ForCES Working Group
Internet Draft

Jamal Hadi Salim
Znyx Networks
Hormuzd Khosravi
Intel
Andi Kleen
Suse
Alexey Kuznetsov
INR/Swsoft
March 2002

Netlink as an IP services protocol
draft-ietf-forces-netlink-02.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 Forwarding 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 linux 2.1 kernel, netlink has been providing the IP service abstraction to a few services other than the classical IPv4 forwarding.

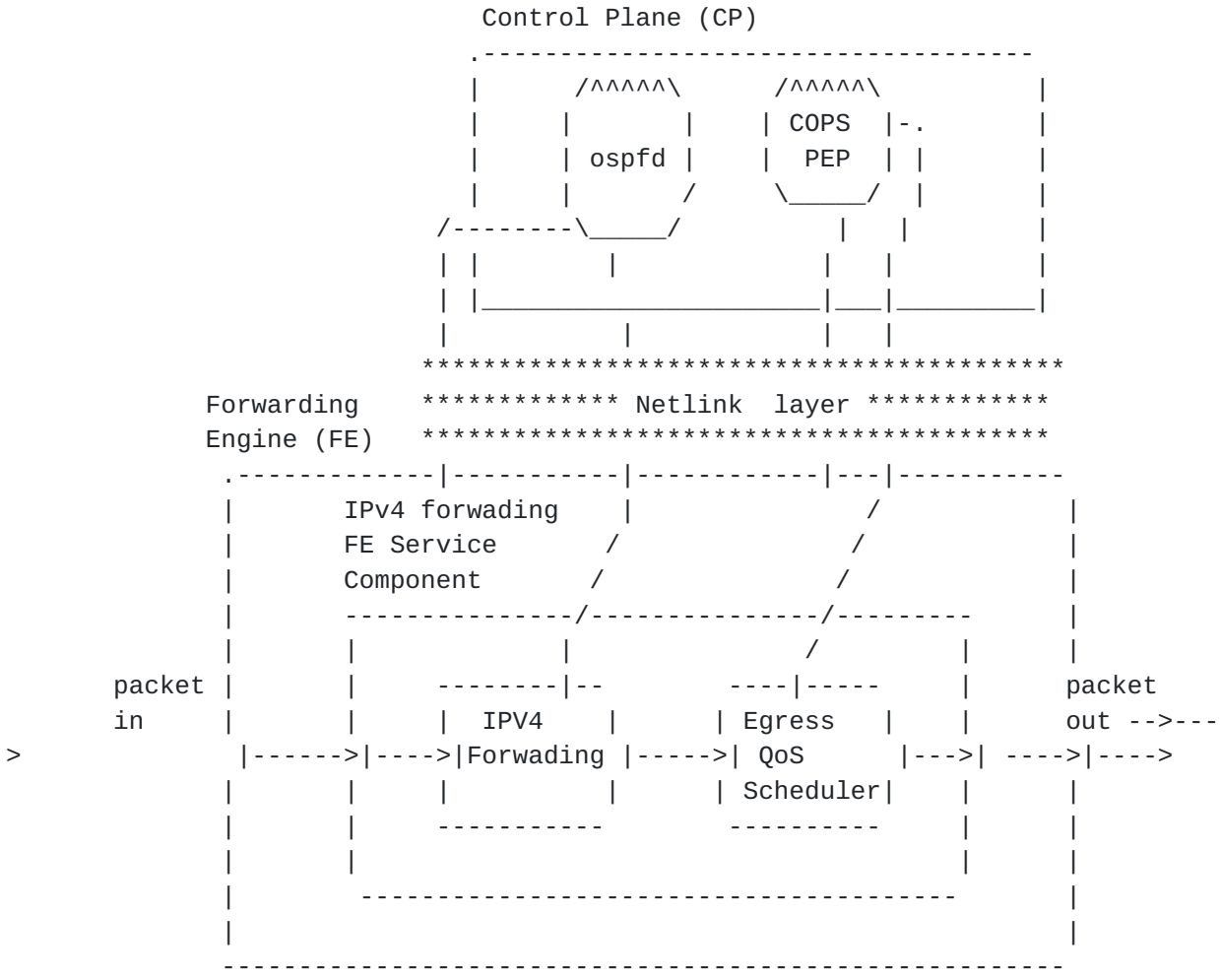
We first give some concept definitions and then describe how netlink fits in.

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.

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 Forwarding. 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 have their own protocol definition. Kernel space and user space just mean different protection domains. Therefore a wire protocol is needed to communicate. The wire protocol would be normally be provided by some privileged service that is able to copy between multiple protection domains. We will refer to this service as the netlink service. Netlink service could also be necapsulated to a different transport layer if the CPC executes on a different node than the FEC. The FEC and CPC, using netlink mechanisms, may choose to define a reliable protocol between each other. By default, however, netlink provides an unreliable communication.

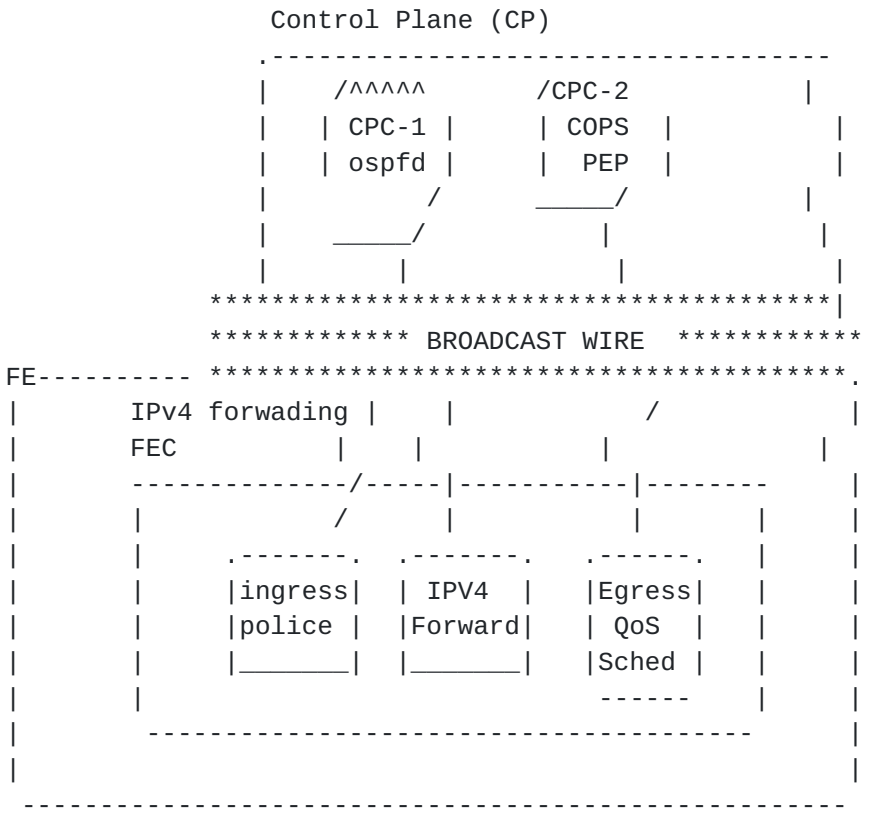
Note that the FEC and CPC can both live in the same memory protection domain and use the connect() system call to create a path to the peer and talk to each other. We will not discuss this further other than to say it is available as a mechanism. Through out this

document we will refer interchangebly to the FEC to mean kernel-space and the CPC to mean user-space. This is not meant, however, to restrict the two components to these protection domains or to the same compute node.

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

3.1. Netlink Logical model

In the diagram below we show a simple FEC<->CPC logical relationship. We use the example of IPV4 forwarding FEC (NETLINK_ROUTE, which is discussed further below) as an example.



Netlink logically models FECs and CPCs in the form of nodes interconnected to each other via a broadcast wire.

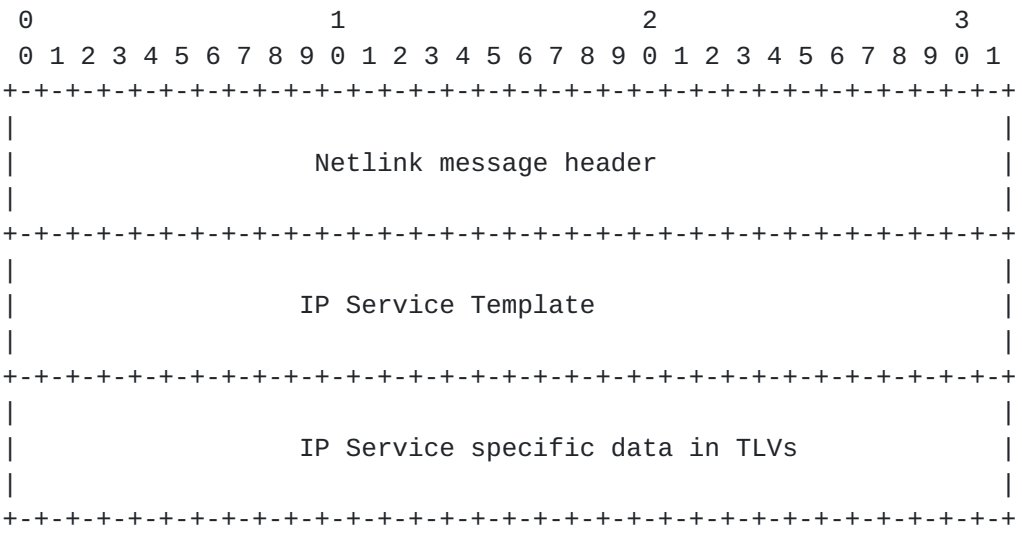
The wire is specific to a service. The example above shows the broadcast wire belonging to the extended IPV4 forwarding service.

Nodes connect to the wire and register to receive specific messages. CPCs may connect to multiple wires if it helps them to control the service better. All nodes(CPCs and FECs) dump packets on the broadcast wire. Packets could be discarded by the wire if malformed or not specifically formated for the wire. Dropped packets are not seen by any of the nodes. The netlink service MAY signal an error to the original if it detects an malformatted netlink packet.

Packets sent on the wire could be broadcast, multicast or unicast. FECs or CPCs register for and pick specific messages of interest for processing or just monitoring purposes.

3.2. 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.



The netlink message is used to communicate between the FEC and CPC for parametrization of the FECs, asynchoronous event notification of FEC events to the CPCs and statistics querying/gathering (typically by the CPC). The Netlink message header is generic for all services whereas the IP Service Template header is specific to a service. Each IP Service then carries parameterization data(CPC->FEC direction) or response (FEC->CPC direction). These are in TLV format and unique just to the service.

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

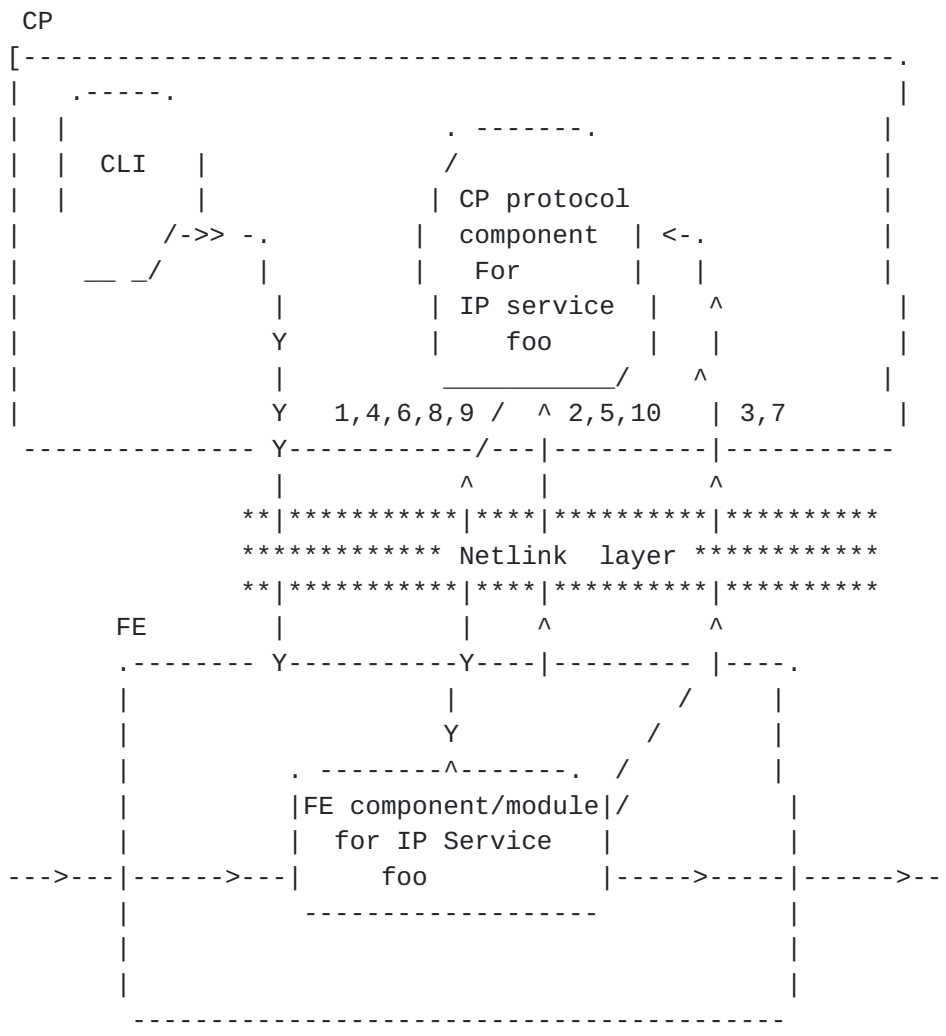
Access is provided by first connecting to the service on the FE. This is done by making a socket() system call to the PF_NETLINK domain. Each FEC is identified by a protocol number. One may open either SOCK_RAW or SOCK_DGRAM type sockets although netlink doesnt distinguish the two. The socket connection 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, or statistics collection.

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(labels 1-3).

We introduce the diagram below to demonstrate CP<->FE addressing. In this section we illustrate only the addressing semantics. In section 4, the diagram is referenced again to define the protocol interaction between service foo's CPC and FEC (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 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)

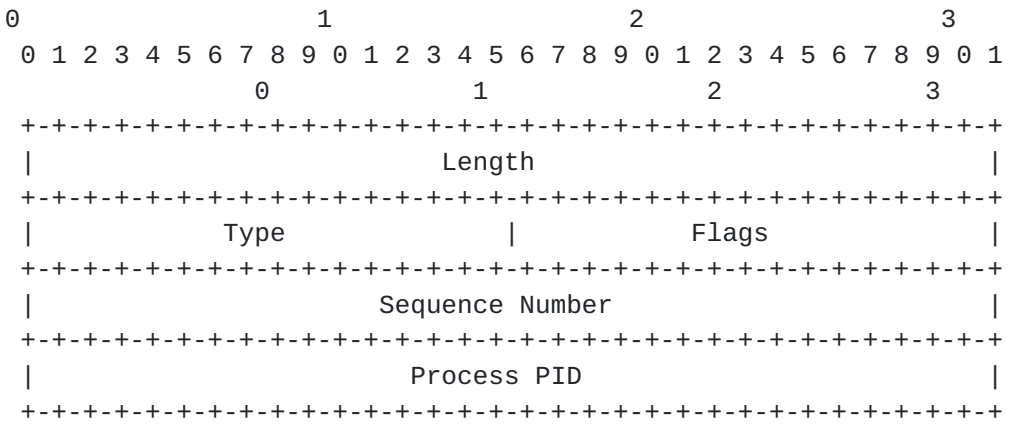
2) Bind to listen to specific async events for service foo

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. If the payload is too big to fit into a single message it can be split over multiple netlink messages. This is called a multipart message. 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
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. This may require special privileges because it has the potential to interrupt service in the FE for a longer time.

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	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 equivalent 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. netlink service fills in an appropiate value when zero.

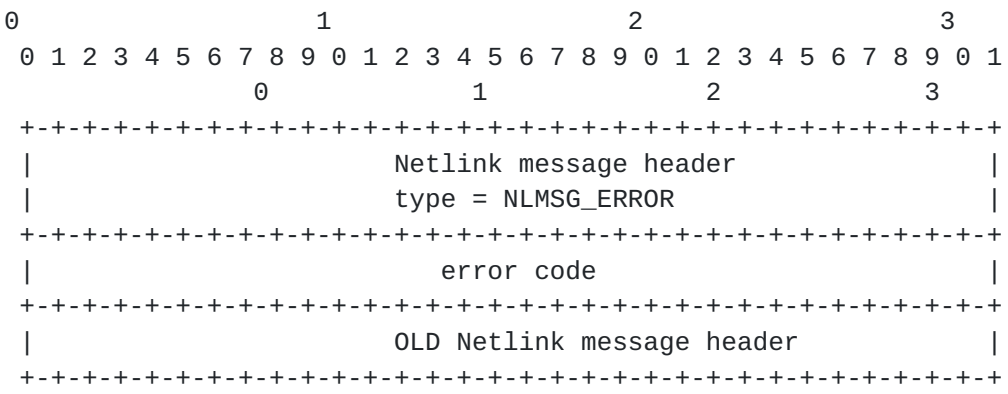
3.3.2.1. Mechanisms for creating protocols

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 ACKs 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 and the NLMSG_NOOP message.

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). 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).

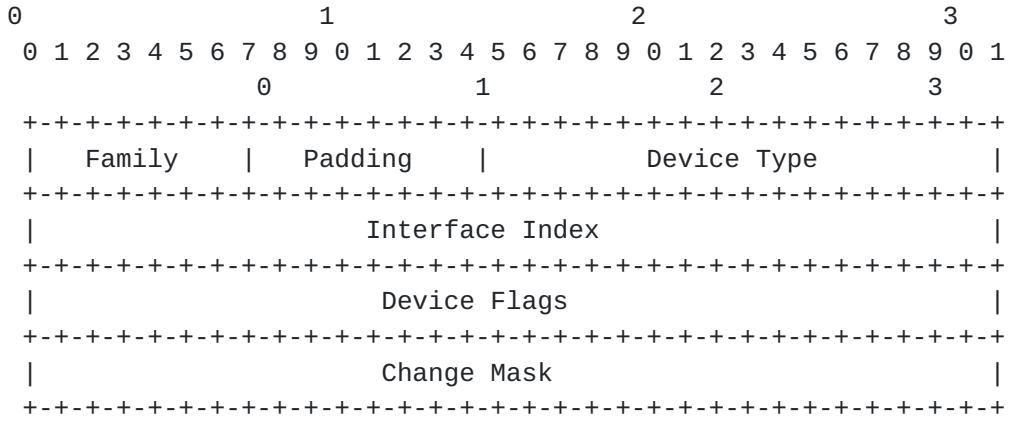
3.3.3. FE System services' templates

These are services that are offered by the system for general use by other services. They include ability to configure, gather statistics and listen to changes in shared resources. 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 network interface could be either physical or virtual and is network protocol independent (example an x.25 interface can be defined via this message). The Interface service message template is shown below.



Family: This is always set to AF_UNSPEC

Device Type: This defines the type of the link. The link could be ethernet, a tunnel etc. Although we are interested only in IPV4, the link type is protocol independent.

Interface Index: uniquely identifies interface.

Device Flags:

IFF_UP	Interface is running.
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	Resources allocated.
IFF_NOARP	No arp protocol
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 Address is not permanent.

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

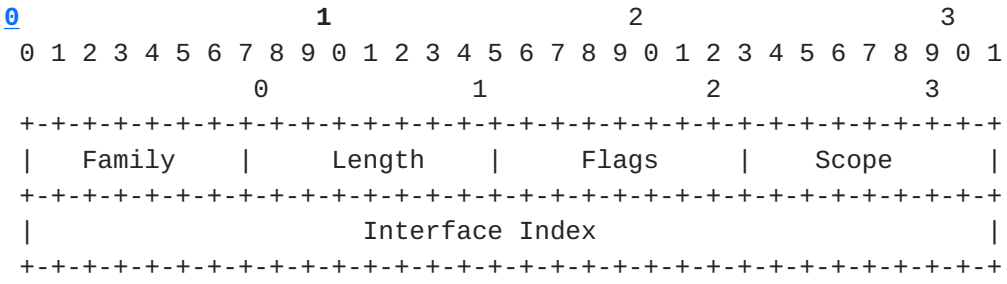
Applicable attributes:

attribute	description
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	Link type.
IFLA_QDISC	ascii string defining Queueing discipline.
IFLA_STATS	Interface Statistics.

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

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.



Family: AF_INET for IPV4 or AF_INET6 for IPV6.
Length: the length of the address mask
Flags: IFA_F_SECONDARY for secondary address (alias interface),
IFA_F_PERMANENT for a permanent address set by the user as opposed to dynamic addresses.
other flags include:
IFA_F_DEPRECATED which defines deprecated (IPV6) address
IFA_F_TENTATIVE which defines tentative (IPV6) address

Scope: the address scope

Applicable attributes:

attribute	description
IFA_UNSPEC	- unspecified.
IFA_ADDRESS	raw protocol address of interface
IFA_LOCAL	raw protocol local address
IFA_LABEL	ascii string name of the interface reffered to.
IFA_BROADCAST	raw protocol broadcast address.
IFA_ANYCAST	raw protocol anycast address
IFA_CACHEINFO	cacheinfo address information.

Define cacheinfo here -- JHS

netlink messages specific to this service: RTM_NEWADDR, RTM_DELADDR, RTM_GETADDR

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:

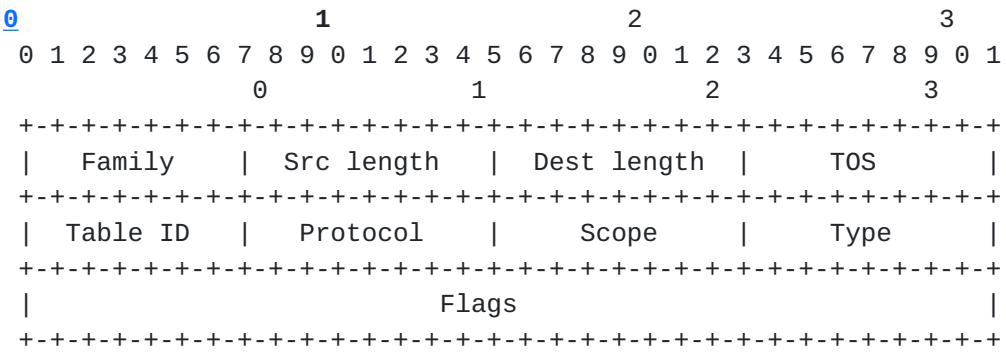
NETLINK_ROUTE,NETLINK_FIREWALL,NETLINK_ARPD,NETLINK_ROUTE6,
NETLINK_IP6_FW

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 as well as collecting statistics.

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.



Family: Address family of route. AF_INET for IPV4 and AF_INET6 for IPV6.

Src length: prefix length of source

Dest length: Prefix length of destination IP address

TOS: the 8 bit tos (should be deprecated to make room for DSCP)

Table ID: Table identifier. Upto 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 arbitary values between RT_TABLE_UNSPEC and RT_TABLE_DEFAULT.

Protocol: identifies what/who added the route. Described further below.

protocol	Route origin.
RTPROT_UNSPEC	unknown
RTPROT_REDIRECT	by an ICMP redirect (currently unused)
RTPROT_KERNEL	by the kernel
RTPROT_BOOT	during boot
RTPROT_STATIC	by the administrator

Values larger than RTPROT_STATIC 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 distingush between multiple routing daemons. See <linux/rtnetlink.h> for the routing daemon identifiers which are already assigned.

Scope: Route scope (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 doesn't exist

The values between RT_SCOPE_UNIVERSE and RT_SCOPE_SITE are available to the user.

Type: 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	a local broadcast route (sent as a unicast)
RTN_MULTICAST	a multicast route
RTN_BLACKHOLE	a packet dropping route
RTN_UNREACHABLE	an unreachable destination
RTN_PROHIBIT	a packet rejection route
RTN_THROW	continue routing lookup in another table
RTN_NAT	a network address translation rule
RTN_XRESOLVE	refer to an external resolver (not implemented)

Flags: further qualify the route.

RTM_F_NOTIFY	if the route changes, notify the user via rtnetlink
RTM_F_CLONED	route is cloned from another route
RTM_F_EQUALIZE	a multicast equalizer (not yet 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	
RTA_METRICS	Route metric
RTA_MULTIPATH	
RTA_PROTOINFO	
RTA_FLOW	
RTA_CACHEINFO	

additional netlink message types applicable to this service:
RTM_NEWROUTE, RTM_DELROUTE, RTM_GETROUTE

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.

Family: Address Family

Interface Index: The unique interface index

State: is a bitmask of the following states:

NUD_INCOMPLETE	a currently resolving cache entry
NUD_REACHABLE	a confirmed working cache entry
NUD_STALE	an expired cache entry
NUD_DELAY	an entry waiting for a timer
NUD_PROBE	a cache entry that is currently reprobed
NUD_FAILED	an invalid cache entry
NUD_NOARP	a device with no destination cache
NUD_PERMANENT	a static entry

Flags: one of:

NTF_PROXY	a proxy arp entry
NTF_ROUTER	an IPv6 router

Attributes applicable to this service:

Attributes	description
NDA_UNSPEC	unknown type
NDA_DST	a neighbour cache network layer destination address
NDA_LLADDR	a neighbour cache link layer address
NDA_CACHEINFO	cache statistics.

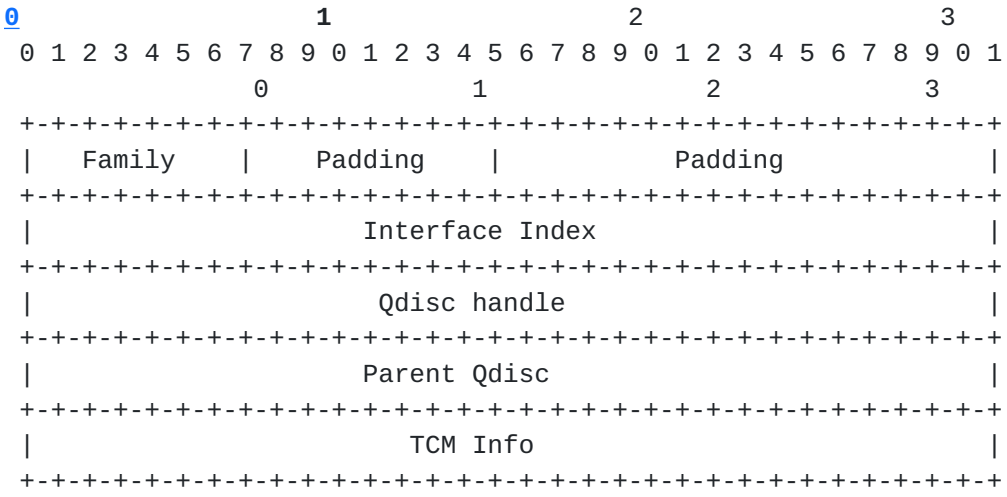
Describe the NDA_CACHEINFO nda_cacheinfo header later --JHS

additional netlink message types applicable to this service:
RTM_NEWNEIGH, RTM_DELNEIGH, RTM_GETNEIGH

5.1.3. Traffic Control Service

This service provides the ability to provision, query or listen to events under the auspicies of traffic control. These include Queueing disciplines (schedulers and queue treatment algorithms eg Priority based scheduler or RED algorithm) and classifiers. Linux Traffic Control Service is very flexible and allows for hierachical cascading of the different blocks for traffic sharing. The service message template which makes this possible is shown below. Each of the specific component of the model has unique attributes which describe it best. The common

attributes as well which are described below.



Family: Address Family

Interface Index: The unique interface index

Qdisc handle: unique identifier for instance of queueing discipline.
Typically this is split into major:minor of 16 bits each. The major
number would also be the major number of the parent of this instance.

Parent Qdisc: This is used in hierarchical layering of queueing
disciplines.
If this value and the Qdisc handle are the same and equal to TC_H_ROOT
then the defined qdisc is the top most layer known as the root qdisc.

TCM Info: This is set by the FE to 1 typically except when the qdisc
instance is in use, in which case it is set to imply a reference count.

Attributes applicable to this service:

Attribute	description
TCA_KIND	canonical name of FE component
TCA_STATS	generic usage statistics of FEC
TCA_RATE	rate estimator being attached to FEC. Takes snapshots of stats to compute rate
TCA_XSTATS	specific statistics of FEC
TCA_OPTIONS	nested FEC-specific attributes

[should we define all FEC-specific attributes? Seems like a lot of work
-- jhs]

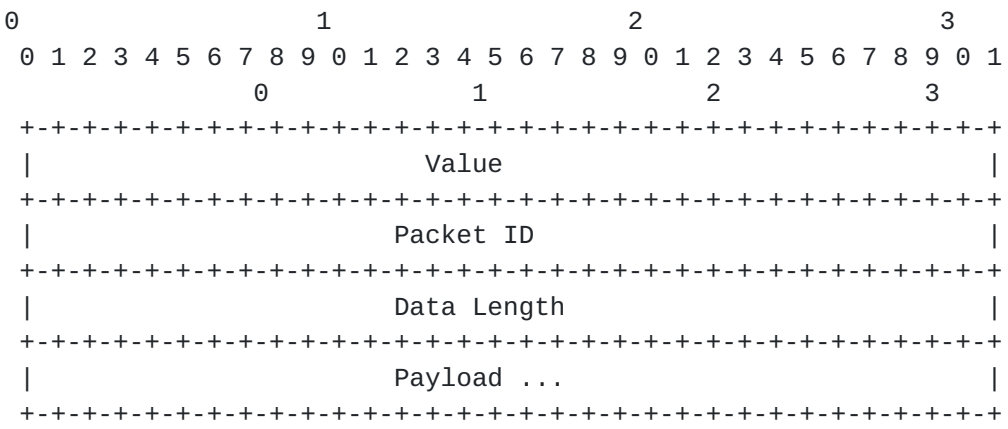
[We still need to talk about classes and filters; later -- jhs]

5.2. IP Service NETLINK_FIREWALL

This service allows CPCs to receive packets sent by the IPv4 fire-
wall service in the FE.

Two types of messages exist that can be sent from CPC to FEC. These
are: Mode messages and Verdict messages. The formats are described
below.

The Verdict message format is as follows



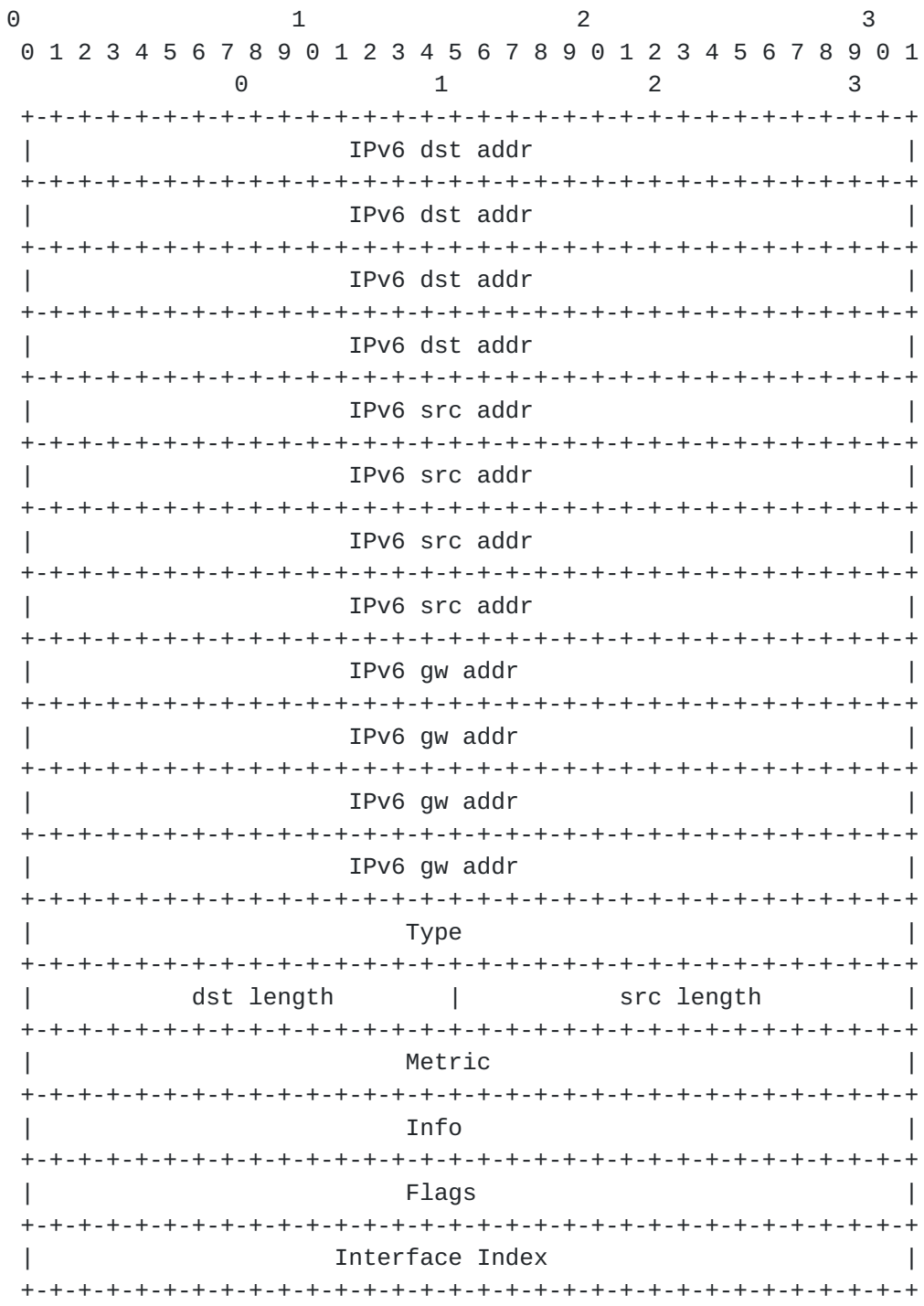
A ipq_packet_msg packet type is sent from the FEC to the CPC. The format is described below ==> We need to complete this later

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.

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
USA
1 503 264 0334
hormuzd.m.khosravi@intel.com

Andi Kleen
SuSE
Stahlgruberring 28
81829 Muenchen
Germany

Alexey Kuznetsov
INR/Swsoft
Moscow
Russia