Watch Where the Linux Kernel Drops a Packet (die.net)

117 points by dedalus on May 20, 2020 | hide | past | favorite | 20 comments

Ao7bei3s on May 20, 2020 | next [–]

dropwatch is great, I've used it to debug why our router firmware drops packets. It has some issues (I really need to verify which are still present and file them) but it works.

You can also achieve mostly the same effect with perf via the skb:kfree_skb tracepoint. This has less performance impact, better post-processing options, and doesn't require the dropwatch specific kernel config option (NET_DROP_MONITOR).

m00dy on May 20, 2020 | prev | next [–]

I recently read a blogpost about post-mortem in google engineering blog. They also mentioned this tool to detect dropped dns packages. Generally, when I see something new or something I never heard of before, when I see it two times at least within frequent time intervals, it almost becomes impossible to forget that for me. Brain is weird.

azinman2 on May 20, 2020 | parent | next [–]

I'm guessing this is posted BECAUSE of that article.

jiveturkey on May 20, 2020 | parent | prev | next [–]

Not sure if you are saying weird because you think you, specifically, are unusual, or weird just as a general comment that the brain is weird (like if you wear glasses that make images upside-down, you'll get used to it).

this is a very well known phenomenon, permeating everything, everything around us. it's how advertising works.

2bluesc on May 20, 2020 | prev | next [–]

I setup the AUR package [0] for this while troubleshooting some dropped UDP data and was kind of shocked at how hard this tool was to used. It reported tons of drops everywhere that I couldn't observe from userspace on any of my tests.

Is anyone aware of an overview showing how to approach and get started with this tool?

[0] https://aur.archlinux.org/packages/dropwatch/

Ao7bei3s on May 20, 2020 | parent | next [–]

dropwatch's definition of drop is literally kfree_skb, ie kernel stops processing a packet (plus changes to drop count at napi device). Don't think of it as recording drops, think of it as recording the ultimate fate of every packet, including e.g. iptables -j DROP, but also e.g. successful transmit by your network driver. dropwatch is a low level tool.

I've found it useful to perform a baseline test to see "normal" drops and their rates, then run intense test traffic and compare.

You pretty much have to understand each reported call site by looking at the kernel source code. As you gain experience with the kernel network code, you learn which locations do what, and how to access the regular statistics for them. Having the (outdated but) relevant kernel networking books on your desk helps.

2bluesc on May 20, 2020 | root | parent | next [–]

> dropwatch's definition of drop is literally kfree_skb

Wow, this changes everything. Thanks for clarifying. With this insight `dropwatch` sounds much harder to use then suggested.

DominoTree on May 20, 2020 | prev | next [–]

Link to the GitHub repo:

https://github.com/nhorman/dropwatch

Ao7bei3s on May 20, 2020 | parent | next [–]

Note there's also kernel side code for this. Theoretically it's generic, but practically it was written specifically for dropwatch and dropwatch is the only user of it that I am aware of.

https://github.com/torvalds/linux/blob/master/net/core/drop_...

nh2 on May 20, 2020 | root | parent | next [–]

If you just want to enable it in your kernel (in case your distribution hasn't already), the config option is called `CONFIG_NET_DROP_MONITOR`.

I've recently made it on-by-default in NixOS and also collected some info about which other distros already have it on-by-default and since when: https://github.com/NixOS/nixpkgs/pull/85119

dropwatch is very useful, I used it to debug interruptions of our VPN setup between servers.

DominoTree on May 20, 2020 | prev | next [–]

Would something similar be achievable using eBPF?

▲ Ao7bei3s on May 20, 2020 | parent | next [–]

Yes, you can attach an eBPF program to the skb:kfree_skb tracepoint.

For example: perf record -e skb:kfree_skb

▲ terrywang on May 21, 2020 | prev | next [–]

Great to know such a tool exists ;-)

Thanks @2bluesc for creating the AUR for Arch (Yes, Arch + Manjaro user ;-)

NOTE: what surprised me was that Fedora 32 has dropwatch in its official repo (so simply dnf install dropwatch worked without fuss). It all made sense when discovering the author works for Red Hat when looking at the GitHub repo, well done.

▲ badrabbit on May 20, 2020 | prev | next [–]

Is it for any layer? If traffic is dropped at the frame/L2 level or at the tcp layer, would this show?

▲ pfundstein on May 20, 2020 | parent | next [–]

Anything the kernel sees, so L2+

▲ egberts1 on May 20, 2020 | prev | next [–]

Nice. Now where's the repository?

▲ PeterCorless on May 20, 2020 | parent | next [–]

https://github.com/nhorman/dropwatch

▲ gigatexal on May 20, 2020 | prev | next [–]

oh boy! i had no idea such a utility existed, very cool.

▲ ggmartins on May 20, 2020 | prev [–]

cool, trying to understand 3. Ambiguity. from the output: 4 drops at unix_stream_connect+4ff (0xffffffffb8e089ef) 4 drops at unix_stream_connect+4ff (0xffffffffb8e089ef) 2 drops at tcp_v4_rcv+48 (0xffffffffb8db1618) 2 drops at unix_stream_connect+4ff (0xffffffffb8e089ef) 2 drops at unix_stream_connect+4ff (0xffffffffb8e089ef)

especially the reason for the drop? what does +48 in "tcp_v4_rcv+48" mean, thanks and sorry if this was documented? and I TL;dr

▲ Ao7bei3s on May 20, 2020 | parent [–]

This is a standard notation. tcp_v4_rcv+48 means, within the binary kernel image, 0x48 bytes after the start of the tcp_v4_rcv function. You can use the addr2line tool to find the source code line. https://serverfault.com/questions/605946

The tcp_v4_rcv function has two obvious kfree_skb calls, here https://elixir.bootlin.com/linux/v5.6.13/source/net/ipv4/tcp... and a few lines below after "discard_it:" (there may be more if there are #define's or inlined function calls), and without your kernel image and debug info I cannot tell which one the offset corresponds to. Also, clean-up code like kfree_skb is often after a label ("out:") referenced by multiple goto's, and you cannot tell which goto was taken. However, often the function return value contains an error code that identifies it, and you can (often) grab that with perf by attaching a dynamic kprobe to the function exit (it's much easier than it sounds). Or attach a gdb to the kernel (easiest is if the kernel is in a qemu VM) and put a breakpoint on tcp_v4_rcv. There's also the inverse problem of "who called tcp_v4_rcv". Either gdb, or perf record -g, can tell you the stacktrace. (perf is less invasive, so better in production)

As an example, take https://elixir.bootlin.com/linux/v5.6.13/source/net/ipv4/tcp... : if the packet is a SYN belonging to a new connection, but has a bad TCP checksum, goto csum_error, and from there fall-through to discard_it: kfree_skb(skb).

This may sound laborious, and it is, but note that often you don't need to go to this effort. To me, as a troubleshooter, the precise reason might not be that relevant. The function name already tells me this packet has gone up into the TCP receive stack, which (basically) rules out entire problem areas like bridging and routing, tells me if this specific drop is even relevant for me, and/or lets me decide which simpler tools to use next.