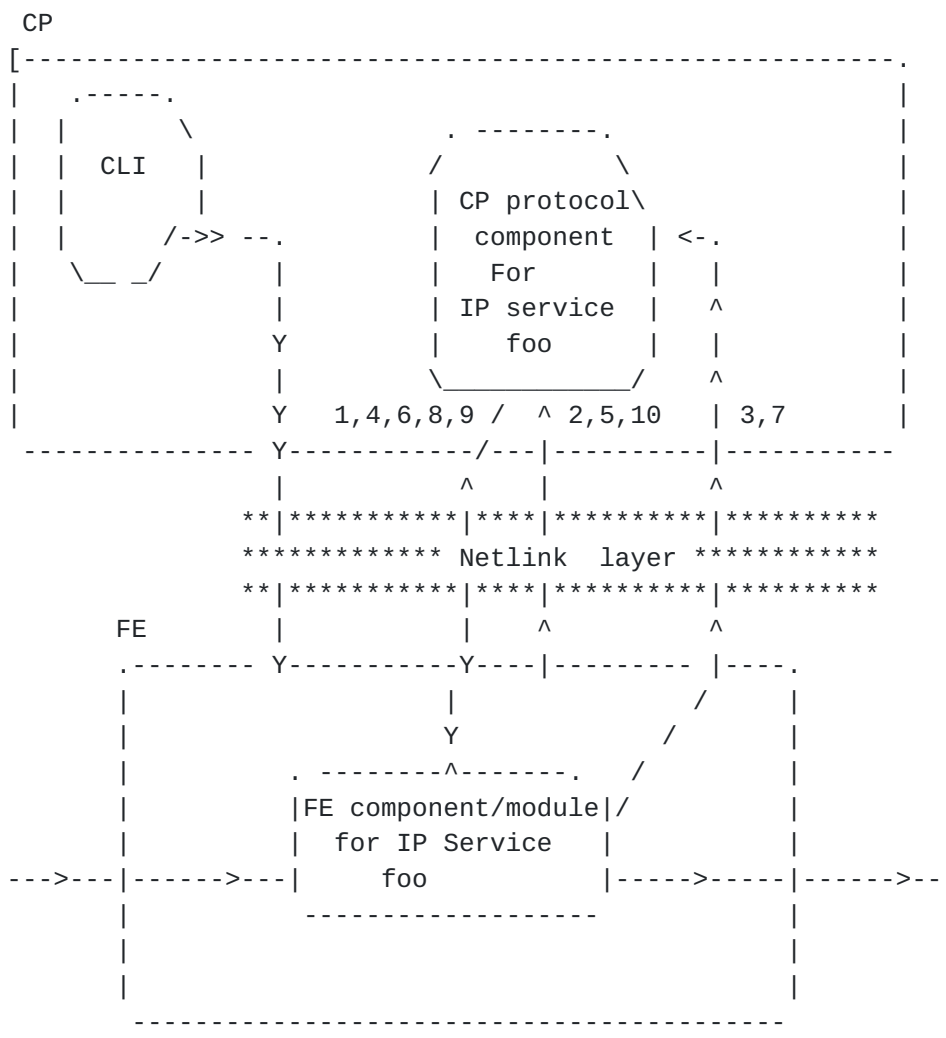
Access is provided by first connecting to the service on the FE. This is done by making a socket() system call to the PF_NETLINK

domain. Each FEC is identified by a protocol number. One may open either SOCK_RAW or SOCK_DGRAM type sockets although netlink doesnt distinguish the two. The socket connection provides the basis for the FE<->CP addressing.

Connecting to a service is followed (at any point during the life of the connection) by issuing either a service specific command mostly for configuration purposes (from the CPC to the FEC) or subscribing/unsubscribing to service(s') events.

3.3.1.1. Sample Service Hierachy

In the diagram below we show a simple IP service, foo, and the interaction it has between CP and FE components for the service.



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

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)

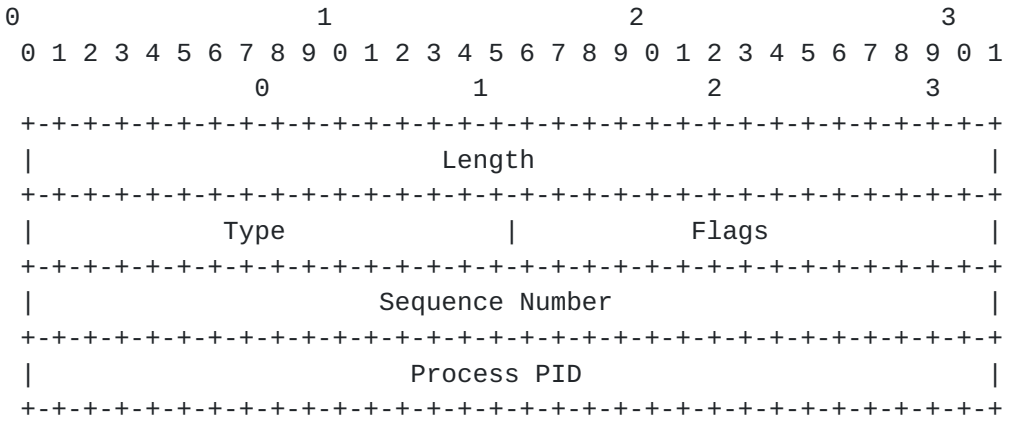
2) Bind to listen to specific async events for service foo

3) Bind to listen to specific async FE events

3.3.2. Netlink message header

Netlink messages consist of a byte stream with one or multiple Netlink headers and associated payload. (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 netlink message header is shown below.



The fields in the header are:

```
          Length: 32 bits
          The length of the 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 ignored in the current implementation
               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 FEC to CPC.
               NLMSG_DONE message terminates a multipart message

          Individual IP Services specify more message types, for e.g.,
          NETLINK_ROUTE Service specifies several types such as RTM_NEWLINK,
          RTM_DELLINK, RTM_GETLINK, RTM_NEWADDR, RTM_DELADDR, 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 user space to kernel space)
                 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 user space to
kernel space.
                 NLM_F_ECHO      Echo this request. Typical direction of
request is from
                                 user space to kernel space.

          Additional flag bits for GET requests on config information in the
FEC.
                 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   Return an atomic snapshot of the table being
referenced.
                 NLM_F_DUMP     Return all that matches in the table. This is a
shortcut
                                having both NLM_F_ROOT and NLM_F_MATCH flags
set.

          Additional flag bits for NEW requests
                 NLM_F_REPLACE   Replace existing matching config object with
```

this
request.

NLM_F_EXCL	Don't replace the config object if it already exists.
NLM_F_CREATE	Create config object if it doesn't already exist.
NLM_F_APPEND	Add to the end of the object list.

For those 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 to NLM_F_REPLACE

BSD Check operation equates to NLM_F_EXCL

BSD APPEND equaivalent is actually mapped to NLM_F_CREATE

Sequence Number: 32 bits
The sequence number of the message.

Process PID: 32 bits
The PID of the process sending the message. The PID is used by the kernel to multiplex to the correct sockets. A PID of zero is used when sending messages to user space from the kernel.

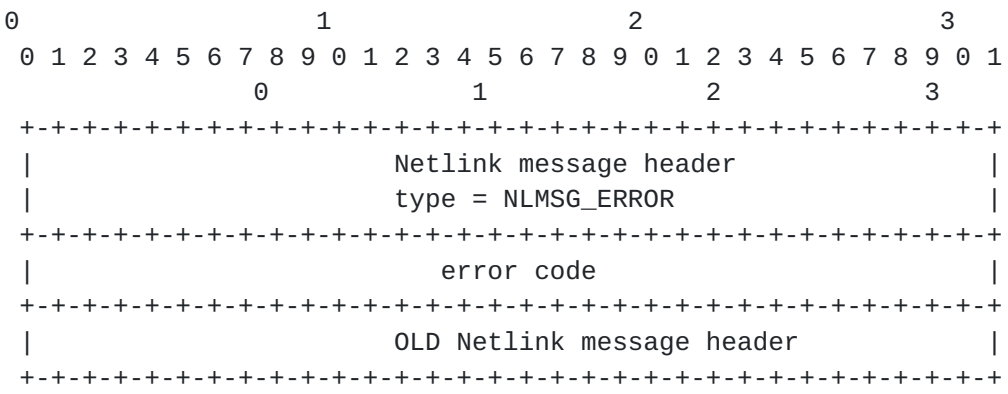
3.3.2.1. Mechanisms for creating protocols

One could create a reliable protocol between an FEC and a CPC by using the combination of sequence numbers, ACKs and retransmit timers. Both sequence numbers and sequence numbers are provided by netlink. Timers are provided by Linux.

One could create a heartbeat protocol between the FEC and CPC by using the ECHO flags.

3.3.2.2. The ACK netlink message

This message is actually used to denote both an ACK and a NACK. Typically the direction is from kernel to user space (in response to an ACK request message that is sent). However, user space should be able to send ACKs back to kernel space when requested. This is IP service specific.



Error code: integer (typically 32 bits)

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

A non-zero error message is equivalent to a Negative ACK (NACK). In such a situation, the netlink data that was sent down to the kernel is returned appended to the original netlink message header. An error code printable via the perror() is also set (not in the message header, rather in the executing environment state variable).

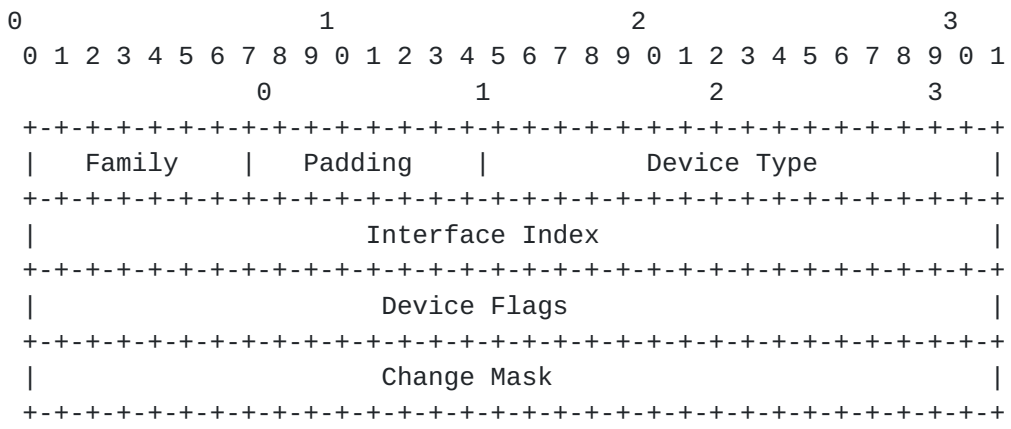
3.3.3. FE services' templates

These are services that are offered by the system for general use by other services. They include ability to configure and listen to changes in resource management. IP address management, link events etc fit here. We separate them into this section here for logical purposes despite the fact that they are accessed via the NETLINK_ROUTE FEC. The reason that they exist within NETLINK_ROUTE is due to historical cruft based on the fact that BSD 4.4 rather narrowly focussed Route Sockets implemented them as part of the IPV4 forwarding sockets.

3.3.3.1.

Network Interface Service Module

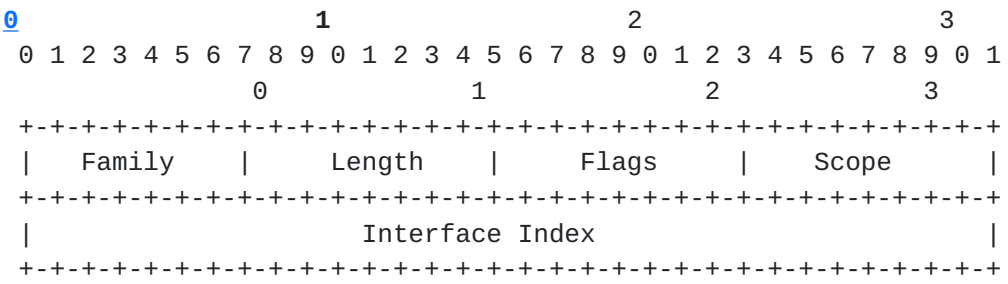
This service provides the ability to create, remove or get information about a specific network interface. The Interface service message template is shown below.



Descriptions of the headers to be added.

3.3.3.2. IP Address Service module

This service provides the ability to add, remove or receive information about an IP address associated with an interface. The Address provisioning service message template is shown below.



Descriptions of the headers to be added.

4. 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 netlink.

These steps are continued from the "Sample Service Hierachy" section.

4) query for current config of FE component

5) receive response to 4) via channel on 3)

6) query for current state of IP service foo

7) receive response to 6) via channel on 2)

9) register the protocol specific packets you would like the FE to forward to you

10) send specific service foo commands and receive responses for them if needed

4.1. Interacting with other IP services

The last diagram 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 to the foo component. If the CLI should issue commands that will affect the policy of the FEC for service "foo" then, then the "foo" CPC is notified. It could then make algorithmic decisions based on this input (example if a policy that foo installed was deleted, there might be need to propagate this to all the peers of service "foo").

5. Currently Defined netlink IP services

Although there are many other IP services defined which are using netlink, we will only mention those integrated into the kernel today (kernel version 2.4.6). These are:

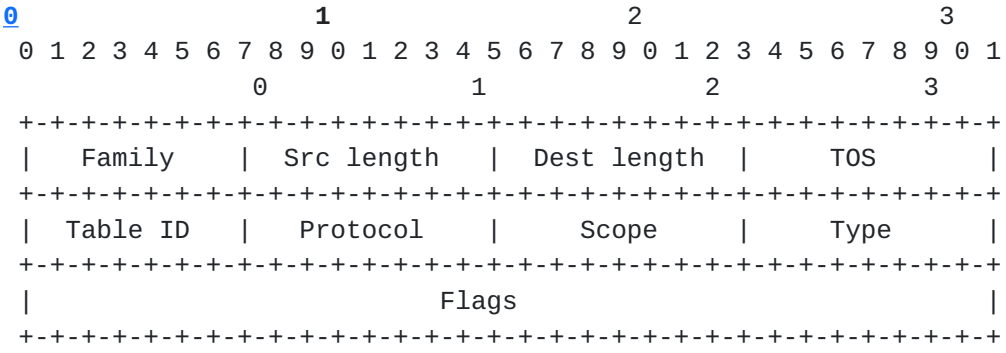
NETLINK_ROUTE,NETLINK_FIREWALL,NETLINK_ARPD,NETLINK_ROUTE6,NETLINK_IP6_FW
NETLINK_TAPBASE,NETLINK_SKIP,NETLINK_USERSOCK.

5.1. IP Service NETLINK_ROUTE

This service allows CPCs to modify the IPv4 routing table in the Forwarding Engine. It can also be used by CPCs to receive routing updates.

5.1.1. Network Route Service Module

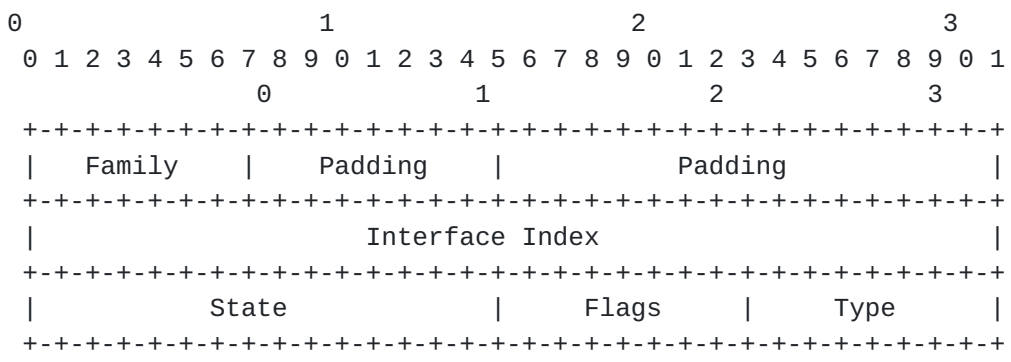
This service provides the ability to create, remove or receive information about a network route. The service message template is shown below.



Descriptions of the headers to be added.

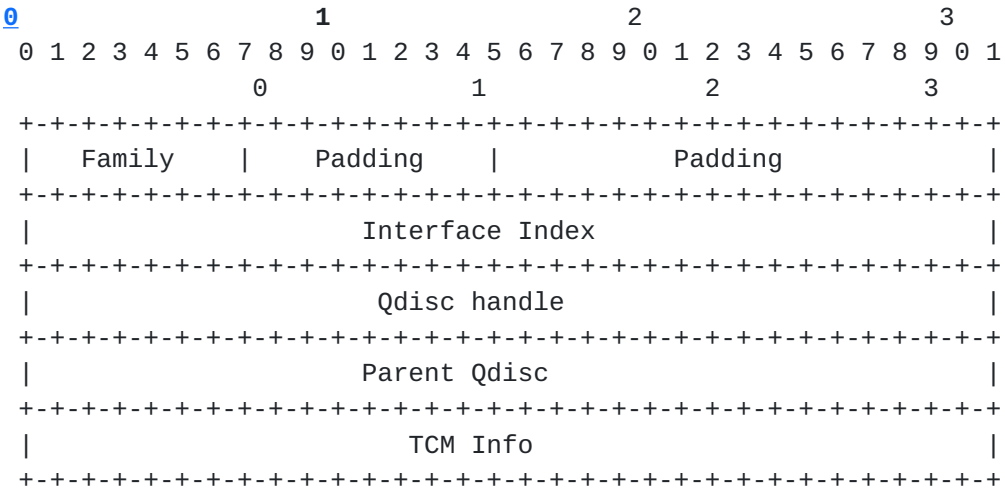
5.1.2. Neighbour Setup Service Module

This service provides the ability to add, remove or receive information about a neighbour table entry (e.g. an ARP entry). The service message template is shown below.



5.1.3. Traffic Control Service

This service provides the ability to add, remove or get a queueing discipline. The service message template is shown below.



5.2. IP Service NETLINK_FIREWALL

This service allows CPCs to receive packets sent by the IPv4 firewall module in the FE.

5.3. IP Service NETLINK_ARPD

This service is used by CPCs for managing the ARP table in FE.

5.4. IP Service NETLINK_ROUTE6

This service allows CPCs to modify the IPv6 routing table in the FE. It can also be used by CPCs to receive routing updates.

5.5. IP Service NETLINK_IP6_FW

This service allows CPCs to receive packets that failed the IPv6 firewall checks by that module in the FE.

5.6. IP Service NETLINK_TAPBASE

This service allows CPCs to simulate an ethernet driver belonging to the FE.

//are the instances of the ethertap device. Ethertap //is a pseudo network tunnel device that allows an //ethernet driver to be simulated from user space.

5.7. IP Service NETLINK_SKIP

This service is reserved for ENskip (?).

5.8. IP Service NETLINK_USERSOCK

This service is reserved for future Control Plane to FE protocols.

6. Security Considerations

Netlink lives in a trusted environment of a single host separated by kernel and user space. Linux capabilities ensures that only someone with CAP_NET_ADMIN capability (typically root user) is allowed to open sockets.

7. References

[RFC1633] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, ISI, MIT, and PARC, June 1994.

[RFC1812] F. Baker, "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

[RFC2475] M. Carlson, W. Weiss, S. Blake, Z. Wang, D. Black, and E. Davies, "An Architecture for Differentiated Services", RFC 2475, December 1998.

[RFC2748] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.

[RFC2328] J. Moy, "OSPF Version 2", RFC 2328, April 1998.

[RFC1157] J.D. Case, M. Fedor, M.L. Schoffstall, C. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[RFC3036] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas "LDP Specification", RFC 3036, January 2001.

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

8. Acknowledgements

1) Andi Kleen for man pages on netlink and rtnetlink.

2) Alexey Kuznetsov is credited for extending netlink to the IP service delivery model. The original netlink character device was written by Alan Cox.

9. Author's Address:

Jamal Hadi Salim
Znyx Networks
Ottawa, Ontario
Canada
hadi@znyx.com

Hormuzd M Khosravi
Intel
2111 N.E. 25th Avenue JF3-206
Hillsboro OR 97124-5961
1 503 264 0334
hormuzd.m.khosravi@intel.com