XDPのテストとCI

日下部雄也(@higebu) BBSakura Networks, Inc ※育児休業中

2021/10/12 Open Mobile Network Infra Meetup #4

アジェンダ

- なぜ #omni_jp でXDPのテストの話をするのか
- XDPのテストの概要
- XDPのテストで気をつけた方がいいところ
- 世の中で行われているXDP(eBPF)のテストのやり方
- BBSakuraでのXDPのテストのやり方

なぜ #omni_jp でXDPのテストの話？

1. さくらのセキュアモバイルコネクトのUプレーンで使っているから
   - 2019年12月から本番環境でXDPを使ったPGWの運用をしている
   - 参考：ENOG63 モバイルネットワークのデータプレーンを XDPで作る話
2. XDPでUPFを実装している人を見かけるから
   - navarrothiago/upf-bpf
3. パケット処理のテストは面倒だけどテスト書かないわけにはいかないので…

XDPのテストの概要

XDPのテスト方法

1. BPF_PROG_TEST_RUN
2. 実際にパケットを送受信

XDPのテスト方法

1. BPF_PROG_TEST_RUN
    - eBPF Syscall — The Linux Kernel documentation
    - ロードしたプログラムの fdと実行回数、パケットデータを渡すと、指定された実行回数プログラムを実行し、プログラムのリターンコード（ XDP_PASSなど）、修正されたパケットデータ、実行時間を返してくれるbpf(2)のサブコマンド
    - 5.15からxdp_mdが指定可能になり、XDP metadataのテストもできるようになる
        - https://lore.kernel.org/bpf/20210707221657.3985075-1-zeffron@riotgames.com/
    - オフラインかつ1つのマシン内でテスト可能なので比較的手軽
        - ただ、IPやMACアドレスを書き換えたり、ルートテーブルを参照するようなプログラムでは、結局環境を整える必要がある

XDPのテスト方法

2. 実際にパケットを送受信
   - よくある性能検証のようにパケット送受信をするテスターを用意し、 XDPのプログラムをロード、アタッチしたテスト対象に対して、実際にパケットを送受信する
   - TRexでやっているのを見かける
   - テスターとDUTは別のサーバなのが普通なので手間がかかる
     - VMでやれば多少マシそう

XDPのテストで
気をつけた方がいいところ

XDPのテストで気をつけた方がいいところ

1. カーネルのバージョンを本番と同じにする
    - カーネルのバージョンが違うとXDP関連の機能がサポートされていなかったり、バグが治っていなかったりするし、機能が同じでも挙動が違う可能性がある
2. NICもできれば本番と同じにする
    - XDP関連の機能はNICのドライバ毎に実装されているため、ドライバが違うとry
3. native/genericも本番と同じにする
    - native modeだったらnative modeでテストする

まとめると、「カーネル内のコードをテストと本番で同じにする」ということ

世の中で行われている
XDP(eBPF)のテストのやり方

世の中で行われているXDPのテストのやり方

- libbpf
  - Linux公式のCでeBPFするためのライブラリ
  - travis-ci/vmtest にテスト用のスクリプトがある
  - 2ヵ月くらい前にTravis CIからGitHub Actionsに移行したように見える
  - vmtestというローカルのActionからrun_vmtest.sh → run.sh と呼ばれて、最小限のLinuxイメージを作りつつ、qemuでVMを起動してテストしている
  - mkrootfs.sh を見ると最小限のArch Linuxのイメージを作っているように見えるが、現在はこれは使われていなそう
  - Linux本体のselftests/bpfを全部実行している

世の中で行われているXDPのテストのやり方

- Linux kernel
  - KernelCI があるが、bpf-nextはここではやっていなくて kernel-patches/bpf でやっている様子
  - v5.12から tools/testing/selftests/bpf/vmtest.sh というスクリプトが入っている
    - [PATCH bpf-next v5 0/2] BPF selftest helper script
  - libbpfのテストを移植していて、同様に qemuでVMを起動してテストを実行する方式
  - 移植時にきれいになっているようで読みやすい（個人の感想です）

世の中で行われているXDPのテストのやり方

- cilium/ebpf
  - GoでeBPFするときのライブラリ
  - Semaphore CI でCIしている(.semaphore/semaphore.yml)
  - run-tests.sh が実行されると virtme 経由でVM上で go test が走る仕組み
  - テスト時に起動するVM用のカーネルイメージは cilium/ci-kernels に置いてある
  - BBSakuraではこれを真似しました

世の中で行われているXDPのテスト

- cilium (本体)
  - GitHub ActionsのworkflowでBPF/Makefile の go_prog_test が実行されている
  - bpf/tests/prog_test にBPF_PROG_TEST_RUNを使ったテストがある
    - gopacketでパケットを作っている

世の中で行われているXDPのテスト

- katran
  - GitHub ActionsのWorkflowがあるが、テストは実行していなそう
  - katran/lib/testing/BpfTester.cpp にテストが書いてある
  - pcapからパケットデータを作って BPF_PROG_TEST_RUN
  - DEVELOPING.md にテストのやり方が書いてあり、os_run_tester.sh を実行すると katran_tester.cpp が走る
  - IPが固定っぽいのでVMでやった方がよさそう
  - Migrate some Katran Tests to VMTests というコミットがあるのでVMでテストしてそうに見える

世の中で行われているXDPのテスト

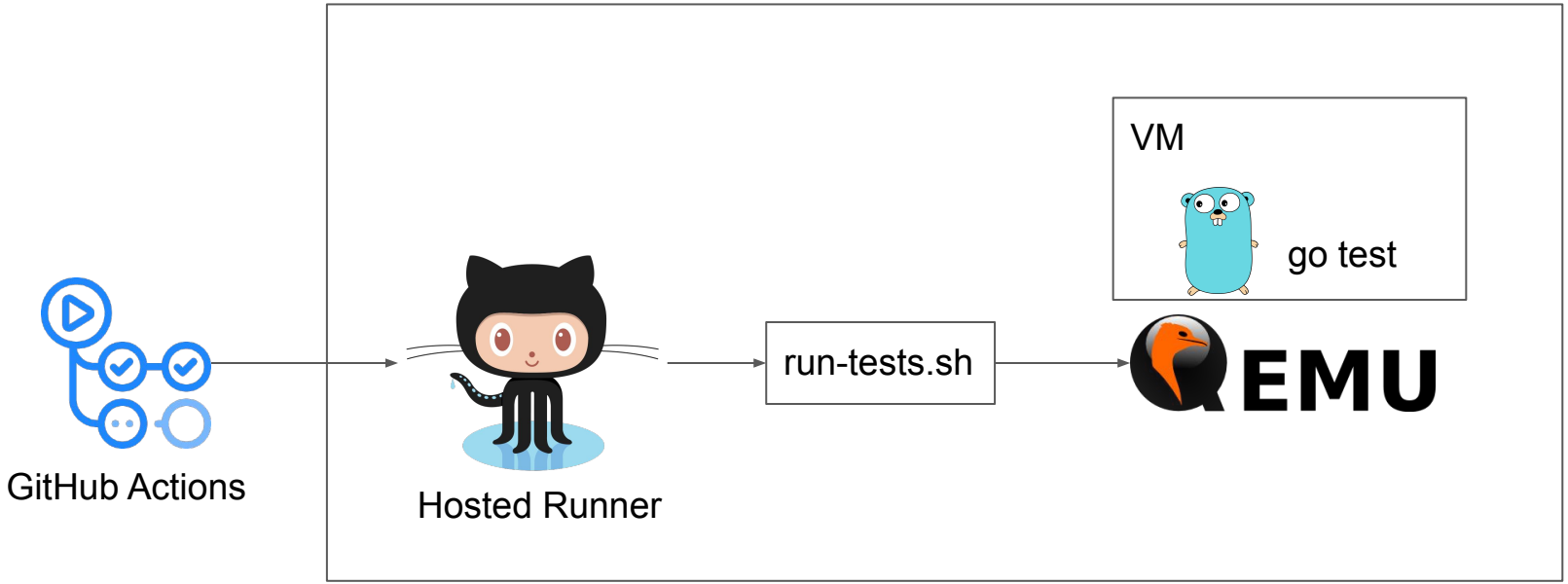
- navarrothiago/upf-bpf
  - XDPを使ったUPF
  - Create a build for CI #40 というIssueがあって、CIはまだやっていなそう
  - tests フォルダ配下にテストコードがある
    - TRexで実際にパケットを送受信する方式

BBSakuraでやっている
XDPのテストのやり方

テスト対象のPGW-Uの概要

- ユーザスペースのGoのプログラムがXDPのプログラムをロードしてNICにアタッチしたり、eBPF Mapを読み書きしている
- XDPのプログラムはeBPF Mapに書かれたセッション情報を元に受信したパケットを書き換えて送信(XDP_TX)したり、ドロップ(XDP_DROP)したり、パス(XDP_PASS)したりしている
- さくらのクラウド（つまりKVM）上で動いている
- 詳しくは ENOG63 モバイルネットワークのデータプレーンをXDPで作る話

CIの流れ



さくらのクラウドの専有ホスト 上のサーバ
Nested Virtualizationが使える

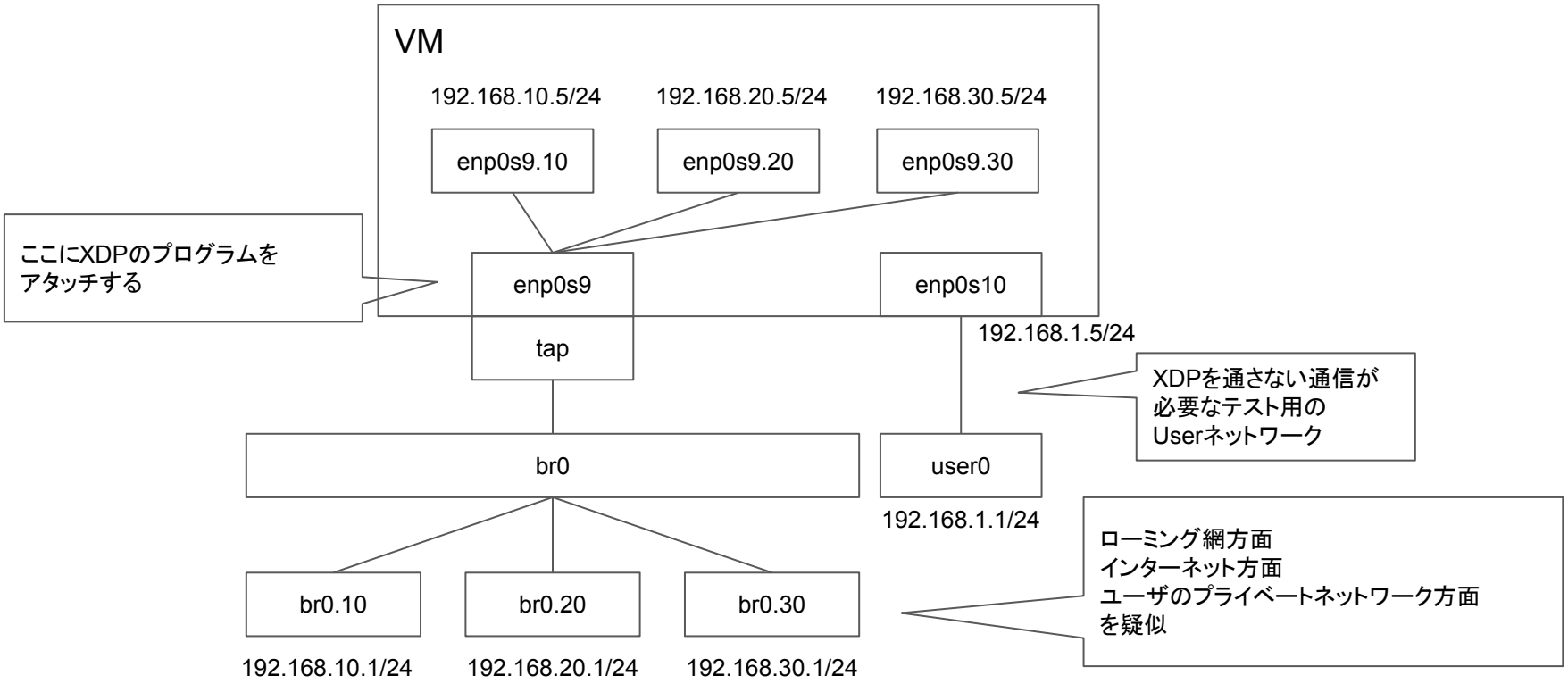
BBSakuraでのXDPのテストのやり方

- cilium/ebpf と同様
- CIはGitHub Actions（with Hosted Runner）
- テストに必要なバージョンの最低限のカーネルイメージを予めビルドしてGitHubリポジトリに置いておく
- テストスクリプトでは、カーネルイメージを取ってきて、virtmeでVMを立ち上げ、VM上でgo testを実行
  - パケットの生成はgopacket
  - 入力用のパケットデータと確認用のパケットデータを用意してBPF_PROG_TEST_RUN
- VMが立ち上がる環境がなるべく本番に近づくようにいろいろお膳建てしている

最低限のカーネルイメージ

- ci-kernels/config at master · cilium/ci-kernels · GitHub
  - シンプルなプログラムのテストであればこれで十分
  - TCと組み合わせたい場合などに足りない
- TCなどが使えるconfigをgistに置いておいたので参考にしてください
  - https://gist.github.com/higebu/145f9e4071258819ba1ad905ce0483ac
- カーネルのビルド方法はここでは説明しませんが、cilium/ci-kernelsのmake.shが参考になります

お膳立ての構成



※VRFなどもあるけどややこしいので省いています

virtme

- カーネルのgitリポジトリにも入っている、VMでカーネルのテストをするための Python製のツール
  - https://git.kernel.org/pub/scm/utils/kernel/virtme/virtme.git
- virtme-configkernel: virtmeで使うために最低限必要なconfigを足してくれるコマンド
- virtme-run: テストを実行するコマンド（qemuのラッパー）
  - script-shオプションにスクリプトを渡すとVM内で実行してくれる

テストスクリプトの中のvirtme-run

```
sudo virtme-run --kimg "${tmp_dir}/${kernel}" --name ${name} --cpus 4 --memory 8192M --pwd \
  --rw \
  --show-command \
  --show-boot-console \
  --force-initramfs \
  --rwdir=/run/input="${input}" \
  --rwdir=/run/output="${output}" \
  --rodir=/run/go-path="$(go env GOPATH)" \
  --rwdir=/run/go-cache="$(go env GOCACHE)" \
  --script-sh "PATH=\"$PATH\" $(realpath "$0") --in-vm /run/output" \
  --qemu-opts -enable-kvm \
  -netdev tap,vhost=on,vnet_hdr=off,queues=8,id=${nic},ifname=${nic},script=${ifup_script},downscript=${ifdown_script} \
  -device virtio-net-pci,mq=on,vectors=12,netdev=${nic},mac=${mac} \
  -netdev user,id=${user_nic},ipv6=off,net=192.168.1.5/24,host=192.168.1.1 -device virtio-net-pci,netdev=${user_nic} < /dev/zero
```

VM内からはインターネットに出られないので、VM起動前に go mod download しておいて、GOPATHとGOCACHEをVMにマウントしている
go testでは “GOFLAGS=-mod=readonly” しておく

vnet_hdr=offにしてるので、いろいろなオフロードのoffがないが、vnet_hdr=onにするならいろいろなオフロードのoffをしないとXDPが使えないので注意
詳細: Virtio-netでXDPを動かすにはqemuのオプション変更が必要 - yunazuno.log
あとmrg_rxbufをon/offするとvirtio_net内で通るコードがかなり変わるので注意
receive_small() or receive_mergeable()
詳しくは “drivers/net/virtio_net.c” 参照

- cilium/ebpfのrun-tests.shをベースにしているので詳しくはそちら。。。
- GitHub Actions等で動かしたいときに最後の < /dev/zero が必要になる
  - --script-sh breaks with /dev/null or closed stdin · Issue #33 · amluto/virtme

gopacketでGTPv1-Uのパケット生成

```
icmpPayload := []byte{...}
opts := gopacket.SerializeOptions{FixLengths: true, ComputeChecksums: true}
buf := gopacket.NewSerializeBuffer()
iph := &layers.IPv4{
        Version: 4, Protocol: layers.IPProtocolUDP, Flags: layers.IPv4DontFragment, TTL: 64, IHL: 5, Id: 1212,
        SrcIP: net.IP{192, 168, 10, 1}, DstIP: net.IP{192, 168, 10, 5},
}
udp := &layers.UDP{SrcPort: 2152, DstPort: 2152}
udp.SetNetworkLayerForChecksum(iph)
gopacket.SerializeLayers(buf, opts,
        &layers.Ethernet{DstMAC: []byte{0x00, 0x00, 0x5e, 0x00, 0x53, 0x01}, SrcMAC: []byte{0x00, 0x00, 0x5e, 0x00, 0x53,
0x02}, EthernetType: layers.EthernetTypeDot1Q},
        &layers.Dot1Q{VLANIdentifier: 100, Type: layers.EthernetTypeIPv4},
        iph, udp,
        &layers.GTPv1U{Version: 1, ProtocolType: 1, Reserved: 0, ExtensionHeaderFlag: false, SequenceNumberFlag: false,
NPDUFlag: false, MessageType: 255, MessageLength: 76, TEID: 2},
        &layers.IPv4{
                Version: 4, Protocol: layers.IPProtocolICMPv4, Flags: layers.IPv4DontFragment, TTL: 64, IHL: 5, Id: 1160,
                SrcIP: net.IP{192, 168, 100, 200}, DstIP: net.IP{192, 168, 30, 1},
        },
        &layers.ICMPv4{TypeCode: layers.CreateICMPv4TypeCode(layers.ICMPv4TypeEchoRequest, 0), Id: 1, Seq: 1},
        gopacket.Payload(icmpPayload),
)
```

長さとチェックサム計算を任せると少し楽

フルバージョン: https://gist.github.com/higebu/9503a3b90c047d5bbf677c0d3eb156df

これを入力用、確認用で全テストケース分書く。。。

GoでBPF_PROG_TEST_RUN

cilium/ebpf を使うとこんな感じになる

```
objs := &ExampleObjects{}
err := LoadExampleObjects(objs, nil)
if err != nil {
    t.Fatal(err)
}
defer objs.Close()

ret, got, err := objs.ExamplePrograms.XdpProg.Test(generateInput(t))
if err != nil {
    t.Error(err)
}

// retern code should be XDP_TX
if ret != 3 {
    t.Errorf("got %d want %d", ret, 3)
}

// check output
want := generateOutput(t)
if diff := cmp.Diff(want, got); diff != "" {
    t.Errorf("output mismatch (-want +got):\n%s", diff)
}
```

フルバージョン：https://github.com/higebu/xdp-example

まとめ？

- XDPでもテストはやれるがお膳立てが大変
- もう少しハイレベルなテストフレームワークっぽいものがあると良いのかもしれない。。。
- 今のところカバレッジ計測が実現できていない
- 当たり前だがCIできるようにしておくとプログラムの変更時に絶大な安心感がある

EOF