The case for eliminating inconsistencies between IPv4 and IPv6 kernel User API

Roopa Prabhu

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

Abstract

The Linux kernel provides a rich Netlink based user API (UAPI) to configure, deploy and manage IPv4 and IPv6. However, the UAPI's for IPv4 and IPv6 are not consistent in some cases. Some of these inconsistencies include: IPv6 addresses are removed on link down, but IPv4 addresses stay. The IPv4 multipath route handling API is different from IPv6. Over the years user-space components have worked around these inconsistencies. In this paper we survey the inconsistencies in the kernel UAPI between IPv4 and IPv6 and present the solutions we adopted to work around these inconsistencies in user space. We show how these inconsistencies cascade into multiple components (routing daemons, user-space Netlink caches and hardware offload drivers) in a system. We show that the resulting implementation is complex enough to justify an effort to eliminate these inconsistencies in the future by unifying the IPv4/IPv6 kernel UAPI.

Keywords

Linux, Kernel, Ipv4, Ipv6, iproute2, Rtnetlink, netlink

Introduction

Linux kernel provides Netlink [1],[2],[3] based UAPI to provision and manage IPv4 and IPv6 addresses and routes.

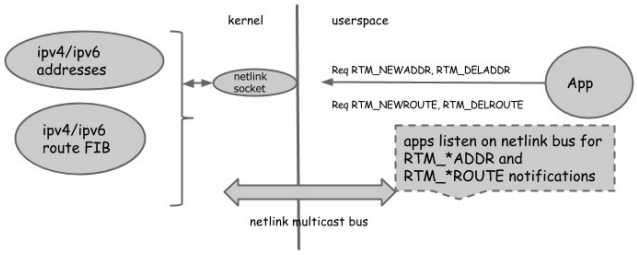

Figure 1. Example application talking to kernel using IPv4 and IPv6 netlink UAPI.

The most common networking applications using the kernel IPv4 and IPv6 Netlink UAPI include routing daemons, network interface managers and hardware offload drivers in users-pace.

- Routing Daemons: A routing daemon like quagga [11] running in user-space uses the netlink kernel UAPI to push IPv4 and IPv6 routes into the kernel FIB.
- Network interface managers: Network interface managers provision network interfaces, addresses and static routes. Network interface managers use the Netlink UAPI to talk to the kernel directly or may use other tools that in-turn use the netlink UAPI
- Network hardware offload drivers in user-space: A hardware offload driver may use the kernel IPv4 and IPv6 UAPI to program kernel routes to hardware. Example [8], [9] These drivers in most cases build Netlink caches in user-space. Hardware offload drivers primarily use these netlink caches to interpret changes in the kernel database via notifications and program the hardware accordingly. They start with a dump from the kernel and rely on netlink notifications to update the cache. Libnl [11] allows building netlink caches in user-space.

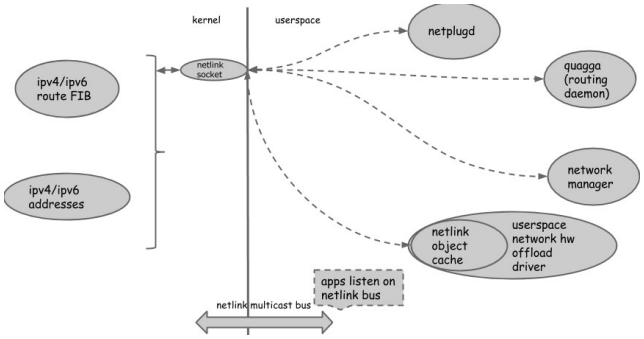

Figure 2. Example applications using netlink api to configure Ipv4/IPv6

In building a network switching platform using Linux as the control plane [8], we have had a chance to see the IPv4/IPv6 U API from the view of all the applications listed above. We have observed inconsistencies in the IPv4 and IPv6 UAPI and also seen that the API inconsistency handling becomes part of more than one component in the system.

The goal of this paper is to provide a survey of the IPv4/IPv6 kernel UAPI inconsistencies, lessons learnt, solutions to handling these in user-space, problems involved and a method to solve this problem in the kernel.

We understand that most of the UAPI inconsistency comes from the fact that the kernel keeps its ABI promise and stands strictly behind supporting all of the legacy UAPI's the user has been exposed to. Hence in most cases there is no fix. Nonetheless, we hope this paper will provide justification for consistent IPv4 and IPv6 kernel UAPI in the future. We hope this paper will also serve as a guide to deploying IPv4 and IPv6 on Linux.

Related Work

There have been attempts [5], [6] to fix some of the inconsistencies. [5] has been accepted recently in the kernel. [6] is pending. Our paper describes these and additional inconsistencies that remain unresolved.

IPv4/IPv6 kernel API inconsistencies

We will discuss inconsistencies in the kernel IPv4/IPv6 UAPI in the following areas:

- Address handling on interface down
- Route delete notifications on interface down
- Multipath route add/del UAPI
- Multipath route netlink notification
- Multipath route replaces
- Multipath route appends
- Handling un-equal cost multipath routes

The rest of the sections in this paper will cover details on the above, discuss solutions in user-space, associated problems and possible in-kernel solution. In each case we show some examples using iproute2 [10].

Address handling on interface down

On interface down, IPv6 addresses are flushed but IPv4 addresses are not. Interface addresses can be configured by user directly using Linux native commands like iproute2 [10], dynamically obtained by a protocol like DHCP [13], configured using network interface managers or routing daemons like quagga[11]. As a result of which in a given system these interface addresses can be owned by multiple components.

```
# interface dummy0 below has an ipv4 address, ipv6 global
# and ipv6 link local address
ip addr show
4:  dummy0:  <BROADCAST,NOARP,UP,LOWER_UP>  mtu
1500 qdisc noqueue state UNKNOWN group default
   link/ether 12:3f:92:73:f7:1f brd ff:ff:ff:ff:ff:ff
   inet 10.0.13.2/24 scope global dummy0
      valid_lft forever preferred_lft forever
   inet6 2001:20:1::2/64 scope global
      valid_lft forever preferred_lft forever
   inet6 fe80::103f:92ff:fe73:f71f/64 scope link
      valid_lft forever preferred_lft forever

# down dummy0
ip link set dev dummy0 down

# ip monitor output displaying address
# delete notification for ipv6 link local and global address
# from the kernel. No deletes were performed for Ipv4
#
ip monitor addr
Deleted 4: dummy0    inet6 2001:20:1::2/64 scope global
      valid_lft forever preferred_lft forever
Deleted 4: dummy0     inet6 fe80::103f:92ff:fe73:f71f/64 scope
link valid_lft forever preferred_lft forever

# bring interface dummy0 up
ip link set dev dummy0 up

# ip monitor output showing ipv6 link local address coming
# back up
ip monitor addr
4: dummy0    inet6 fe80::103f:92ff:fe73:f71f/64 scope link
      valid_lft forever preferred_lft forever

# ipv6 global scope address 2001:20:1::2/64, never came back
# and is lost
ip addr show
4:  dummy0:  <BROADCAST,NOARP,UP,LOWER_UP>  mtu
1500 qdisc noqueue state UNKNOWN group default
   link/ether 12:3f:92:73:f7:1f brd ff:ff:ff:ff:ff:ff
   inet 10.0.13.2/24 scope global dummy0
      valid_lft forever preferred_lft forever
   inet6 fe80::103f:92ff:fe73:f71f/64 scope link
      valid_lft forever preferred_lft forever
```

- Solutions in user-space: All interested daemons can listen to link notifications from the kernel and restore IPv6 addresses on link up (netplugd can be configured to do this. But since netplugd does not own interface address configuration, it will require all interested daemons to hook into netplugd or have netplugd understand address configuration in the daemons).

  Problems: Having to remember and handle this specific behavior in more than one component is error-prone. This

aggravates the problem when you have network namespaces and each namespace must now handle the same behavior (run netplugd instances). Caution must be taken to make sure not more than one daemon/component is trying to fix the problem and step on each other

- Solution in kernel: Kernel IPv6 behavior can be made to be consistent with IPv4. Kernel should not flush IPv6 static addresses on link down (This can be made conditional on a sysctl [12] and this was done recently in [5])

Route delete notifications on interface down

Kernel notifies user-space of IPv6 dead routes on interface down but user-space is not notified of IPv4 dead routes on interface down.

```
# interface dummy0 below has an ipv4 address, ipv6 global
# and ipv6 link local address

ip addr show
4: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu
1500 qdisc noqueue state UNKNOWN group default
   link/ether 12:3f:92:73:f7:1f brd ff:ff:ff:ff:ff:ff
   inet 10.0.13.2/24 scope global dummy0
      valid_lft forever preferred_lft forever
   inet6 2001:20:1::2/64 scope global
      valid_lft forever preferred_lft forever
   inet6 fe80::103f:92ff:fe73:f71f/64 scope link
      valid_lft forever preferred_lft forever

# showing IPv4 connected routes installed by the kernel
# for the IPv4 address
ip -4 route show
10.0.13.0/24 dev dummy0 proto kernel scope link src
10.0.13.2

# showing IPv6 connected routes installed by the kernel
# for the IPv6 address
 ip -6 route show
2001:20:1::/64 dev dummy0 proto kernel metric 256
fe80::/64 dev dummy0 proto kernel metric 256

# As you can see below, only notifications for IPv6 were
# generated by the kernel. There were no notifications for
# IPv4 route delete.
ip monitor route
Deleted 2001:20:1::/64 dev dummy0 proto kernel metric 256
Deleted fe80::/64 dev dummy0 proto kernel metric 256
Deleted ff00::/8 dev dummy0 table local metric 256
Deleted local 2001:20:1::2 dev lo table local proto none
metric 0
Deleted local fe80::103f:92ff:fe73:f71f dev lo table local proto
none metric 0

# Both IPv4 and IPv6 connected routes were deleted by
# the kernel
ip -4 route show
ip -6 route show
```

- Solutions in user-space: An application can listen to link notifications and purge all IPv4 dead routes i.e. purge all routes pointing to down interfaces (routing daemons already do this)

  Problems: Every application managing routes needs to remember this and have efficient purging functions to purge routes on link down. With large routing databases, it also becomes necessary to maintain an efficient data structure linking interfaces to routes they are part of. Mimicking the kernel hashtable of links (fib_info_devhash) and the nexthops it points to becomes necessary.

- Solution in kernel: IPv4 UAPI can be fixed to generate notifications on all dead routes similar to IPv6. Kernel does not generate notifications for dead routes today because user-space can figure this out. Which we believe might be the right thing to do given that this can generate a notification storm on interface down. Nonetheless, this is still an inconsistency in IPv4 and IPv6 UAPI.

Multipath route add/del API

Kernel supports two separate netlink message formats for IPv6 equal cost multipath route add and deletes: each nexthop can be added as a separate route with the same prefix or all nexthops can be grouped into a single message. The latter makes the IPv6 multipath route API consistent with IPv4. The kernel supports the old API to not break existing users and hence there is no solution to this inconsistency. Nonetheless, there are two versions of the API and we see a mix of these two API's in use today.

```
# Add a IPv4 multipath route
ip route add 10.0.12.2 nexthop via 10.0.13.2 dev dummy0
nexthop via 10.0.14.2 dev dummy1

# dump of kernel IPv4 routes showing the multipath route
ip -4 route show
10.0.12.2
   nexthop via 10.0.13.2 dev dummy0 weight 1
   nexthop via 10.0.14.2 dev dummy1 weight 1

# Add a IPv6 multipath route
# First approach: Add individual nexthops separately
ip -6 route add 3ffe:304:124:2306::/64 nexthop via
fe80::b077:f0ff:fe23:5cc7 dev dummy0
ip -6 route add 3ffe:304:124:2306::/64 nexthop via
```

```
fe80::d850:e7ff:fe87:cf6a dev dummy1

# dump of kernel IPv6 routes showing the multipath route
ip -6 route show
3ffe:304:124:2306::/64 via fe80::b077:f0ff:fe23:5cc7 dev dummy0 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024

# The ipv6 route added above is removed to show the
# second approach below

# Second approach: All nexthops can be added together,
# same as ipv4
ip -6 route add 3ffe:304:124:2306::/64 nexthop via fe80::b077:f0ff:fe23:5cc7 dev dummy0 nexthop via fe80::d850:e7ff:fe87:cf6a dev dummy1

# dump of kernel IPv6 routes showing the multipath route
ip -6 route show
3ffe:304:124:2306::/64 via fe80::b077:f0ff:fe23:5cc7 dev dummy0 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024
```

Multipath route netlink notification

IPv6 multipath netlink route notification format is different from IPv4. Kernel generates a separate netlink route notification for each nexthop in an IPv6 multipath route. Kernel generates a single notification with all nexthops in the same notification message for IPv4 multipath route. This inconsistency exists because of the way IPv6 multipath routes are stored in the kernel route database. They are stored as separate routes through each nexthop and linked as siblings if they have the same destination and metric.

```
# Add IPv4 multipath route
$ip route add 10.0.12.2 nexthop via 10.0.13.2 dev dummy0 nexthop via 10.0.14.2 dev dummy1

# below ip monitor output shows notification for multipath route
# add from kernel
$ip monitor route
10.0.12.2
   nexthop via 10.0.13.2  dev dummy0 weight 1
   nexthop via 10.0.14.2  dev dummy1 weight 1

# Add a IPv6 multipath route
$ip -6 route add 3ffe:304:124:2306::/64 nexthop via fe80::b077:f0ff:fe23:5cc7 dev dummy0 nexthop via fe80::d850:e7ff:fe87:cf6a dev dummy1
```

```
# Notifications for the IPv6 multipath route add from kernel
ip monitor route
3ffe:304:124:2306::/64 via fe80::b077:f0ff:fe23:5cc7 dev dummy0 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024
```

- Solution in user-space: An application in user-space can maintain a cache of IPv6 routes and use nexthop netlink notifications from the kernel to accrue IPv6 nexthops into a single multipath route for the same prefix. i.e. follow the kernel approach of linking IPv6 multipath routes into siblings if they have the same destination. One such patch was posted to libnl [7]. As we will see in the later sections, this approach will also need to handle route multipath appends and replaces.

  Problems: User-space rebuilding the multipath route from notifications can be error prone in the below cases: During large route dumps with iproute adds and deletes in progress in parallel, there is a potential of duplicate msgs and duplicate next-hops. As will be described later in the paper, this will also need to be ordered correctly for route replaces (NLM_F_REPLACE) and route appends (NLM_F_APPEND)

- Solution in kernel: Kernel can fix IPv6 multipath route notifications to be in the same format as IPv4. i.e. All nexthops in a multipath route belong to the same msg. This change in behavior can be wrapped in a sysctl [12] to not break existing users.

Multipath route replaces

IPv6 multipath route replace handling and notification is different from IPv4. As described in the last section, this is because of the way IPv6 multipath routes are stored in the kernel route database. Route replace is an efficient way to request route updates in the kernel. It replaces two messages (add + del) with a single message and this can speed up performance when syncing large routing tables to the kernel from routing daemons. Replaces are indicated to the kernel using the NLM_F_REPLACE flag in the netlink message flags [4]. Interpreting kernel route replace notifications in user-space has always been tricky. This is because replace notifications do not contain enough information of the route being replaced in the kernel. It only contains information about the new route that got inserted in the routing database.

Unlike IPv4, IPv6 allows replacing a single nexthop in a multipath route and the kernel generates replace notification for the replaced nexthop. IPv4 does not allow replacing a single nexthop. However, it does allow replacing a multipath route with a new one. This also results in the multipath route replace notification containing the full new multipath route making it easier in user-space to handle route replaces.

```
# ip route show
10.0.12.2
   nexthop via 10.0.13.2  dev dummy0 weight 1
   nexthop via 10.0.14.2  dev dummy1 weight 1

#ip route replace 10.0.12.2 nexthop via 10.0.15.2 dev dummy2

#ip monitor route
10.0.12.2 via 10.0.15.2 dev dummy2

# ipv4 route was replaced
ip route show
10.0.12.2 via 10.0.15.2 dev dummy2

#ipv6
ip -6 route show
3ffe:304:124:2306::/64 via fe80::b077:f0ff:fe23:5cc7 dev dummy0 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024

ip -6 route replace 3ffe:304:124:2306::/64 nexthop via fe80::c26:cdff:feca:18f2 dev dummy2

ip monitor route
3ffe:304:124:2306::/64 via fe80::c26:cdff:feca:18f2 dev dummy2 metric 1024

# ipv6 nexthop was replaced
ip -6 route show
3ffe:304:124:2306::/64 via fe80::c26:cdff:feca:18f2 dev dummy2 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024
```

- Solutions in user-space: As described in previous section, IPv6 next-hop notifications can be accrued in user-space to build an IPv6 multipath that is similar to IPv4. Replace notifications include the NLM_F_REPLACE flag which can be used to always replace the first nexthop.

  Problems: Always replacing the first nexthop might not be the right thing to do and could be error prone.

- Solution in kernel: As discussed previously, making kernel IPv6 multipath notification format to be same as IPv4 makes this problem easier to handle in user-space. Additionally, kernel replace notifications could contain a new nested attribute with hints to the user about the route or nexthop being replaced in the kernel

Multipath route append

Unlike IPv4, IPv6 allows appending a nexthop to an existing multipath route. User-space IPv6 netlink route handling functions must also take into account that IPv6 nexthop notifications can be due to an append and handle nexthop appends to an existing route accordingly. Appends are indicated to the kernel using the NLM_F_APPEND flag in the netlink message flags [3]

```
#ipv4

ip route show
10.0.12.2
   nexthop via 10.0.13.2  dev dummy0 weight 1
   nexthop via 10.0.14.2  dev dummy1 weight 1

ip route append 10.0.12.2 nexthop via 10.0.15.2 dev dummy2

ip monitor route
10.0.12.2 via 10.0.15.2 dev dummy2

# A new route was appended
ip route show
10.0.12.2
   nexthop via 10.0.13.2  dev dummy0 weight 1
   nexthop via 10.0.14.2  dev dummy1 weight 1
10.0.12.2 via 10.0.15.2 dev dummy2

#ipv6

ip -6 route show
3ffe:304:124:2306::/64 via fe80::b077:f0ff:fe23:5cc7 dev dummy0 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024

ip monitor route
3ffe:304:124:2306::/64 via fe80::c26:cdff:feca:18f2 dev dummy2 metric 1024

ip -6 route append 3ffe:304:124:2306::/64 nexthop via fe80::c26:cdff:feca:18f2 dev dummy2

# A new nexthop was appended to the existing multipath route
ip -6 route show
3ffe:304:124:2306::/64 via fe80::b077:f0ff:fe23:5cc7 dev dummy0 metric 1024
3ffe:304:124:2306::/64 via fe80::d850:e7ff:fe87:cf6a dev dummy1 metric 1024
3ffe:304:124:2306::/64 nexthop via fe80::c26:cdff:feca:18f2 dev dummy2
```

- Solution in user-space: As described in the previous sections, IPv6 next-hop notifications can be accrued in

user-space to build an IPv6 multipath route that is similar to IPv4. User-space can then use the NLM_F_APPEND flag in the route nexthop notification to append the nexthop to the tail of nexthops.

Problems: guessing appends in user-space can be error prone

- Solution in kernel: As discussed previously, making kernel IPv6 multipath notification format to be same as IPv4 makes this problem easier to handle in user-space This can be wrapped in a sysctl [12] to not break existing users.

Un-equal cost multipath routes

In an un-equal cost multipath route there are two ways to assign weights to nexthops:

1. Repeat the nexthop times equal to the weight of the nexthop (duplicate nexthops). Most hardware routing ASICS program multipath routes in this format.

2. Use 'weight' attribute to assign weights to nexthops

IPv4 supports both ways 1) and 2) above to handle un-equal cost multipath routes. IPv6 supports only 2).

Since most hardware routing ASICS do accept format 1), some IPv4 deployments use approach 1) today. And, when these deployments have to move to IPv6, they need to now tweak their handling of non-equal cost multipath routes for IPv6 because of this inconsistency.

```
ip route add 10.0.12.2 nexthop via 10.0.13.2 dev dummy0
nexthop via 10.0.14.2 dev dummy1 nexthop via 10.0.14.2 dev
dummy1

ip route show
10.0.12.2
   nexthop via 10.0.13.2  dev dummy0 weight 1
   nexthop via 10.0.14.2  dev dummy1 weight 1
   nexthop via 10.0.14.2  dev dummy1 weight 1

ip route add 3ffe:304:124:2306::/64 nexthop via
fe80::b077:f0ff:fe23:5cc7 dev dummy0 nexthop via
fe80::d850:e7ff:fe87:cf6a dev dummy1 exthop via
fe80::d850:e7ff:fe87:cf6a dev dummy2
/* error */
```

- Solutions in user-space: Weights can always be used to indicate un-equal cost multipath routes avoiding the need to use duplicate nexthops. This inconsistency can also be made hidden in a netlink library API which can take duplicate nexthops but convert it into weights before pushing it to the kernel.

- Solutions in kernel: For consistency, the kernel should allow duplicate nexthops in a IPv6 multipath route.

Conclusions

There are many solutions possible in ironing out the kernel inconsistencies in IPv4 and IPv6 UAPI. An ideal solution would not require these inconsistencies to be handled in network applications. User-space netlink libraries can be used to abstract out the inconsistencies. An in-kernel solution which can unify the kernel IPv4 and IPv6 UAPI would simplify development of network applications.

Acknowledgements

The author thanks Shrijeet Mukherjee and others at Cumulus networks for insights into IPv4/IPv6 handling in hardware accelerated switches.

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 by David Ahern 'net: ipv6: Make address flushing on ifdown optional' http://permalink.gmane.org/gmane.linux.network/346229
6. patch by Nicolas Dichtel to fix route delete notifications on linkdown for iPv4: http://patchwork.ozlabs.org/patch/195516/
7. patch by shrijeet Mukerjee and Roopa Prabhu to accrue ipv6 nexthops in libnl http://lists.infradead.org/pipermail/libnl/2012-December/000836.html
8. Cumulus Networks user-space hardware switching daemon: http://cumulusnetworks.com/product/architecture/
9. Open route cache: http://www.e-side.co.jp/okinawaopendays/2014/document/12_Rob-Sherwood.pdf
10. iproute: advanced routing tools for Linux. Web pages at: http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2
11. Quagga team. Quagga Routing Software Suite. Web pages at: http://www.quagga.net
12. Linux kernel sysctl documentation https://www.kernel.org/doc/Documentation/sysctl/
13. DHCP https://www.isc.org/downloads/dhcp/

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