Rtnetlink dump filtering in the kernel

Roopa Prabhu

Cumulus Networks,
Mountain View, CA, USA,
roopa@cumulusnetworks.com

Abstract

Rtnetlink dump handlers supported by the kernel are a useful way to query state of the kernel network databases. Most rtnetlink dump handlers return data for all network objects in the corresponding networking subsystem databases today, e.g. RTM_GETLINK returns data for all network interfaces in the system. With no rtnetlink dump filtering support in the kernel, the burden is on user-space to filter such dumps. This does not scale on systems with a large number of network interfaces or large routing databases. Such systems are not uncommon given that Linux is being deployed on network switches, routers, hypervisors and other devices in the data center today. Filtering in user-space is not scalable. This paper looks at scalability problems with rtnetlink dumps and discusses possible solutions to filter such dumps in the kernel. We will look at a consistent way to filter such dumps across all network object types using existing infrastructure provided by the kernel.

Keywords

Netlink, Rtnetlink, iproute2

Introduction

The Linux kernel netlink API is one of the main interfaces the Linux kernel provides to user-space applications [3]. Most Linux network applications today use netlink to talk to the kernel. Rtnetlink is a netlink bus used by all kernel networking subsystems including network interfaces, routing, fdb and neighbour. Some kernel networking subsystems also provide services on the generic netlink bus [2]. The Linux kernel networking subsystems register handlers with Rtnetlink core with a message type and family. For further details on Netlink families and msg types see [1], [3], [4].

User-space applications can talk to a subsystem on the Rtnetlink bus by using the corresponding family, msg type and attributes with the socket API. Throughout the paper we refer to a network subsystem entry as a 'networking object'. This can be a network interface, a route, an address, a forwarding database (FDB) entry or a neighbour table entry.

Rtnetlink handlers support the following message types today:

- RTM_NEW* to create or set networking attributes on a network object
- RTM_DEL* to delete an object
- RTM_GET* to get an object or when used with NLM_F_DUMP netlink flag dump the networking subsystem database (eg all links)

Example message types supported for links [4]:

- RTM_NEWLINK to create or set attributes on a link
- RTM_DELLINK to delete a link
- RIM_GETLINK to get a link or dump the kernel link database when NLM_F_DUMP flag is set

In this paper we focus on Rtnetlink dump handlers in the kernel (RTM_GET* with NLM_F_DUMP). Rtnetlink dump handlers are invoked when the user is requesting a dump of a kernel object database (links, routes, fdb, neigh). We discuss Rtnetlink dumps in the context of short lived and long running networking applications:

- A short lived application requests a dump from the kernel, processes dump entries and exits. e.g. an application (like iproute2 [7]) polling for 'stale' neighbour table entries to refresh them every 30s

- A long running application (or daemon) requests for a dump from the kernel at startup to build a cache of kernel objects. It then listens to notifications from the kernel to update the cache and uses the cache as a copy of the kernel database objects for further processing and actions. e.g. a user-space hardware accelerating switch driver [10], [11]

Problem

Most Rtnetlink dump handlers today dump the entire kernel networking subsystem databases. e.g. address dump returns addresses on all interfaces in the system. The application filters these dumps in user-space. This does not scale well on systems with large kernel network databases or large number of network interfaces when all the user wants is addresses on one interface. A short lived application requesting a dump from the kernel every poll interval will suffer from parsing and processing large dumps. e.g. in a system with thousands of neighbour table entries, a short lived application looking for a few stale neighbour entries will request a dump of the neighbour table from the kernel every poll interval.

```
# the below iproute command execution requires requesting the
# kernel for a full dump of all interface details in the system and
# then looking for eth0 in users-space

ip addr show dev eth0


# showing all bridge interfaces in the system requires iproute2
#to get a dump of details of all interfaces in the system and
#filter bridge devices in user-space

ip link show type bridge
```

Kernel provides a few ways to dump efficiently or filter dumps in the kernel in some cases:

- Netlink messages can be filtered in the kernel using BPF socket filters at the socket level [6]
- Memory mapped IO can be used with netlink for better performance and reduced overhead during dumps [8]
- Netlink attribute IFLA_EXT_MASK to indicate pre-defined filter masks [12]
- Netlink messages can also be filtered using attributes passed by the user in the incoming dump request message [5]. This method is the focus of this paper. We propose incremental improvements to this method covering all networking subsystems.

In some cases the rtnetlink doit handler has been proven to be useful to query a targeted object in the kernel as shown in [9].
In this paper we focus on filtering dumps using Netlink attributes in the dump request message. The Linux kernel currently supports required infrastructure to parse and filter dumps based on incoming messages.

Related Work

Recent work in the bridge FDB area [5] supports filtering of FDB entries in the kernel based on attributes in the dump request message. The FDB filtering needs changes to extend it to filter by other attributes of an FDB entry. This paper proposes an implementation for these changes.

Rtnetlink dump filtering in the kernel

The next few sections will look at ways to provide consistent dump filtering in kernel networking subsystems. This can be achieved by making dump request message format consistent with the corresponding RTM_NEW* or RTM_SET* message format. This makes possible for filtering in kernel based on similar attributes available during sets. This is implied in most cases but in this paper we would like to call it out because the format mismatch can break the ability to filter as it already has in some cases in the kernel.

The following guidelines can be used to add or support filtering of netlink dump requests in the kernel:

- Use the incoming dump request message as a filter message
- The dump message format must be same as the RTM_SET* or RTM_NEW* message
- Parse the incoming dump message into a filter of netlink attributes
- Netlink filter attributes may need to be passed or cascaded to subsequent subsystem handlers:

We will look at filtering in the kernel for the following dumps:

- Link dumps
- FDB dumps
- Neighbour table dumps
- Address dumps

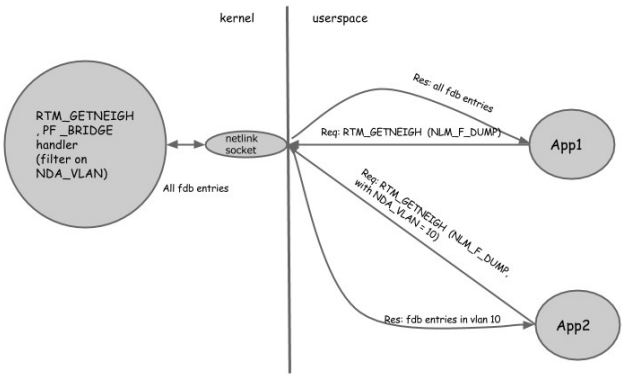

Figure 1: Dump filtering in kernel (example shows filtering fdb entries by vlan)

Link dumps

Rtnetlink Link dump handlers use RTM_GETLINK msg type and IFLA_* attributes. Link dump filtering can be achieved by enforcing a 'struct ifinfomsg' and IFLA_* attributes in the link dump request message. The RTM_GETLINK dump handler in the kernel can parse the incoming link attributes into a filter structure and pass it to subsequent netdev getlink handlers.

Link dumps can be filtered on any fields in the incoming 'struct ifinfomsg', like interface flags. They can also be filtered based on the supported netlink attributes. e.g.,

- IFLA_GROUP to filter interfaces belonging to a group
- IFLA_MASTER to filter interfaces with a specific master interface
- IFLA_LINK to filter logical interfaces with this interface as the link

```
# iproute2 examples showing filtering links

# show all 'up' interfaces
ip link show up
```

```
# show links with group test
ip link show group test

# show links with master br0
ip link show master br0

# show interfaces with link eth0
ip link show link eth0
```

FDB Dumps

Kernel forwarding database dumps are supported by most kernel drivers maintaining a forwarding database. e.g. bridge and vxlan drivers. FDB dump handlers use the RTM_*NEIGH msg type with NDA_* attributes. FDB dump filtering can be achieved by enforcing a 'struct ndmsg' and NDA_* attributes in the FDB dump request messages. Bridge and vxlan FDB dumps can be filtered on any of the below fields in 'struct ndmsg':

- ndm_state – sta of the entry (NUD_PERMANENT, NUD_REACHABLE and others)
- ndm_type - type of entry (static or local)
- ndm_ifindex – interface the fdb entry points to

They can also be filtered based on any of the NDA_* netlink neigh attributes:

- bridge fdb entries can be filtered based on the below attributes:
  - NDA_DST - filter by dst
  - NDA_LLADDR - filter by addr
  - NDA_VLAN - filter by vlan
  - NDA_MASTER - filter by master interface index
- vxlan fdb entries can be filtered based on the below attributes:
  - NDA_DST filter by dst
  - NDA_LLADDR filter by addr
  - NDA_PORT filter by remote port
  - NDA_VNI filter by vni id for vxlan fdb
  - NDA_IFINDEX filter by remote port ifindex for vxlan fdb

```
# iproute2 example showing bridge fdb dump filtering

# show fdb for bridge br0
bridge fdb show br br0

# show fdb for bridge port eth0
bridge fdb show brport eth0

# show static fdb entries
bridge fdb show static

# show fdb entries with dst 172.16.20.103
bridge fdb show dst 172.16.20.103

# show fdb entries with vlan 10
bridge fdb show vlan 10

# show vxlan fdb entries with vni 100
bridge fdb show vni 100

# show vxlan fdb entries with remote port 4783
bridge fdb show port 4783
```

Neighbour table dumps

Neighbour tables dump handlers use the msg type RTM_GETNEIGH and a subset of NDA_* attributes (similar to FDB dumps described in the previous section). Neighbour table entries can be filtered by fields in 'struct ndmsg':

- ndm_state (NUD_PERMANENT, NUD_REACHABLE and others)
- ndb_type - neighbour entry type (static or local)
- ndm_ifindex – neighbour entry pointing to an interface

```
# iproute2 examples filtering neigh dumps

# show reachable neigh entries
ip neigh show nud reachable

# show permanent neigh entries
ip neigh show nud permanent

# show stale neigh entries
ip neigh show nud stale

# show neigh entries for dev eth0
ip neigh show dev eth0
```

Address dumps

Address dump handlers use the messag type RTM_GETADDR and IFA_* attributes. RTM_GETADDR dump filtering can be achieved by enforcing a 'struct ifaddrmsg' and IFA_* attributes in the dump request messages. Address table entries can be filtered on fields in 'struct ifaddrmsg':

- ifa_flags filter addresses with address flags
- ifa_scope filter address with given scope
- ifa_index dump addresses belonging to an interface

They can also be filtered based on the below netlink attributes:

- IFA_LABEL filter addresses with a given label
- IFLA_FLAGS filter on flags like permanent, dynamic, secondary, primary

```
# show addresses belonging to an interface

ip addr show dev eth0


# Timing measurements on a system with 2000 interfaces with addresses

# Before: filtering in userspace

$time ip addr show dev eth0

3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000

link/ether 00:01:00:00:01:cc brd ff:ff:ff:ff:ff:ff

inet 192.168.0.15/24 brd 192.168.0.255 scope global eth0

valid_lft forever preferred_lft forever

real 0m0.060s

user 0m0.040s
sys 0m0.020s


# After: filtering in kernel space

# time ip addr show dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 00:01:00:00:01:cc brd ff:ff:ff:ff:ff:ff
inet 192.168.0.15/24 brd 192.168.0.255 scope global eth0
valid_lft forever preferred_lft forever

real 0m0.028s
user 0m0.004s
sys 0m0.020s
```

Conclusions

Filtered dump support in the kernel will benefit short lived applications and will avoid the need for building netlink caches in user-space for these applications. This paper proposes a method to implement filtered dump support.

References

1. J. Hadi Salim, H. Khosravi, A. Kleen, A. Kuznetsov, Linux Netlink as an IP Services Protocol, RFC 3549, July 2003
2. Generic netlink: http://lwn.net/Articles/208755/
3. Pablo Neira Ayuso,Rafael M. Gasca, Laurent Lefevre**.** Communicating between the kernel and user-space Linux using Netlink sockets. Software: Practice and Experience, 2010
4. Understanding and programming with Netlink sockets http://people.redhat.com/nhorman/papers/netlink.pdf
5. Patch http://patchwork.ozlabs.org/patch/367086/ Filtering bridge fdb entries in the kernel by Jamal Hadi Salim
6. Linux kernel BPF https://www.kernel.org/doc/Documentation/networking/filter.txt
7. iproute: advanced routing tools for Linux. Web pages at: http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2.
8. Memory mapped IO with netlink: https://www.kernel.org/doc/Documentation/networking/netlink_mmap.txt
9. iproute2 patch, 'link dump filter', Roopa Prabhu http://lists.openwall.net/netdev/2014/07/03/94
10. Cumulus Networks user-space hardware switching daemon: http://cumulusnetworks.com/product/architecture/
11. Open route cache: http://www.e-side.co.jp/okinawaopendays/2014/document/12_Rob-Sherwood.pdf
12. Patch IFLA_EXT_MASK support for a predefined filtering mask info: http://comments.gmane.org/gmane.linux.network/220770

Future Work

Explore other possible ways to improve dump filtering in the kernel. Get time measurements with the BPF approach (which the author was not able to provide in time for this paper).

Author Biography

Roopa Prabhu is a member of technical staff at Cumulus Networks. At Cumulus she works on networking in the Linux kernel and user-space, Network interface management and other system infrastructure areas. Her previous experience includes Linux clusters, ethernet drivers and Linux KVM virtualization platforms. She has an MS in Computer Science from the University of Southern California