vNetTracer: Efficient and Programmable Packet Tracing in Virtualized Networks

Kun Suo[*], Yong Zhao[*], Wei Chen[^], Jia Rao[*]

University of Texas at Arlington[*], University of Colorado, Colorado Springs[^]

UNIVERSITY OF TEXAS ARLINGTON

UCCS University of Colorado Colorado Springs

ICDCS 2018

38th IEEE International Conference on Distributed Computing Systems, July 2 – 5, 2018, Vienna, Austria

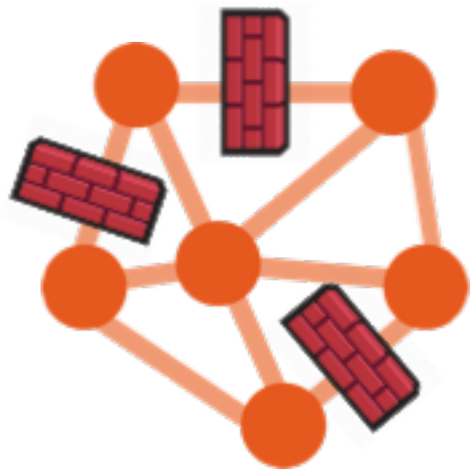
The Rise of Virtualized Networks

- Virtualized networks (VN) are becoming increasingly important to on-demand, elastic, and cost-effective cloud services

- Key benefits of virtualized networks
  - ✓ rapid deployment
  - ✓ ease of management
  - ✓ programmability

- Examples of VN: ***Software defined network (SDN), network function virtualization (NFV)***
  - ✓ Microsoft shows virtualized networks can improve network utilization while offering better quality-of-service (QoS) guarantees

Challenges in Monitoring Virtualized Networks



Virtualized networks usually span multiple protected domains, e.g. host OS or hypervisor, virtual network devices

The performance of virtualized networks is sensitive to tracing overhead

The complexity of virtualized networks requires real time, reconfigurable and programmable tracing for performance diagnosis

Related Work

- **Monitoring based on system logs**
  - ✓ Manual or static instrumentations: Ftrace, Perf, Blackbox debug [SOSP-03], Sherlock [SIGCOMM-07], Wap5 [WWW-06], Spectroscope [NSDI-11], etc.
  - ✓ Machine learning-based log analysis: mystery machine [OSDI-14], DISTALYZER [NSDI-12], Soroban [HotCloud-15], OtterTune [SIGMOD-17], Xu Detecting [SOSP-09], etc.

✗ High overhead

✗ Hard to meet diverse user requirements

- **Dynamic instrumentation**
  - ✓ DTrace, SystemTap, and DARC [SIGMETRICS-08], Pip [NSDI-06] , Pinpoint [DSN-02], X-Trace [NSDI-07], etc.

✗ Require changes To applications

✗ Only effective within a protected domain

- **Tracing in distributed systems**
  - ✓ Pivot tracing [SOSP-15], Appinsight [OSDI-12], Timecard [SOSP-13], etc.

✗ Inflexible

Our Approach: vNetTracer

- **What is vNetTracer?**
  - ✓ An efficient and programmable packet profiler based on eBPF
- **Features of vNetTracer**
  - ✓ End-to-end tracing across boundaries of separate, protected domains
  - ✓ Negligible runtime overhead in highly consolidated and optimized virtualized networks
  - ✓ Rich performance monitoring metrics and customized network packet tracing

extended Berkeley Packet Filter (eBPF)

What is BPF?

- The Berkeley Packet Filter (BPF) provides a raw interface in the kernel, permitting raw link-layer packets to be sent and received.

eg. Ubuntu:

- 3.18: bpf syscall
- 3.19: sockets
- 4.1: kprobes
- 4.4: bpf_perf_event_output — 16.04
- 4.6: stack traces
- 4.7: tracepoints — 16.10
- 4.9: profiling

What is eBPF?

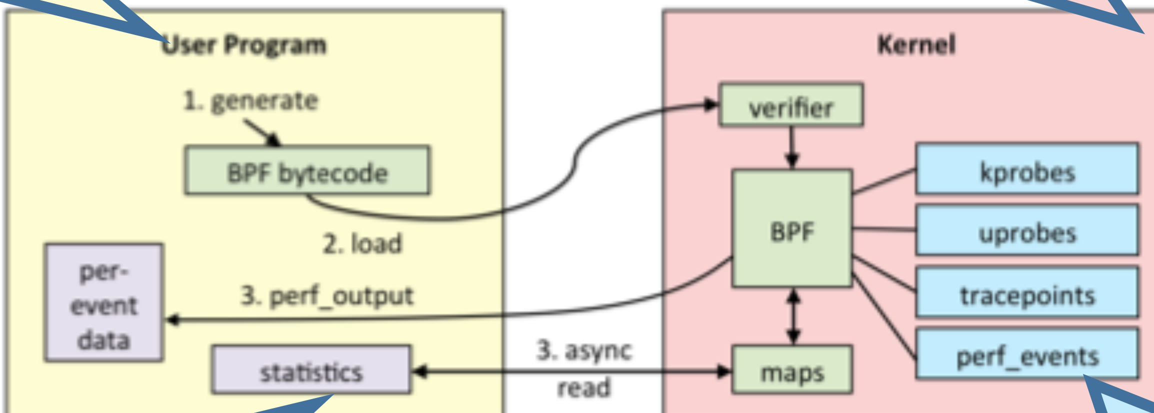
- eBPF is extended BPF.
- eBPF has raw tracing capabilities similar to those of DTrace and SystemTap.

classic BPF	extended BPF
2 registers + stack	**10 registers** + stack
32-bit registers	**64-bit registers** with 32-bit sub-registers
4-byte load/store to stack	1-8 byte load/store to stack, maps, context
1-4 byte load from packet	Same + store to packet
Conditional jump forward	Conditional jump forward and backward
+, -, *, … instructions	Same + signed_shift + endian
	Call instruction
	tail_call
	map lookup/update/delete helpers
	packet rewrite, csum, clone_redirect
	sk_buff read/write

extended Berkeley Packet Filter (eBPF)

eBPF allows programmers to attach user-defined programs into the kernel

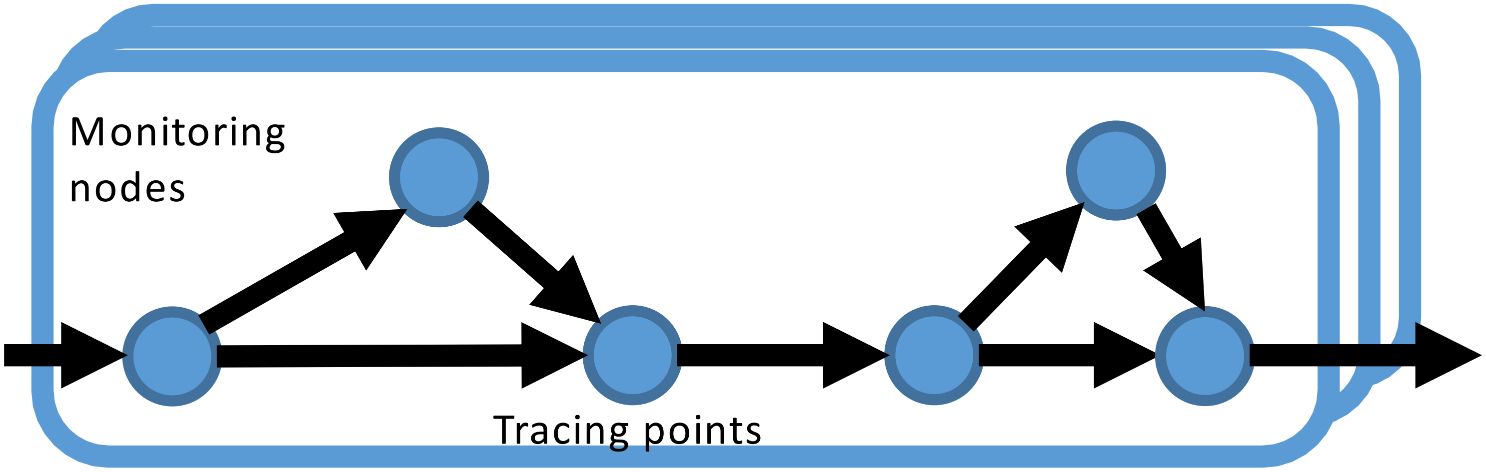
Without adding inflexible and dull log inside the systems



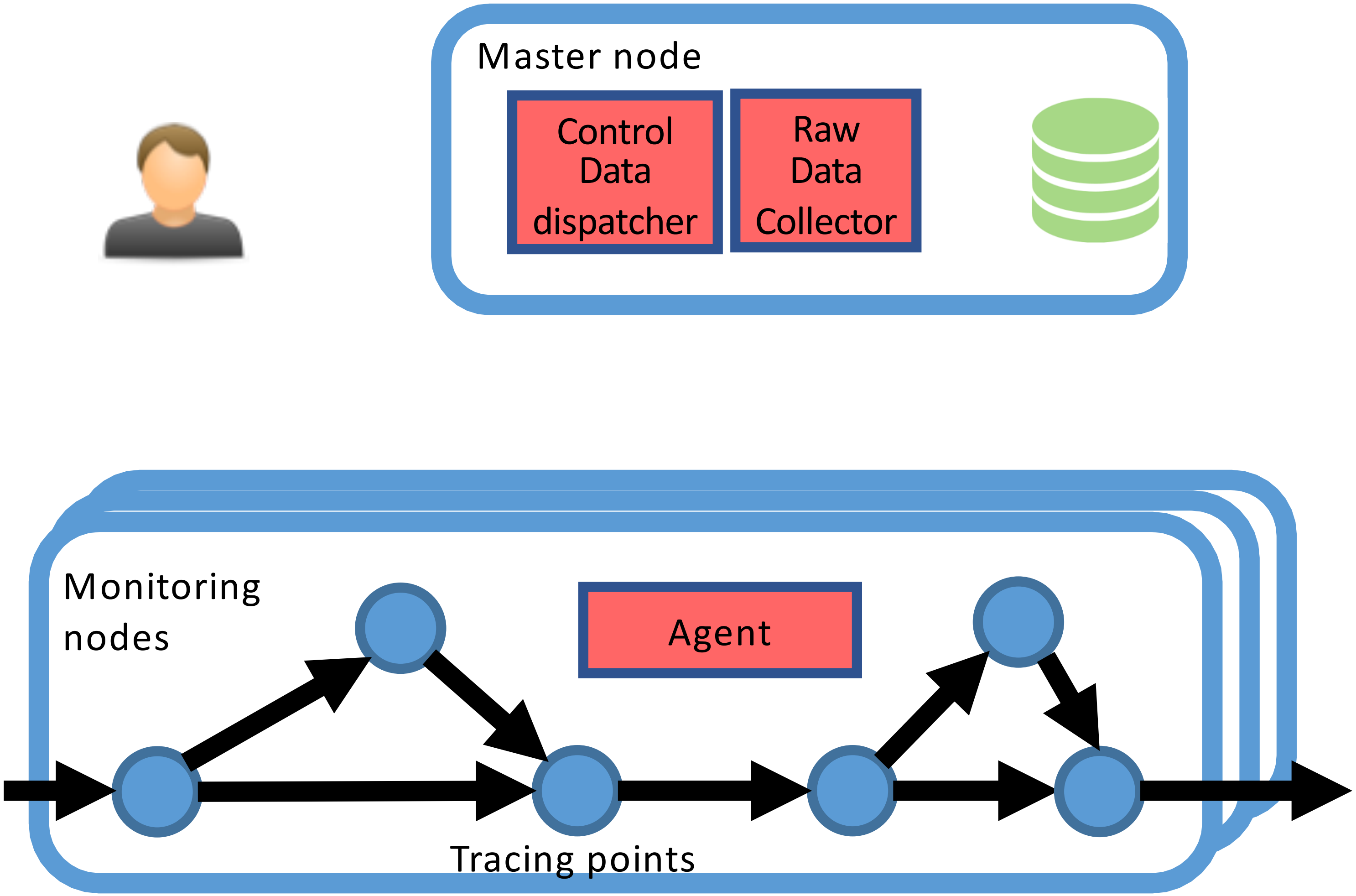
Monitoring data can be either temporarily stored in the eBPF data structures inside kernel or collected asynchronously from user space

eBPF enables users to trace high frequency events, such as context switches or packet processing

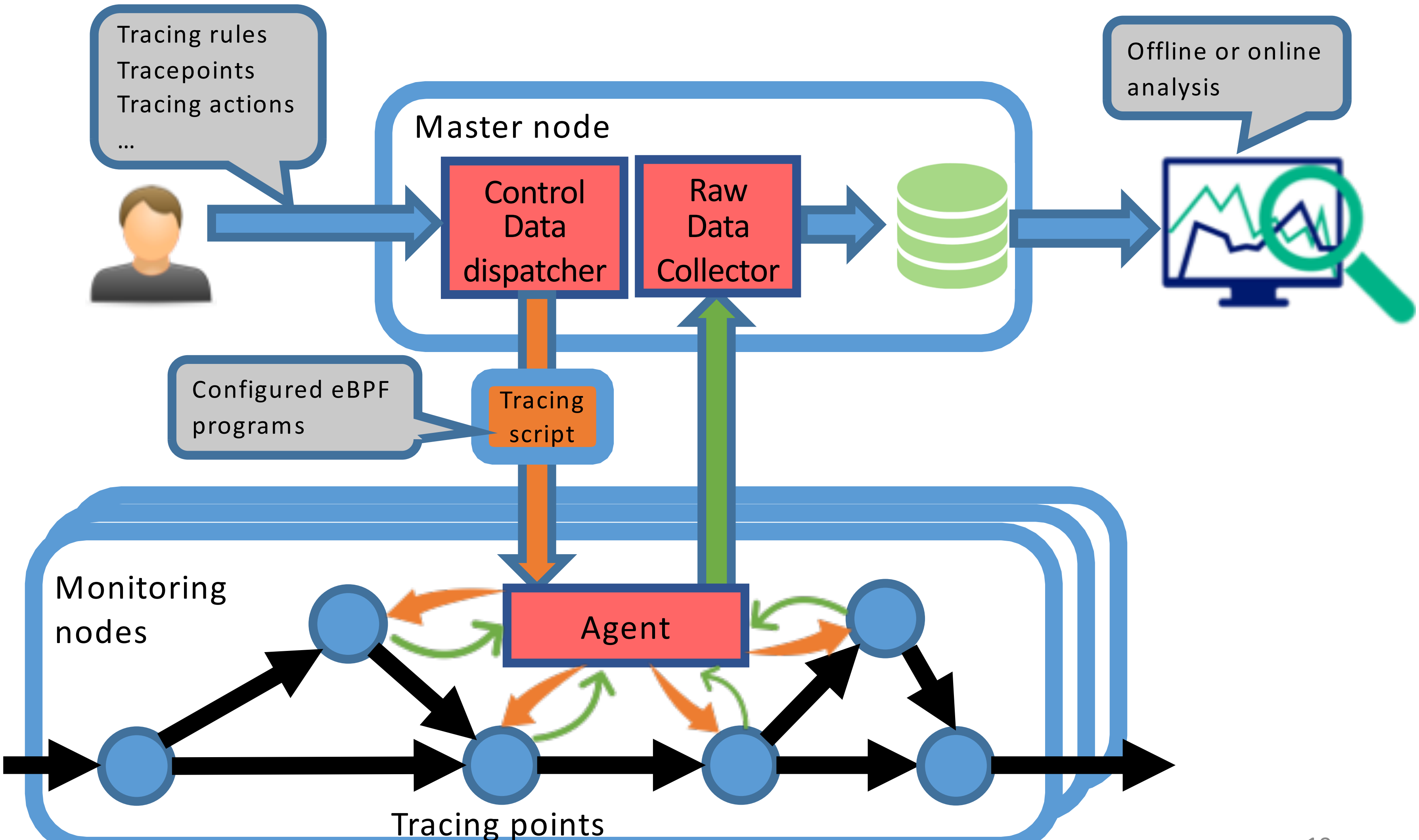
vNetTracer Overview



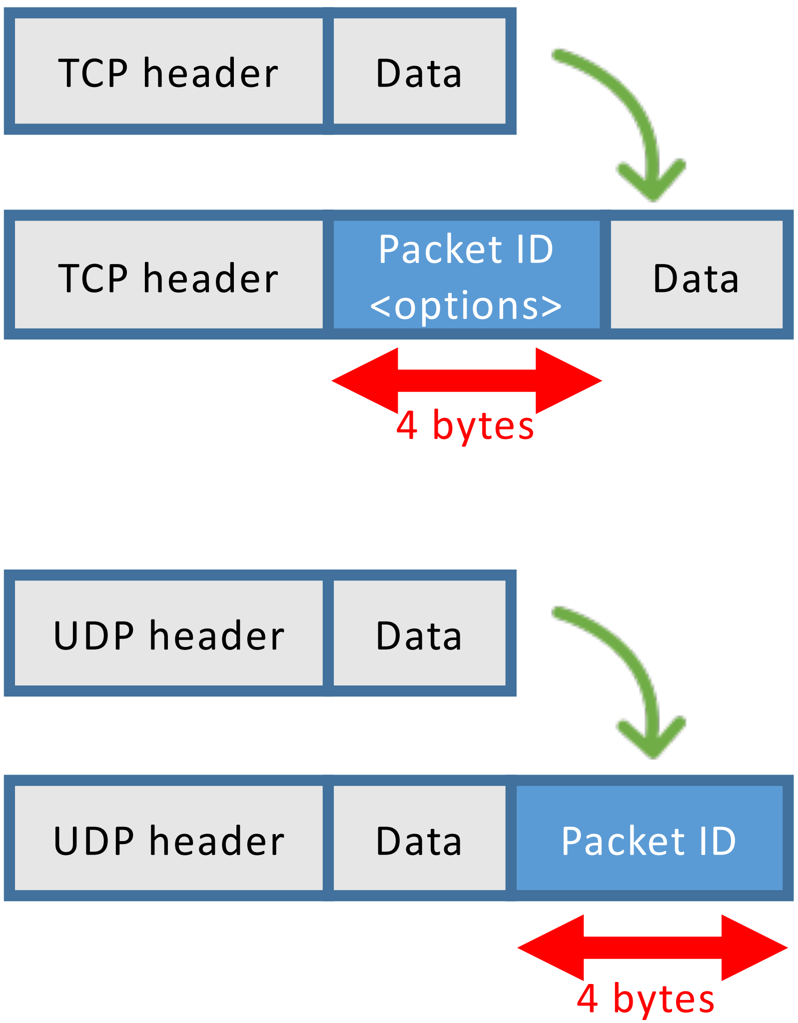
vNetTracer Overview

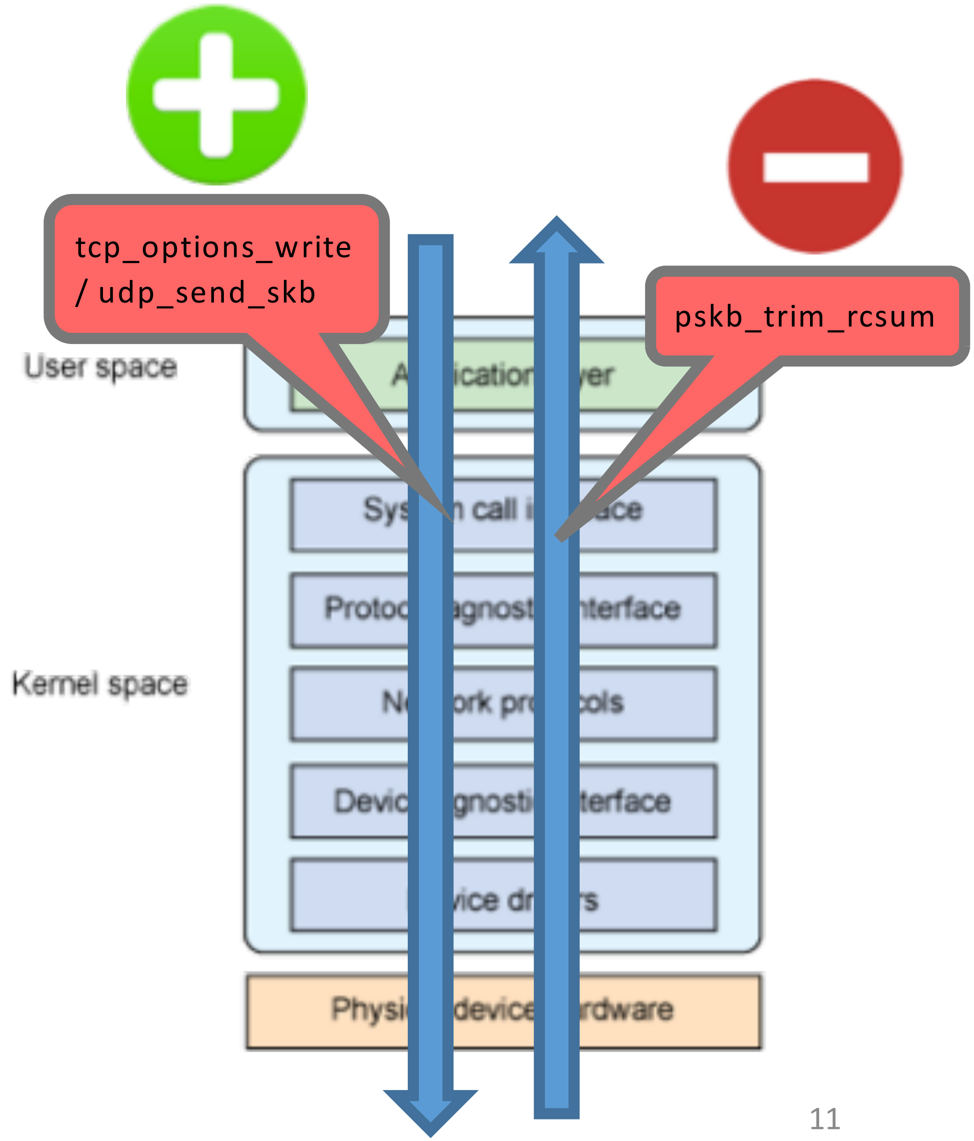


vNetTracer Overview

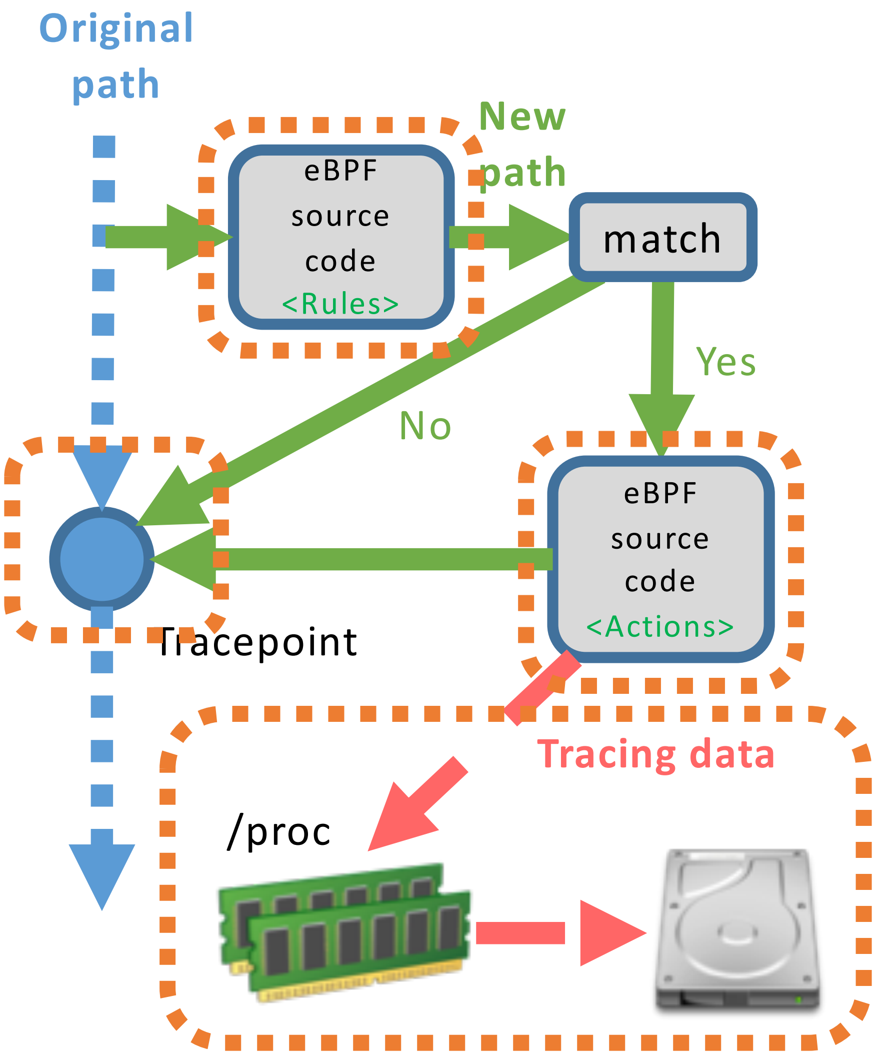


Tracing across Boundaries





Runtime Efficiency

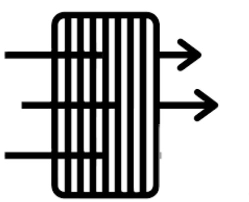


- The position of tracepoints, rules and actions are defined by users through configuration files
- We used mmap() to map a kernel buffer to the /proc file system in user space
- When the tracing scripts generate some intermediate data, it is first copied to the memory buffer
- It periodically dumps the tracing data from the buffer onto disk, clears the buffer and then collects the raw tracing data to a centralized data processing node

Programmability

- Traced items: packet id, packet length, timestamp, device name, etc.

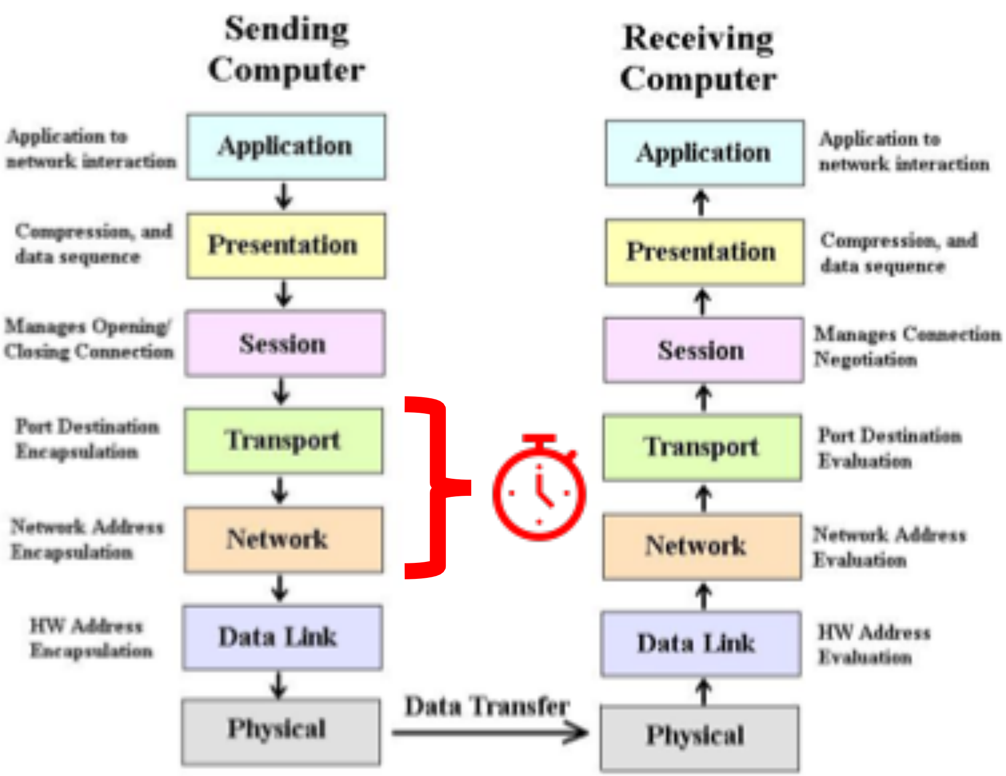
- Basic performance metrics: throughput, latency, packet loss rate, jitter



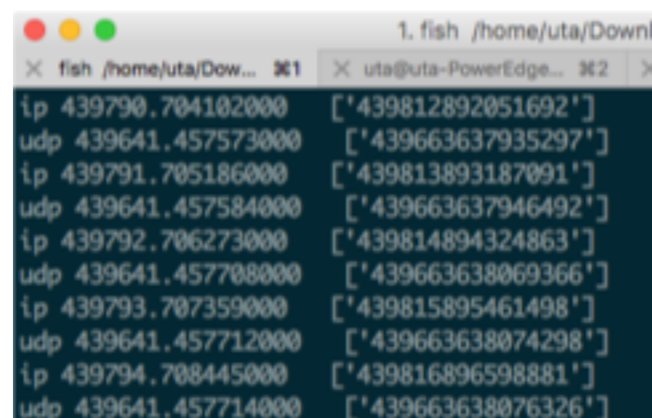
- Advanced metrics: per-flow throughput, the decomposition of end-to-end latency, etc.

- vNetTracer allows users to reconfigure tracing at runtime

Example: Trace Time between Network Stack Layer 3 and Layer 4



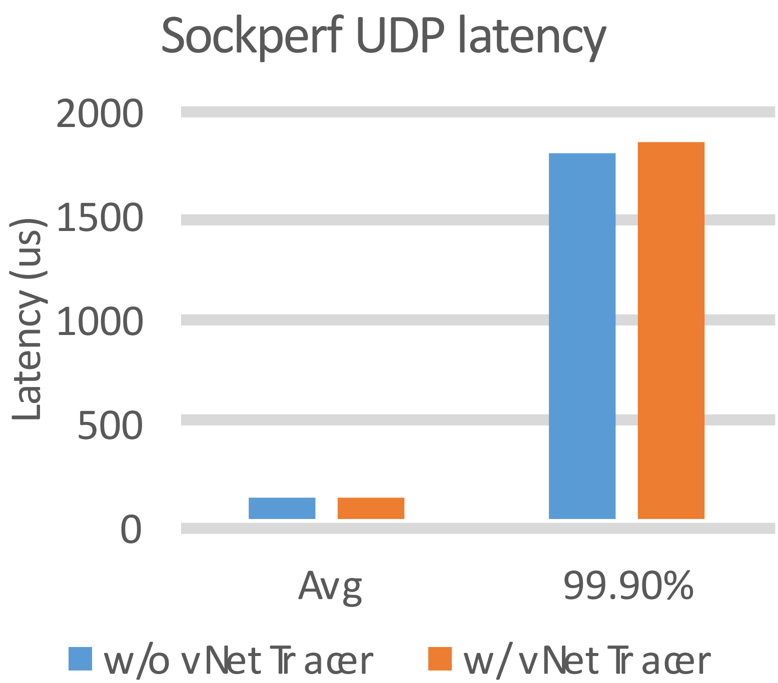

```
from __future__ import print_function
from bcc import BPF

bpf_text = """
#include <uapi/linux/ptrace.h>
#include <bcc/proto.h>

void trace_start(struct pt_regs *ctx) {
        u64 ts = bpf_ktime_get_ns();
        bpf_trace_printk("%llu\\n", ts);
}
"""

# load BPF program
b = BPF(text=bpf_text)
b.attach_kprobe(event="udp_send_skb", fn_name="trace_start")

# header
print("%-18s %-8s" % ("TIME(s)", "get time(ns)"))

# format output
while 1:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
        (time)=msg.split()
        print("udp %-18.9f %s" % (ts, time))
```

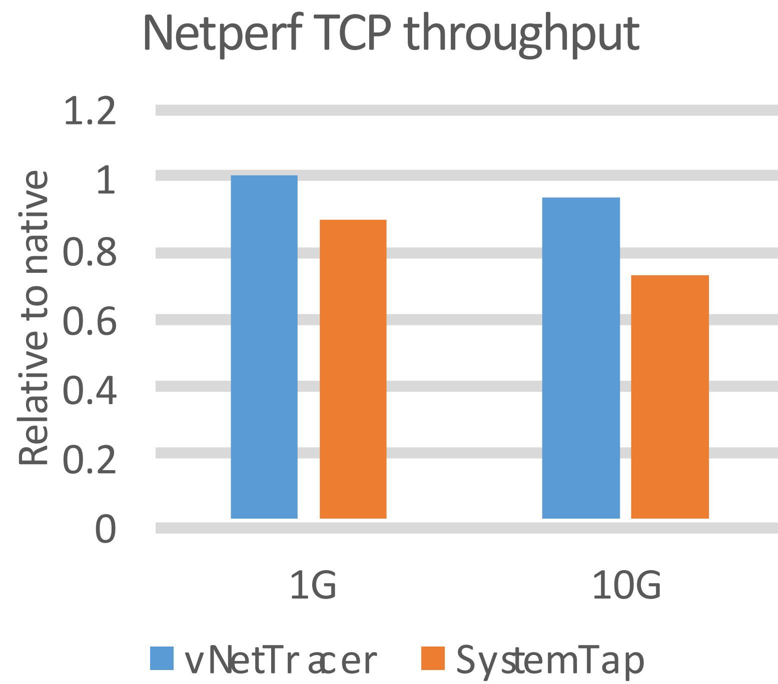


Evaluation Settings

- **Hardware**
  - ✓ Two DELL PowerEdge T430 servers equipped with dual ten-core Intel Xeon E5-2640 2.6GHz processors
  - ✓ 64GB memory and a 2TB 7200RPM SATA hard disk
  - ✓ NICs: 1Gbps Ethernet and 10Gbps Ethernet
- **Software**
  - ✓ Ubuntu 16.10 and Linux kernel 4.10 as the host and the guest OS
  - ✓ Open vSwitch 2.6.0 to connect various VMs on the same host
  - ✓ Hypervisor: KVM 2.6.1 or Xen 4.8.1
  - ✓ Container runtime: Docker 1.12.1
- **Benchmarks**
  - ✓ Netperf, sockperf, iPerf, Cloudsuite benchmark 3.0

Overhead of vNetTracer





Four tracing scripts attached to OVS port ovs-br1 in the hypervisor and virtual ethernet port ens3 in the VM on the two physical servers

The performance of vNetTracer with SystemTap tracing tcp_recvmsg

Case Studies of vNetTracers

- Case Study I

  Network Delay in the Open vSwitch

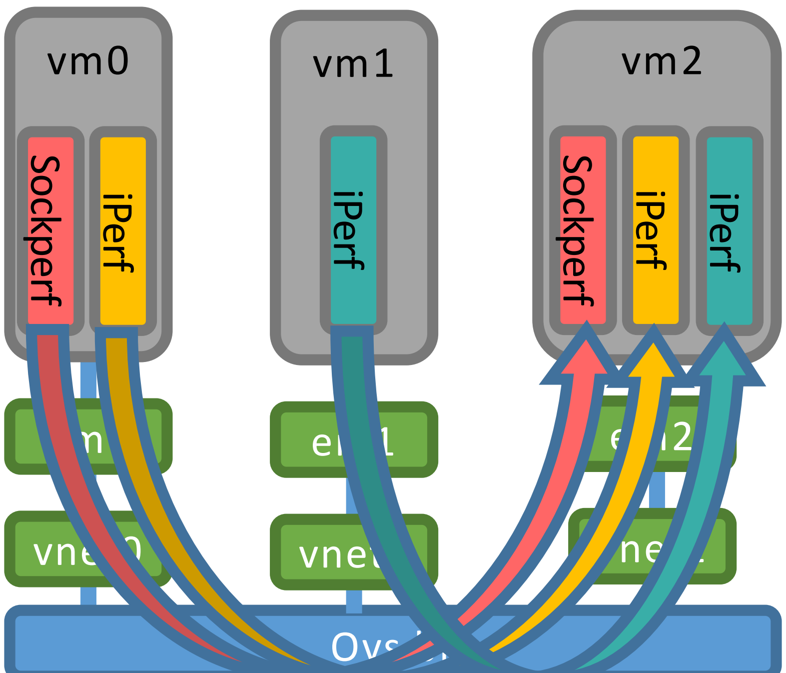
- Case Study II

  Tuning the Scheduler in Hypervisors

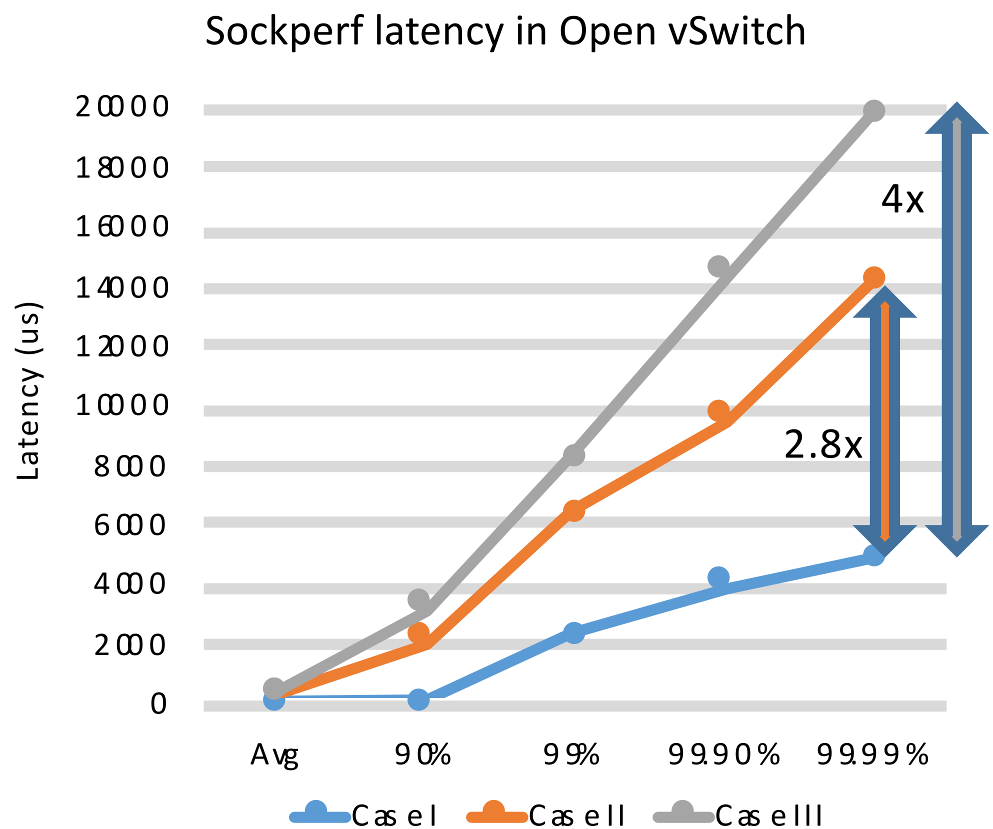
- Case Study III

  Bottlenecks in Container Networks

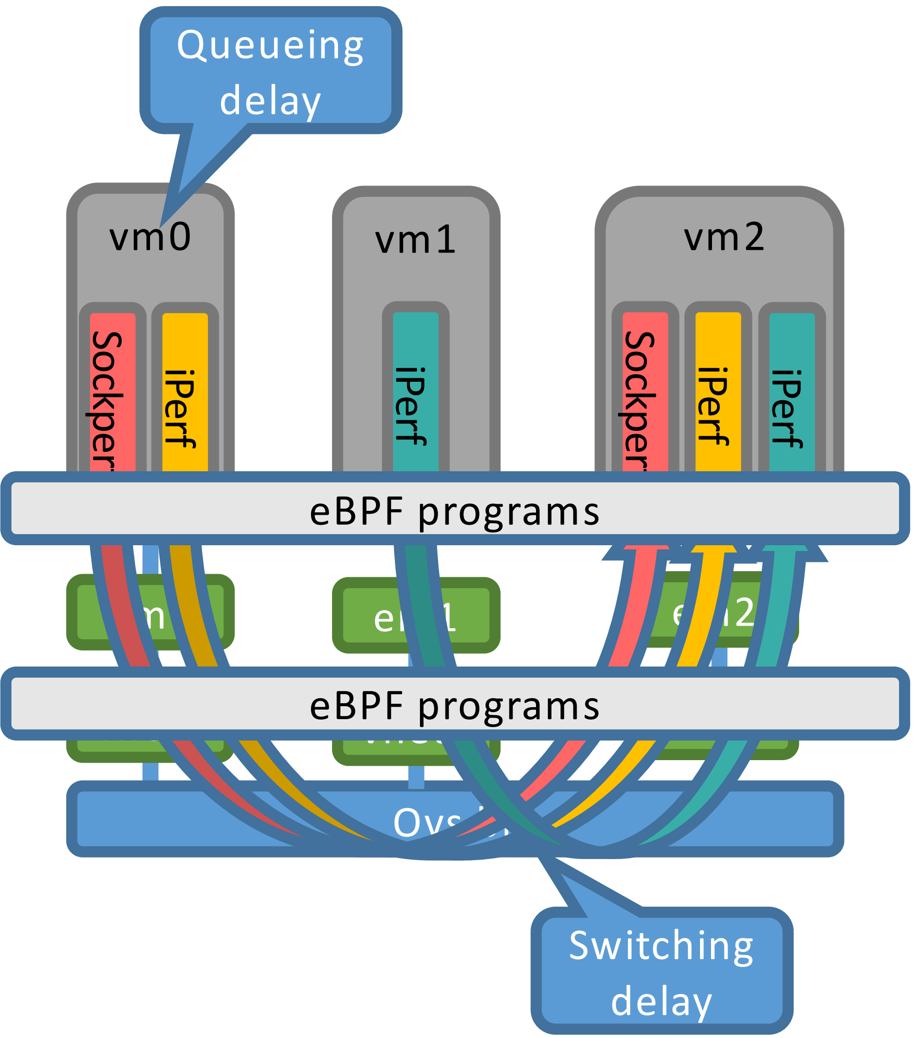
Case Study I: Network Delay in the Open vSwitch

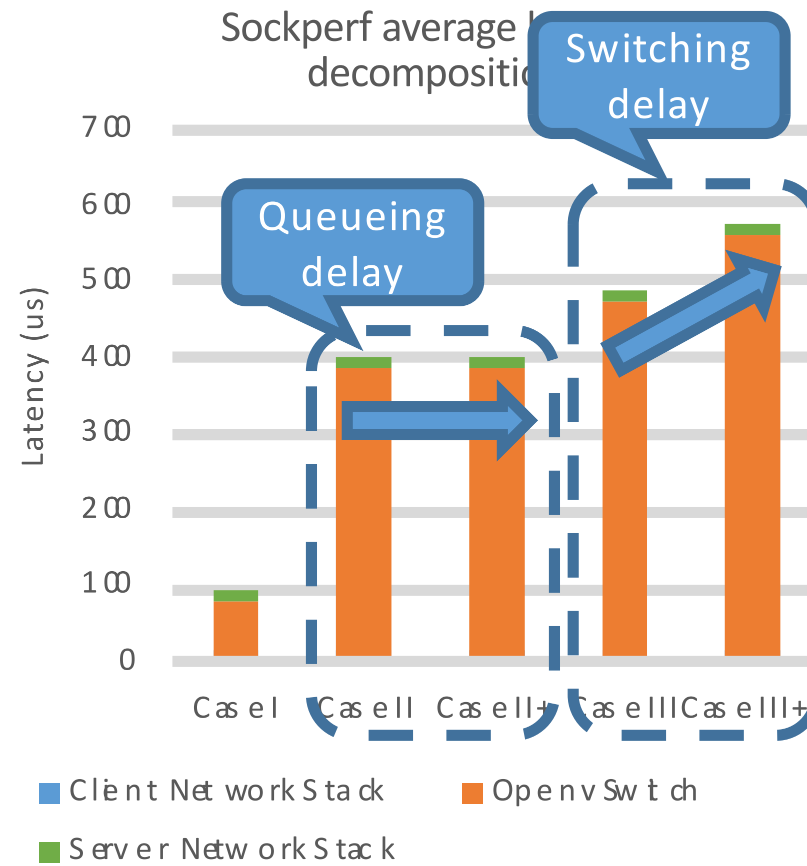


Case I Case II Case III

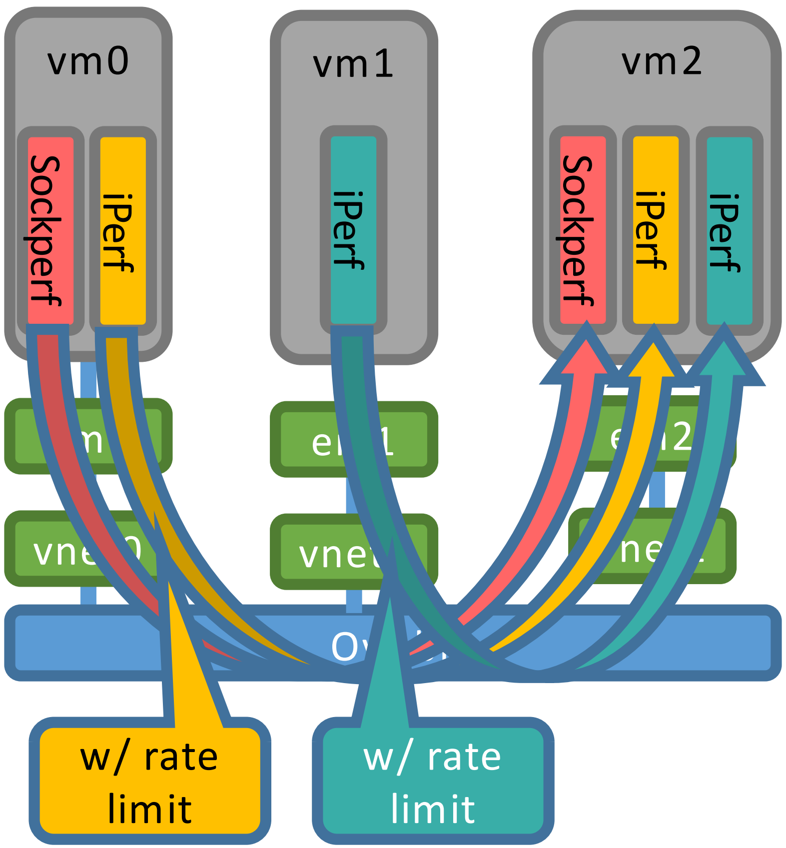


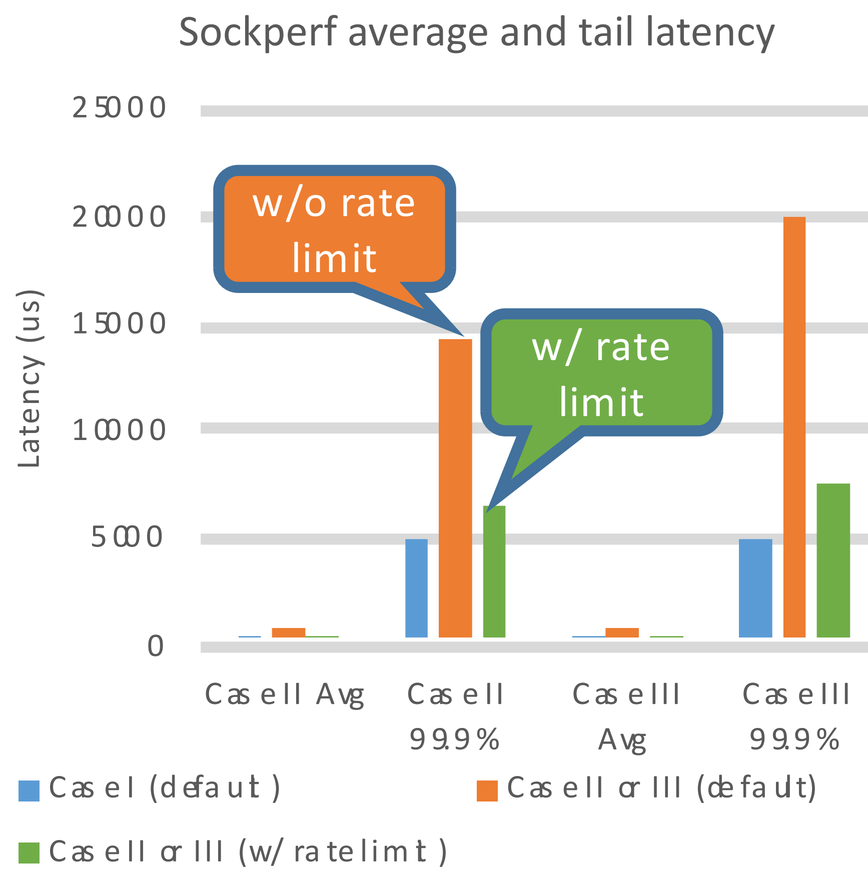
Case Study I: Network Delay in the Open vSwitch



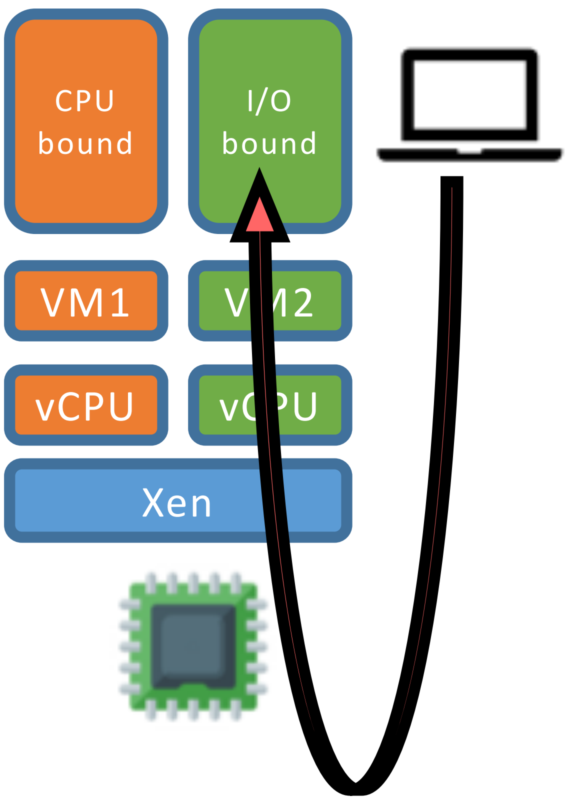


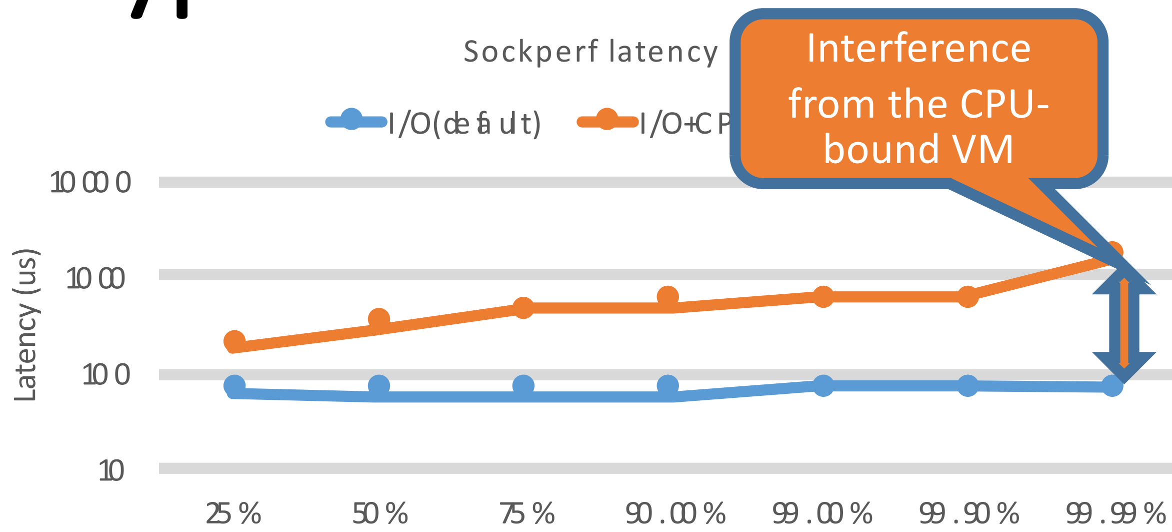
Case Study I: Network Delay in the Open vSwitch

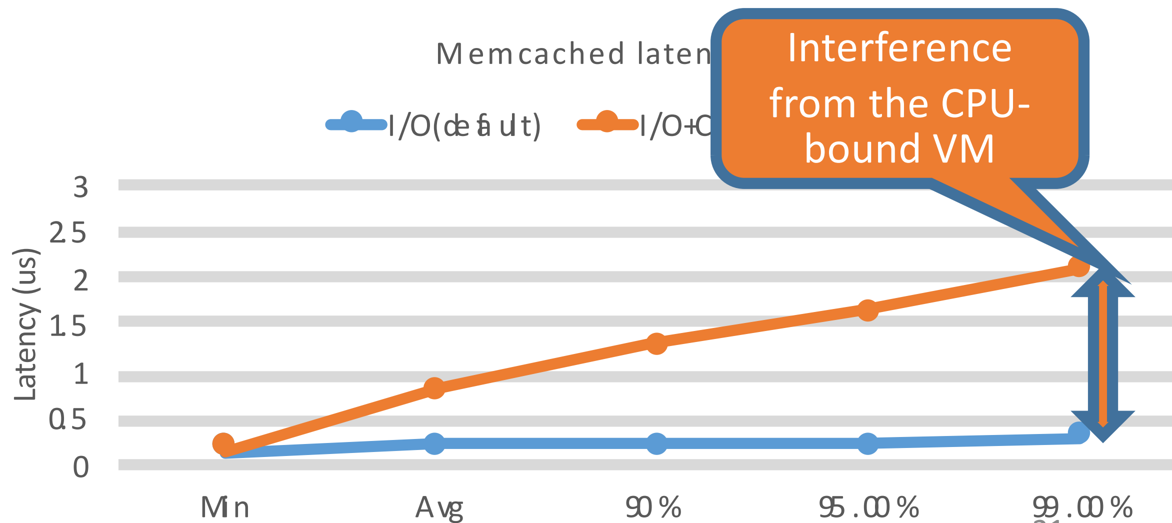




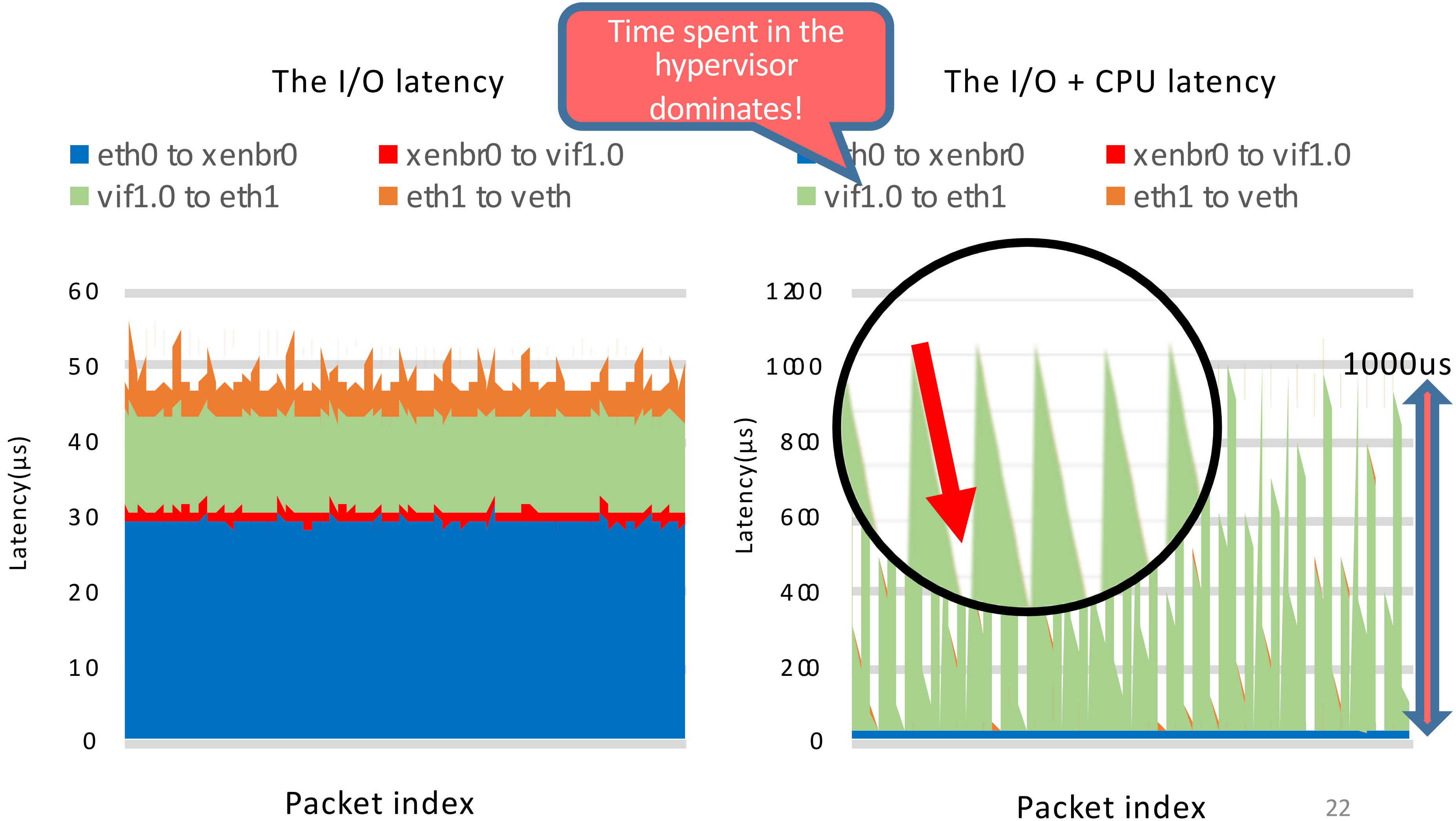
Case Study II: Tuning the Scheduler in Hypervisors



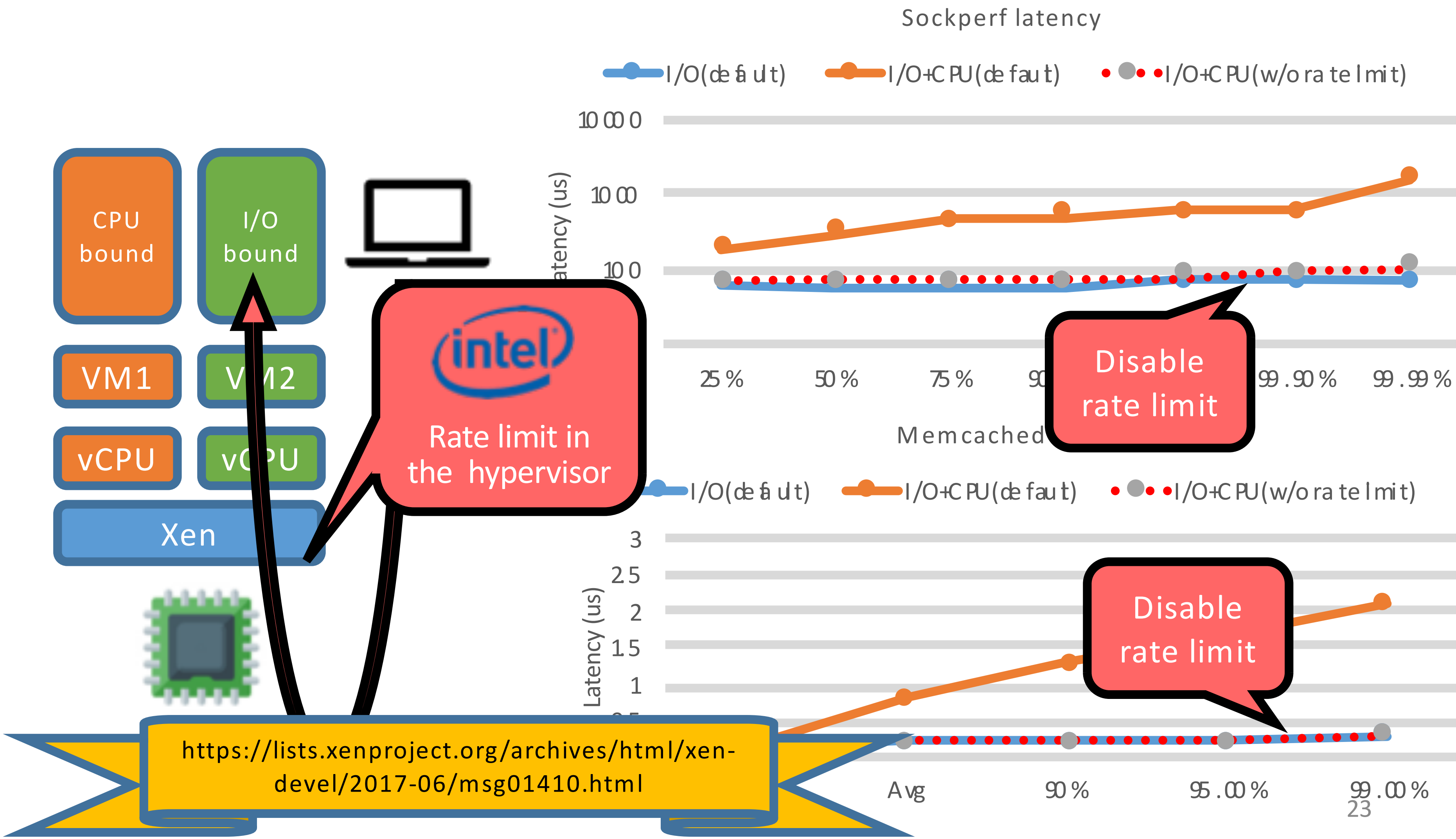




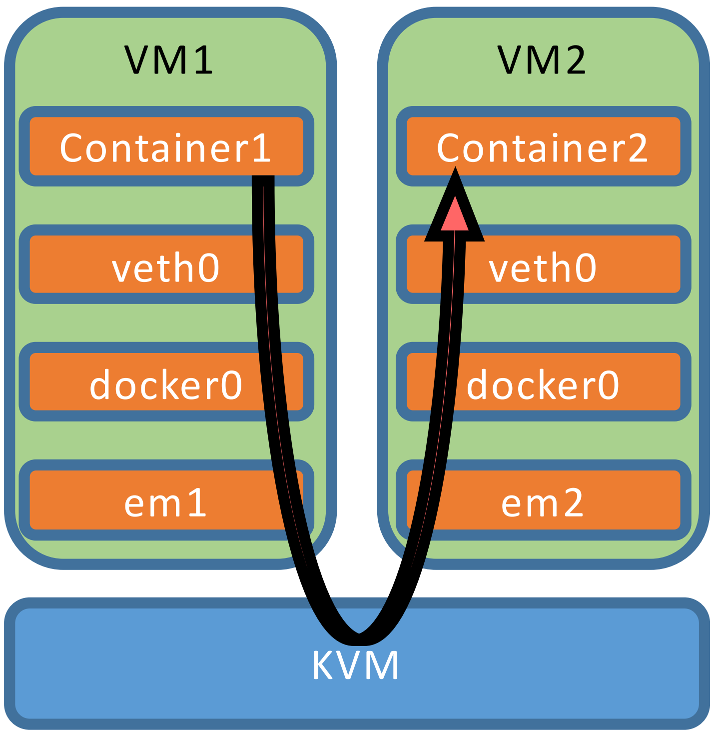
Case Study II: Tuning the Scheduler in Hypervisors

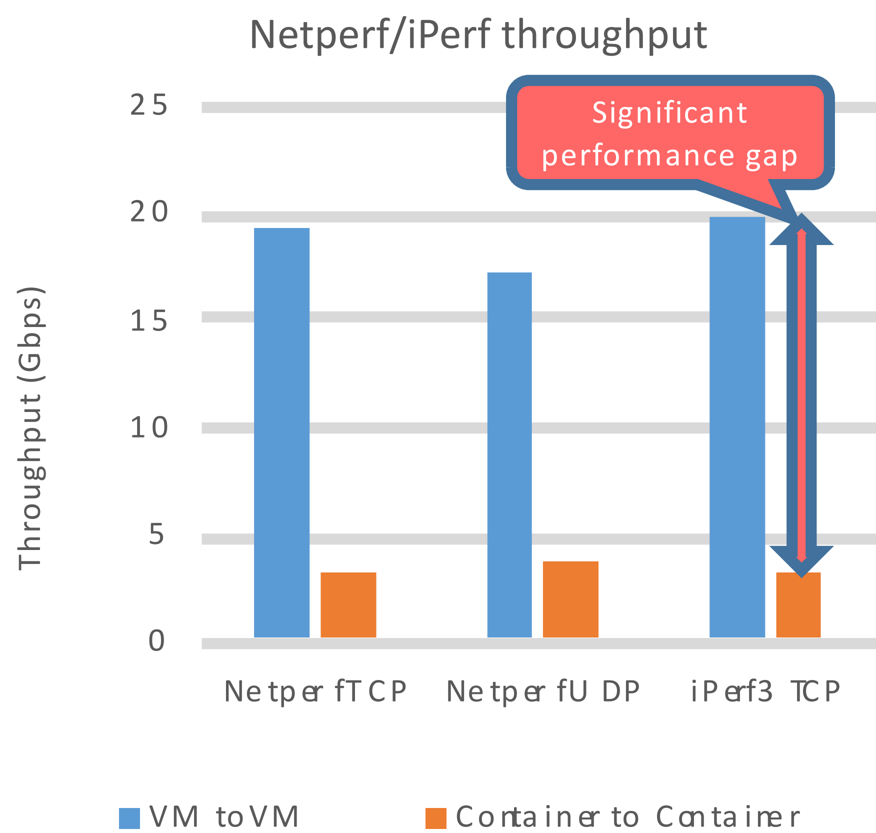


Case Study II: Tuning the Scheduler in Hypervisors

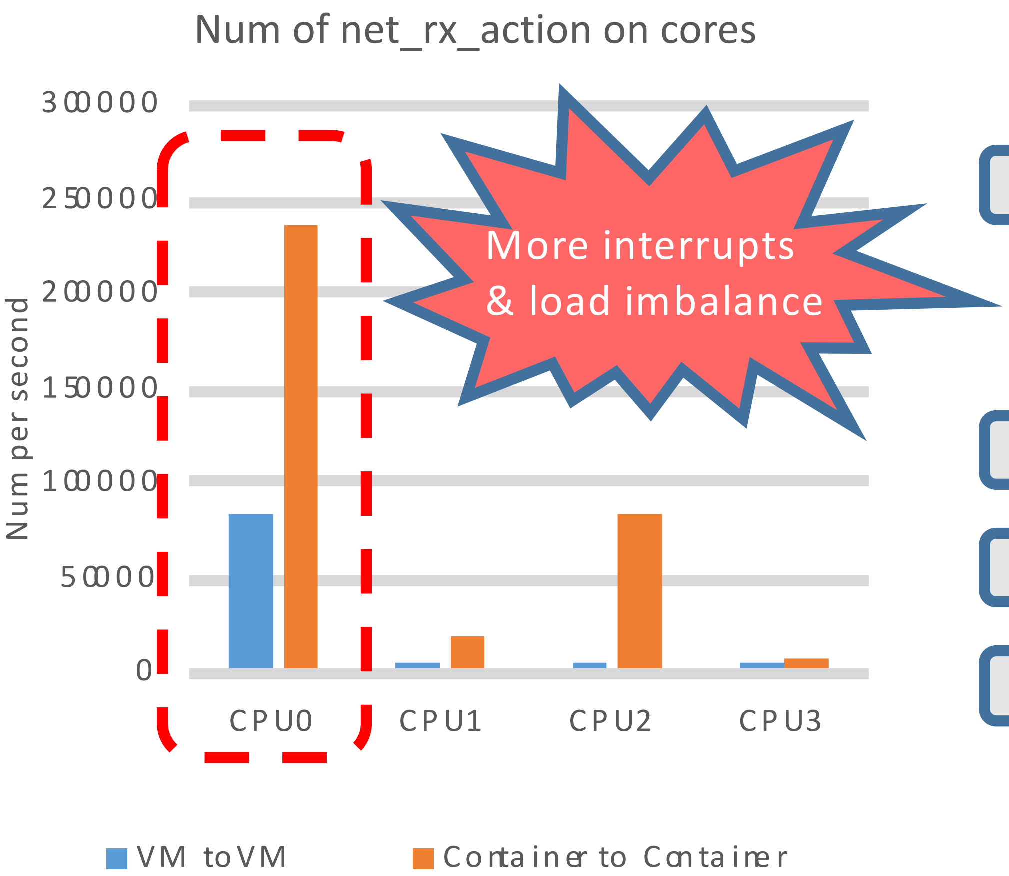


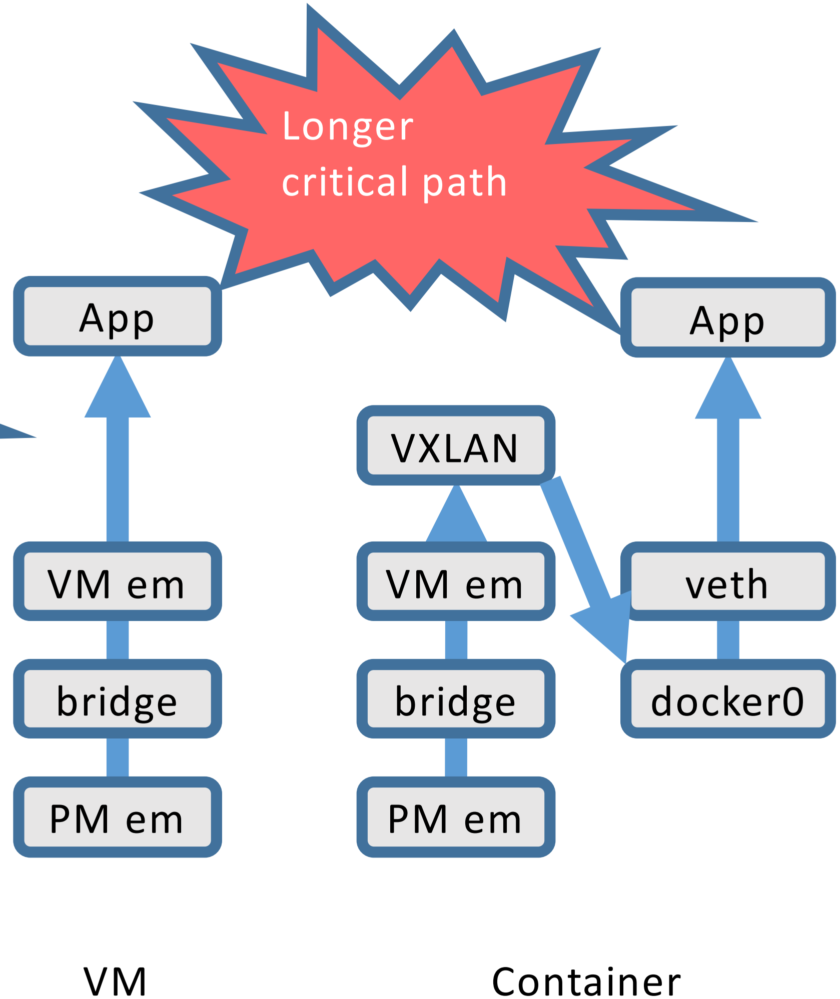
Case Study III: Bottlenecks in Container Networks





Case Study III: Bottlenecks in Container Networks





Conclusions

- **Challenges in tracing virtualized networks**
  - ✓ Need to cross the boundaries of protected domains
  - ✓ Sensitive to tracing overhead
  - ✓ Real-time and reconfigurable tracing
- **vNetTracer**
  - ✓ Feather-light tracing with eBPF
  - ✓ Instrument packet header to correlate tracing info across protected domains
  - ✓ A rich set of performance metrics
- **Results**
  - ✓ Negligible overhead
  - ✓ Shed light on system inefficiencies and bottlenecks in 3 case studies

Thank you !

Questions?