Time Capsule: Tracing Packet Latency across Different Layers in Virtualized Systems

Kun Suo[*], Jia Rao[*], Luwei Cheng[◆], Francis C. M. Lau[^]

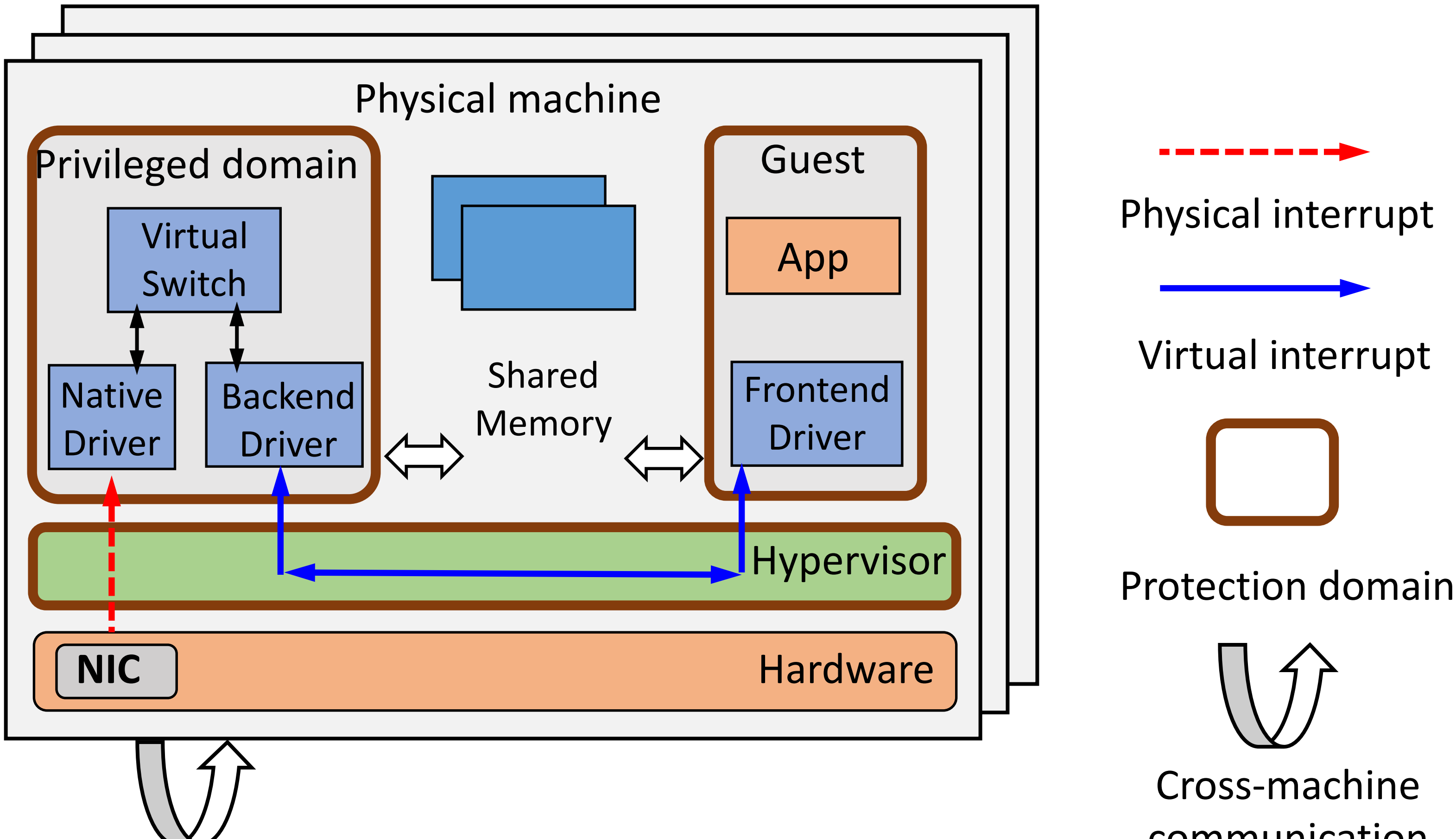
UT Arlington[*], Facebook[◆], The University of Hong Kong[^]

Virtualization and Multi-Tenancy

- Mainstream in data centers
  - ✓ Fault isolation and enhanced security
  - ✓ Improved hardware utilization
- Challenging to guarantee QoS
  - ✓ Complex virtualization stacks
  - ✓ Performance interference

Latency-sensitive network apps
suffer poor and unpredictable performance

Para-virtualized Network I/O



Possible Causes of Long Latency

- Additional layers of software stack
  - ✓ Asynchronous notifications
  - ✓ Data copy
- Resource contention
  - ✓ Time-sharing CPU
  - ✓ Contentions on data structures, e.g., locks and queues
- Packet transmission in DC network

End-to-end latency monitoring and analysis is key to identifying the causes

Challenges in Monitoring Packet Latency in Virtualized Systems

- Across the **boundaries** of protection domains
  - ✓ Machines, privileged domains, guests, and hypervisor
- **Correlating events** in various components
  - ✓ Asynchronous packet processing
- **Fine-grained** tracing with **low overhead**
  - ✓ Troubleshooting at packet level
- Application **transparency**
  - ✓ A wide spectrum of network apps, no access to code

Related Work

- Tracing tools
  - ✓ App: gperf
  - ✓ OS: SystemTap, Dtrace, Perf, and bcc
  - ✓ Hypervisor: Xentrace
- Distributed tracing
  - ✓ Causal tracing: Pip[NSDI’06], X-Trace[NSDI’07], Dapper [Google], Fay [SOSP’11], Magpie [HotOS]
  - ✓ Log mining: Draco [DSN’12], [Nagaraj NSDI’12], [Xu SOSP’09]

Related Work (cont’)

- Tracing metadata propagation
  - ✓ Pivot Tracing [SOSP’15], X-Trace [NSDI’07], Dapper[Google]
  - ✓ Propagating task IDs and timestamps using task containers or embedding the info into protocol headers

Tracing virtualized network I/O:

✓ Fine-grained tracing at packet level

✓ Packet processing at multiple hosts and different layers.

Trace metadata propagation is difficult

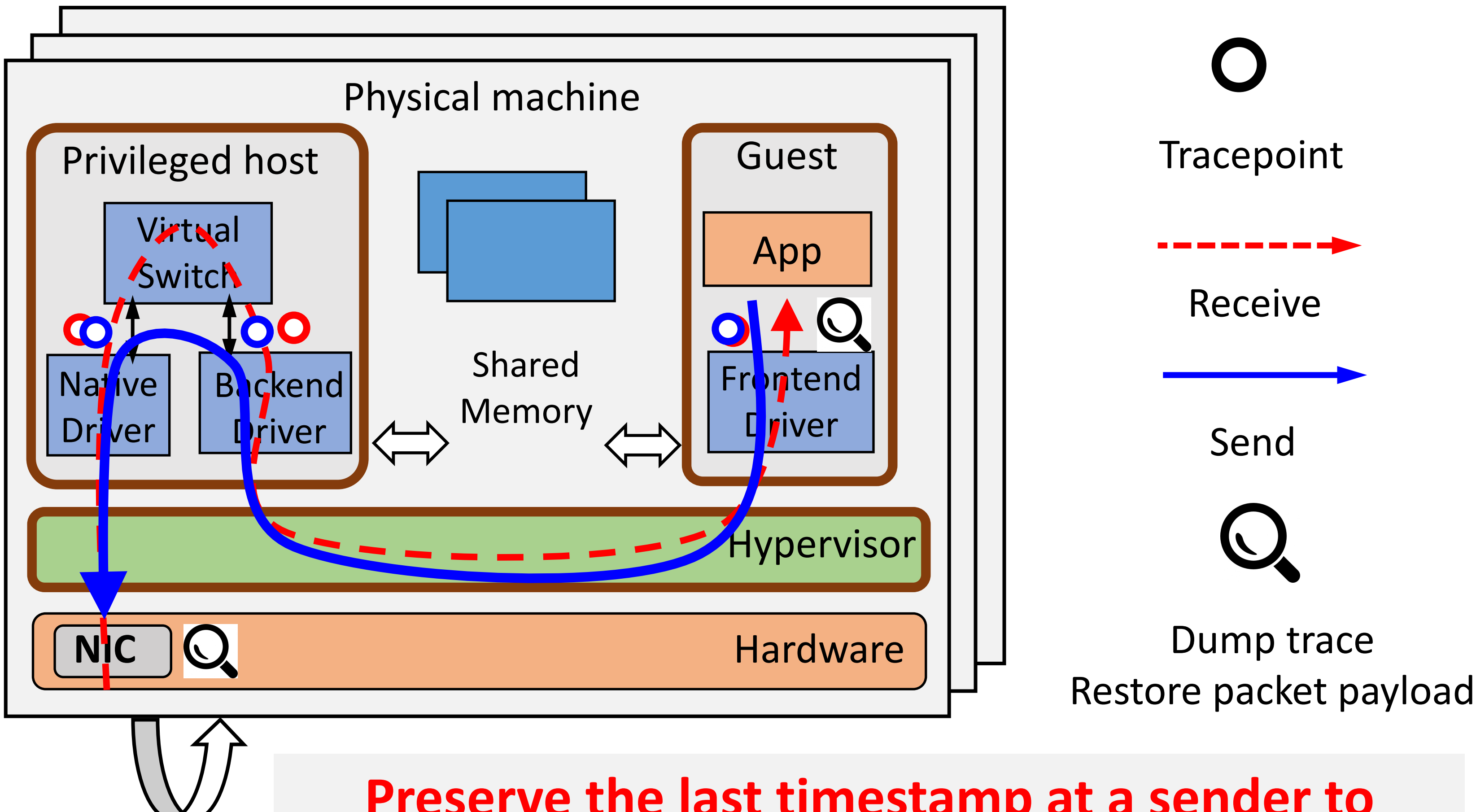
Time Capsule

Timestamp packet processing at each tracepoint and append the tracing info to the packet **payload**

Advantages:

- ✓ **Traces embedded in packets go across the boundaries of protection domains**
- ✓ **Timestamps taken at different points have happened-before relation. No need to capture causality**

Time Capsule in Action



Preserve the last timestamp at a sender to capture causality relationship across machines

Timestamping at Tracepoints

- Challenges
  - ✓ Low overhead
  - ✓ Available in separate protection domains
  - ✓ Dealing with time drift on multiple hosts
- Solution
  - ✓ Para-virtualized clocksouce `xen` or `kvm-clock`
  - ✓ Function `native_read_tscp` to read tsc values, nanosecond granularity, ~20ns overhead for each reading
  - ✓ Set `constant_tsc` to ensure consistent tsc readings across difference cores

Timestamping across Machines

- Network transmission time

$$tsc_b - tsc_a$$

- Transmission time can be negative or inaccurate
  - ✓ TSC ticks at different rates on machines *a* and *b*
  - ✓ TSC resets at different times on machines *a* and *b*

TSC Calibration

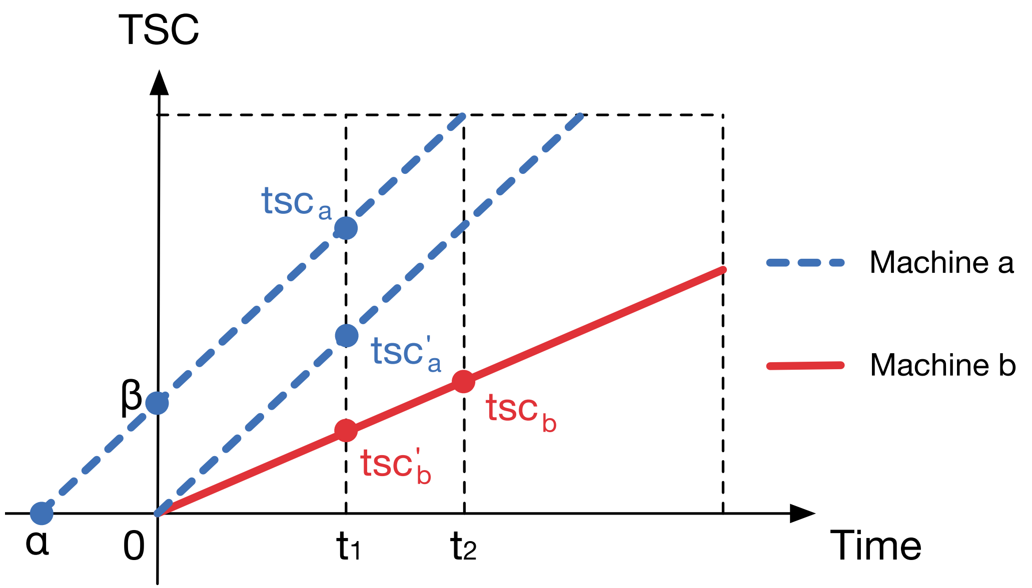
Objective:

Estimate t2 – t1 using readings of tsc_b and tsc_a

Steps:

1) Estimate machine *a*'s tsc reading tsc'_a at t1 as if both machines reset tsc at the same time
2) Convert tsc'_a to tsc'_b, the equivalent tsc reading on machine *b*
3) Calculate packet transmission time as tsc_b – tsc'_b

More details in the paper



Notation:

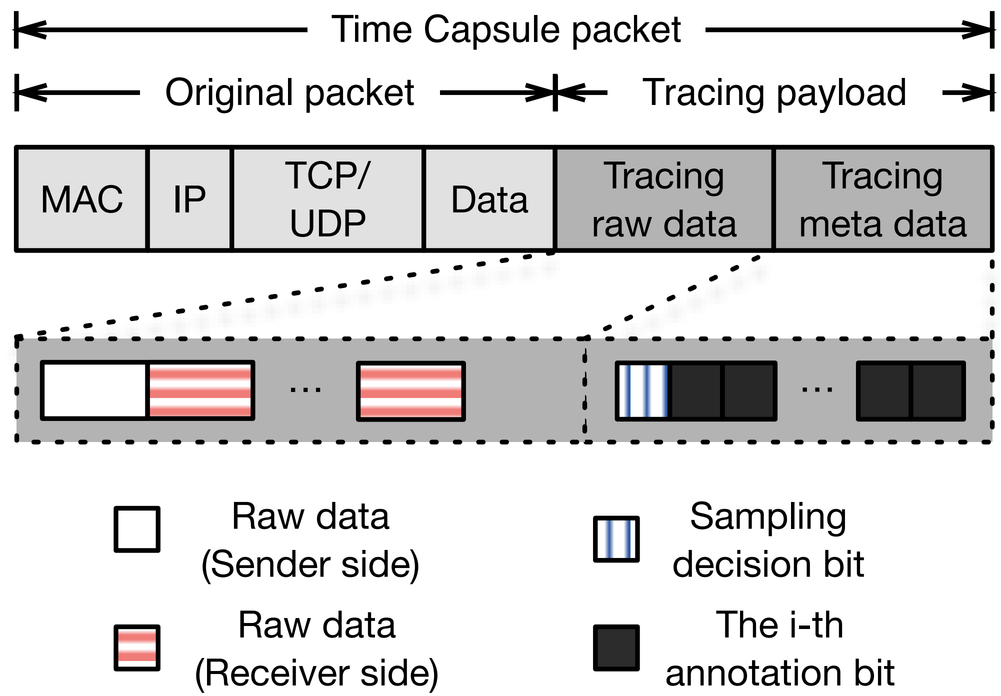
α : time difference when two machines' tsc was reset

β: absolute tsc difference

$cpufreq$: CPU frequency of a machine

Tracing Payload

- Use `__skb_put` to append trace data to the original payload
- Each timestamp is 8 bytes
- Sampling decision bit determines the sampling rate
- Annotation bit decides which tracepoint(s) to enable



Trace Collection

- Ring buffers in physical NIC driver (Tx) and guest OS network stack (Rx)
- Tracing data is removed from the packet payload and copied to the ring buffers
    - ✓ Before packet is copied to user space (Rx)
    - ✓ Before packet is transmitted by NIC driver (Tx)
- `mmap` the ring buffers to `/proc` file systems in user space
- Periodically dump trace to storage for latency analysis

Evaluation

Hardware

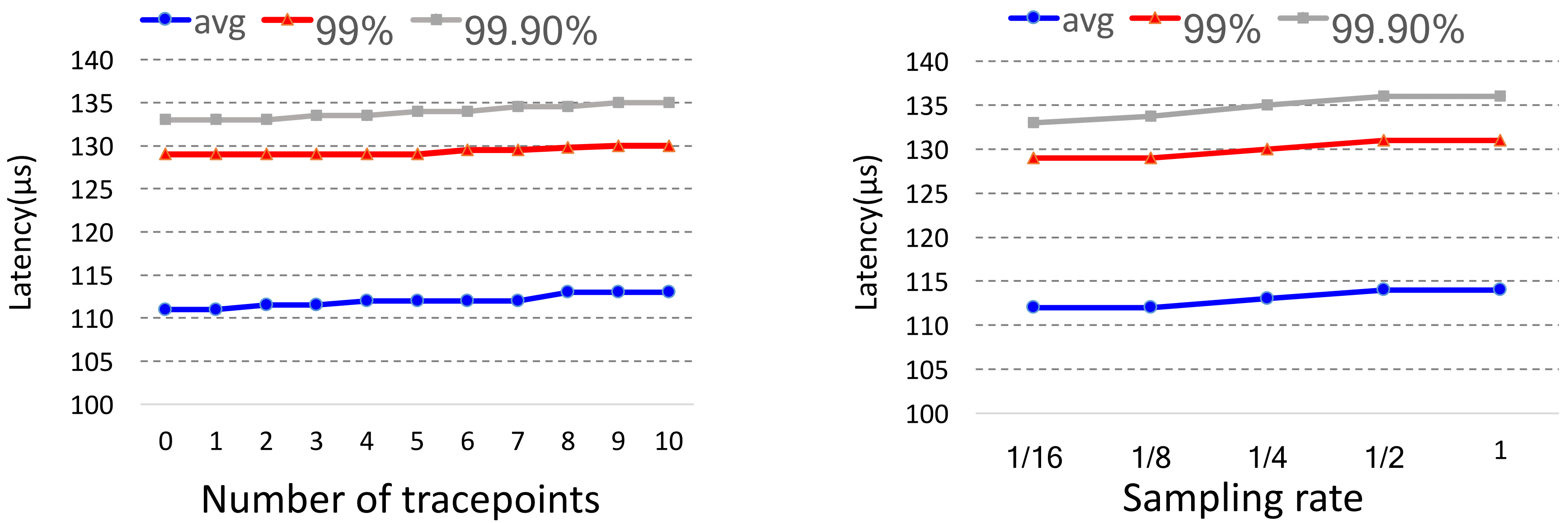
- two PowerEdge T420 servers
- two 6-core 1.90GHz Intel Xeon E5-2420 CPUs
- 32GB memory
- Gigabit Ethernet

Software

- Hypervisor: Xen 4.5
- Dom 0 and Dom U kernel: Linux 3.18.21
- VM: 1 vCPU + 4GB memory

Time Capsule Overhead

Time Capsule incurs no more than 2% latency increase

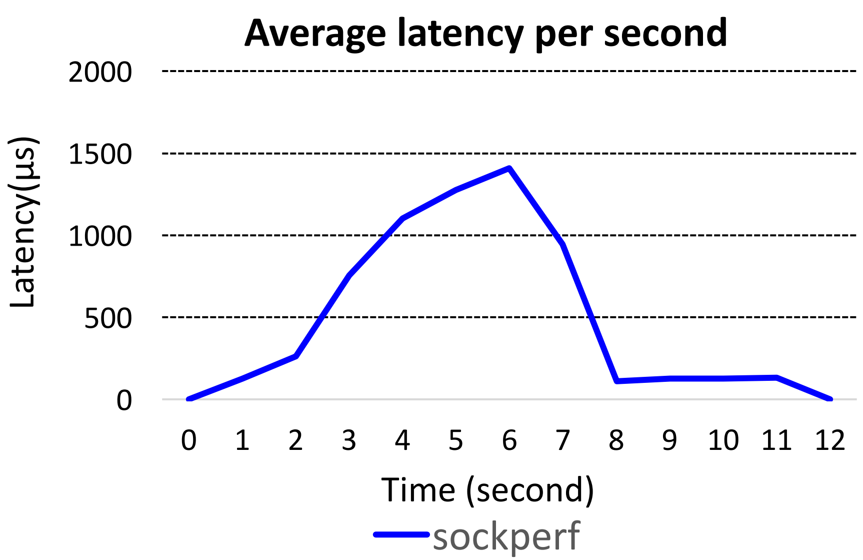


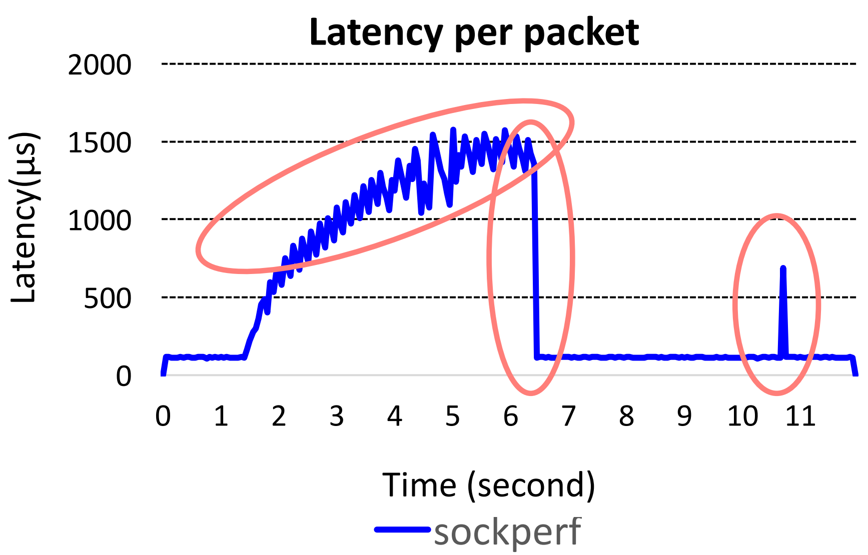
Sockperf UDP latency

Per Packet Latency

Packet level latency monitoring:

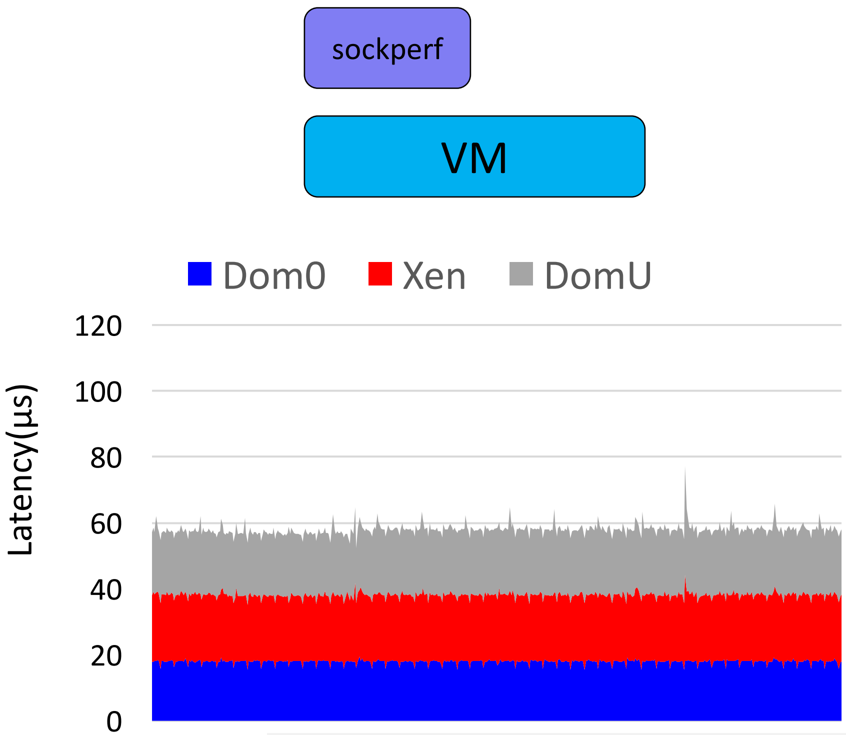
1. **Capture latency fluctuation**
2. **More responsive to traffic change**
3. **Capture transient spikes in user-perceived latency**

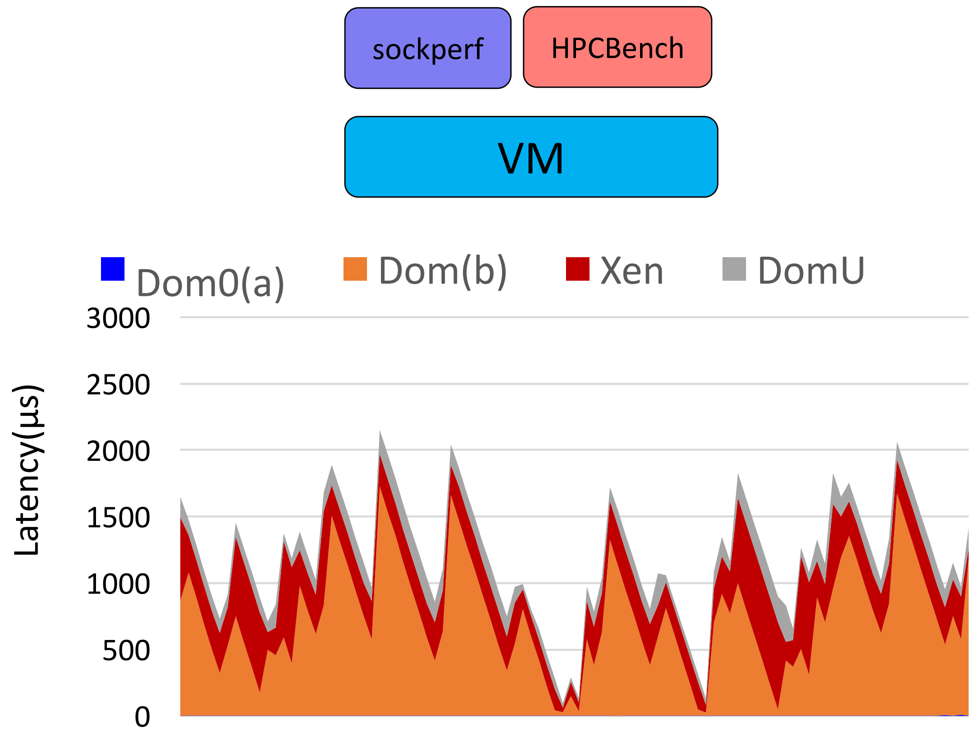




Foreground sockperf UDP request
Background netperf interference arrive at the 1.5th second and left at the 6.5th second

Diagnosis with Latency Breakdown

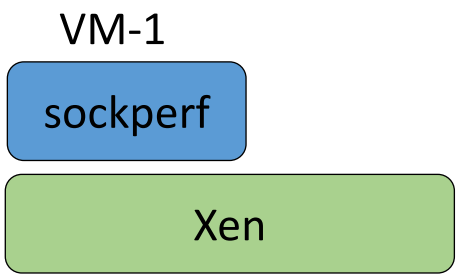




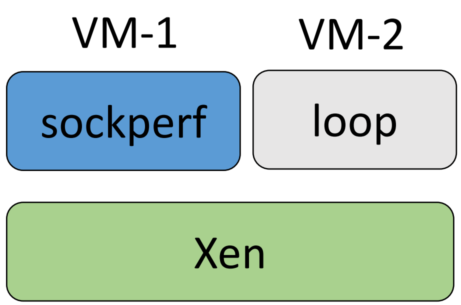
Colocation of latency-sensitive and throughput-intensive workloads in the same VM incurs long and unpredictable latency

Latency breakdown reveals that packet batching at the backend NIC driver was the culprit

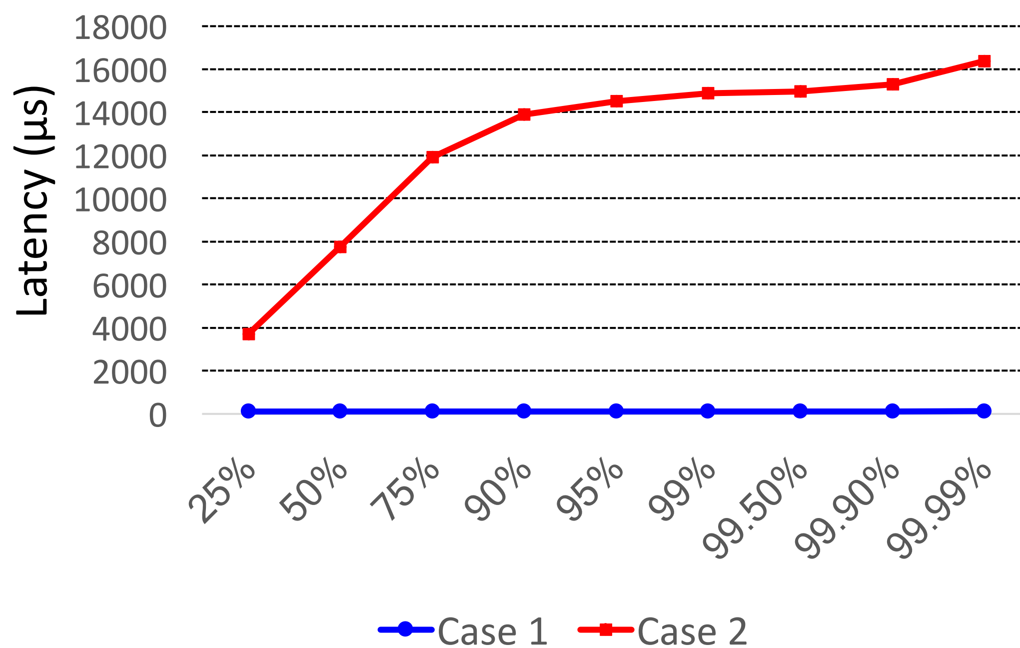
Taming Long Tail Latency in Xen



Case 1



Case 2



Bugs in Xen's credit scheduler
Time capsule helps find them!

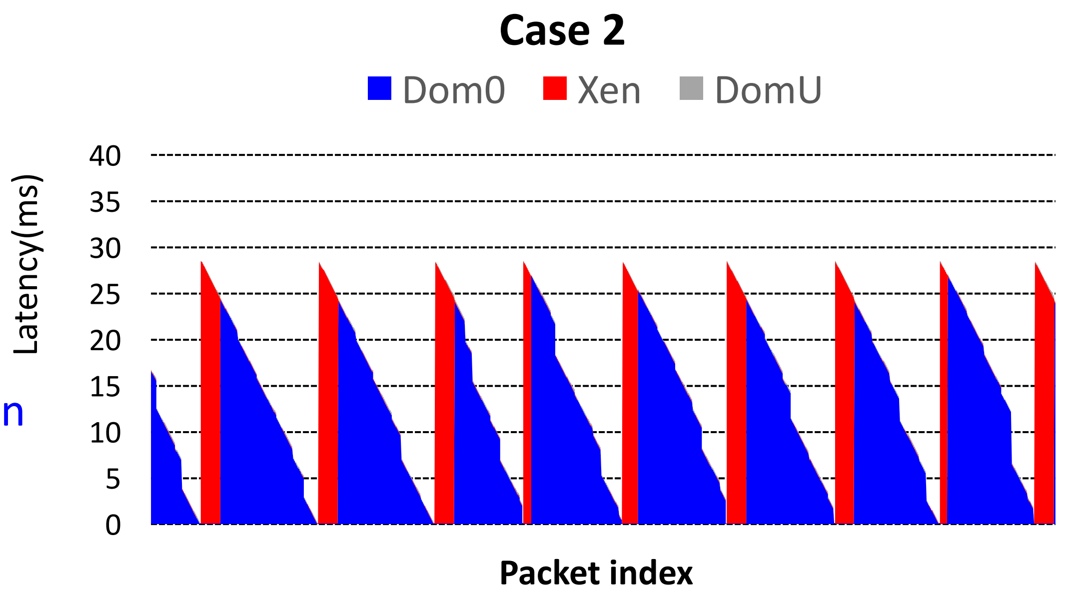
BUG-1

Observations:

1. Predictable latency spike every 250 packets

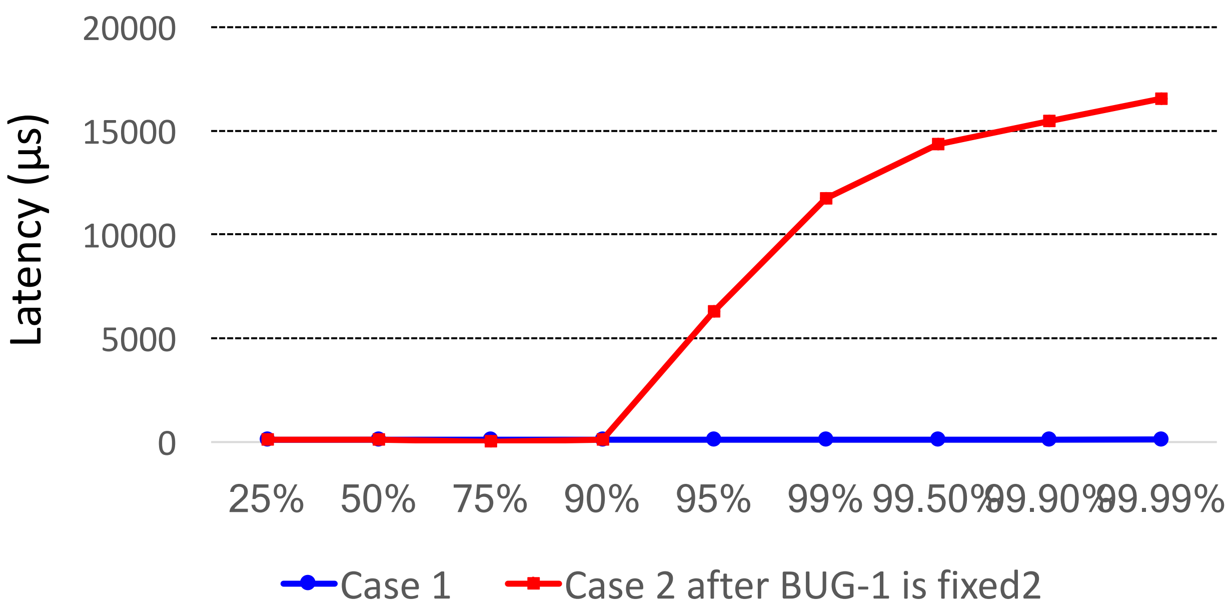
2. The tail latency close to 30ms

3. Spike always starting with delay in Xen



BUG-1:Xen mistakenly boosts the priority of CPU-intensive VMs, which causes long scheduling delays of the I/O-bound VM in Xen
http://lists.xenproject.org/archives/html/xen-devel/2015-10/msg02853.html

After BUG-1 is Fixed



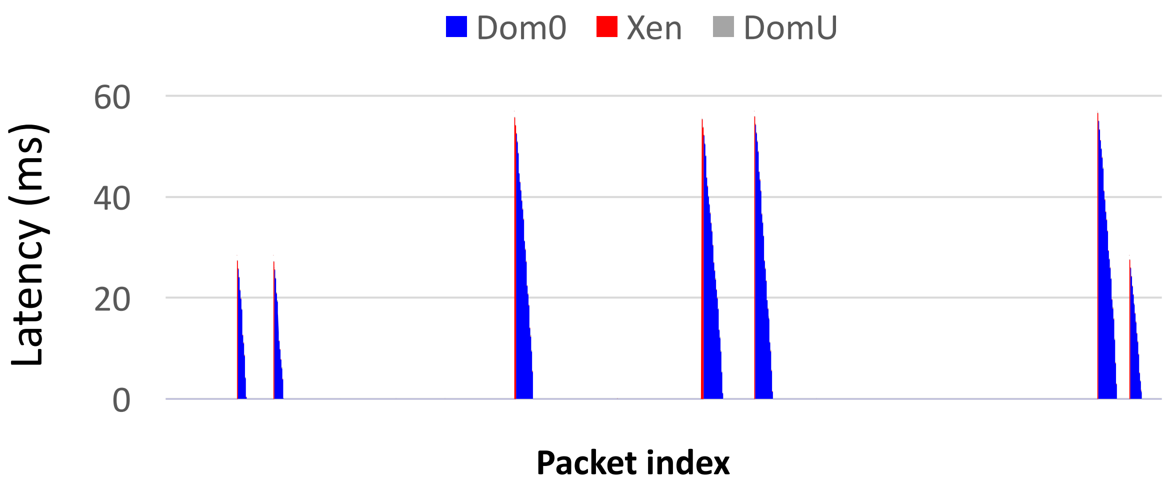
Tail latency is still not bounded

Additional Xen Bugs

Observations:

1. The occurrence and magnitude of the latency spike are unpredictable

1. Spike always starting with delay in Xen



BUG-2: Xen does not timely activate I/O VMs that are deactivated due to long idling
https://lists.xenproject.org/archives/html/xen-devel/2016-05/msg01362.html

BUG-3: I/O VMs' BOOST priority can be prematurely demoted
https://lists.xenproject.org/archives/html/xen-devel/2016-05/msg01362.html

Future Work

- **Dynamic instrumentation**
  - ✓ Extend Time Capsule to dynamically add tracepoints using BPF
- **Automated analysis**
  - ✓ use machine learning to extract better information from packet traces
- **Disk I/O**
  - ✓ extend TC to disk I/O requests. Challenge is the lack of a commonly shared data structure, such as `skb` in networking, across layers in virtualized block I/O stacks

Conclusions

- **Motivation**

  Tracing latency in virtualized systems is challenging due to the isolation of protection domains and requirements for low overhead and application transparency

- **Time Capsule**

  An in-band profiler to trace network latency at packet level in virtualized environments

- **Evaluation**

  Time Capsule incurs low overhead, enables fine-grained packet level tracing, and latency breakdown, which helps to detect bugs that cause long tail latency in Xen

Thank you !

Questions?

FAQ

- **Will time capsule affect the user-perceived packet size?**

  No and yes. When packets are received, MTU is no longer a limit for packet size. Time capsule is able to append as much data as needed to the payload and the tracing payload is removed before the packet arrives at the user space. Thus, the users are unaware of the tracing activities. However, time capsule needs to append 8 bytes to the packet to store the last timestamp at the sender side. In rare cases, this will affect the number of packets transmitted. For example, if the original packet size is 2999 bytes (a little less than two MTUs), adding 8 bytes to the original payload will require one more packet to be transmitted.