XDP and some lower level Linux networking

RITLUG

ritlug.com

To Preface….

- I am by no means an expert on this subject
  - I found this through a Cloudflare blog post (This one in fact!) and thought it was the perfect tool for a class I was taking
  - Everything I know is through trial and error with my experiments, if any of this is interesting to you I encourage you to look further into this!
- I am also no guru when it comes to C in general, I’m sure I will get corrected at some point :)

With all of that being said… Let’s begin!

- XDP (eXpress Data Path) is a high performance data path used to TX/RX network packets at very high speeds
  - By high speeds, I mean *very* high speeds
- It is based off eBPF (“extended berkeley packet filter”, but should be referred to by acronym as a technology, like LLVM)
- XDP was first implemented in the kernel in version 4.8, released around October of 2016
- In the simplest terms, it is a very early RX hook in the kernel that allows a user to supply a eBPF program to decide the fate of a packet.

So, what is BPF then?

- BPF is a mechanism for userspace programs to efficiently specify a filter program to selectively receive packets
  - It is also a mechanism to provide a raw interface to the data link layer (think your NIC), allowing raw packets to be sent/received directly.
- BPF is available on most Unix based systems to this day
- For the most part, the Linux world refers to BPF by just its filtering capabilities, since the Linux kernel provides other ways to access the data link layer directly.
- A popular program that uses BPF is tcpdump
  - The arguments you pass tcpdump are interpreted as a BPF filter program

What if we extend it?

- Then, we would have eBPF
  - To condense about 10 years of history, here’s what this did.

According to Wikipedia

- At the lowest level, [eBPF] introduced the use of 10 64-bit registers in lieu of 2 32-bit long registers, different jump semantics, a call instruction with register passing conventions, “new instructions”, and a different encoding

Let’s make it faster

- In April of 2011, a JIT compiler for cBPF got merged in-kernel
- In July of 2016, eBPF gained the ability to be attached to a network driver’s core receive path, also known as **XDP**.
- A new socket family was added to Linux in release 4.18 named AF_XDP
  - This is a raw socket optimized for high performance and allows zero-copy between kernel and userspace. Since you can both send and receive, you can implement high speed userspace network applications.

Let’s make it more extensible

- Not necessarily a part of XDP, but still cool. We have this great filtering framework, why limit it to just networking?
  - That’s right, you can use eBPF programs to filter syscalls
  - This ability is used mostly by seccomp, which is used by Android, QEMU, OpenSSH, Flatpak, Firejail

systemd

Anyways, what exactly makes XDP so special?

- The fact that it comes so early in the packet processing pipeline.
- In fact, it comes so early that processing occurs before the network stack performs any needed memory allocations
  - The eBPF program runs right after the interrupt processing is complete.
  - In some cases, the NIC itself will execute the XDP code if supported, completely offloading that burden from the main CPU.

So, being that early, how fast is XDP?

- Very fast
  - ***Very***, very fast.
- According to tests done in this repo, we are looking at the following statistics:
  - Tests were performed on an Intel Xeon E5-1650 v4 @ 3.6GHz (a CPU released in 2016)
  - Packets were able to be dropped at a rate of 26Mpps per core
  - Packets were able to be redirected at a rate of 8.5Mpps per core

Wow

- Yeah.. it's fast. It's no wonder why big companies like Amazon, Google, Facebook, and Cloudflare use it for various tasks
  - For example, Cloudflare re-implements iptables rules using XDP to perform high performance DDOS protection
  - Facebook's Layer 4 load balancer uses XDP to route packets

So, what’s the catch?

- At a very high level, you trade extensibility for speed.
- First off, it can only perform these operations:
  - PASS → pass the packet to the network stack
  - DROP → silently drop the packet
  - ABORTED → drop the packet with a trace point exception
  - TX → bounce the packet back to the receiving NIC
  - REDIRECT → redirect the packet to another NIC or userspace AF_XDP socket

- Also, most changes require a program to be re-compiled (usually using clang)
  - Some changes can be done dynamically, which I will get to in a moment
- Due to the privileged nature of this code, all eBPF programs must run through a pre-verifier test within the kernel
  - This step ensures that there are no out of bound memory accesses (memory safety!), infinite or otherwise non-returning loops or functions, anything that may crash or hang, or contain any global variables.
  - It’s like C, but a lot safer!

Alright, I’m sold, how do I make a XDP Program?

- First, make sure you have the following dependencies installed:
  - Linux Headers
  - Libbpf headers
  - Libxdp headers
  - Clang
- Now, all we need is a C file
  - Let’s make a simple program that drops everything

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC(“xdpentry”)
int xdp_dropper(struct xdp_mp *ctx) {
  return XDP_DROP;
}

char _license[] SEC(“license”) = “GPL”;
```

What makes it safe?

- It doesn’t allow any memory accesses that may be out of bounds
  - …. which means you will be seeing this code a whole lot

```
if (icmph + 1 > data_end) {
   // More bounds checking
   return XDP_PASS;
}
```

How can you communicate?

- So, the thing is, the program is run fresh on every received packet, there is no “continuous execution”, or shared state between packets
- Which begs the question, how do I share state between packets and invocations of the program?
  - Plus, how do I communicate with userspace?
- The answer: BPF maps (and other data structures)
  - BPF maps are a data structure that allows you to set and get values by keys from both userspace and kernel space, meaning data and state can be shared (to a degree)

Enough slides, let’s see an example!