firewalld ↔ iptables (continued)

Or, better said as,
Understanding Linux Firewall Changes and Tools
A firewall evolution and system management process

Presented to SLUUG
By David Forrest
August 9, 2017

Bio

I am David Forrest, a businessman in the housing and construction materials industry. Always keen to use the open and supportable solution even if it means getting my hands dirty.

I was there, I did that, I have the t-shirt. And, I'm retired so now I can work on the “bleeding edge” - so on to the testing kernel!

Why tonight?

Why should we switch to firewalld?

I felt a continuation was in order to address the problems that are caused by the virtual world and the interaction of processes within today's machines.

Our various distributions seem to be jumping to the systemd init setup as it appears to offer better administration control to Linux Kernel machines.

Firewalld just one of many efforts to see the future.

In recent years, operating system virtualization has taken the industry by storm.

But I'm still on CentOS7 and it uses firewalld as its default firewall along with systemd

https://wiki.debian.org/Debate/initsystem/systemd

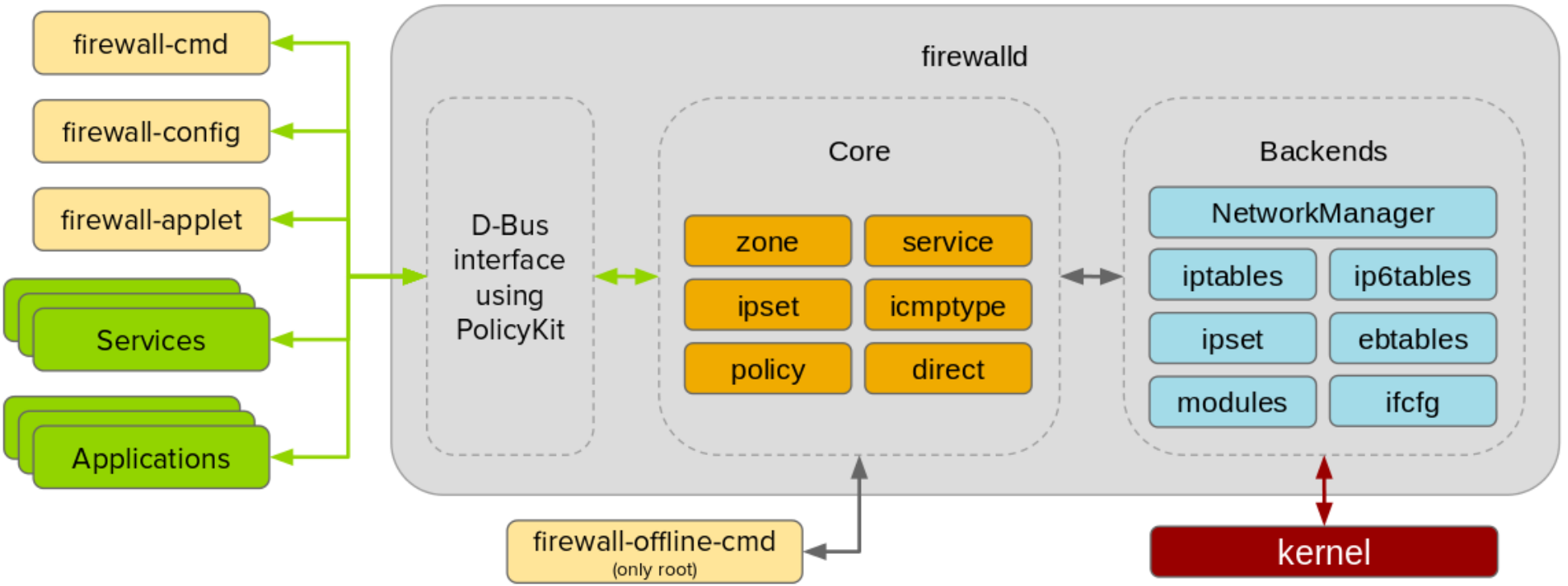
firewalld

It's a daemon and a command line interface to all the backends!

One can start it as a service with a default setup and change it dynamically with a command line or with the daemon using D-Bus or NetworkManager.

And with the new nftables release, we'll be able to combine several rules in one rich rule.

The firewalld Architecture



Firewalld and nft

Systems have also moved toward Software Defined Networking (SDN) and system density has increased. Work has been done to manage and isolate networking within a physical host bringing the full Open System Interconnection (OSI) inside the operating system. …

With all the new tools and technology in play management is extremely important. We will look at how firewalld is used to administrate the firewall. The process of managing and maintaining a firewall has come a long way with big changes in the works.

Be prepared to toss your old confusing scripts and enjoy firewalld. And let's just take a few baby steps.

Baby Steps

Some distros including CentOS7 have enabled firewalld and disabled iptables by default as firewalld does provide all the same capabilities as {ip|ip6|eb}tables have. But, as iptables rules are not dynamic, one has to preselect just one of the “zones” for a side-to-side comparison. There are several defined zones provided by firewalld.

Zones are an implementation of the “flow” of the rules.

A special “direct” zone is active for processing before the selected zone for common things like pass-through, lock-downs, and whitelists.

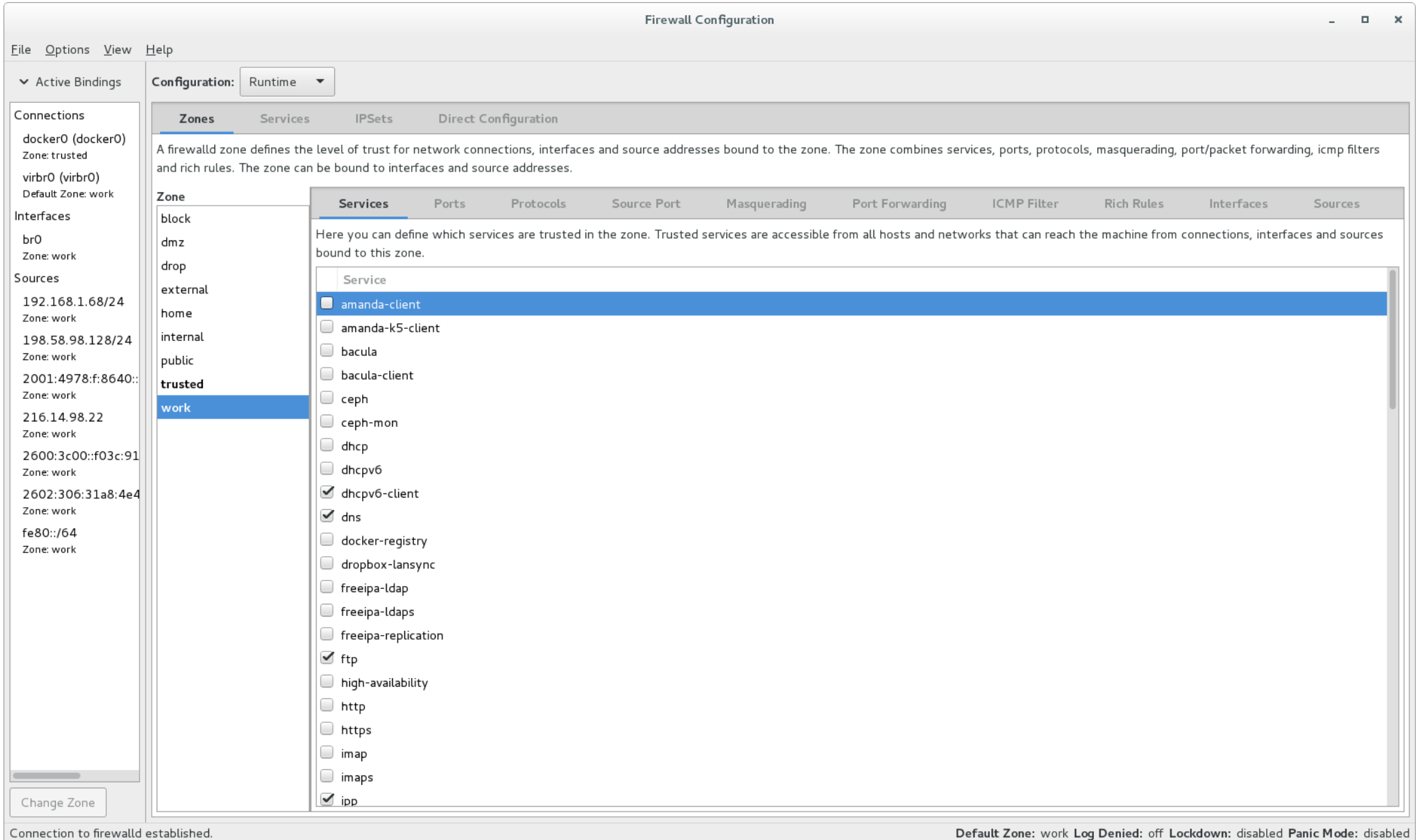
Select a zone

I have selected the “work” zone as I do, “mostly trust the other computers on networks to not harm my computer”.

Only selected incoming connections are accepted. I used the firewalld gui to set it up.

Firewalld does also have a cli, firewall-cmd for scripting.

Firewalld gui (firewall-applet)



My “work” zone

~]$ firewall-cmd --info-zone=work

- work (active)
- target: default
- icmp-block-inversion: no
- interfaces: br0
- sources: 192.168.1.68/24 198.58.98.128/24 2001:4978:f:8640::/64 216.14.98.22 2600:3c00::f03c:91ff:fe56:7e17/64 2602:306:31a8:4e40::/64 fe80::/64
- services: dhcpv6-client dns ftp ipp ipp-client mdns ntp openvpn ssh
- ports: 1941/tcp
- protocols: 41
- masquerade: no
- forward-ports:
- sourceports:
- icmp-blocks:
- rich rules:

Service file

[Unit]

- Description=firewalld - dynamic firewall daemon
- Before=network.target
- Before=libvirtd.service
- Before=NetworkManager.service
- After=dbus.service
- After=polkit.service
- Conflicts=iptables.service ip6tables.service ebtables.service ipset.service
- Documentation=man:firewalld(1)
-

Service file (cont.)

- [Service]
- EnvironmentFile=-/etc/sysconfig/firewalld
- ExecStart=/usr/sbin/firewalld --nofork --nopid $FIREWALLD_ARGS
- ExecReload=/bin/kill -HUP $MAINPID
- # supress to log debug and error output also to /var/log/messages
- StandardOutput=null
- StandardError=null
- Type=dbus
- BusName=org.fedoraproject.FirewallD1

Service file (cont)

- [Install]
- WantedBy=basic.target
- Alias=dbus-org.fedoraproject.FirewallD1.service
- (END)
-

Xml service file

- <short>Work</short>
- <description>For use in work areas. You mostly trust the other computers on networks to not harm your computer. Only selected incoming connections are accepted.</description>
- <interface name="br0"/>
- <source address="2600:3c00::f03c:91ff:fe56:7e17/64"/>
- <source address="192.168.1.68/24"/>
- <source address="fe80::/64"/>
- <source address="216.14.98.22"/>
- <source address="2001:4978:f:8640::/64"/>
- <source address="2602:306:31a8:4e40::/64"/>
- <source address="198.58.98.128/24"/>
- <service name="ftp"/>
- <service name="mdns"/>
- <service name="ipp-client"/>
- <service name="dhcpv6-client"/>
- <service name="ipp"/>
- <service name="ssh"/>
- <service name="dns"/>
- <service name="openvpn"/>
- <service name="ntp"/>
- <port protocol="tcp" port="1941"/>
- <protocol value="41"/>
- </zone>

  <?xml version="1.0" encoding="utf-8"?>
- <zone

Additional Capabilities

Firewalld is a gui and includes a cli, firewall-cmd, letting it be fully scripted much like iptables with some additional features such as:

Ipset – allows sets of ipset. Can be used to set up, maintain and inspect so called IP sets in the Linux kernel. Depending on the type of the set, an IP set may store IP(v4/v6) addresses, (TCP/UDP) port numbers, IP and MAC address pairs, IP address and port number pairs, etc. Iptables matches and targets referring to sets create references, which protect the given sets in the kernel. A set cannot be destroyed while there is a single reference pointing to it.

D-Bus Capable

Allows interface changes to be noticed by the kernel and configured into firewall results thereby noticing virtual machines creation or deletion and effect actions.

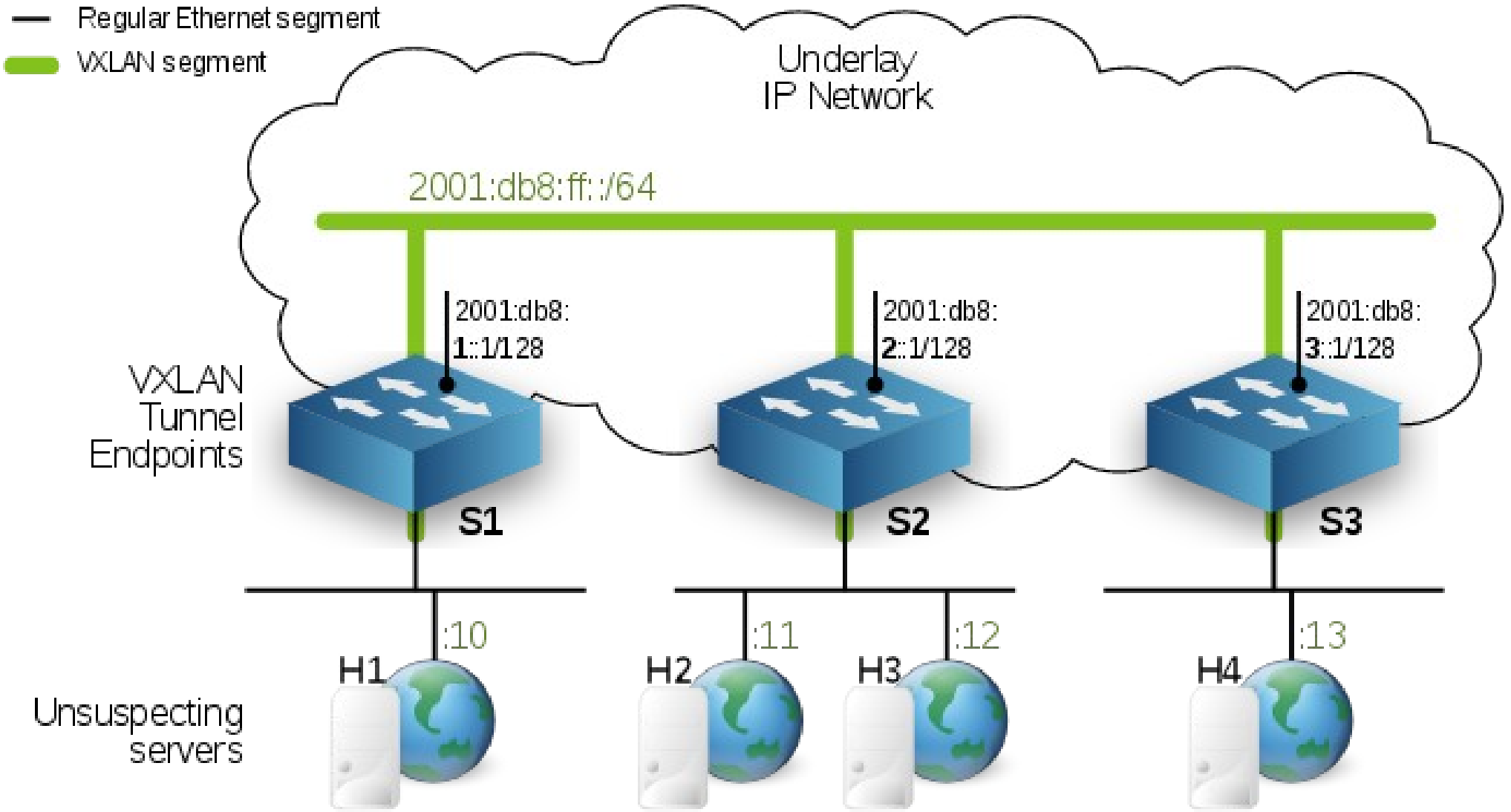
Getting a good handle on the format change is enough for me at this time as CentOS7 is still on the 3.10 Linux kernel. But 4.12.0 is now the latest stable released kernel and 4.12 brings nftables.

I am holding my breath!!!

So where do we go from here?

VXLAN is an overlay network to carry Ethernet track over an existing (highly available and scalable) IP network while accommodating a very large number of tenants. It is defined in RFC7348. Starting from Linux 3.12, the VXLAN implementation is quite complete as both multicast and unicast are supported as well as IPv6 and IPv4. Let’s explore the various methods to configure it.

VXLAN deployment example


Regular Ethernet segment
VXLAN segment
Underlay
IP Network
2001:db8:ff::/64
2001:db8:
1::1/128
2001:db8:
2::1/128
2001:db8:
3::1/128
VXLAN
Tunnel
Endpoints
S1
S2
S3
:10
:11
:12
:13
H1
H2
H3
H4
Unsuspecting
servers

Those of you who saw my Multicast IPv6 presentation will (or should) see this as a simple multicast (*,G) tree but I then showed it using IPv4. The link below calls this a “Basic” and also details a much more complicated IPv4 description using broadcast ARP connections in the underlay network.

One complication is using private IPv4 on different subnets for the “unsuspecting” servers.

From: https://vincent.bernat.im/en/blog/2017-vxlan-linux

Basic setup

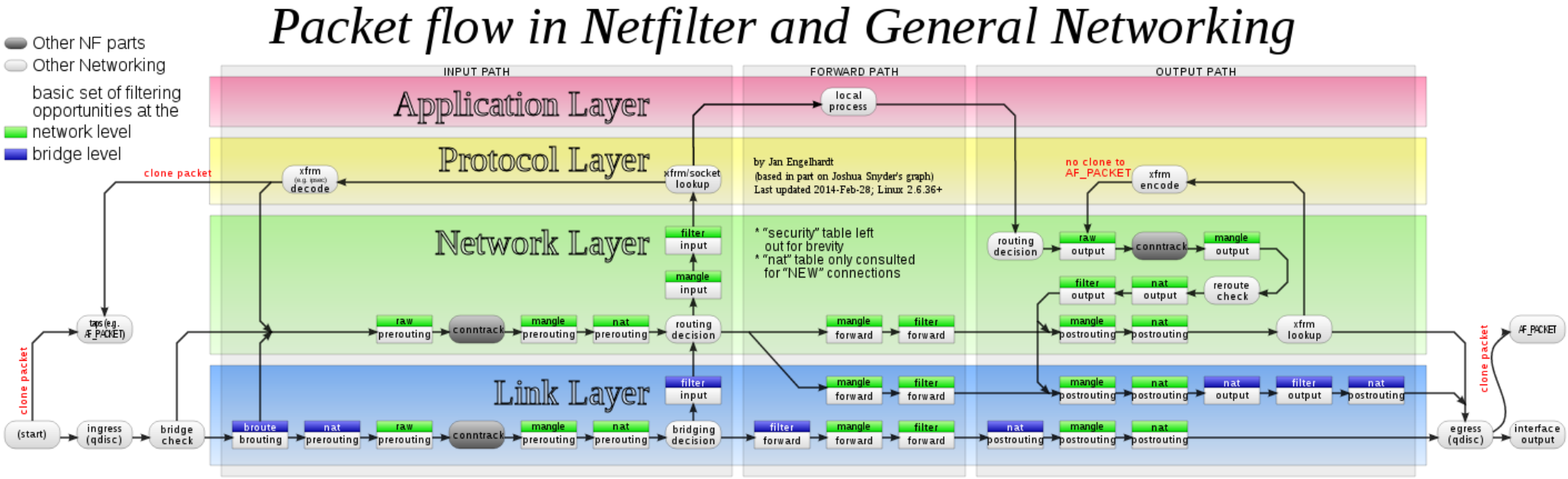
A VXLAN tunnel extends the individual Ethernet segments across the three bridges, providing a unique (virtual) Ethernet segment. From one host (e.g. H1), we can reach directly all the other hosts in the virtual segment:

$ ping -c10 -w1 -t1 ff02::1%eth0

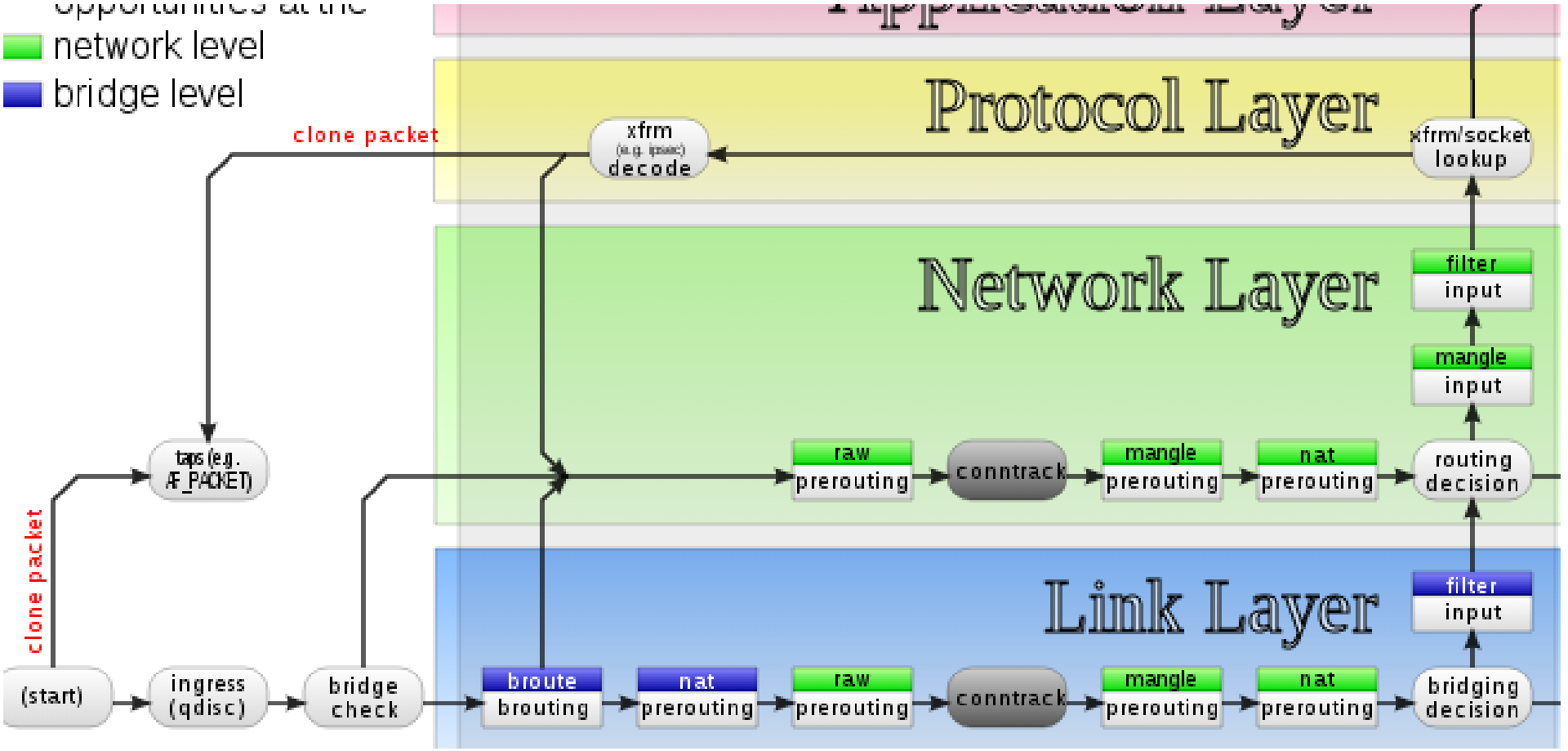
- PING ff02::1%eth0(ff02::1%eth0) 56 data bytes
- 64 bytes from fe80::5254:33ff:fe00:8%eth0: icmp_seq=1 ttl=64 time=0.016 ms
- 64 bytes from fe80::5254:33ff:fe00:b%eth0: icmp_seq=1 ttl=64 time=4.98 ms (DUP!)
- 64 bytes from fe80::5254:33ff:fe00:9%eth0: icmp_seq=1 ttl=64 time=4.99 ms (DUP!)
- 64 bytes from fe80::5254:33ff:fe00:a%eth0: icmp_seq=1 ttl=64 time=4.99 ms (DUP!)
- --- ff02::1%eth0 ping statistics ---
- 1 packets transmitted, 1 received, +3 duplicates, 0% packet loss, time 0ms
- rtt min/avg/max/mdev = 0.016/3.745/4.991/2.152 ms

This solution uses IPv6 local node addressing, FE80::/28, which can be tied up in IPv4 also by referring to the MAC addresses using an arptable setup although it is a little more complicated. The easy way is ebtables, virtual Ethernet Bridge interfaces. It's just an Ethernet bridge in the virtual world.

What happened?



A little bigger?



Things get more connected

Iptables (we all know it), ip6tables (some know it), ebtables (some network people know it), ARP & arptables (few get that deep) all have been connected in a new suite called nftables. And firewalld was developed to make the maintenance MUCH easier as it uses another factor, zones, to handle much of flowcharting logic (but still backwards compatible).

Nftables (nft)

nftables replaces the popular {ip,ip6,arp,eb}tables. This software provides a new in-kernel packet classification framework that is based on a network-specific Virtual Machine (VM) and a new nft userspace command line tool. Nftables reuses the existing Netfilter subsystems such as the existing hook infrastructure, the connection tracking system, NAT, userspace queueing and logging subsystem.

https://netfilter.org/projects/nftables/

https://linoxide.com/firewall/configure-nftables-serve-internet/

Nftables features

- Network-specific VM: the nft command line tool compiles the ruleset into the VM bytecode in netlink format, then it pushes this into the kernel via the nftables Netlink API. When retrieving the ruleset, the VM bytecode in netlink format is decompiled back to its original ruleset representation. So nft behaves both as compiler and decompiler.
- High performance through maps and concatenations: Linear ruleset inspection doesn't scale up. Using maps and concatenations, you can structure your ruleset to reduce the number of rule inspections to find the final action on the packet to the bare minimum.

Nftables features

- Smaller kernel codebase. The intelligence is placed in userspace nft command line tool, which is considerably more complex than iptables in terms of codebase, however, in the midrun, this will potentially allow us to deliver new features by upgrading the userspace command line tool, with no need of kernel upgrades.

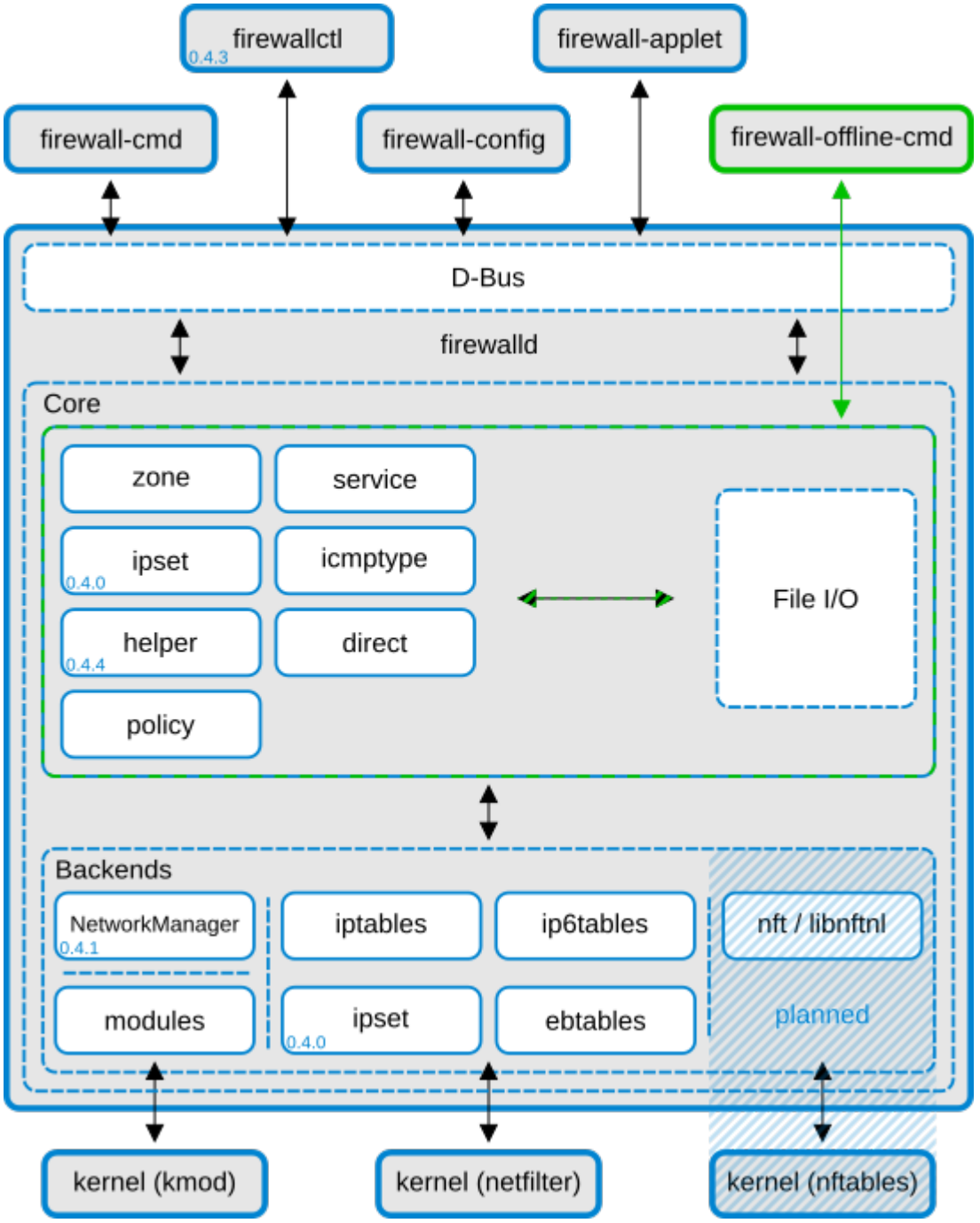
- Unified and consistent syntax for every support protocol family, contrary to xtables utilities, that are well-known to be full of inconsistencies.

Many have used iptables and some have knowledge of ebtables, ip6tables, arptables and the netfilter project at large. These userspace programs connect to the API for netfilter in the Linux kernel and setup rules and hooks for network packets that pass though the network stack.

These administration tools have always had a hammer approach of do one thing with no persistence.

The various tools over the years have created duplicate code in the kernel. The netfilter team designed nftables as a replacement in the same way as ipfwadm was replaced by ipchains, an evolution.

Nftables (nft) is a code refactor, code de-duplication and clean up effort.


firewallctl
0.4.3
firewall-applet
firewall-cmd
firewall-config
firewall-offline-cmd
D-Bus
firewalld
Core
zone
service
ipset
0.4.0
icmptype
helper
0.4.4
direct
policy
File I/O
Backends
NetworkManager
0.4.1
iptables
ip6tables
nft / libnftnl
modules
ipset
0.4.0
ebtables
planned
kernel (kmod)
kernel (netfilter)
kernel (nftables)

That's all for now

- Questions?

- This presentation can be downloaded in a pdf:

  http://gobble.de.gook.maplepark.com/20170809_More_Firewalld_Iptables_2.pdf

  (or as an .odp or .pptx by using the desired extension)
- Thanks for your attention!
- (to sid!) https://www.digitalocean.com/community/tutorials/upgrading-debian-to-unstable