

Network Working Group
Request for Comments: 3549
Category: Informational

J. Salim
Znyx Networks
H. Khosravi
Intel
A. Kleen
Suse
A. Kuznetsov
INR/Swsoft
July 2003

Linux Netlink as an IP Services Protocol

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This document describes Linux Netlink, which is used in Linux both as an intra-kernel messaging system as well as between kernel and user space. The focus of this document is to describe Netlink's functionality as a protocol between a Forwarding Engine Component (FEC) and a Control Plane Component (CPC), the two components that define an IP service. As a result of this focus, this document ignores other uses of Netlink, including its use as an intra-kernel messaging system, as an inter-process communication scheme (IPC), or as a configuration tool for other non-networking or non-IP network services (such as decnet, etc.).

This document is intended as informational in the context of prior art for the ForCES IETF working group.

Table of Contents

| | | |
|--------|---|----|
| 1. | Introduction | 2 |
| 1.1. | Definitions | 3 |
| 1.1.1. | Control Plane Components (CPCs)..... | 3 |
| 1.1.2. | Forwarding Engine Components (FECs)..... | 3 |
| 1.1.3. | IP Services | 5 |
| 2. | Netlink Architecture | 7 |
| 2.1. | Netlink Logical Model | 8 |
| 2.2. | Message Format..... | 9 |
| 2.3. | Protocol Model..... | 9 |
| 2.3.1. | Service Addressing..... | 10 |
| 2.3.2. | Netlink Message Header..... | 10 |
| 2.3.3. | FE System Services' Templates..... | 13 |
| 3. | Currently Defined Netlink IP Services..... | 16 |
| 3.1. | IP Service NETLINK_ROUTE..... | 16 |
| 3.1.1. | Network Route Service Module..... | 16 |
| 3.1.2. | Neighbor Setup Service Module..... | 20 |
| 3.1.3. | Traffic Control Service..... | 21 |
| 3.2. | IP Service NETLINK_FIREWALL..... | 23 |
| 3.3. | IP Service NETLINK_ARPD..... | 27 |
| 4. | References..... | 27 |
| 4.1. | Normative References..... | 27 |
| 4.2. | Informative References..... | 28 |
| 5. | Security Considerations..... | 28 |
| 6. | Acknowledgements..... | 28 |
| | Appendix 1: Sample Service Hierarchy | 29 |
| | Appendix 2: Sample Protocol for the Foo IP Service..... | 30 |
| | Appendix 2a: Interacting with Other IP services..... | 30 |
| | Appendix 3: Examples..... | 31 |
| | Authors' Addresses..... | 32 |
| | Full Copyright Statement..... | 33 |

1. Introduction

The concept of IP Service control-forwarding separation was first introduced in the early 1990s by the BSD 4.4 routing sockets [9]. 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, could 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 and management 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 RFC 1812 IPv4 forwarding.

The motivation for this document is not to list every possible service for which Netlink is applied. In fact, we leave out a lot of services (multicast routing, tunneling, policy routing, etc). Neither is this document intended to be a tutorial on Netlink. The idea is to explain the overall Netlink view with a special focus on the mandatory building blocks within the ForCES charter (i.e., IPv4 and QoS). This document also serves to capture prior art to many mechanisms that are useful within the context of ForCES. The text is limited to a subset of what is available in kernel 2.4.6, the newest kernel when this document was first written. It is also limited to IPv4 functionality.

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

1.1. Definitions

A Control Plane (CP) is an execution environment that may have several sub-components, which we refer to as CPCs. Each CPC provides control for a different IP service being executed by a Forwarding Engine (FE) component. This relationship 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 an FE component is the service abstraction.

1.1.1. Control Plane Components (CPCs)

Control Plane Components encompass signalling protocols, with diversity ranging from dynamic routing protocols, such as OSPF [5], to tag distribution protocols, such as CR-LDP [7]. Classical management protocols and activities also fall under this category. These include SNMP [6], COPS [4], and 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 Network Element (NE) component: the FE. The result of the configuration defines the way that packets traversing the FE are treated.

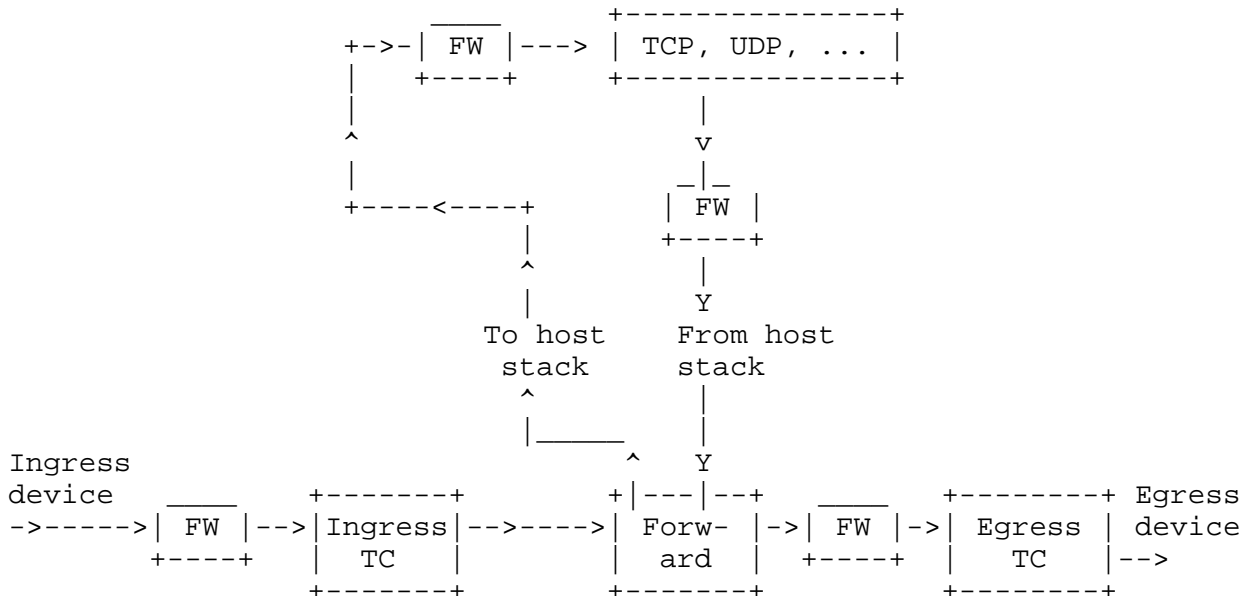
1.1.2. Forwarding Engine Components (FECs)

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 an IP service, as defined by the Control Plane Components for that IP service. Different services will utilize different FECs. Service modules may be chained to achieve a

more complex service (refer to the Linux FE model, described later). When built for providing a specific service, the FE service component will adhere to a forwarding model.

1.1.2.1. Linux IP Forwarding Engine Model



The figure above shows the Linux FE model per device. The only mandatory part of the datapath is the Forwarding module, which is RFC 1812 conformant. The different Firewall (FW), Ingress Traffic Control, and Egress Traffic Control building blocks are not mandatory in the datapath and may even be used to bypass the RFC 1812 module. These modules are shown as simple blocks in the datapath but, in fact, could be multiple cascaded, independent submodules within the indicated blocks. More information can be found at [10] and [11].

Packets arriving at the ingress device first pass through a firewall module. Packets may be dropped, munged, etc., by the firewall module. The incoming packet, depending on set policy, may then be passed via an Ingress Traffic Control module. Metering and policing activities are contained within the Ingress TC module. Packets may be dropped, depending on metering results and policing policies, at this module. Next, the packet is subjected to the only non-optional module, the RFC 1812-conformant Forwarding module. The packet may be dropped if it is nonconformant (to the many RFCs complementing 1812 and 1122). This module is a juncture point at which packets destined to the forwarding NE may be sent up to the host stack.

Packets that are not for the NE may further traverse a policy routing submodule (within the forwarding module), if so provisioned. Another firewall module is walked next. The firewall module can drop or munge/transform packets, depending on the configured sub-modules encountered and their policies. If all goes well, the Egress TC module is accessed next.

The Egress TC may drop packets for policing, scheduling, congestion control, or rate control reasons. Egress queues exist at this point and any of the drops or delays may happen before or after the packet is queued. All is dependent on configured module algorithms and policies.

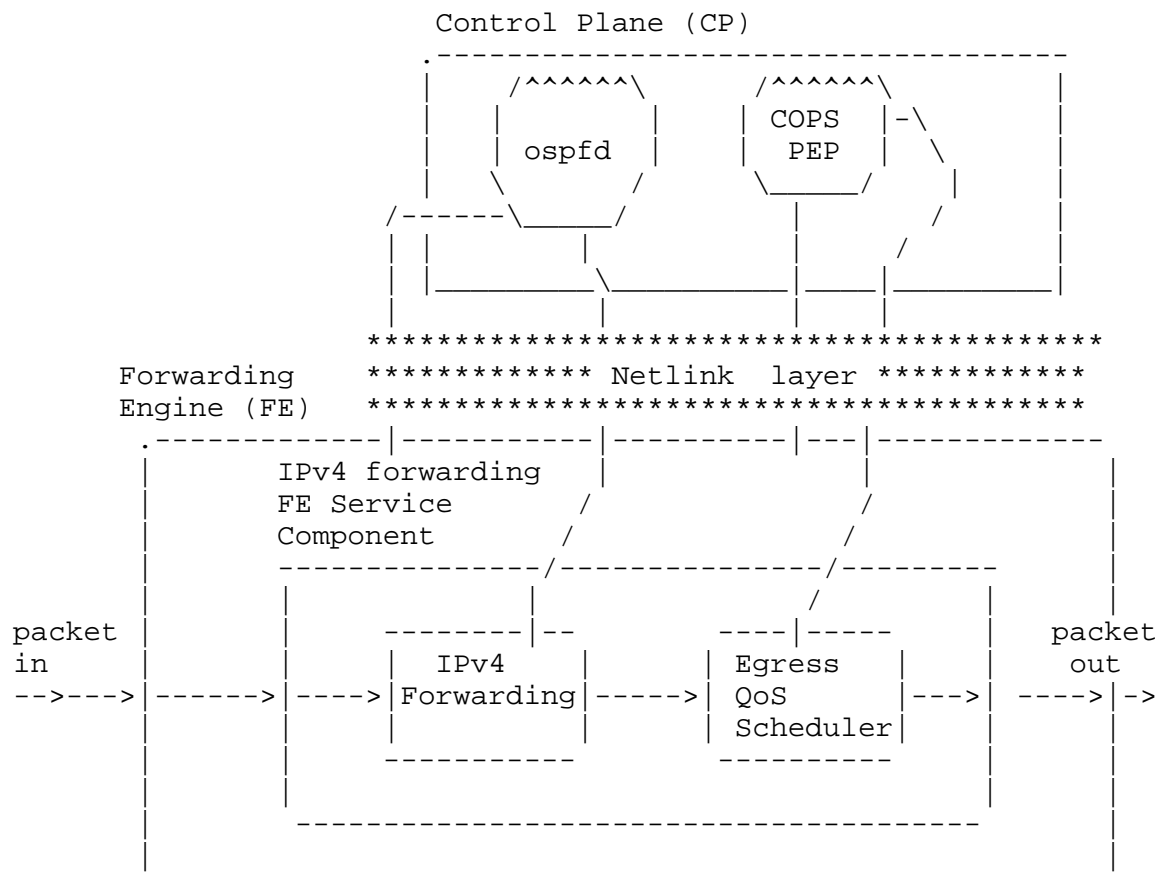
1.1.3. IP Services

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 the FEC.

The time span of the service is from the moment when the packet arrives at the NE to the moment that it departs. In essence, an IP service in this context is a Per-Hop Behavior. CP components running on NEs define the end-to-end path control for a service by running control/signaling protocol/management-applications. These distributed CPCs unify 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) for 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.

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 illustrate how the setup fits in this described model.



The above diagram illustrates `ospfd`, an OSPF protocol control daemon, and a COPS Policy Enforcement Point (PEP) as distinct CPCs. The IPv4 FE component includes the IPv4 Forwarding service module as well as the Egress Scheduling service module. Another service might add a policy forwarder between the IPv4 forwarder and the QoS egress scheduler. A simpler classical service would have constituted only the IPv4 forwarder.

Over the years, it has become important to add additional services to routers to meet emerging requirements. More complex services extending classical forwarding have been 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 service headers) include firewalling, QoS in Diffserv and RSVP, NAT, 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 for which a service provider would be able to charge.

2. Netlink Architecture

Control of IP service components is defined by using templates.

The FEC and CPC participate to deliver the IP service by communicating using these templates. The FEC might continuously get updates from the Control Plane Component on how to operate the service (e.g., for v4 forwarding or for route additions or deletions).

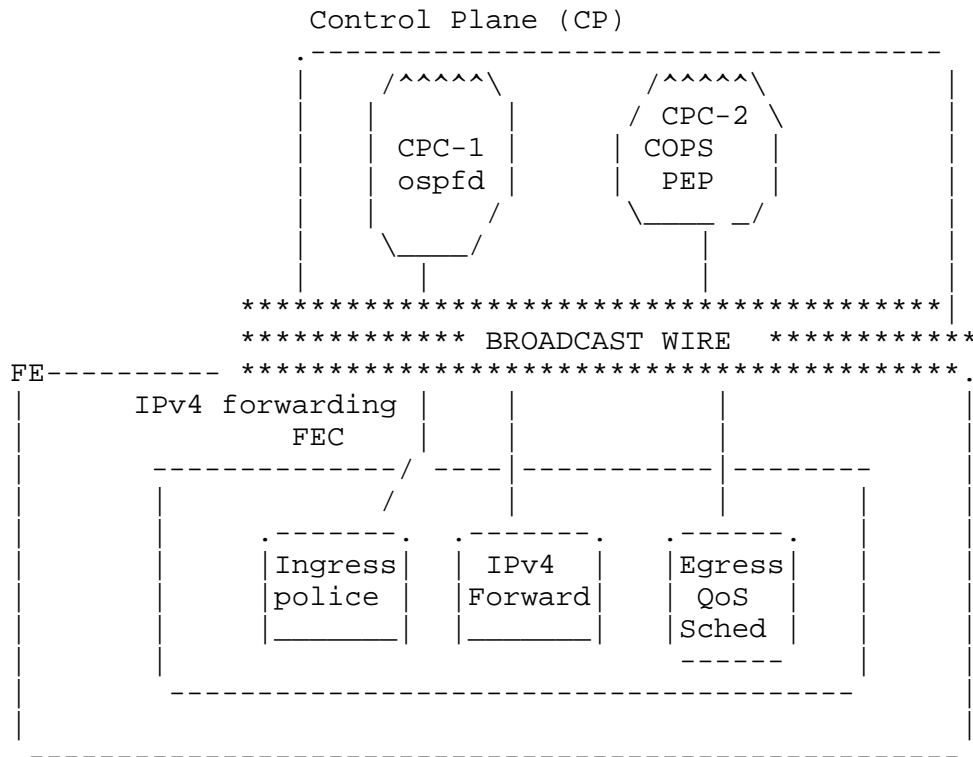
The interaction between the FEC and the CPC, in the Netlink context, defines a protocol. Netlink provides mechanisms for the CPC (residing in user space) and the FEC (residing in kernel space) to have their own protocol definition -- kernel space and user space just mean different protection domains. Therefore, a wire protocol is needed to communicate. The wire protocol is normally provided by some privileged service that is able to copy between multiple protection domains. We will refer to this service as the Netlink service. The Netlink service can also be encapsulated in 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 mechanism further other than to say that it is available. Throughout this document, we will refer interchangeably to the FEC to mean kernel space and the CPC to mean user space. This denomination 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.

2.1. Netlink Logical Model

In the diagram below we show a simple FEC<->CPC logical relationship. We use the 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.

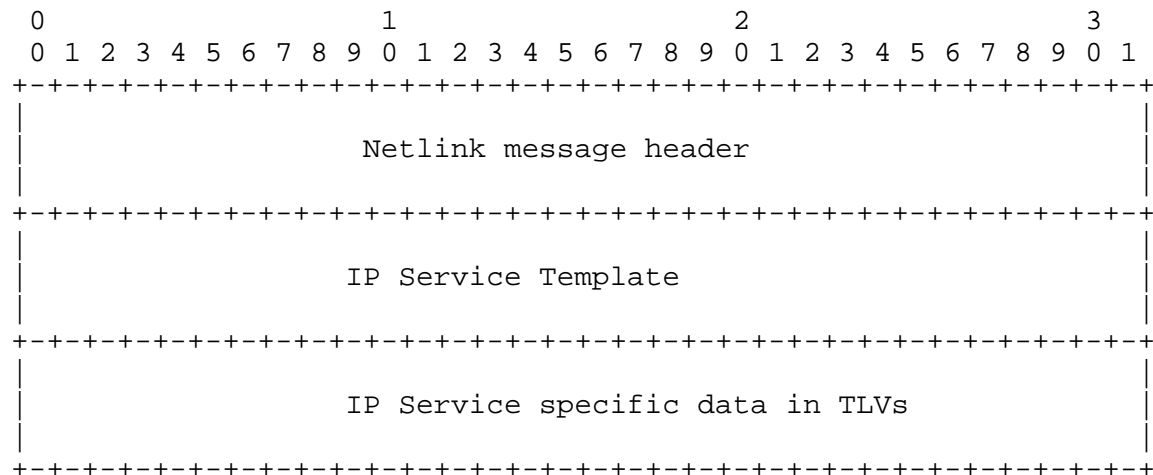
Nodes (CPCs or FECs as illustrated above) connect to the wire and register to receive specific messages. CPCs may connect to multiple wires if it helps them to control the service better. All nodes (CPCs and FECs) dump packets on the broadcast wire. Packets can be discarded by the wire if they are malformed or not specifically formatted for the wire. Dropped packets are not seen by any of the nodes. The Netlink service may signal an error to the sender if it detects a malformed Netlink packet.

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

Appendices 1 and 2 have a high level overview of this interaction.

2.2. Message Format

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



The Netlink message is used to communicate between the FEC and CPC for parameterization of the FECs, asynchronous event notification of FEC events to the CPCs, and statistics querying/gathering (typically by a 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 parameterizations are in TLV (Type-Length-Value) format and are unique to the service.

The different parts of the netlink message are discussed in the following sections.

2.3. Protocol Model

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

2.3.1. Service Addressing

Access is provided by first connecting to the service on the FE. The connection is achieved 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 does not distinguish between 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 either issuing a service-specific command (from the CPC to the FEC, mostly for configuration purposes), issuing a statistics-collection command, or subscribing/unsubscribing to service events. Closing the socket terminates the transaction. Refer to Appendices 1 and 2 for examples.

2.3.2. Netlink Message Header

Netlink messages consist of a byte stream with one or multiple Netlink headers and an associated payload. If the payload is too big to fit into a single message it, can be split over multiple Netlink messages, collectively 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.

```

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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     Length                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               Type               |               Flags               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               Sequence Number               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               Process ID (PID)               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

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, 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 (CPC) to kernel space (FEC). |
| NLM_F_ECHO | Echo this request. Typical direction of request is from user space (CPC) to kernel space (FEC). |

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 entries 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 ID (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.

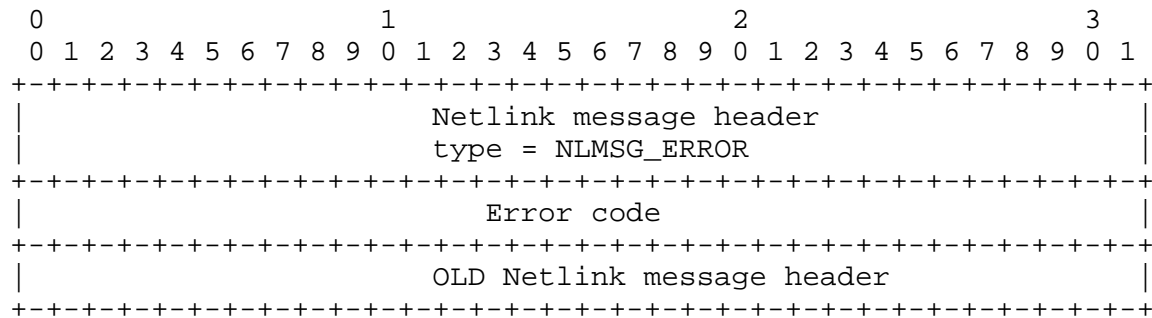
2.3.2.1. Mechanisms for Creating Protocols

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.

2.3.2.2. The ACK Netlink Message

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



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 Netlink message header, which 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 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).

2.3.3. FE System Services' Templates

These are services that are offered by the system for general use by other services. They include the ability to configure, gather statistics and listen to changes in shared resources. IP address management, link events, etc. fit here. We create this section for these services for logical separation, despite the fact that they are accessed via the NETLINK_ROUTE FEC. The reason that they exist within NETLINK_ROUTE is due to historical craft: the BSD 4.4 Route Sockets implemented them as part of the IPv4 forwarding sockets.

2.3.3.1. Network Interface Service Module

This service provides the ability to create, remove, or get information about a specific network interface. The network interface can be either physical or virtual and is network protocol independent (e.g., an x.25 interface can be defined via this message). The Interface service message template is shown below.

| 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 |
| Family | Reserved | Device Type | |
| Interface Index | | | |
| Device Flags | | | |
| Change Mask | | | |

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, a tunnel, etc. We are interested only in IPv4, although the link type is L3 protocol-independent.

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:

| 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 | 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

2.3.3.2. 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.

```

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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  Family   |      Length   |      Flags   |      Scope   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     Interface Index                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

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.

le 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. |
| 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.

3. Currently Defined Netlink IP Services

Although there are many other IP services defined that are using Netlink, as mentioned earlier, we will talk only about a handful of those integrated into kernel version 2.4.6. These are:

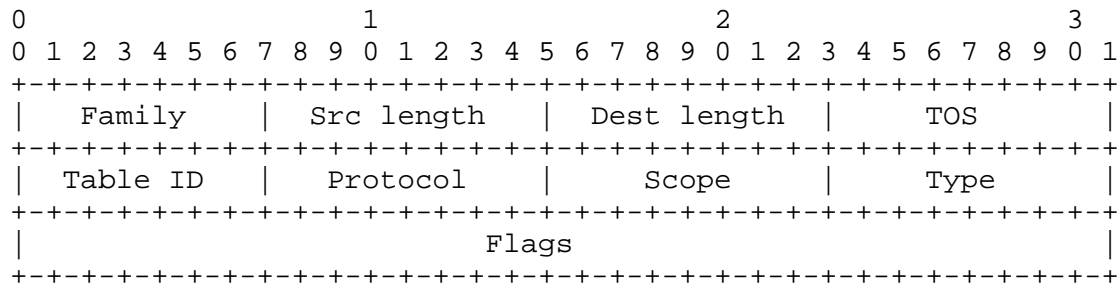
NETLINK_ROUTE, NETLINK_FIREWALL, and NETLINK_ARPD.

3.1. 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 to collect statistics.

3.1.1. 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: 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. See <linux/rtnetlink.h> for the routing daemon identifiers that are already assigned.

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

3.1.2. Neighbor Setup Service Module

This service provides the ability to add, remove, or receive information about a neighbor table entry (e.g., an ARP entry or an IPv4 neighbor solicitation, etc.). The service message template is shown below.

```

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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   Family   |   Reserved1  |   Reserved2   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     Interface Index                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               State                 |   Flags   |   Type   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

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. |

queuing discipline and has a queue associated with it. The queue may be subject to a simple algorithm, like FIFO, or a more complex one, like RED or a token bucket. The outermost queuing discipline, which is referred to as the parent is typically associated with a scheduler. Within this scheduler hierarchy, however, may be other scheduling algorithms, making the Linux Egress TC very flexible.

The service message template that makes this possible is shown below. This template is used in both the ingress and the egress queuing disciplines (refer to the egress traffic control model in the FE model section). Each of the specific components of the model has unique attributes that describe it best. The common attributes are described below.

```

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
+-----+-----+-----+-----+-----+-----+-----+-----+
|   Family   |   Reserved1   |           Reserved2           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Interface Index                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Qdisc handle                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Parent Qdisc                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     TCM Info                                       |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Family: 8 bits

Address Family: AF_INET for IPv4; and AF_INET6 for IPV6.

Interface Index: 32 bits

The unique interface index.

Qdisc handle: 32 bits

Unique identifier for instance of queuing 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: 32 bits

Used in hierarchical layering of queuing 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: 32 bits

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. From the CPC towards the direction of the FEC, this is typically set to 0

except when used in the context of filters. In that case, this 32-bit field is split into a 16-bit priority field and 16-bit protocol field. The protocol is defined in kernel source `<include/linux/if_ether.h>`, however, the most commonly used one is `ETH_P_IP` (the IP protocol).

The priority is used for conflict resolution when filters intersect in their expressions.

Generic attributes applicable to this service:

| Attribute | Description |
|--------------------------|---|
| ----- | |
| <code>TCA_KIND</code> | Canonical name of FE component. |
| <code>TCA_STATS</code> | Generic usage statistics of FEC |
| <code>TCA_RATE</code> | rate estimator being attached to FEC. Takes snapshots of stats to compute rate. |
| <code>TCA_XSTATS</code> | Specific statistics of FEC. |
| <code>TCA_OPTIONS</code> | Nested FEC-specific attributes. |

Appendix 3 has an example of configuring an FE component for a FIFO Qdisc.

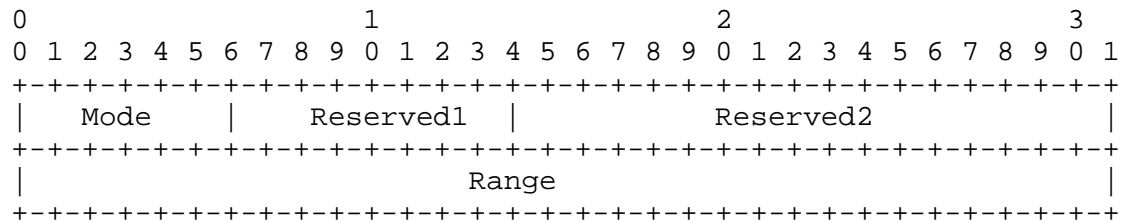
Additional Netlink message types applicable to this service:
`RTM_NEWQDISC`, `RTM_DELQDISC`, `RTM_GETQDISC`, `RTM_NEWTCLASS`,
`RTM_DELTCLASS`, `RTM_GETTCLASS`, `RTM_NEWTFILTER`, `RTM_DELTFILTER`, and
`RTM_GETTFILTER`.

3.2. IP Service `NETLINK_FIREWALL`

This service allows CPCs to receive, manipulate, and re-inject packets via the IPv4 firewall service modules in the FE. A firewall rule is first inserted to activate packet redirection. The CPC informs the FEC whether it would like to receive just the metadata on the packet or the actual data and, if the metadata is desired, what is the maximum data length to be redirected. The redirected packets are still stored in the FEC, waiting a verdict from the CPC. The verdict could constitute a simple accept or drop decision of the packet, in which case the verdict is imposed on the packet still sitting on the FEC. The verdict may also include a modified packet to be sent on as a replacement.

Two types of messages exist that can be sent from CPC to FEC. These are: Mode messages and Verdict messages. Mode messages are sent immediately to the FEC to describe what the CPC would like to receive. Verdict messages are sent to the FEC after a decision has been made on the fate of a received packet. The formats are described below.

The mode message is described first.



Mode: 8 bits

Control information on the packet to be sent to the CPC. The different types are:

- IPQ_COPY_META Copy only packet metadata to CPC.
- IPQ_COPY_PACKET Copy packet metadata and packet payloads to CPC.

Range: 32 bits

If IPQ_COPY_PACKET, this defines the maximum length to copy.

A packet and associated metadata received from user space looks as follows.

```

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
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Packet ID                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Mark                                          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     timestamp_m                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     timestamp_u                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     hook                                          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     indev_name                                   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     outdev_name                                  |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          hw_protocol          |          hw_type          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          hw_addrlen          |          Reserved          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          hw_addr             |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          data_len            |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          Payload . . .      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Packet ID: 32 bits

The unique packet identifier as passed to the CPC by the FEC.

Mark: 32 bits

The internal metadata value set to describe the rule in which the packet was picked.

timestamp_m: 32 bits

Packet arrival time (seconds)

timestamp_u: 32 bits

Packet arrival time (useconds in addition to the seconds in timestamp_m)

hook: 32 bits

The firewall module from which the packet was picked.

indev_name: 128 bits
ASCII name of incoming interface.

outdev_name: 128 bits
ASCII name of outgoing interface.

hw_protocol: 16 bits
Hardware protocol, in network order.

hw_type: 16 bits
Hardware type.

hw_addrln: 8 bits
Hardware address length.

hw_addr: 64 bits
Hardware address.

data_len: 32 bits
Length of packet data.

Payload: size defined by data_len
The payload of the packet received.

The Verdict message format is as follows

```

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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Value                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Packet ID                           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Data Length                         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Payload . . .                      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Value: 32 bits

This is the verdict to be imposed on the packet still sitting in the FEC. Verdicts could be:

| | |
|-----------|--|
| NF_ACCEPT | Accept the packet and let it continue its traversal. |
| NF_DROP | Drop the packet. |

Packet ID: 32 bits

The packet identifier as passed to the CPC by the FEC.

Data Length: 32 bits

The data length of the modified packet (in bytes). If you don't modify the packet just set it to 0.

Payload:

Size as defined by the Data Length field.

3.3. IP Service NETLINK_ARPD

This service is used by CPCs for managing the neighbor table in the FE. The message format used between the FEC and CPC is described in the section on the Neighbor Setup Service Module.

The CPC service is expected to participate in neighbor solicitation protocol(s).

A neighbor message of type RTM_NEWNEIGH is sent towards the CPC by the FE to inform the CPC of changes that might have happened on that neighbor's entry (e.g., a neighbor being perceived as unreachable).

RTM_GETNEIGH is used to solicit the CPC for information on a specific neighbor.

4. References

4.1. Normative References

- [1] Braden, R., Clark, D. and S. Shenker, "Integrated Services in the Internet Architecture: an Overview", RFC 1633, June 1994.
- [2] Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, June 1995.
- [3] Blake, S., Black, D., Carlson, M., Davies, E, Wang, Z. and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.
- [4] Durham, D., Boyle, J., Cohen, R., Herzog, S., Rajan, R. and A. Sastry, "The COPS (Common Open Policy Service) Protocol", RFC 2748, January 2000.
- [5] Moy, J., "OSPF Version 2", STD 54, RFC 2328, April 1998.
- [6] Case, J., Fedor, M., Schoffstall, M. and C. Davin, "Simple Network Management Protocol (SNMP)", STD 15, RFC 1157, May 1990.

- [7] Andersson, L., Doolan, P., Feldman, N., Fredette, A. and B. Thomas, "LDP Specification", RFC 3036, January 2001.
- [8] Bernet, Y., Blake, S., Grossman, D. and A. Smith, "An Informal Management Model for DiffServ Routers", RFC 3290, May 2002.

4.2. Informative References

- [9] G. R. Wright, W. Richard Stevens. "TCP/IP Illustrated Volume 2, Chapter 20", June 1995.
- [10] <http://www.netfilter.org>
- [11] <http://diffserv.sourceforge.net>

5. Security Considerations

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

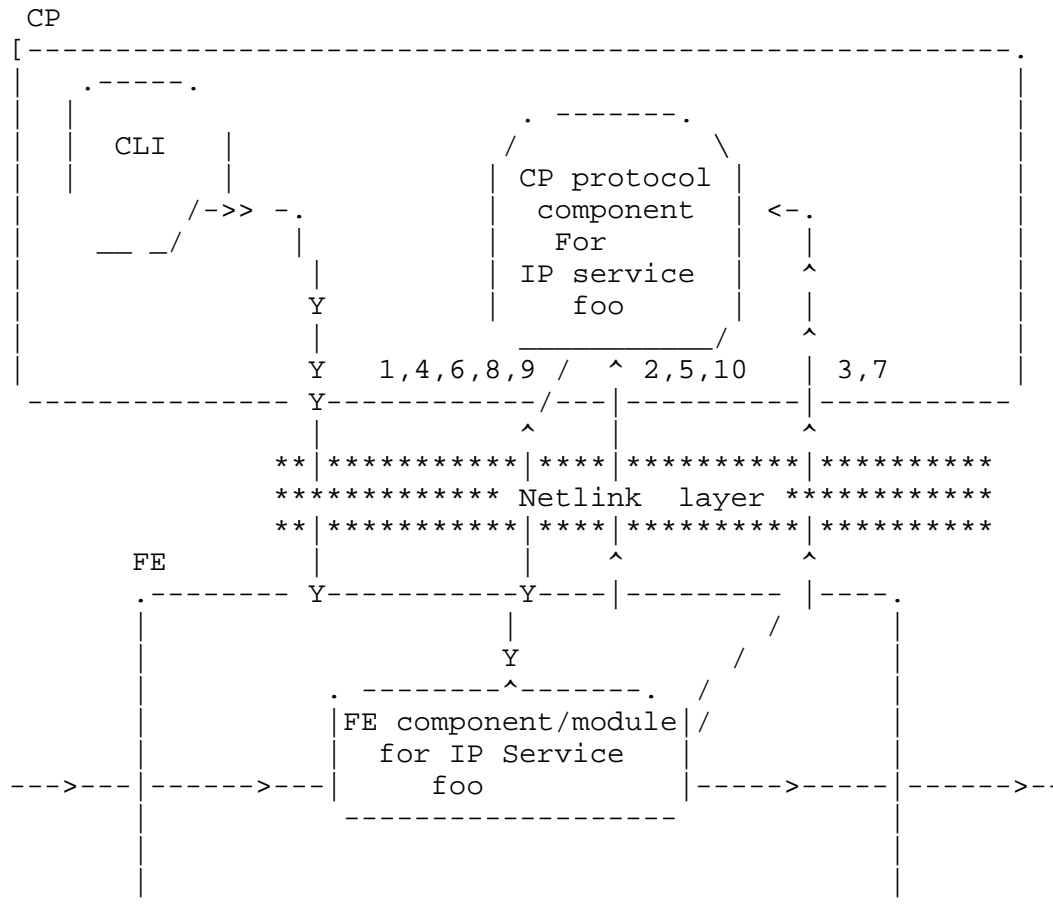
6. Acknowledgements

- 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.
- 3) Jeremy Ethridge for taking the role of someone who did not understand Netlink and reviewing the document to make sure that it made sense.

Appendix 1: 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 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 the 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 asynchronous events for service foo.
- 3) Bind to listen to specific asynchronous FE events.

Appendix 2: Sample Protocol for the Foo IP Service

Our example IP service foo is used again to demonstrate how one can deploy a simple IP service control using Netlink.

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

- 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).
- 8) Register the protocol-specific packets you would like the FE to forward to you.
- 9) Send service-specific foo commands and receive responses for them, if needed.

Appendix 2a: 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 to the foo component. If the CLI issues 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. For example, if an 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 3: Examples

In this example, we show a simple configuration Netlink 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 100 packets. We assume that the queue is in a hierarchical setup with a parent 100:0 and a classid of 100:1 and that it is to be installed on a device with an ifindex of 4.

```

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
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Length (52)                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Type (RTM_NEWQDISC) | Flags (NLM_F_EXCL |                                     |
|                     | NLM_F_CREATE | NLM_F_REQUEST) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Sequence Number(arbitrary number) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Process ID (0) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Family(AF_INET) | Reserved1 | Reserved1 |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Interface Index (4) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Qdisc handle (0x1000001) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Parent Qdisc (0x1000000) |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     TCM Info (0) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Type (TCA_KIND) | Length(4) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value ("pfifo") |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Type (TCA_OPTIONS) | Length(4) |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Value (limit=100) |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Authors' Addresses

Jamal Hadi Salim
Znyx Networks
Ottawa, Ontario
Canada

EMail: hadi@znyx.com

Hormuzd M Khosravi
Intel
2111 N.E. 25th Avenue JF3-206
Hillsboro OR 97124-5961
USA

Phone: +1 503 264 0334
EMail: hormuzd.m.khosravi@intel.com

Andi Kleen
SuSE
Stahlgruberring 28
81829 Muenchen
Germany

EMail: ak@suse.de

Alexey Kuznetsov
INR/Swsoft
Moscow
Russia

EMail: kuznet@ms2.inr.ac.ru

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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

