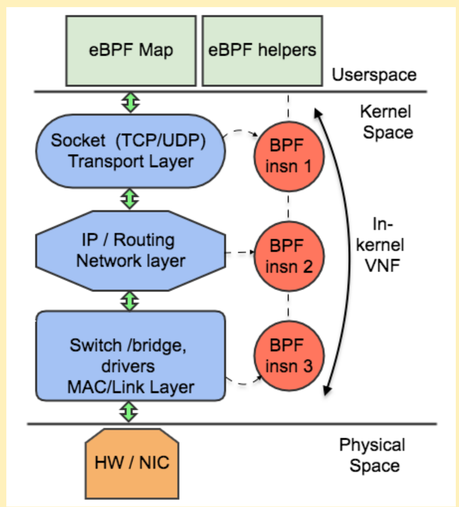
eBPF Filtering and Test Setup for eXpress Data Path on Virtual Network Environment

Sabur Hassan Baidya and Yan Chen
Software Network Virtualization Group, Futurewei Technologies, Santa Clara, CA 95050

Introduction

BPF :

- Berkeley Packet Filter : In-kernel Virtual Machine for packet & syscall filtering
- eBPF: extended BPF for filtering, tracing, networking etc.
- In-kernel JIT compilation
- eBPF Maps: Share data between userspace and Kernel space
- Enhanced 64 bit register, supports instruction call, load, store, conditional jump
- eBPF programs can be used as hook to different layers of the kernel stack
- bcc : bpf compiler collection

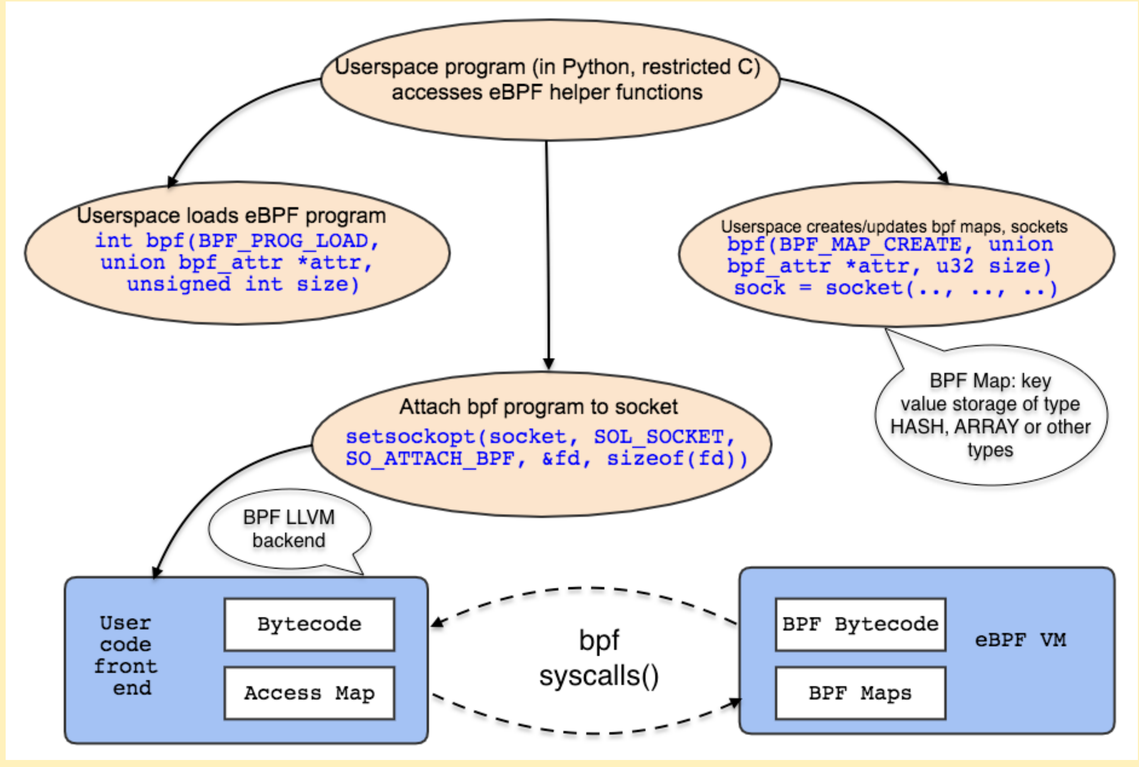


eBPF hook from userspace to kernel space

XDP :

- XDP: eXpress Data Path under IOVisor project
- Programmable, high performance data path at the lowest point of Linux Kernel Software stack
- Very fast packet processing before allocation of skbuff inside device driver Rx function
- Flow table managed by BPF program which is portable to userspace and other OS
- Aim of XDP is to replace the OVS data path with eXpress Data Path

eBPF Implementation and flows



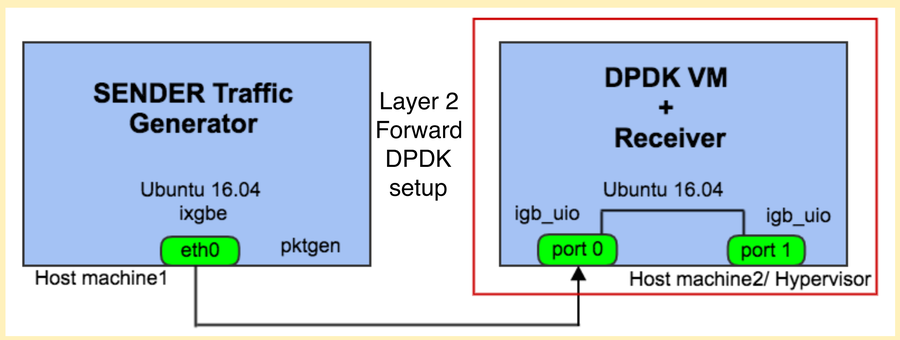
Network Virtualization and Test Setup

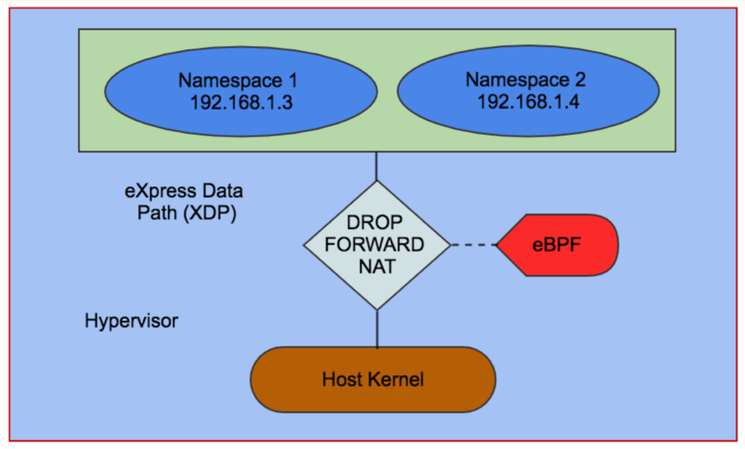
VM vs Container - Comparison and Coexistence

- VM is abstraction of hardware with full hardware stack, virtualized network adapters, memory and CPU.
- Multiple VMs can be instantiated by a hypervisor running on the physical host where a separate OS run for each VM.
- In container, abstraction is mostly done at OS level where the containers share the same Kernel space with the OS but userspace is abstracted.
- The main aim of a container is to provide a runtime environment for applications.
- Containers are lightweight compared to VMs, so many containers can run on a single machine with less overhead compared to VMs.
- In terms of security containers are more vulnerable compared to VMs.
- VM and containers can be used as complementary to each other. Containers can run within a VM providing more isolation, enhanced security and easy manage of hardware.

Test Setup:

- Fast L2 forwarding using dpdk application. The sender generates packet using pktgen and the packets go through dpdk l2fwd application to the receiver.
- XDP testing for fast packet drop in the iovisor container
- NAT implementation with eBPF, testing over two namespaces





XDP with eBPF for different control functions

L2 Forwarding with DPDK

- DPDK (Data Plane Development Kit) is utility program containing data plane libraries and network interface controller driver for fast packet processing.
- Layer2 forwarding for each received packet
- Memory pools and port queues are needed for l2fwding
- Source port and destination port are paired by port mask
- Source and destination MAC address of the received packets are modified to forward to adjacent port.

eBPF experiments

- IOVisor opensource code with eBPF helper functions
- Userspace frontend python program calls BPF system call wrapper functions
- For XDP fast packet drop, the eBPF hook attaches its context to the kernel driver and monitors packet header.
- It drops packets before skbuff allocation
- Source port and destination port are paired by port mask
- For NAT application, it monitors IP header and perform NAT as per control mapping introduced by user

Experiments & Preliminary Results

DPDK l2fwd:

- Connect the traffic generator host with the DPDK host/VM with open switch.
- Install DPDK application on the VM with target as x86_64-native-linuxapp-gcc on ubuntu 16.04
- Setup the device driver virtio or igb_uio and bind the network interfaces
- Allocate huge pages in the system and compile the l2fwd application from dpdk example
- Run the dpdk l2fwd application with command :

```
# /examples/l2fwd/build/l2fwd -c 1 -n 1 — -q 1  -p 1
```

XDP Fast packet drop:

- XDP fast packet drop functionality is tested with Mellanox 40 Gbits/s NIC.
- Sender is running hight speed packetgen application on Ubuntu 16.04 with Linux kernel 4.7
- Receiver machine is running a VM where we execute the XDP code for fast packet drop.
- Test the speed of packet drop using ‘nicstat’
- Got ~13 Mbps packet drop speed with our current setup

NAT functionality with eBPF:

- New NAT functionality is implemented with eBPF using BPF Maps with bcc
- The BPF code reads the IP header and maps it to a destination address which the userspace can see through the BPF map.
- Tested the NAT functionality by creating two namespace containers and NATing from one namespace to another.

Conclusion

- As a third party software DPDK needs licensing and special hardware whereas XDP uses tools from Linux kernel.
- Dedicated CPUs are not required for XDP; also no huge pages is needed.
- XDP is not a kernel bypass, it is a fast path in the kernel stack accessed by the userspace.
- XDP is very flexible in header parsing and packet rewriting and packet steering.
- Fast packet drop by XDP results are currently constrained by hardware limitations
- The NAT functionality is currently tested with containers which can be extended with VMs.

References

[1] https://www.iovisor.org/

[2] http://dpdk.org/

[3] http://events.linuxfoundation.org/sites/events/files/slides/bpf_collabsummit_2015feb20.pdf

[4] Z. Ahmed, M. H. Alizai, Affan. A. Syed. “InKeV: In-kernel Distributed Network Virtualization for DCN”, ACM SIGCOMM 2016.

[5] Fernando Sanchez, Brenden Blanco, “Extended BPF and Data Plane Extensibility: An overview of Networking and Linux”, PLUMgrid Inc.

[6] S. Dutt Sharma, “LTTng’s Trace Filtering and Beyond” , TracingSummit (LinuxCon, Seattle), 2015

[7] Yunsong Lu. “Evolving Virtual Networking with IO Visor”, Huawei Technologies Co., Ltd.

printed by MegaPrint Inc. www.postersession.com