XDP hands-on tutorial

Jesper Dangaard Brouer
Toke Høiland-Jørgensen

Bornhack
Gelsted, August 2019

Outline

Introduction - what is XDP?

About this tutorial - plan for today

Bonus tasks

What is XDP?

XDP basically: New layer in the kernel network stack

- Before allocating the SKB
- Driver level hook at DMA level

Means: Competing at the same “layer” as DPDK / netmap

- Super fast, due to
  - Take action/decision earlier (e.g. skip some network layers)
  - No memory allocations

Not kernel bypass; data-plane is kept inside the kernel

- Via eBPF: makes early network stack run-time programmable
- Cooperates with the kernel stack

About this tutorial

This tutorial is meant as a living document, developed on Github:

https://github.com/xdp-project/xdp-tutorial

This session is the second beta test of the live version.

- Please send feedback; or even better, pull requests!

Plan for today's session

- This introduction
- You each go through the tutorial in the git repo
- We will help answer questions
- Plenary follow-ups as needed

Structure of the tutorial

Comprised of seven topical lessons, in the numbered directories in the git repo.

We recommend you complete them in this order:

- basic01-xdp-pass
- basic02-prog-by-name
- basic03-map-counter
- basic04-pinning-maps
- packet01-parsing
- packet02-rewriting
- packet03-redirecting

Read the README.org file in each directory to get started.

Basic introduction and understanding of eBPF

Basic introduction to

- eBPF bytecode
- Compiling restricted-C to eBPF
  - compiler storing it in ELF-format
  - loading this into the Linux kernel

eBPF bytecode and kernel hooks

XDP 'just' a Linux kernel hook that can run eBPF-bytecode

- Many more eBPF hooks (tracepoint, all function calls via kprobe)

The eBPF bytecode is:

- Generic Instruction Set Architecture (ISA) with C-calling convention
  - Read: the eBPF assembly language
- Designed to run in the Linux kernel
  - It is not a kernel module
  - It is a sandbox technology; BPF verfier ensures code safety
  - Kernel provides an eBPF runtime environment, via BPF helper calls

Compiling restricted-C to eBPF into ELF

LLVM compiler has an eBPF backend (to avoid writing eBPF assembly by hand)

- Write Restricted C – some limits imposed by sandbox BPF-verfier

Compiler produces an standard ELF “executable” file

- Cannot execute this file directly, as the eBPF runtime is inside the kernel
- Need our own ELF loader that can:
  - Extract the eBPF bytecode and eBPF maps
  - Do ELF relocation of eBPF maps references in bytecode
  - Create/load eBPF maps and bytecode into kernel
- Attaching to hook is separate step

libbpf

This tutorial uses libbpf as our ELF loader for eBPF

- libbpf is part of Linux kernel tree
- Facebook fortunately exports this to https://github.com/libbpf
  - Tutorial git repo, use libbpf as git-submodule

Please userspace apps: Everybody should use this library

- Unfortunately several loaders exists
- Worst case is iproute2 have its own
  - cause incompatible ELF object, if using eBPF maps
  - (stalled?) plans for converting to libbpf

eBPF concepts: context, maps and helpers

Each eBPF runtime hook gets a pointer to a context struct

- BPF bytecode has access to context (read/write limited)
  - verifier may adjust the bytecode for safety

The BPF program itself is stateless

- Concept eBPF maps can be used to create state
- Maps are basically key = value construct

BPF helpers are used for

- calling kernel functions, to obtain info/state from kernel

Testlab on your laptop!

XDP performance comes from running at driver level

- as close as possible to NIC hardware, just after DMA-sync to CPU

In this tutorial, we create a virtual network environment

- Disadvantage: obviously not as fast
- Advantage: can be setup directly on your Linux laptop
  - use network namespaces and veth (like containers do)

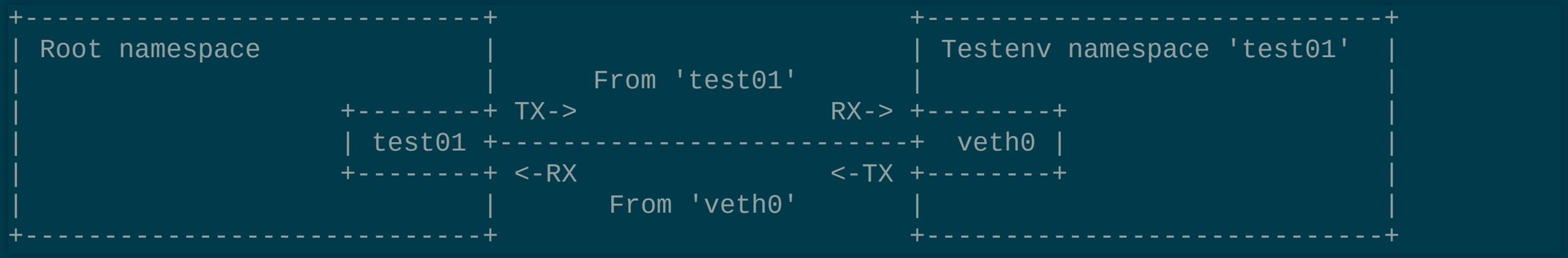
The test environment helper script

The `testenv` directory contains a helper script to setup a test environment.

- Uses network namespaces and virtual network devices to simulate a real setup
- Requires kernel version 4.19 or higher
  - Due to `veth` driver getting native-XDP support (incl. fixes)
  - Preferred kernel is 4.20 as `veth` got ethtool statistics
- See README.org in the testenv directory for instructions
- Easy alias:
  - `eval $(./testenv alias)`,
  - then `t setup`

Namespaces and virtual ethernet devices

- The testenv script uses network namespaces and virtual ethernet devices to simulate a real environment.



- XDP programs are installed on the test01 interface in root namespace
- Generate traffic from inside the namespace

Bonus tasks

As we said, this is the second beta test. So some of you may finish all tasks before we run out of time.

Here are some suggestions for extra tasks:

- Complete some of the other lessons not mentioned above
- Improve the tutorial and send a pull request
- Implement your own use case and test it (we'll help!)
- Write a blog post about your experience with XDP

Getting started

```
$ git clone https://github.com/xdp-project/xdp-tutorial
$ cd xdp-tutorial
$ git submodule update --init
$ less README.org
```