Rtnetlink dump filtering in the kernel

Roopa Prabhu

Agenda

- Introduction to kernel rtnetlink dumps
- Applications using rtnetlink dumps
- Scalability problems with rtnetlink dumps
- Better Dump filtering in the kernel

Introduction

- Rtnetlink is a Netlink protocol bus:
  - provides an UAPI to manage Linux kernel networking object database

- Networking subsystems register handlers to manage kernel networking objects (with family and message type)

- Rtnetlink dump handlers:
  - registered with the RTM_GET* message type
  - and invoked when the netlink reqest contains RTM_GET* message with the NLM_F_DUMP flag

Applications: short lived

Mostly poll for kernel database changes:

- Connect to kernel
- Get kernel database dump
- Process messages
- Filter msgs
- Throw away all the data until next poll interval

Applications: short lived example

Look for stale neighbour entries every 30s

```
$ip neigh show | grep 'stale'
```

Applications: Long running apps/daemons

Build userspace kernel object database caches:

- Connect to kernel
- Get kernel database dump
- Listen to kernel netlink notifications to keep the cache current
- App traverses the cached objects to do work

Applications: Long running daemons example

Userspace routing daemons:

- Push routes to kernel
- Build cache of what the kernel has
- React to notifications from the kernel

Current Problems:

- In most cases there is no way to query the kernel via RTnetlink based UAPI on a few attributes

- short lived apps suffer:
  - Its a problem if the neigh database is 16k entries with only a few stale entries

    $ip neigh show | grep 'stale'

example

```
# 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
```

Existing Solutions for efficient dumps:

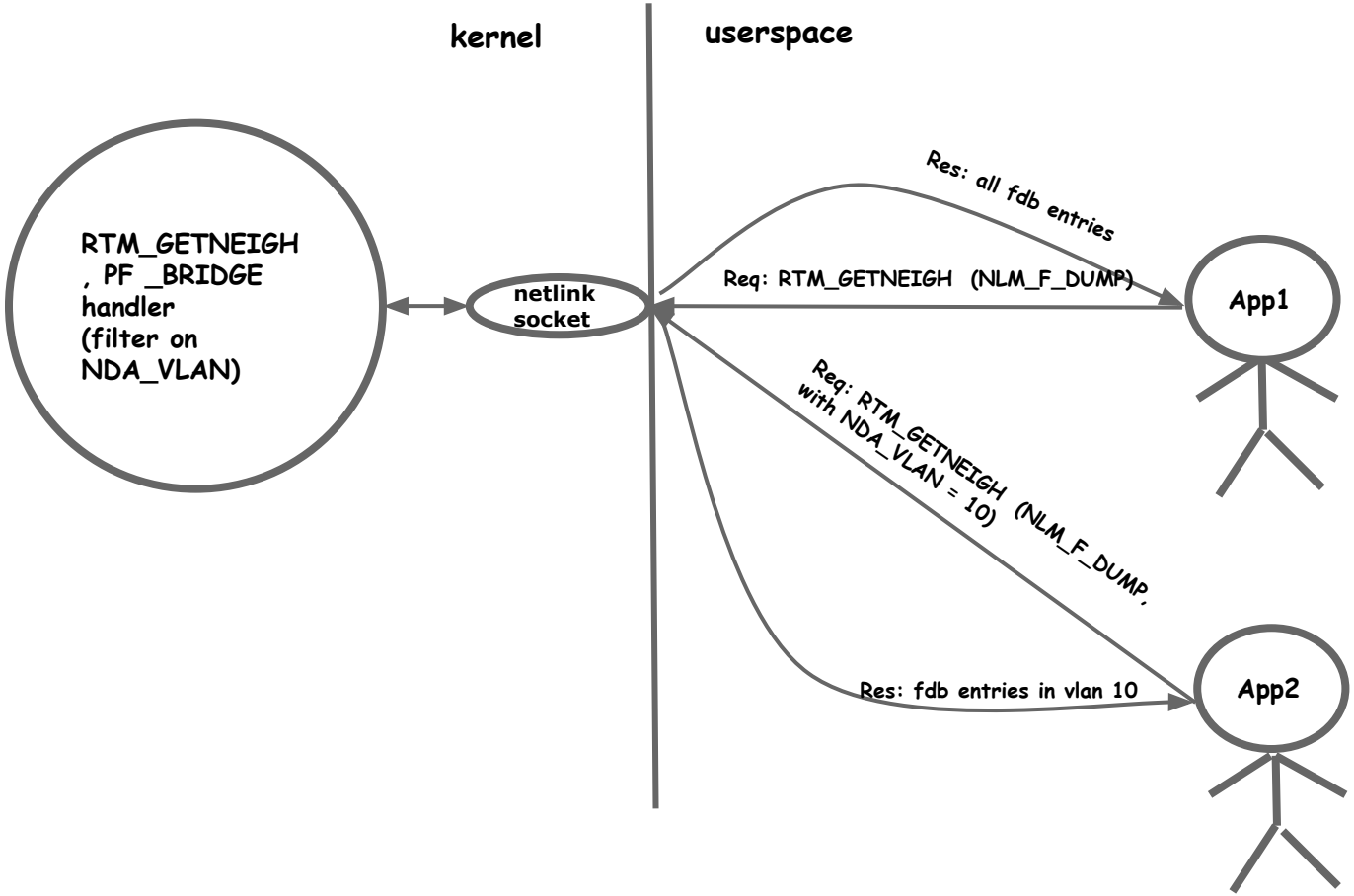
1. BPF socket filters for netlink messages
2. Use netlink mmap to speed up large dumps
3. IFLA_EXT_MASK (u32) netlink attribute which takes a few predefined mask values to filter dumps
4. Filter dump responses with attributes in the dump request messages

This talk is about 4) and in the context of short lived applications

Guidelines for dump request messages:

- RTM_GET* messages with and without NLM_F_DUMP flags must follow the same message format as the RTM_NEW* message

(This is not a new requirement, but is required for consistent dump filtering across subsystems)


kernel
userspace
RTM_GETNEIGH
, PF _BRIDGE
handler
(filter on
NDA_VLAN)
netlink
socket
Res: all fdb entries
Req: RTM_GETNEIGH (NLM_F_DUMP)
App1
Req: RTM_GETNEIGH (NLM_F_DUMP,
with NDA_VLAN = 10)
Res: fdb entries in vlan 10
App2

Next few slides walks through a few such messages

Link dumps: RTM_GETLINK

- 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

example

ip link show type bridge
ip link show group test
ip link show master br0
ip link show link eth1

Fdb dumps: RTM_GETNEIGH

- Filter fdb dumps on any fields in the incoming 'struct ndmsg'
- Bridge and vxlan FDB dumps can be filtered on any of the below fields in 'struct ndmsg':
  - ndm_state – state of the fdb entry (NUD_PERMANENT, NUD_REACHABLE and others)
  - ndm_type - type of entry (static or local)
  - ndm_ifindex – interface the fdb entry points to

Fdb dumps: RTM_GETNEIGH (Contd)

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

example

```
# 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
```

Neigh table dumps: RTM_GETNEIGH

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

example

```
# 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 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

Example

```
# show addresses belonging to an interface
ip addr show dev eth0
```

Numbers: address filtering in kernel with 2000 interfaces

No filtering in kernel: 2000 interfaces with ip addresses (orig)

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

Filtering in kernel: 2000 interfaces with ip addresses

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

Futures

- Post patches
- Explore other ways to filter dumps in the kernel