Internship Report on Network Virtualization Project

By
Sabur Hassan Baidya
Mentor: Yan Chen

www.huawei.com

Outline of Activities

- Test Environment Setup for Network Virtualization
- Experiment with Ceth driver
- Test setup and experiment with RancherOS
- eBPF and IO Visor

Test Environment Setup

- Kernel upgrade and install from source
- Creating VM from scratch with Qemu virtualizer
- Creating containers e.g. dockers, namespaces and connect with virtual interfaces
- VM interconnection with Open Vswitch
- Experiment with DPDK

Documentation:

http://swlab-conf.huawei.com:8090/display/CON/Test+Environment+Setup

Experiment with Ceth driver

- Integrated the patch for ceth driver for mellanox cards on Kernel 4.7
- Tested the performance with pktgen

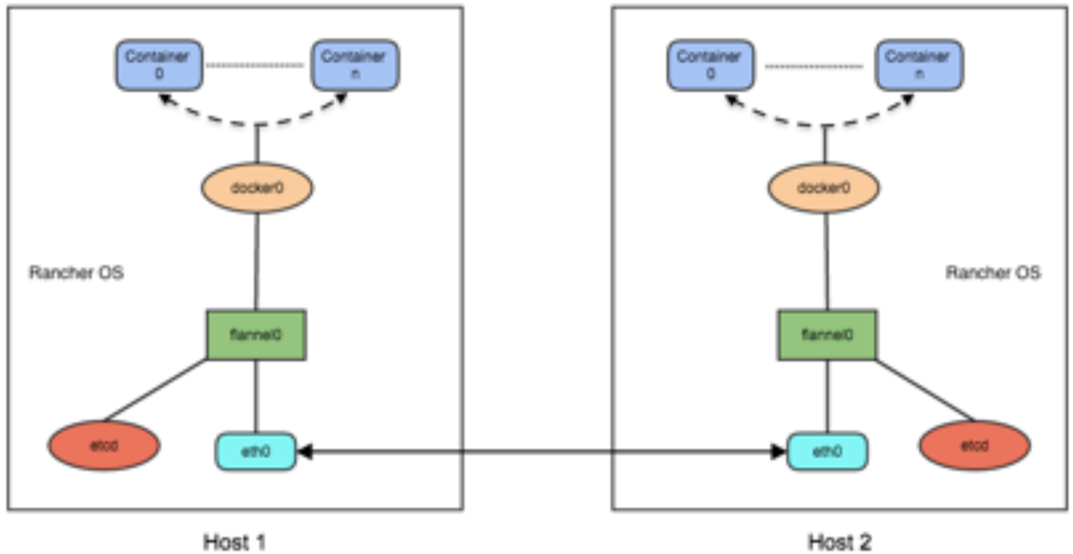
Documentation:

Created a patch for ceth changes on fresh kernel 4.7 and submitted to Yan

Test setup and experiment with RancherOS

- Container OS : docker + system docker
- System docker launches additional system services
- Docker manages all the user containers
- Installed RancherOS on Physical hosts
- Challenges for installation on hosts :
    - Set ssh and static IP in cloud_config.yml while installing

Experiment with flannel & etcd



- Measured performance using netperf
- netclient running on all containers on Host 1
- netserver running on all containers on Host 2

<u>Documentation:</u>

Will be uploaded in the confluence

eBPF and IO Visor

Outline:

- Learning BPF basics
- eBPF features as a part of IO Visor bcc
- Experiment with already implemented XDP code
- Adding Virtual Network functions with eBPF
- eBPF usecase and its implementation

Packet Filtering

- Kernel facility for packet classification based on user defined criteria
- CSPF : CMU/Stanford Packet Filter (1987)
  - Motivation: Implement some network protocols at userspace
  - Uses a boolean expression Tree Model
- BPF : Berkeley Packet Filter (1993)
  - Motivation: High speed network monitoring
  - Uses directed acyclic Control Flow Graph (CFG)
- FFPF : Fairly Fast Packet Filter (2004)
- Swift : Fast Dynamic Packet Filter (2008)

References

[1] J. C. Mogul, R. F. Rashid and M. J. Accetta. “The Packet filter: An efficient mechanism for user-level network code”. Proceedings of 11th ACM SOSP, pages 39-51, 1987

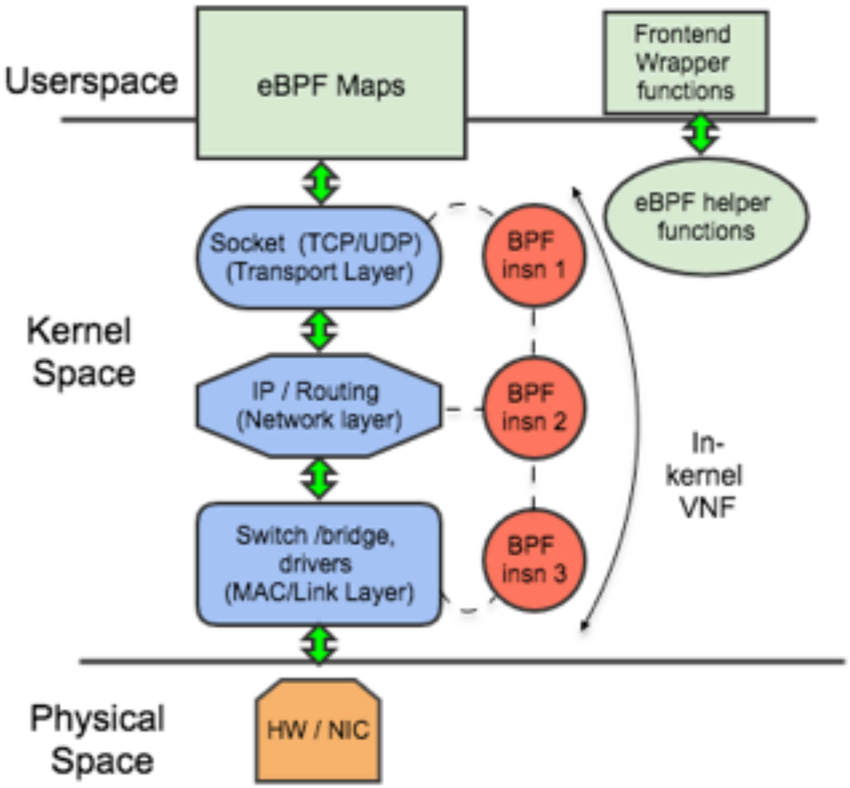
[2] S. McCanne and V. Jacobson. “The BSD packet filter: A new architecture for user-level packet capture”. Proceedings of 1993 Winter USENIX Technical Conference, 1993

[3] H. Bos, W. de Bruijn, M. Criesta, T. Nguyen and G. Portokalidis. “FFPF: Fairly fast packet filters”. Proceedings of USENIX OSDI’04, 2004

[4] Z. Wu, M. Xie, H. Wang. “Swift: A Fast Dynamic Packet Filter”. NDSI’08: 5th USENIX Symposium on Networked Systems Design and Implementation, 2008

Berkeley Packet Filter (BPF)

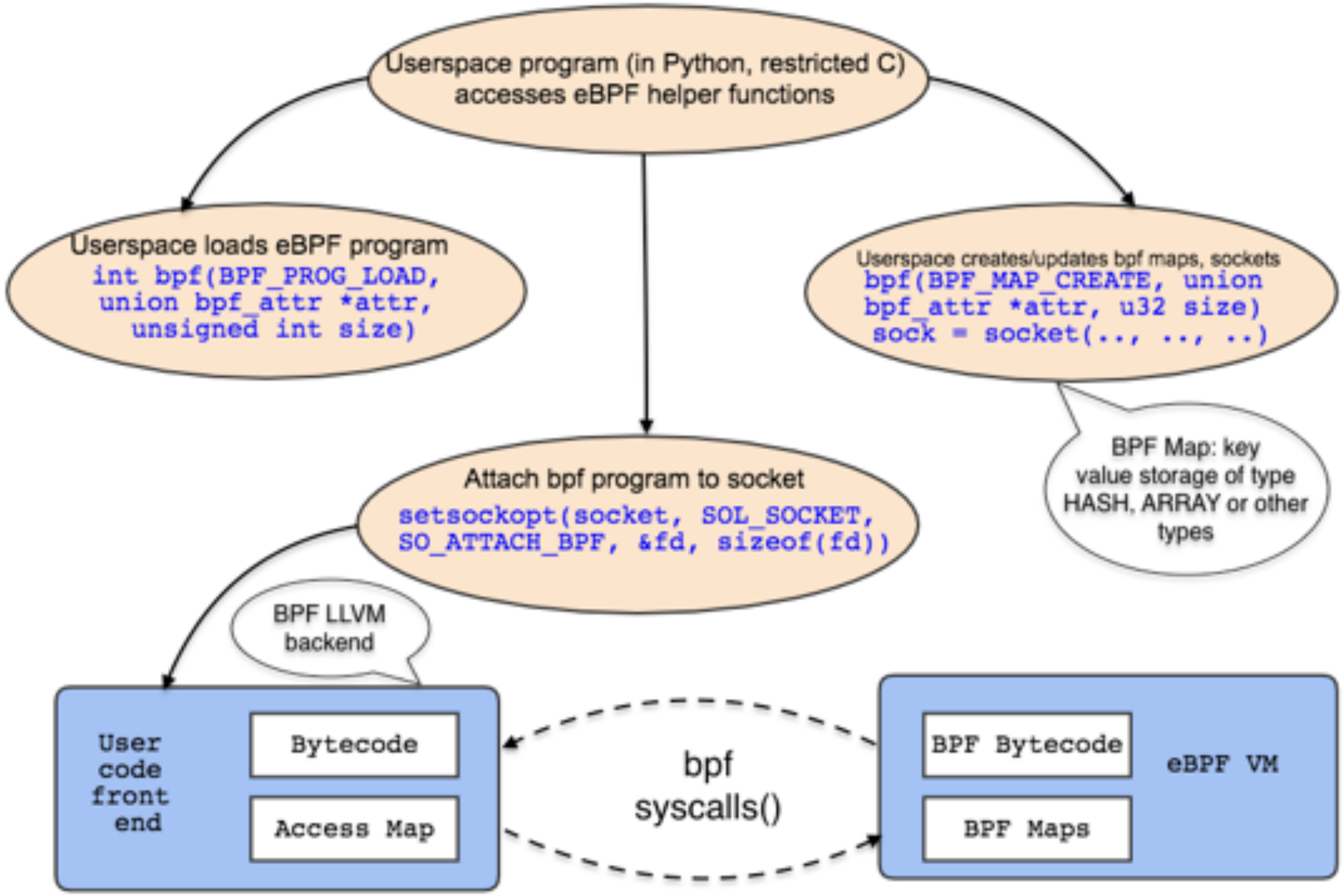
- In-kernel Virtual Machine for packet & syscall filtering
- BPF => Classic BPF (CBPF)
- Introduced in Linux Kernel 2.1.75 (1997)
- Two 32 bit registers, 32-bit slots stack
- Supports full Integer arithmetic
- Supports conditional jump, forward
- Can be hooked to different layer of kernel protocol stack



Extended BPF or eBPF

- Introduced in kernel 3.8 and above
- Enhanced 64 bit register, 10 registers, stack
- Supports instruction call, load, store, conditional jump
- eBPF Maps: Share data between userspace and Kernel space and also among eBPF programs
- Helpers for lookup, update, delete maps
- Fast interpretation and in-kernel JIT compilation (2011)
- bcc : bpf compiler collection (IO Visor project)
- LLVM backend, clang frontend

eBPF Program Flow



eBPF: Much more than tracing

- Initial motivation : socket filtering (tcpdump/libpcap)
- seccomp for syscall filtering
- Packet Classifier for Traffic Control (TC)
- Kernel Probe (kprobe) tracing
- Packet Processing (parse, update, modify)
- Hardware modelling , disk latency monitoring
- IO Visor tools for Networking, security, tracing
- One major focus of IO Visor: Programmable Data Plane + high speed data path in the kernel

High Speed Datapath : L2 forwarding

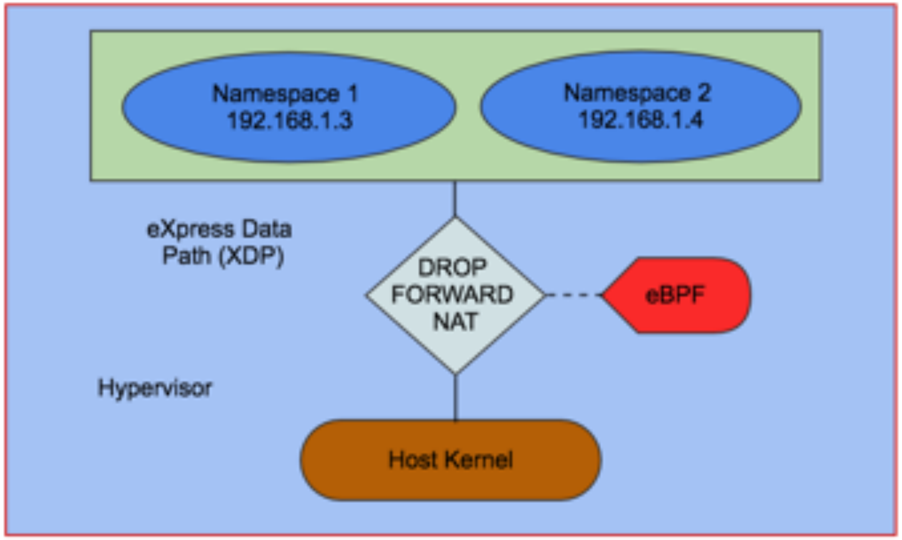
- Networking at userspace: High performance frameworks
- Kernel Bypass for Datapath
- Netmap : Luigi Rizzo and Matteo Landi, Università di Pisa (BSD License)
  - Memory mapped access to network devices
- Openonload : From Solarflare (Open Source, GPL)
  - Ethernet packets can be processed in application context based on flow information in header
- DPDK : From Intel (BSD License)
  - Data Plane Development Kit
  - utility program containing data plane libraries and network interface controller driver for fast packet processing

eXpress DataPath : XDP

- 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
- XDP aims at replacing the OVS data path with eXpress Data Path
- XDP can perform different functionalities e.g. packet drop, packet forward or NAT

eXpress DataPath : XDP

- For XDP, the eBPF hook attaches its context to the kernel driver and monitors packet header
- XDP fast packet drop functionality is tested with Mellanox 40 Gbits/s NIC.



- Sender is running hight speed pktgen 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

eBPF Switch / NAT

- Performs the switching/NAT functionality.
- The bpf program is hooked at the qdisc traffic classifier.
- The userspace accesses the bpf map and defines the switching rules by updating the shared hash table.
- The hash table is read in the bpf kernel space and it modifies the outgoing interface based on the map.

Example:

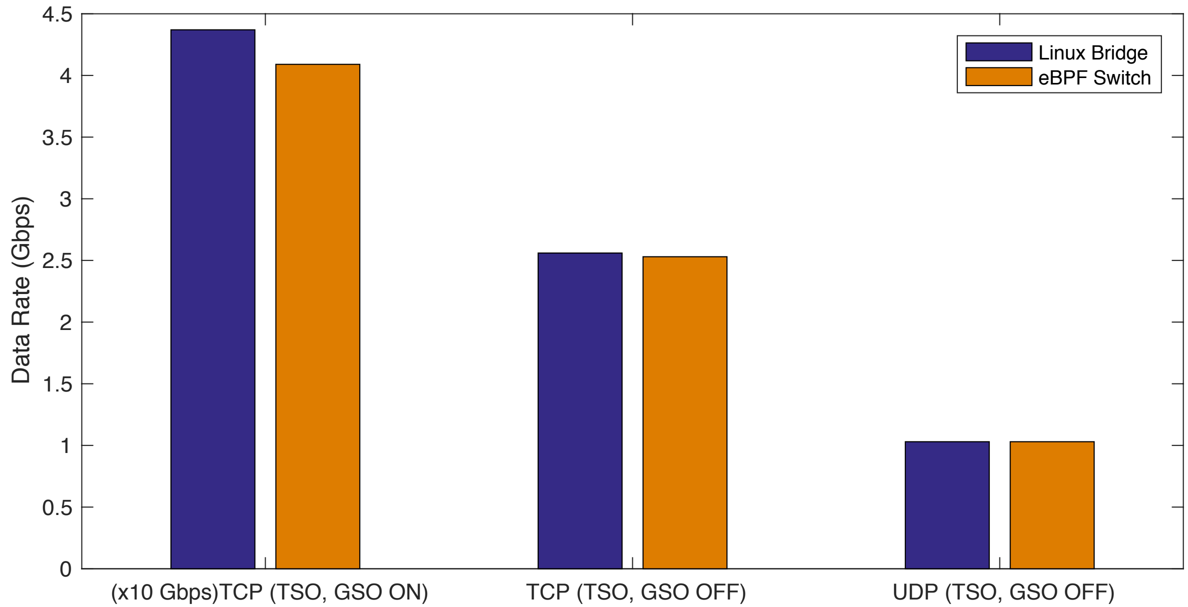
```
# python test_ebpf.py enp3s0 virbr0

Interface In   MAC address In     IP address In    Interface Out  MAC address Out    IP address Out
-------------  ---------------    -------------    -------------  ---------------    --------------

enp3s0       18:03:73:d4:4d:52   10.145.240.201     virbr0     fe:54:00:7e:a3:41   192.168.122.1

virbr0       fe:54:00:7e:a3:41   192.168.122.1      enp3s0     18:03:73:d4:4d:52  10.145.240.201
```

eBPF Switch vs Linux Bridge



Data rate measurement and comparison on Linux bridge and eBPF switch

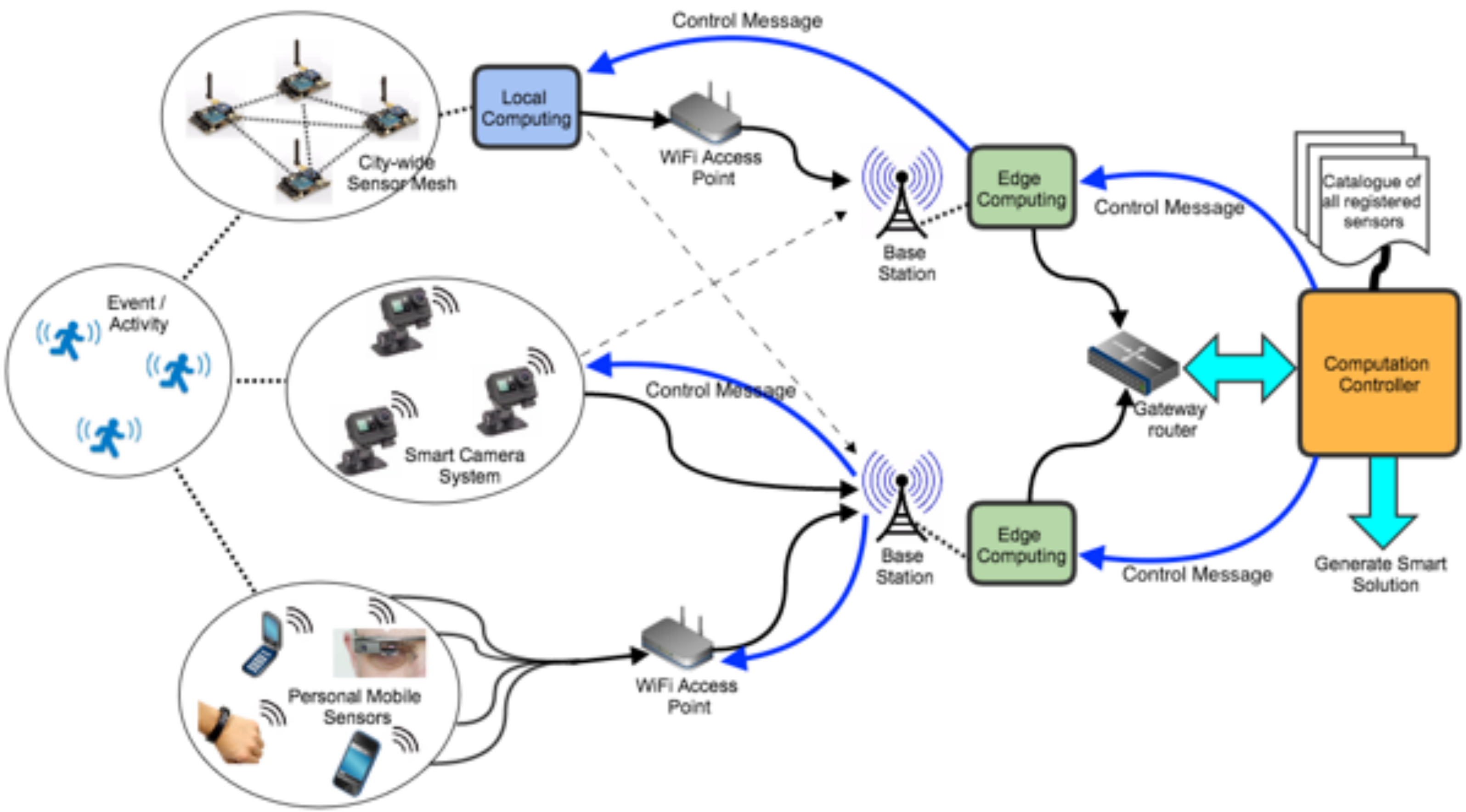
<u>CPU Utilization:</u> Linux bridge : 0.7% - 1% eBPF switch : 0.7 - 0.8%

- Speed of eBPF switch can be improved by hooking it in lower level of kernel
- Also eBPF offers dynamic programability from the user control which Linux bridge cannot

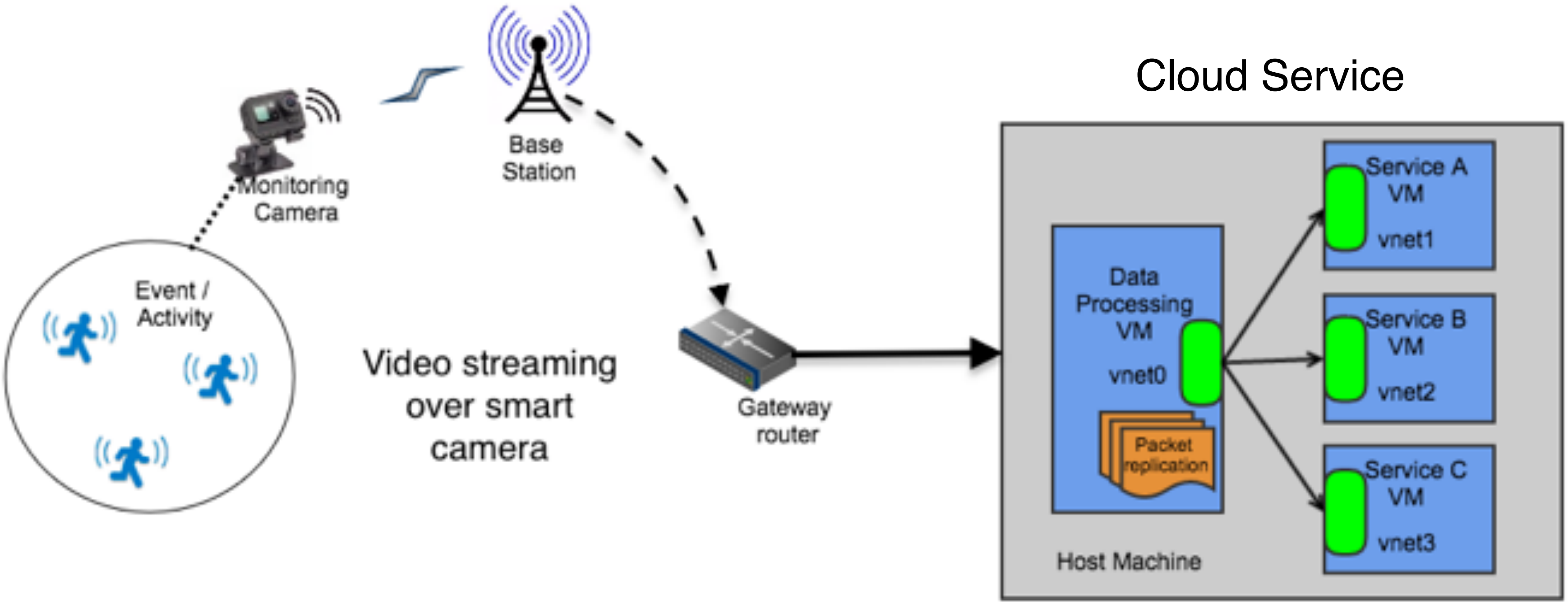
Use case of eBPF

- Take advantage of in-kernel VM
  - suitable for real-time applications
- Programmability from userspace frontend
  - suitable for dynamic QoS requirement
- Usecase: Quality Varying Parallel Video Streaming for Smart City Applications

Real-time Smart City Applications

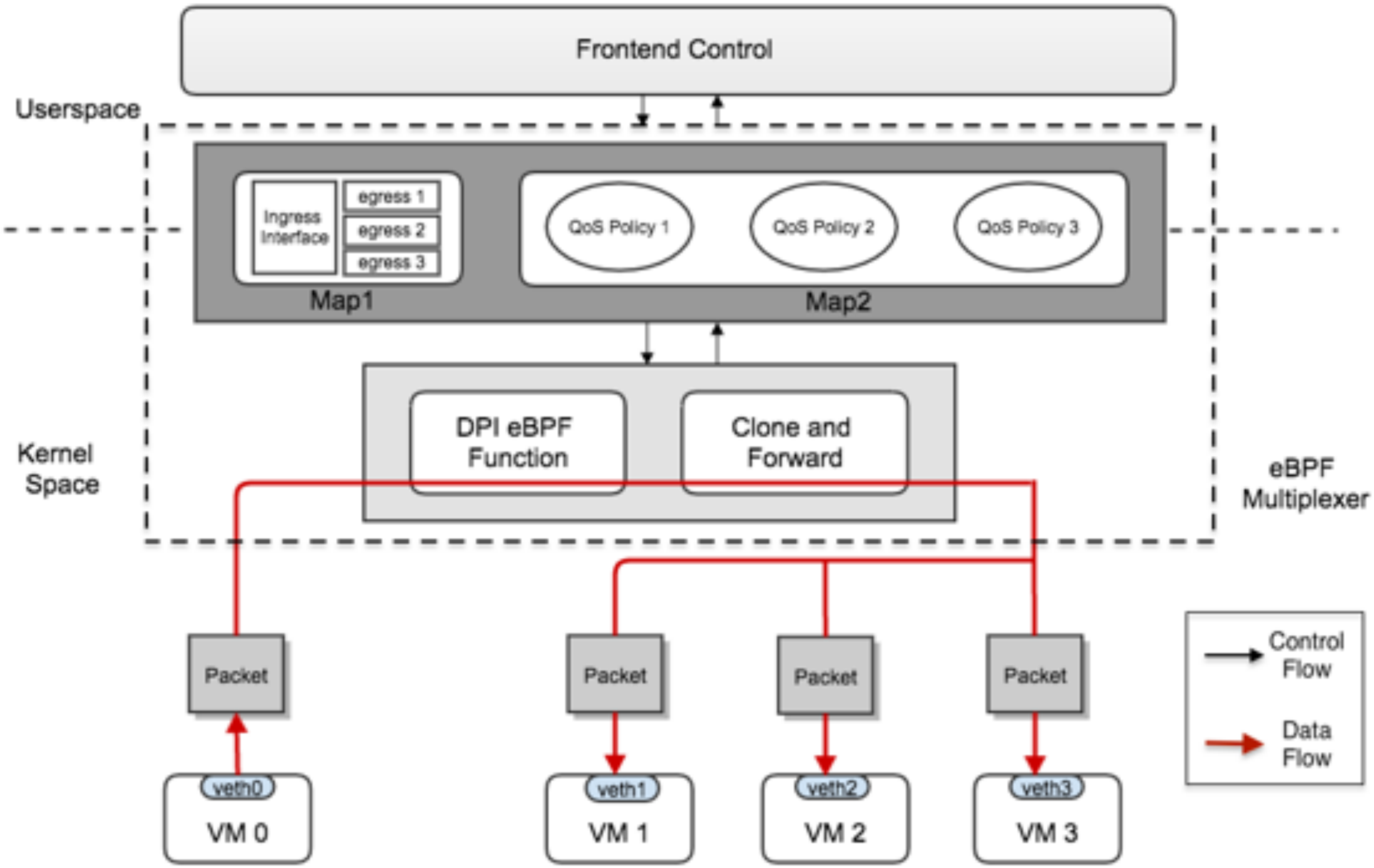


Programmable Multi-streaming Framework for real-time Smart City Applications



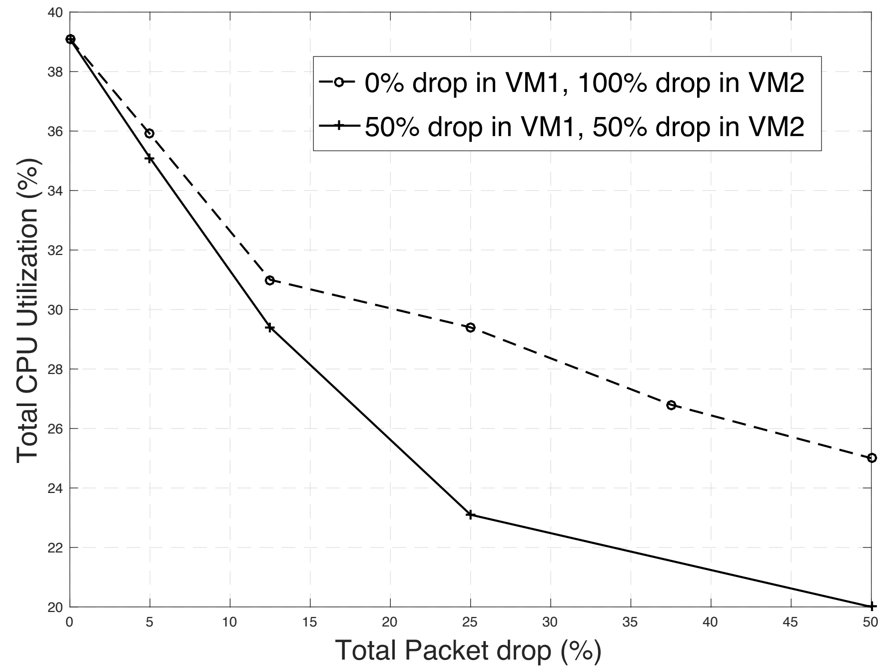
One video streaming source used for multiple smart city services

Clone-n-Cast Framework for Multi-streaming



Clone-n-Cast Performance on VMs

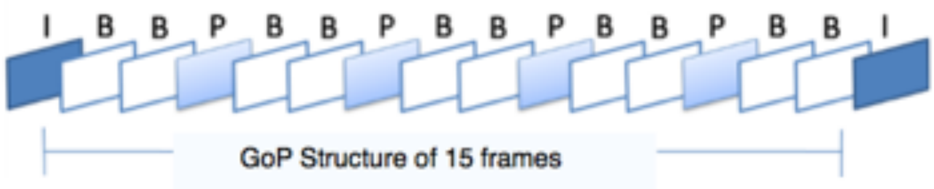
- Programmability
- Resource Utilization
- Scalability
- Security
- Fault Tolerance

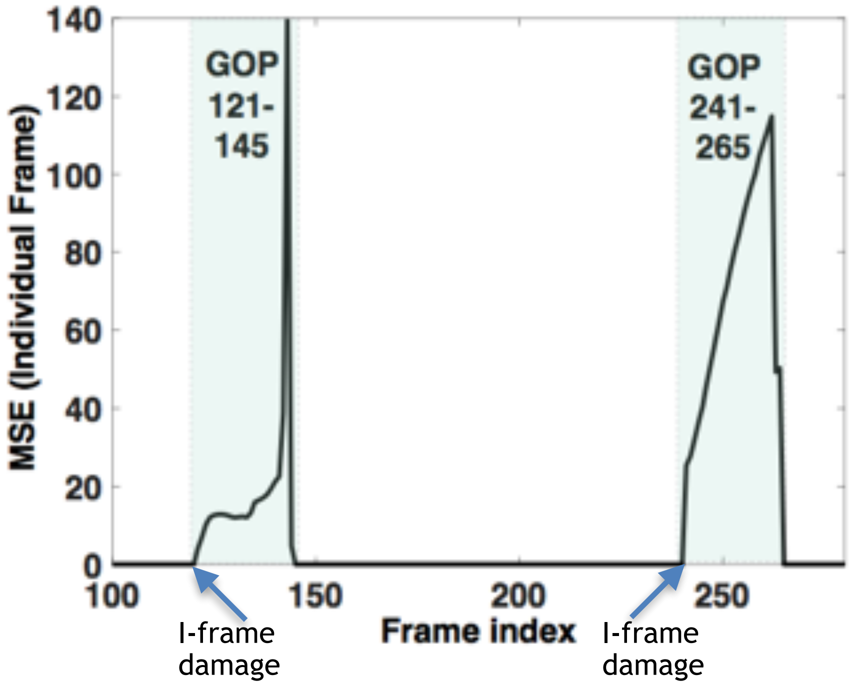


Advantages over - broadcast / Multicast
- DPI at userspace

Video Streaming

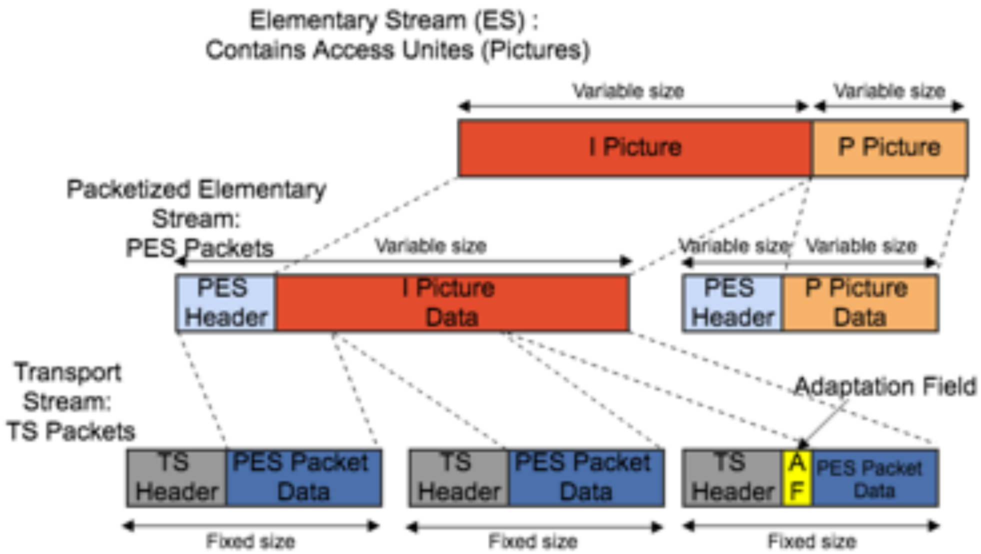
- **Video Compression:**
  - **Spatial Compression:**
    - Discrete Cosine Transform
  - **Temporal Compression:**
    - GoP : Group of Pictures
    - Reference Frame / Intra-coded frame (I-frames)
    - Differentially encoded / Inter-coded frames e.g. Predicted frames (P-frames), Bi-directional predicted frames (B-frames)

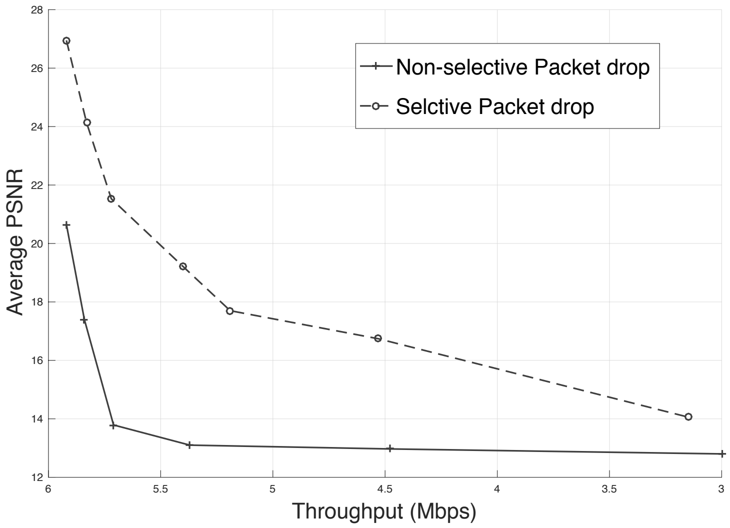




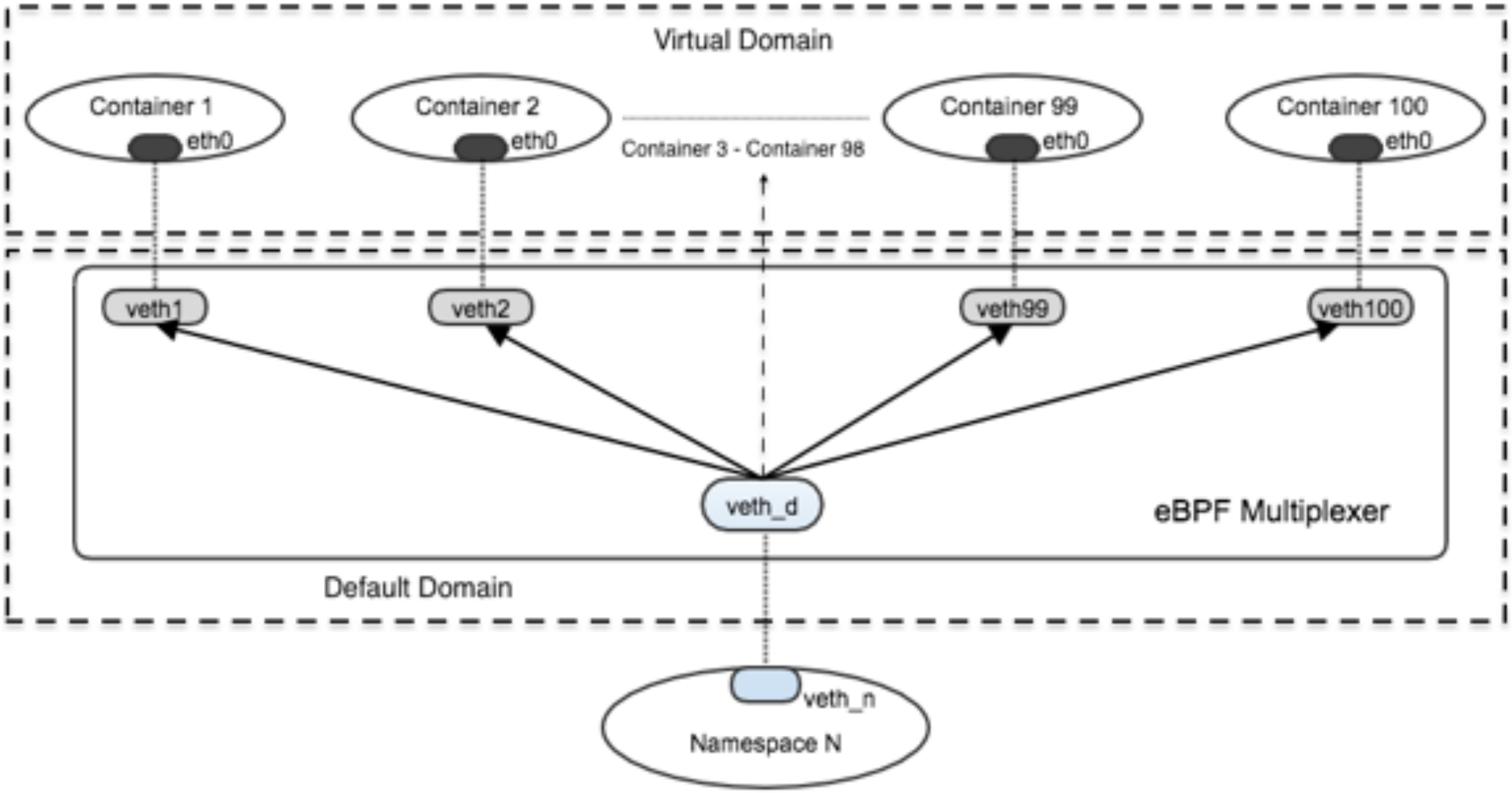
Clone-n-Cast Performance on Video

Transport stream (TS) packets are encapsulated into UDP/TCP packets



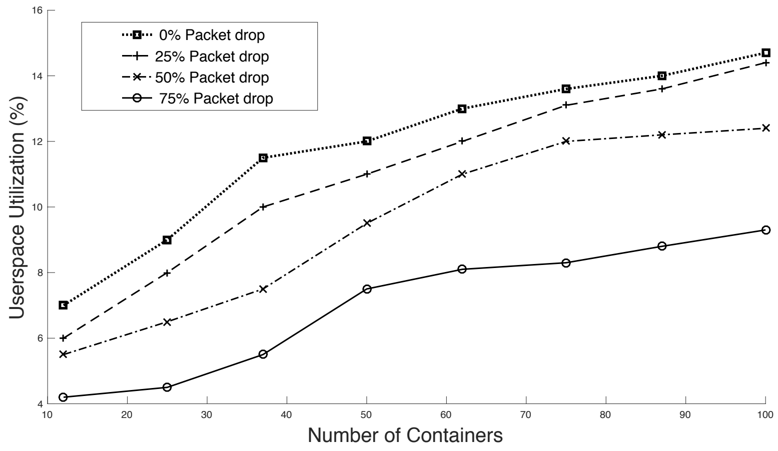


Clone-n-Cast Scalability



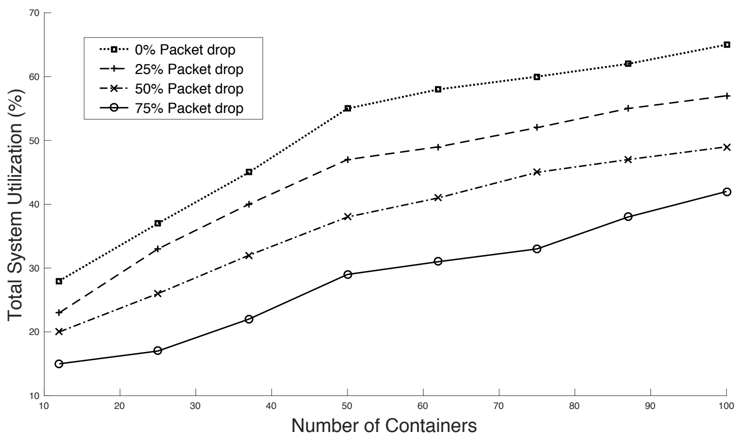
- Challenges in setting up the virtual interfaces in docker containers externally

Clone-n-Cast Scalability Performance

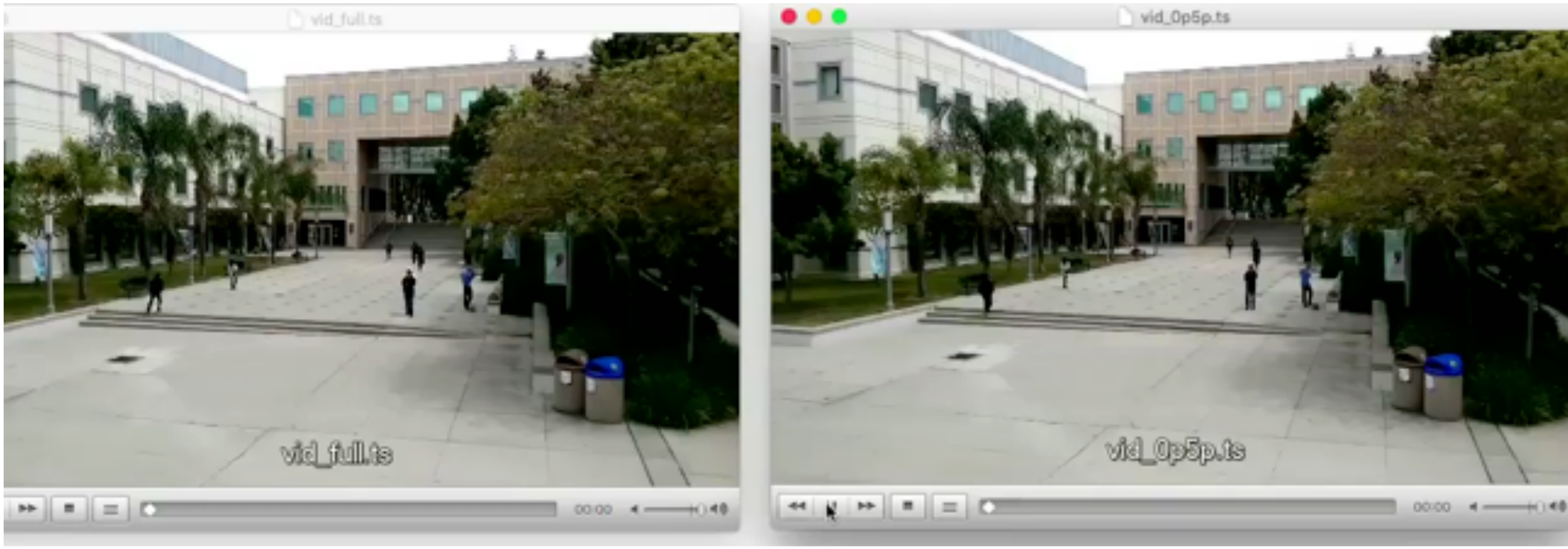


- Userspace utilization for all the running containers

- Total system utilization for all the running containers including software and hardware interrupts



Clone-n-Cast Framework for Multi-streaming



- Comparison of two videos

: left - with all packets

: right - with 5% packet loss

Limitations and Future Work

- Challenges with size of BPF instruction set
- Variable payload size may cause problems
- Different packet loss patterns can be explored
- Add intelligence to the clone-n-cast for dynamic QoS requirement
- Github links for code:
- https://github.com/saburhb/ebpf-switch
- https://github.com/saburhb/ClonenCast

Acknowledgement

Special Thanks

to

Yan

THANK YOU

THANK YOU