eBPF use in Android Networking

(and Android Networking vs Linux Kernel in general)

September 10, 2019

Who am I?

Maciej Żenczykowski (maze@google.com)

At Google since 2006, initially Crawl SRE.

2009-2018 Linux Kernel Networking on servers

(performance, tuning, configuration, frontend serving and load balancing, AnyIp, IPv6, 'glue')

Since mid-2018 (**Q+**): **Android Core Networking**

With a focus on:
- *Kernel (upstreaming)*
- *eBPF & performance*
- *low level C stuff:*
  libs, core utils:
  iproute2, iptables,
  ethtool

android

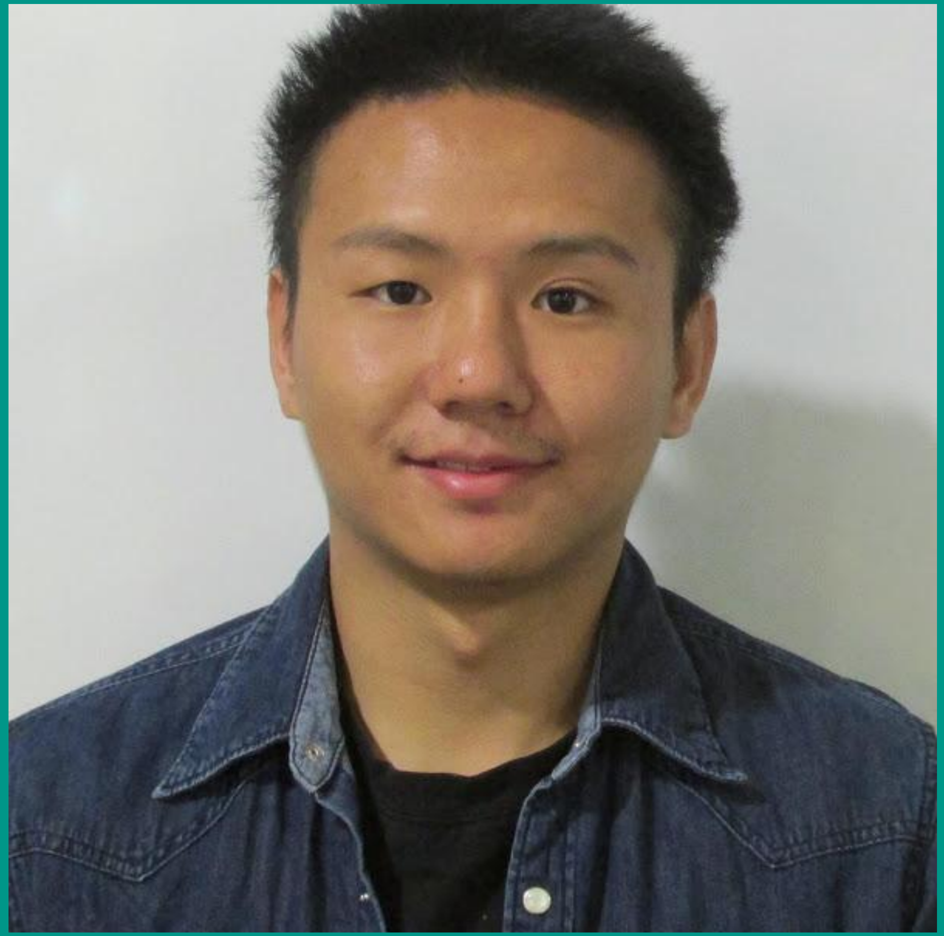
My Team

Lorenzo Colitti (lorenzo@)

Manager of Android Core Networking (5.5 years)

Root cause of this work, code/design reviews, etc.

& others from Android Core Networking and Kernel Teams



Chenbo Feng (fengc@)
ex-Android Kernel Team

Did most of the **P** and some of the **Q** work.

Now working on other non-Kernel Networking things.

Why eBPF?

Because:

- **Get rid of code divergence vs upstream Linux**
  - moving kernel hacks into eBPF thus eliminating them (xt_qtaguid, Paranoid Android)
- **Performance improvements *(and simplicity)***
  - Clatd xlat464 user space daemon -> eBPF offload *(& it's actually easier!!!)*
  - Due to multi-network architecture Android has a really complex firewall + routing setup - this (and/or rndis performance) requires tethering to use HW offload, we hope to offload this to eBPF SW bypass and simplify things while improving performance.
- **More nuanced & more dynamic** (then capabilities like CAP_NET_ADMIN)
- Real hard to upgrade kernel on shipped devices, even new devices run old kernels...
- **It's cool** *(personal bias? I like low level stuff...)*

To quote another talk, cause: '***eBPF is eating the world***'

What we did in Android P

First use of eBPF in Android - eBPF compilation (tool chain) support, bpfloader, etc.

Requires:

- Linux 4.9+ (for Kernel eBPF support)
- Devices launching with Android P
  - Treble means we can't require kernel changes on device upgrade,
    even if it's just enabling kernel config options or backporting fixes (policy may change).
  - Trouble figuring out if a 4.9+ device which launched with Android N/O actually supports eBPF or not - have to assume it does not.
- *for Google devices this means: Pixel 3+*

Replaced use of ancient 'xt_qtaguid' custom Android Linux kernel netfilter patch with eBPF bandwidth statistics collection. (reverted finally in Q kernels)

Mostly done by fengc@ and predates me.

What we did in Android P (continued)

See 2017 LPC talks on this topic.

Replacing xt_qtaguid with an upstream eBPF implementation

https://blog.linuxplumbersconf.org/2017/ocw/sessions/4786.html

https://blog.linuxplumbersconf.org/2017/ocw//system/presentations/4786/original/Replacing%20xt_qtaguid%20with%20an%20upstream%20eBPF%20implementation.pdf

eBPF cgroup filters for data usage accounting on Android

https://blog.linuxplumbersconf.org/2017/ocw/sessions/4791.html

http://www.linuxplumbersconf.net/2017/ocw//system/presentations/4791/original/eBPF%20cgroup%20filters%20for%20data%20usage%20accounting%20on%20Android.pdf

What we did in Android 10 (Q)

‘Paranoid Android’ kernel patch: unpriv apps cannot create AF_INET/AF_INET6 sockets

- this is the app ‘internet’ permission
- now implemented in eBPF (fengc@), requires 4.14+ *(and thus newer then Pixel 3)*
- kernel patch not ported to Android common 4.19-Q and reverted in 4.14-R
- *patch also had a feature auto-granting CAP_NET_ADMIN / CAP_NET_RAW based on process supplementary group membership, which couldn’t be replicated...*

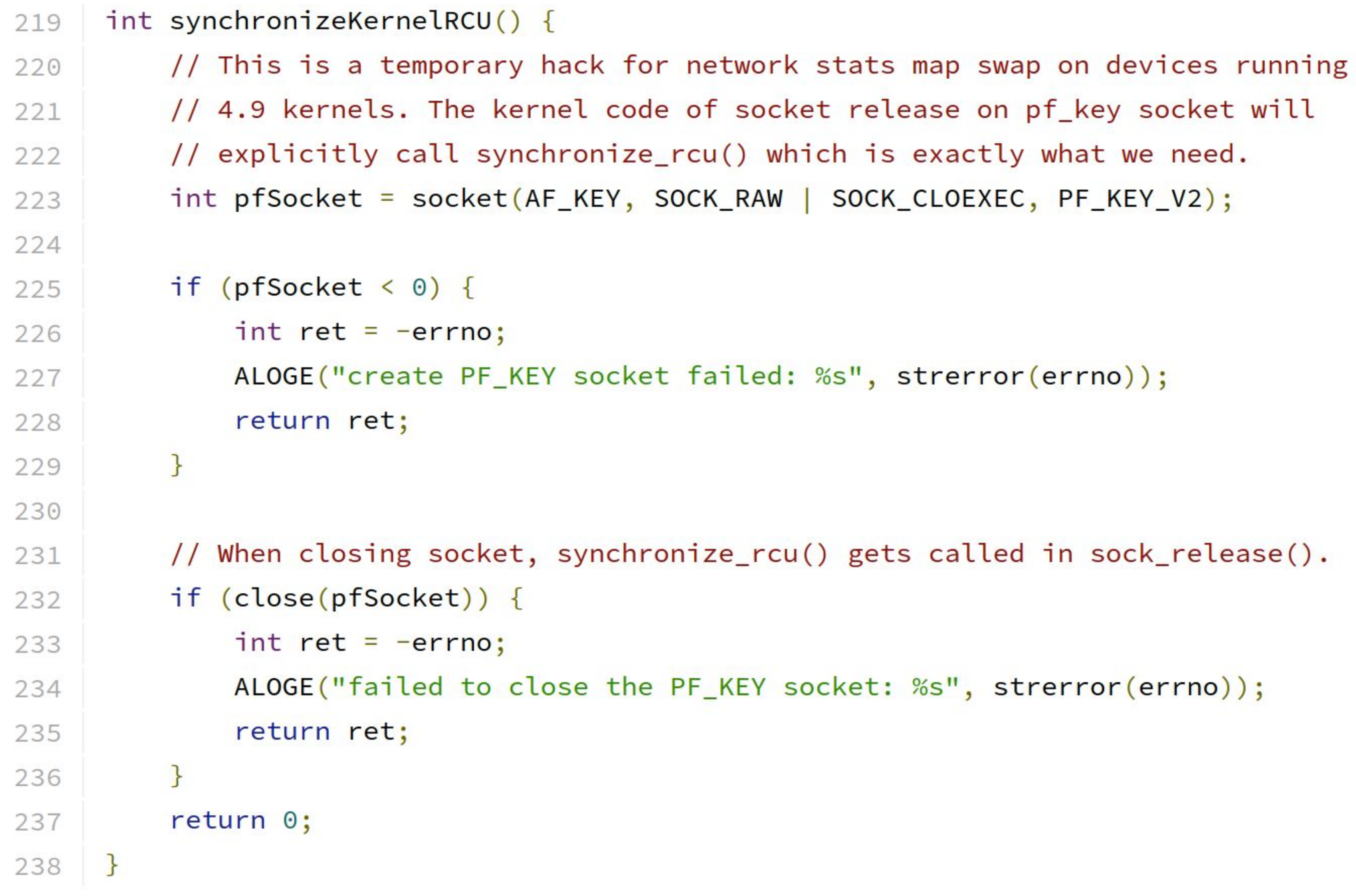
Android has long supported **IPv6-only** networks (many cell networks are such for simplicity).
The ‘**Clatd**’ userspace daemon doing **XLAT464 packet translation** was a **performance problem**.

- added RX-only TCP & UDP non-fragmented packet eBPF offload (maze@)
- requires 4.9+ with LTS fixes and config changes
- all Android Q devices launched on 4.9+ (and some launched on P)
- *ie. Pixel 3+ (launched on 4.9-P, kernel upgraded, now on 4.9-Q)*
- **tcpdump visibility hack**: tc ingress + bpf_redirect(same_ifindex, BPF_F_INGRESS)

See also Android Bootcamp 2019 slides (at end of slide deck)

SynchronizeKernelRCU

or how to synchronize map flips on a multithreaded system…
https://android.googlesource.com/platform/system/bpf/+/refs/heads/master/libbpf_android/BpfUtils.cpp#219



```cpp
int synchronizeKernelRCU() {
    // This is a temporary hack for network stats map swap on devices running
    // 4.9 kernels. The kernel code of socket release on pf_key socket will
    // explicitly call synchronize_rcu() which is exactly what we need.
    int pfSocket = socket(AF_KEY, SOCK_RAW | SOCK_CLOEXEC, PF_KEY_V2);

    if (pfSocket < 0) {
        int ret = -errno;
        ALOGE("create PF_KEY socket failed: %s", strerror(errno));
        return ret;
    }

    // When closing socket, synchronize_rcu() gets called in sock_release().
    if (close(pfSocket)) {
        int ret = -errno;
        ALOGE("failed to close the PF_KEY socket: %s", strerror(errno));
        return ret;
    }
    return 0;
}
```

Our plans for Android R+

More CLAT:

- TX: normal tcp/udp non-fragmented packets *(this is the big performance win)*
- RX & TX: ip fragments [*] *(covers large udp packets)*
- RX & TX: icmp packets - *complex and ugly*
- Hopefully eliminate need for userspace clat daemon entirely

[*] this may be hard to accomplish on 4.9…
bpf_skb_change_proto() only supports IPv4 [20] <-> IPv6 [40], not IPv4 frag [20] <-> IPv6 frag [40+8]
IPv6 -> IPv4 can cheat by using 8 bytes of NOP ip options...

But:
How to make a tx packet visible in tcpdump both pre and post mutation (tc egress)?

May not actually be worth fully completing before Android S, since Android R still needs to support 4.4 kernel devices and thus userspace clatd daemon has to exist on at least some devices anyway...

Our plans for Android R+ (continued)

Offload tethering, incl. NAT…

Android multinetwork support causes a pretty complex routing and netfilter setup: presumed to be at least a cause of poor tethering performance (no multi-gigabit 5G speeds on a phone CPU in 3W power budget).

We'd like to get rid of the current reliance on hardware offload solutions:
hard to debug, and every device is different

Move to some sort of eBPF packet mangling/forwarding offload (or perhaps better call it bypass?).

Try to use XDP…

But: *there's so many different drivers…*
Cellular (many...), wireless (many… some don't even do checksum offload), usb dongles, usb ***rndis*** *(or cdc?)*

Enable BPF JIT *(basically an oversight it wasn't enabled, but CLANG CFI is currently incompatible)*

...but been busy dealing with Android Q fallout...

Challenges

Security

- loading only ebpf programs that are signed and/or from a dm-verity partition.
- no dynamically generated programs

Filtering and/or modifying netlink messages

- for example MAC addresses in Netlink Route messages *(‘ip link show’ output)*
- or kernel filtering of uevent to prevent spurious wakeups

Making it all work on old kernels

- or only on new ones and having extra support code for old ones

Huge variance from device to device

- testing is a nightmare… *(and eBPF is hard to debug too…)*

Android Common Kernel (Net) vs Upstream

Linus -> Stable/LTS -> Android Common -> Chipset Vendor (ie. Qualcomm) -> Device/OEM (ie. Google Pixel 3)

Android Common is many branches: {3.18, 4.4, 4.9, 4.14, 4.19, 5.x LTS, mainline} x {O, P, Q, R} ...
Similarly **many** trees from chipset vendors, and *most* OEMs have one tree per device. **It's a true forest out there.**

Android Networking Tests (UML or QEMU based) pass on 4.20.17 + 11 core networking changes

- 5 patches from upstream (three from 5.0, one from 5.1, one from 5.3) + 1 gki_defconfig
- net: xfrm: make PF_KEY SHA256 use RFC-compliant truncation **[1 liner: 96 -> 128]**
- xfrm: remove in_compat_syscall() checks **[2x remove 'if (in_compat_syscall()) return -EOPNOTSUPP;']**
- net: ipv6: autoconf routes into per-device tables **[4 files: 58 +, 39 -]** -> VRF? But also used by Chrome OS
- netfilter: xt_quota2: adding the original quota2 from xtables-addons **[4 files: 450 +]** -> update xt_quota?
- netfilter: xt_IDLETIMER: Add new netlink msg type **[2 files: 245 +, 12 -]** -> simplify and upstream?

5.0+ fails (not yet figured out why), 5.1+ required test adjustment (SO_RCV/SNDBUF changes), 5.2+ required upstream fixes (RTF_ADDRCONF, just landed in net/master, not in 5.3-rc8, hopefully in 5.3?)
But: even further core networking stack divergence in chipset/device/OEM kernels…

Resources

Code location in AOSP: ***(development is entirely in AOSP, including code review visible in AOSP's Gerrit)***

https://android.googlesource.com/kernel/common/ (android-mainline, android-4.9-q, ...)

https://android.googlesource.com/kernel/configs/ (master branch)

https://android.googlesource.com/kernel/tests/ *(network tests only a.t.m.)*

https://android.googlesource.com/platform/system/bpf/ *(support, loader)*

https://android.googlesource.com/platform/system/netd/ *(In particular /bpf_progs/...)*

https://android.googlesource.com/platform/external/android-clat/ *(daemon)*

Docs

https://source.android.com/devices/tech/datausage/ebpf-traffic-monitor

Questions?

Thank You

android Bootcamp 2019

eBPF implementation in Android networking

March 12, 2019

Introduction

What is eBPF

- Kernel bytecode program that can be loaded and attached at run time
- Can be attached to cgroup, iptables, tc (traffic controller) for networking purposes
- Uses in-kernel data structure eBPF maps to store data and control program behavior

Motivation

- Replace Android kernel modules that cannot be submitted to upstream Linux
- Possibly allow upgrading functionality without upgrading kernel

Usage

- Stats collection
- Packet filtering

Changes released in Android Pie

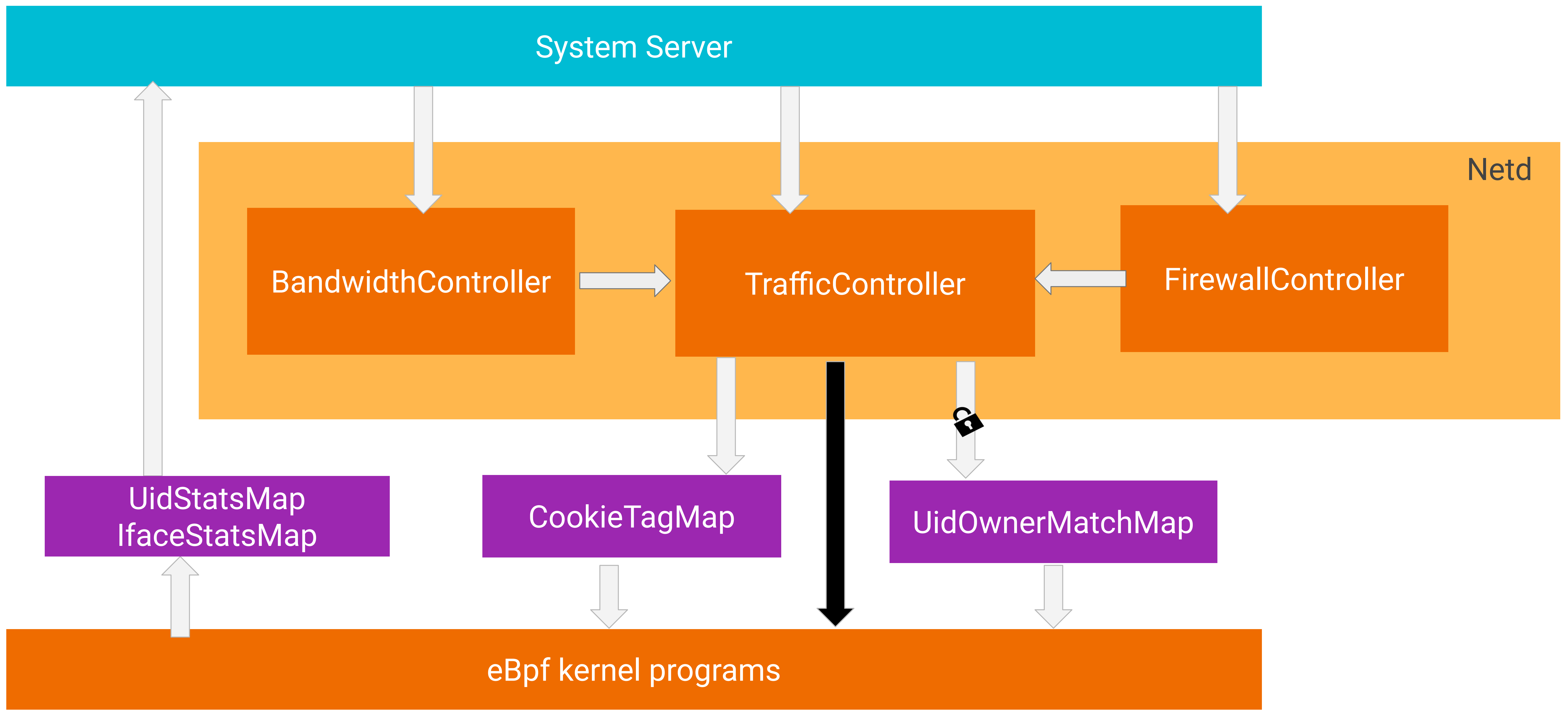
Network stats monitoring

- Attach eBPF program to cgroup root directory to monitor per-UID network stats
- Replaced xt_qtaguid to support socket tagging and background foreground stats counting
- xt_bpf iptables module uses eBPF program to collect per-interface networkStats

Packet filtering

- Replaced xt_owner module in FirewallController and BandwidthController to do per-UID packet filtering.

General design



Next release (Q)

More applications

New cgroup socket filter to control ipv4/6 socket creation, new tc bpf action to handle ipv6 to ipv4 translation.

Better performance

Simplified design for loading eBPF programs and creating eBPF maps. Speed up the process of pulling stats from eBPF maps. Faster packet processing speed.

More coverage

Completely replace xt_qtaguid module and paranoid network feature in Android kernel.

Module deprecation

- **xt_qtaguid**
  - Devices shipping with Q and kernel >= 4.9 must have the config turned off
  - Code is removed from 4.9+ common kernels
  - User space support will be removed once the minimum required kernel is 4.9 (presumably S)
- **Paranoid network**
  - Devices shipping with Q and kernel >= 4.14 must have the config turned off
  - Code is removed from 4.19 kernel
  - Planned to remove from 4.14 common kernel as well

Cgroup socket filter

Functionality

- Control INET/INET6 socket creation
- Only services and apps that have android.Permission.INTERNET are allowed

Motivation

- Replace the out of tree kernel feature "android paranoid network".

Requirement

- Required for devices shipping with Q and kernel version >= 4.14

464XLAT eBPF offload

We tentatively plan to have in-kernel translation of incoming unfragmented ipv6/{tcp,udp} packets to ipv4. Bypasses the userspace clat daemon. Performance improvement for download direction.

In the future, we expect to handle the remaining receive edge cases and have support for transmit side offload as well. Our goal is to remove the clat daemon.

Requirements

- Kernel version >= 4.9
- The kernel configs MUST be turned on:
  - **CONFIG_NETFILTER_XT_MATCH_QTAGUID=n**
  - **CONFIG_NETFILTER_XT_MATCH_OWNER=y**
  - **CONFIG_NET_CLS_BPF=y**
  - CONFIG_CGROUP_BPF=y
  - CONFIG_BPF=y
  - CONFIG_BPF_SYSCALL=y
  - CONFIG_NETFILTER_XT_MATCH_BPF=y
  - CONFIG_INET_UDP_DIAG=y
- The device MEM_LOCK rlimit MUST be set to 8 MB or more.

Related tests

- Unit tests:
  - kernel net_tests
  - netd_unit_test
  - libbpf_test
- Integration test:
  - netd_integration_test
  - vtsKernelNetBpfTest
- Debugging tools:
  - adb shell dumpsys netd trafficcontroller

Future releases

Broader coverage
Will expect more applications of eBPF in Android in the future.

Better performance
With more upstream support, eBPF tools can be faster in the future.

Easier to adopt
Simplify the adoption process to have no additional changes required for partners.

android

Resources

Docs

https://source.android.com/devices/tech/datausage/ebpf-traffic-monitor

Code location is in AOSP:

https://android.googlesource.com/platform/system/netd/

https://android.googlesource.com/platform/system/bpf/

Tl;dr

Summary

Android Q will introduce various improvements and more functionalities to eBPF networking in Android.

Next steps

Try out the new eBPF features available in AOSP on new devices, provide feedback, suggestions, and bug reports.

Thank You