Let’s Be a Little Less Indirect…

•••

David S. Miller
Netconf 2018 - Boston

All Hail Spectre!

We all now know that speculative executation can be exploited.

One of the biggest wake up calls in the industry in a long time.

Maybe we were sleeping at the wheel and we deserved this.

The number of susceptible cpus out there in the world is nearly incomprehensible.

If you said “every computing device is effected” you’d be accurate by at least an order of magnitude.

We are stuck with this for a long time, so…..

Why does networking care?

How can this matter particularly to networking?

All traditional packet processing paths are dispatched as method ops.

Not just at one level, but at many different protocol levels

→ ethernet → ipv4 → tcp → TLS → etc....

Each and every one of these protocol op demuxes is an indirect call

Worse are filtering engines like netfilter and tc, one demux or more for every rule

Spectre’s Effect...

Indirect calls are significantly more expensive.

They always were non-trivial in cost.

But now the order of magnitude of that cost is larger.

Since the parameters have changed, we need to look at optimization ideas (both old and new) differently.

Batching is Awesome

But thanks to Spectre, it’s now _really_ awesome.

Processing more packets at a time decreases full stack traversals.

Less full stack traversals means less indirect calls.

This was always true, but now the gap is wider.

So, old ideas that didn’t pan out because the cost of batching outweighed the gain, need to be re-investigated and re-evaluated.

Ideas that wouldn’t even be considered before, will now be taken seriously.

Eliminating Indirectness Entirely.

How can we do that?

For filtering engines, it can be done. That is, once we know the ruleset.

- Run time code generation into eBPF of filtering paths

No indirect calls except the single one necessary to run eBPF program.

This is one of the benefits we hope to see from the bpfilter project.

Can this idea be extended to other areas of the stack?

More Generally, Code Path Linearization

Can this be generalized for cases we think are worth optimizing?

Instead of demuxing L2, L3, L4, etc. have a pre-cooked linear code path for specific combinations. Such as ethernet+ipv4+TCP

Before you shout “layering violation!” remember:

> Layering is how you design protocols, not how you implement them.

Any other ideas?

Side Topic: SKB Lists...

For real, this time!

skb->next and skb->prev are pointers generally (but not always) used for lists

The type of lists used isn’t even consistent!

Some code hangs arbitrary state in one of the pointers while they own the SKB

Over time the divergance of usage gets more baked in

Therefore, cleaning it up becomes harder and harder

SKB Linked List Conversion

Change skb->{next,prev} to struct list_head, but then what?

Users of SKB queue and list interfaces will be easy.

Custom list handling cases (f.e. GRO) will be harder.

Worst issue is transition points:

- Where SKB is passed from one owner to another
- Who is responsible for clearing old SKB list state
- Will all SKBs which are freed have sane SKB list state which can be asserted?

Idea: Do the work backwards.

Audit transition points and SKB list usage.

Make existing SKB list code follow the list_head state rules explicitly.

Add assertions to transition points and SKB queue helpers.

Then, and only then, use list_head in sk_buff.

Divergent SKB List Types

How to resolve the singly vs. doubly linked list conundrum?

skb_queue_head uses doubly linked list

GRO uses singly linked list

Options:

- Make everyone use doubly linked list, if that is even possible
- Make SKB list object a union (hard to test state of arbitrary SKB)

Neither option sounds great.

Side Topic: Robots, Like Them or Not...

Syzbot, kbuild, etc.

Increasingly dominating the discovery of bugs and build problems.

Kbuild often finds build bugs before I formally review a patch in patchwork.

Over time developers are becoming dependent upon these robot helpers.

Kbuild Robot Seems Uncontroversial

A pure positive in my view.

Whether something builds or not is completely unambiguous.

It saves maintainer time significantly.

First thing I do is run an allmodconfig build on a patch (ok, sometimes I cheat)

It saves reviewer time as well.

If reviewer has choice between looking at code that does vs. doesn’t build, well duh…

Syzbot

This guy is another animal altogether.

Finding bugs in code already in the tree.

Not just any bugs, but sometimes very scary and very old bugs.

It’s very effective at what it does, perhaps a little too much so.

Good developers often now overwhelmed working on syzbot reports.

Developers can start using syzbot as a “crutch” during development.

Anecdotal Evidence...

I see less and less bugs found by code auditing.

Bug fixes are primarily in response to syzbot reports or a failure due to a recent change in a user's active setup which they test every day.

Developers do have "run syzbot on it" as part of the pre-submission routine.

- Is this "in addition" to what testing they would have done
- Or is it "instead of" what testing they would have done

Another great argument for adding more selftests.

Side Topic: Testing…

Selftests is growing more quickly lately, this is great!

Let’s hope this pattern continues.

Existence of tests in a patch series definitely increases it’s karma.

The old joke about: “Did you actually test this?”

How do we move from “nice to have” to “require” for networking submissions?

Other projects thrive with the “require” element, look at GCC etc.

Side Topic: Maintainership

This net-next series was crazy.

Both for networking and BPF.

It took more than two weeks after net-next openning for things to slow down.

Quality reviews are the main bottleneck when things get crazy.

Perhaps it is less about availability of review resources and more about distribution

This doesn’t even get into -stable, which I believe is a maintainer responsibility

Thanks

Alexei and Daniel for maintaining a separate BPF tree, helps a lot!

Eric Dumazet and others for attacking syzbot reports aggressively

Jakub, Jiri, Andrew, Florian, etc. for making sure we have a sane device interface

David Ahern for the routing improvements

And Linus Torvalds, for tolerating us...