Where Has My Time Gone?

Noa Zilberman, Matthew Grosvenor, Diana Andreea Popescu, Neelakandan Manihatty-Bojan, Gianni Antichi, Marcin Wójcik, and Andrew W. Moore

Computer Laboratory, University of Cambridge
firstname.surname@cl.cam.ac.uk

Abstract. Time matters. In a networked world, we would like mobile devices to provide a crisp user experience and applications to instantaneously return results. Unfortunately, application performance does not depend solely on processing time, but also on a number of different components that are commonly counted in the overall system latency. Latency is more than just a nuisance to the user, poorly accounted-for, it degrades application performance. In fields such as high frequency trading, as well as in many data centers, latency translates easily to financial losses. Research to date has focused on specific contributions to latency: from improving latency within the network to latency control on the application level. This paper takes an holistic approach to latency, and aims to provide a break-down of end-to-end latency from the application level to the wire. Using a set of crafted experiments, we explore the many contributors to latency. We assert that more attention should be paid to the latency within the host, and show that there is no silver bullet to solve the end-to-end latency challenge in data centers. We believe that a better understanding of the key elements influencing data center latency can trigger a more focused research, improving the user's quality of experience.

1 Introduction

Time plays a major role in computing, as it translates directly to financial losses [6,13]. User demands for a highly interactive experience (e.g., online shopping, web search, online gaming etc.) has put stringent demands on applications to consistently meet tight deadlines. Nowadays, the question *Can the application (job) meet a deadline?* is replaced by *Will the application get the consistent, low latency, guarantees needed to meet user demands?*.

In the past, large propagation delays and unoptimized hardware have eclipsed inefficiencies in end-system hardware and software: operating systems and applications. Yet decades ago, latency was identified as a fundamental challenge [2,11]. The emergence of data centers increased the importance of the long tail of latency problem: due to the scaling effect within a data center, every small latency issue is having an increasing effect on the performance [1]. Only 5 years ago a switch latency of 10μs and an OS stack latency of 15μs were considered the norm [12], however, since then, a significant improvement has been achieved [5,3]. To fully understand this latency improvement, this paper takes an end-to-end approach, focusing upon the latency between the time a request is issued by an application to the time a reply has returned to that application.

This approach has the advantage of maximizing the throughput of a system, which is the main goal of a user, rather than optimizing discrete parts of the system. We consider the best-possible configurations, which may not be identical to the most realistic configuration, and further focus on the Ethernet-based systems common in data centers.

In this paper we use bespoke experiments (described in §2) to derive a breakdown to the end-to-end latency of modules in commodity end-host systems (discussed in §3). We identify the latency components that require the most focus for improvement and propose trajectories for such work. Finally, we contribute a taxonomy of latency contributors: low-latency/low-variability: the "Good", high-latency/high-variability: the "Bad", and heavy-tailed or otherwise peculiar latency: the "Ugly", while also noting the challenge of profiling application network performance.

1.1 Motivation

The contribution of latency affects a user-experience in a significant, sometimes subtle, manner. More than a simple, additive, increase in run-time, application performance can be dramatically decreased with an increase in latency. Figure 1 illustrates the impact of latency upon performance for several common data center applications.

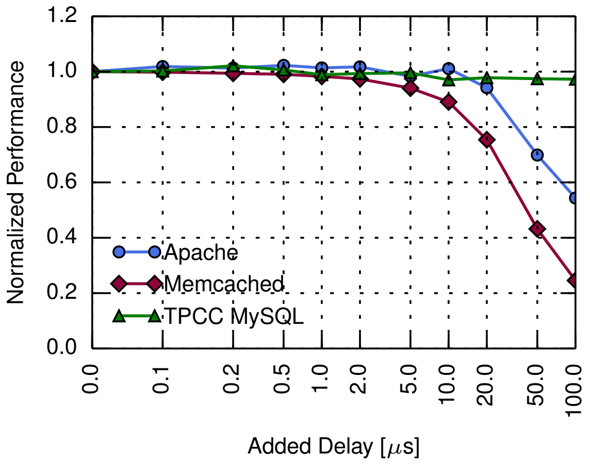

Fig. 1. Delay Effect on Application Performance.

Using an experimental configuration described in Section 2, Figure 1 illustrates experimental results for three application-benchmarks. Each benchmark reports results for an application-specific performance metric. These application-specific benchmarks are normalized to allow comparisons to be made among the applications.

The three benchmarks we use are Apache benchmark[1] reporting mean requests per second, Memcached benchmark[2] reporting throughput, and TPC-C MySQL benchmark[3] reporting New-Order transactions per minute, (where New-Order is one of the database's tables).

Between the two hosts of the experimental configuration described in Section 2, we insert a bespoke hardware device to inject controlled latency. We implemented a

[1] `https://httpd.apache.org/docs/2.4/programs/ab.html`
[2] `http://docs.libmemcached.org/bin/memaslap.html`
[3] `https://github.com/Percona-Lab/tpcc-mysql`

latency-injection appliance[4] that allows us to add arbitrary latency into the system. Past latency injection has been done with approaches such as NetEm [4], yet this proved inappropriate for our work. Alongside limited granularity, such approaches may not reliably introduce latency of less than several tens of microseconds [8]. In contrast, our latency gadget adds 700ns of base latency and permits further additional latency, at 5ns granularity, up to a maximum[5] determined by the rate of operation.

Each test begins by measuring a baseline, which is the performance of each benchmark under the default setup conditions, taking into account the base latency introduced by the latency-injection appliance. Latency is then artificially inserted by the appliance, and the application-specific performance is measured. We can derive the impact on experiments of the artificially inserted latency by removing the baseline measurement. For the three benchmarks, Figure 1 shows the effect of added latency. Each benchmark was run 100 times for the baseline and for each added latency value. The graph plots the average values, and standard errors are omitted for clarity, as they are below 0.005. In one run, the Apache benchmark sends 100000 requests and the Memcached benchmark sends 10 million requests. The TPC-C benchmark runs continuously for 1000 seconds, with an additional time of 6 minutes of warm-up, resulting in 100 measurements over 10 seconds periods. The application most sensitive to latency is Memcached: the addition of 20μs latency leads to a performance drop of 25%, while adding 100μs will reduce its throughput to 25% of the baseline. The TPC-C benchmark is the least sensitive to latency, although still exhibits some performance loss: 3% reduction in performance with an additional 100μs. Finally, the Apache benchmark observes a drop in performance that starts when 20μs are added, while adding 100μs leads to a 46% performance loss.

While the results above are obtained under optimal setup conditions, within an operational data center worse-still results would be expected as latency is further increased under congestion conditions and as services compete for common resources. The results of Figure 1 show clearly that even a small increase in latency, of the scale shown in this paper, can significantly affect an application's performance.

2 Experiments

This section presents experiments we used to provide a decomposition of the latency between the application and the physical-wire of the host. Full results of these experiments are given in Section 3 with the outcome of successive tests presented in Table 1. Each experiment in this section is annotated with the corresponding entry number in Table 1.

2.1 Tests Setup

For our tests setup we use two identical hosts running Ubuntu server 14.04LTS, kernel version 4.4.0-42-generic. The host hardware is a single 3.5GHz Intel Xeon E5-2637 v4

[4] Our latency-injection appliance is an open-source contributed project as part of NetFPGA SUME since release 1.4.0.

[5] The maximum latency introduced is a function of the configured line-rate. The appliance can add up to 700μs of latency at full 10Gb/s rate, and up to 7s at 100Mbps.

on a SuperMicro X10-DRG-Q motherboard. All CPU power-saving, hyper-threading and frequency scaling are disabled throughout our tests. Host adapter evaluation uses commodity network interface cards (NICs), Solarflare SFN8522, and Exablaze X10, using either standard driver or a kernel bypass mode (test dependent). For minimum latency, interrupt hold-off time is set to zero. Each host uses identical NICs for that particular NIC experiment and we only consider Ethernet-based communication. As illustrated in Figure 3, an Endace 9.2SX2 DAG card (7.5ns time-stamping resolution) and a NetOptics passive-optical tap are used to intercept client-server traffic and permit independent measurement of client & server latency. The experiments are reproducible using the procedures documented at `http://www.cl.cam.ac.uk/research/srg/netos/projects/latency/pam2017/`.

2.2 In-host latency

Figure 2 illustrates the various elements contributing to the experienced latency within the host.

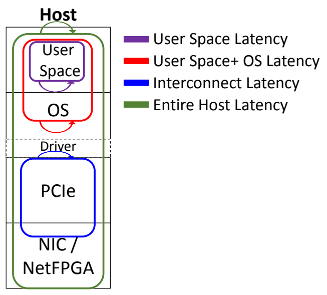

Fig. 2. End Host Tests Setup.

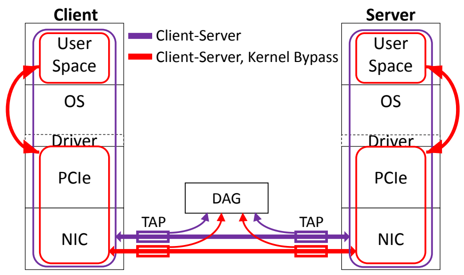

Fig. 3. Client-Server Tests Setup.

Timestamp Counter Latency (1) To accurately measure latency, we set an accuracy baseline for our methods. Our latency measurements are based on the CPU's Time Stamp Counter (TSC). TSC is a 64-bit register, present on the processor, it counts the number of cycles since reset and thus provides a resolution of approximately 288ps-per-cycle although realistically there is tens of nanoseconds resolution due to CPU pipeline effects. Access to TSC is done using *rdtsc* x86 assembly instruction. In order to understand hidden latency effects, and following the Intel recommendations for TSC access [10], we conduct two read operations consecutively. We repeat this simple TSC read operation a large number of times (order of 10^{10} events), and study the time gaps measured between every pair of consecutive reads. Results are saved into previously allocated and initialized buffers, and access to the buffers is outside the measurement mainline.

This test is conducted in three different modes: firstly, *Kernel Cold Start* (1a) which serves as our approximation of a bare metal test. Kernel Cold Start measures very early within the kernel boot process, before the scheduler, multiprocessing and multicore support have been started. The second test, *Kernel Test* (1b), runs from within the kernel, and represents an enhanced version of the recommended test described in [10]. The third test, *User Space Test* (1c), provides high-accuracy time stamping measurement from within a user-space application. The application is pinned to a single CPU core and all other tasks and interrupts are moved to other cores. This is representative of real-time application operation. In contrast to the Kernel Test, interrupts, such as scheduling preemption, are not disabled so as to represent the runtime conditions of real applications.

User Space + OS Latency (2) This experiment investigates the combined latency of the (user-space) application and the operating system. The test sets up two processes and opens a datagram socket between them, measuring the round trip time (RTT) for a message sent from a source process to the destination process, and back. TSC is used to measure the latency and the time is measured by reading TSC before and after the message reply is received. While this does not fully exercise the network stack, it does provide useful insight into the kernel overhead.

Virtualized Environment (1d) The contribution of a virtualized environment is examined by repeating the TSC tests from within a Virtual Machine (VM). We used VirtualBox [9] version 4.3.36 as the hypervisor, with an Ubuntu VM (same version as the base OS). The VM was configured to run the guest OS on a single dedicated CPU core with no co-located native operating system activities.

Host Interconnect (3) To evaluate the latency of the host interconnect (e.g., PCI-Express), we used the NetFPGA SUME platform [16], which implements x8 PCIe Gen3 interface. The DMA design is instrumented to measure the interconnect latency. As the hardware and the processor use different clock sources, the one-way latency can not be directly measured. Instead, the round trip latency of a read operation (a non-posted operation that incorporates an explicit reply) is measured. Every read transaction from the NetFPGA to the CPU is timestamped at 6.25ns accuracy within the DMA engine when each request is issued and when its reply returns. The cache is warmed before the test, to avoid additional latency due to cache misses, and the memory address is fixed. The measured latency does not account for driver latency, as neither the driver nor the CPU core participate in the PCIe read transaction.

Host Latency (4) To measure the latency of an entire host we use a bespoke request-reply test to measure the latency through the NIC, PCIe Interconnect, Kernel and network stack, the application level, and back to the NIC. Contrast to the *User Space + OS Latency* experiment, here packets traverse the networks stack only once in each direction. Packets are injected by a second host, and using the DAG card we isolate the host latency, measuring the latency from the packet's entrance to the NIC and until it returns from the NIC.

Kernel Bypass influence (5) Kernel bypass is promoted as a useful methodology and we consider the latency contribution of the operating-system kernel alone and the impact of kernel-bypass upon latency. Using tests comparable to those of *Host Latency* experiment we can then measure latency using the kernel bypass supported by our NICs (X10, SFN8522). Our performance comparison contrasts the kernel with bypass enabled and disabled.

2.3 Client-Server Latency (6)

Experiments are extended from single-host (and, where appropriate, hardware request-reply server) to a pair of network-hosts as shown in Figure 3. The two servers are directly connected to each other. Using a test method based upon that described in the *Host Latency* experiment, we add support for request-reply at both hosts. This allows us to measure latency between the user-space application of both machines. We further extend this experiment to measure the latency of queries (both get and set) under the Memcached benchmark, indicative of realistic user-space application latency.

2.4 Network Latency

We measure three components that contribute to network latency: networking devices within the network, cabling (e.g., fiber, copper), and networking devices at the edge. The network device at the edge is represented in this study by the NIC. For networking devices within the network we focus on electrical packet switches (EPS) as the most commonly used networking devices within data center today. Networking devices such as routers will inherently have a latency that is the same or larger than a switch, thus we do not study them specifically.

Our focus in this work is on the minimum latency components within a system. We therefore do not evaluate latency components of networking devices such as queueing and buffering or congestion. We consider these out of scope in our attempt to understand the most-ideal latency situation.

Cabling. The propagation delay over a fiber is 4.9ns per meter, and the delay over a copper cable varies between 4.3ns and 4.4ns per meter, depending on the cable's thickness and material used. We corroborate these numbers by sending packet trains over varying lengths of cable and measuring using DAG the latency between transmit and receive[6]. In our reported tests we use fiber exclusively.

NIC Latency (7). Measuring NIC-latency is a subtle art. At least three components contribute to a typical NIC latency figure: the NIC's hardware, the Host Bus Adapter (a PCI-Express interconnect in our case) and the NIC's driver. There are two ways to measure the latency of a NIC: the first is injecting packets from outside the host to the NIC, looping the packets at the driver and capturing them at the NIC's output port. The second is injecting packets from the driver to the NIC, using a (physical or logical) loopback at the NIC's ports and capturing the returning packet at the driver. Neither of these ways allows us to separate the hardware-latency contribution from the rest of its latency components or to measure one way latency. Acknowledging these limitations, we opt for the second method, injecting packets from the driver to the NIC. We use a loopback test provided by Exablaze with the X10 NIC[7]. The test writes a packet to the driver's buffer, and then measures the latency between when the packet starts to be written to PCIe and when the packet returns. This test does not involve the kernel.

[6] We note that the resolution of the DAG of 7.5ns puts short fiber measurements within this range of error

[7] The source code for the test is provided with the NIC, but is not open source.

A similar open-source test provided by Solarflare as part of Onload (eflatency), which measures RTT between two nodes, is used to evaluate SFN8522 NIC. The propagation delay on the fiber is measured and excluded from the NIC latency results.

Switch Latency (8). We measure switch latency using a single DAG card to timestamp the entry and departure time of a packet from a switch under test. The switch under test is statically configured to send packets from one input port to another output port. No other ports are being utilized on the switch during the test, so there is no crosstalk traffic. We vary the size of the packets sent from 64B to 1514B.

We evaluate two classes of switches, both of them cut-through switches: an Arista DCS-7124FX layer 2 switch, and an ExaLINK50 layer 1 switch. The latency reported is one way, end of packet to end of packet.

Caveats: Latest generation cut through switching devices, such as Mellanox Spectrum and Broadcom Tomahawk, opt for lower latency than we measure, on the order of 330ns. We were not able to obtain these devices. As a result, later discussion of these, as well as of large store-and-forward spine switches (e.g., Arista 7500R) relies on results taken from vendors' datasheet and industry analysis [15].

3 Latency Results

	Experiment	Minimum	Median	99.9th	Tail	Observation Period
1a	TSC - Kernel Cold Start	7ns	7ns	7ns	11ns	1 Hour
1b	TSC - Kernel	9ns	9ns	9ns	6.9μs	1 Hour
1c	TSC - From User Space	9ns	10ns	11ns	49μs	1 Hour
1d	TSC - From VM User Space	12ns	12ns	13ns	64ms	1 Hour
2a	User Space + OS (same core)	2μs	2μs	2μs	68μs	10M messages
2b	User Space + OS (other core)	4μs	5μs	5μs	31μs	10M messages
3a	Interconnect (64B)	552ns	572ns	592ns	608ns	1M Transactions
3b	Interconnect (1536B)	976ns	988ns	1020ns	1028ns	1M Transactions
4	Host	3.9μs	4.5μs	21μs	45μs	1M Packets
5	Kernel Bypass	895ns	946ns	1096ns	5.4μs	1M Packets
6a	Client-Server (UDP)	7μs	9μs	107μs	203μs	1M Packets
6b	Client-Server (Memcached)	10μs	13μs	240μs	20.3ms	1M Queries
7a	NIC - X10 (64B)	804ns	834ns	834ns	10μs	100K Packets
7b	NIC - SFN8522 (64B)	960ns	985ns	1047ns	3.3μs	100K Packets
8a	Switch - ExaLINK50 (64B)	0^α	2.7ns $^\alpha$	17.7ns $^\alpha$	17.7ns $^\alpha$	1000 Packets
8b	Switch - ExaLINK50 (1514B)	0^α	2.7ns $^\alpha$	17.7ns $^\alpha$	17.7ns $^\alpha$	1000 Packets
8c	Switch - 7124FX (64B)	512ns	534ns	550ns	557ns	1000 Packets
8d	Switch - 7124FX (1514B)	512ns	535ns	557ns	557ns	1000 Packets

Table 1. Summary of Latency Results. Entries marked $^\alpha$ return results that are within DAG measurement error-range.

The results of the experiments described in Section 2 are presented in Table 1. The accuracy of time-measurements in kernel space, user space, or within a VM is on the order of tens of CPU clock cycles (approximately 10ns in our system). Any operation beyond that is on the order of between hundreds of nanoseconds and microseconds. To better understand this, Figure 4 shows the relative latency contribution of each component. The figure makes it clear that there is no single component that contributes overwhelmingly to end-host latency: while the kernel (including the network stack) is certainly important, the application level also makes significant contribution to latency as, even in our straightforward evaluation example, applications incur overheads due to user-space/kernel-space context switches.

Deriving the latency of different components within the network is not as straightforward as within the host, and depends on the network topology.

To illustrate this impact we use four typical networking topologies, depicted in Figure 6, combined with the median latency results reported in Table 1. Representing the store-and-forward spine switch we use the latency of Arista-7500R switch. Figure 5 shows the relative latency contribution within each network topology.

While differences in latency contribution here are enormous, just as in the end-host case single huge contributor to network latency. Furthermore, the latency of the fibers, which is often disregarded, has a magnitude of microseconds in big data centers and becomes a significant component of the overall latency. However, unlike any other component, propagation delay is one aspect that can not be improved, hinting that minimizing the length of the traversal path through data centers needs to become a future direction of research.

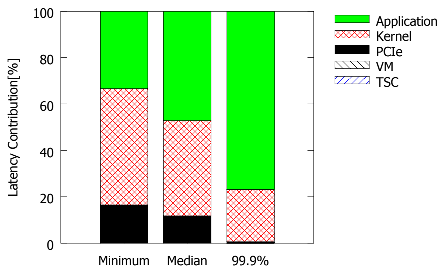

Fig. 4. End Host Latency Contribution.

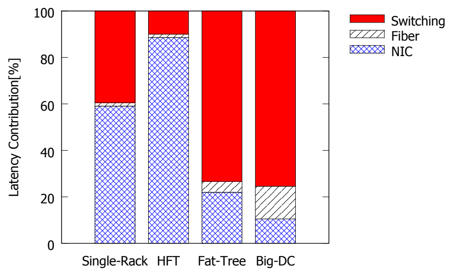

Fig. 5. Network Latency Contribution.

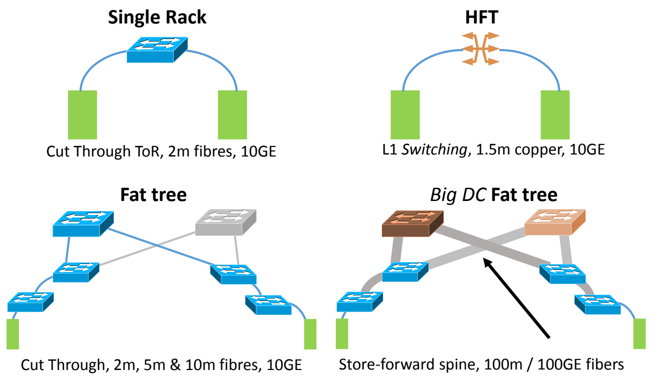
Single Rack
Cut Through ToR, 2m fibres, 10GE
HFT
L1 Switching, 1.5m copper, 10GE
Fat tree
Cut Through, 2m, 5m & 10m fibres, 10GE
Big DC Fat tree
Store-forward spine, 100m / 100GE fibers

Fig. 6. Different Network Topologies.

4 Tail Latency Results

Results in the previous section range between their stated minimum and the 99.9th percentile. However, our experiments also provide insight into heavy-tail properties of the measured latency. Such results, which are not caused by network congestion or other oft-stated causes of tail-latency, are briefly discussed in this section.

The relative scale of these tail latency cases is usefully illustrated by considering the TSC (1). The tail latency values are clearly illustrated when using the TSC experiment (§ 2.2) and all subsequent experiments using the TSC measurement.

While up to 99th percentile for the typical TSC measurements, the latency is in the order of $10\mathrm{ns}$, in both kernel and user space, TSC latencies can be in the order of microseconds or hundreds of microseconds. VMs show even greater sensitivity with higher-still outlier values. The CDF of these results is shown in Figure 7. While long latency events may be infrequent, even a single outlier event can overshadow hundreds to thousands of other operations. This is keenly illustrated in Figure 8 with a complementary CDF (CCDF) the aggregated time wasted on tail events. This graph illustrates that while only 364 out of 22G events of TSC latency in VM user space are $1\mathrm{ms}$ or longer, these events take almost 5% of the observation period.

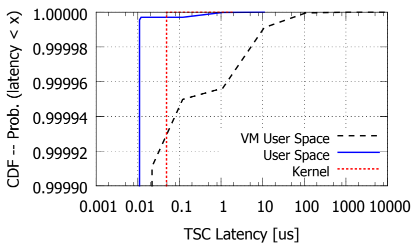

Fig. 7. CDF of TSC Tail Latency.

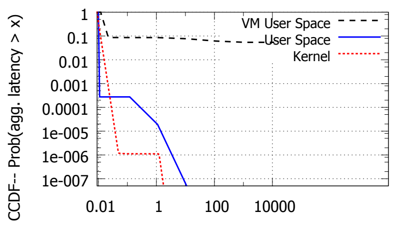

Fig. 8. CCDF of Aggregated TSC Tail Latency.

The OS kernel is a natural source of latency. While in Kernel Cold Start tests (1a) we did not find any outliers that approach a microsecond, microsecond-long gaps do occur in a TSC Kernel test (1b) run at the end of our initialization sequence. In user space (1c), gaps can reach tens of microseconds, even under our best operating conditions. Some of these events are the clear results of scheduling events, as disabling pre-emption is not allowed in user space. Experimenting with different (Linux) OS schedulers (e.g., NOOP, CFQ and Deadline) show that such events may shift in time, but remain at the same magnitude and frequency. Further, changing some scheduler parameters, e.g. CFQ's "low latency" and "Idle slice", does not reduce the frequency of microsecond-long gaps.

The most prominent cause of long time-gaps is not running an application in real time or pinned to a core. While the frequency of gaps greater than $1\mu\mathrm{s}$ does not change significantly, the latency does increase. When pinned in isolation on a CPU, 99.9th percentile of the $1\mu\mathrm{s}$-or-more gaps are less than $10\mu\mathrm{s}$. Without pinning and running in real time, over 10% of the gaps are $10\mu\mathrm{s}$ or longer, and several hundreds-of-microsecond long gaps occur every second. A pinned application sharing a core with other processes

```
while (!done) {
    //Read TSC twice, one immedately after the other
    do_rdtscp(tsc, cpu);
    do_rdtscp(tsc2,cpu2);
    //If the gap between the two reads is above a threshold, save it
    if ((tsc2 - tsc > threshold) && (cpu == cpu2))
       buffer[samples++] = tsc2-tsc; }
```

Listing 1.1. Reading and Comparing TSC - Code 1.

```
while (!done) {
    //Read TSC once
    do_rdtscp(tsc, cpu);
    //If the gap between the current and the previous reads is above a
         threshold, save it
    if ((tsc - last > threshold) && (cpu == lastcpu))
       buffer[samples++] = tsc-last;
    last = tsc;
    lastcpu = cpu; }
```

Listing 1.2. Reading and Comparing TSC - Code 2.

exhibits latency in-between the aforementioned results - which makes clear VMs are more prone to long latencies, especially when the VM is running on a single core.

A different source of latency is coding practice: Listings 1.1 and 1.2 show two ways to conduct the TSC user-space test. While Listing 1.1 measures the exact gap between two consecutive reads, it potentially misses longer events occurring between loops. Listing 1.2 overcomes this problem, but also captures gaps caused by the code itself. Consequently, Listing 1.2's minimal gap grows from 9ns to 14ns, while the maximal gap is about twofold longer. In addition, page faults lead to hundreds of microseconds latencies that can be avoided using e.g. *mlock*.

5 Discussion

This paper contributes a decomposition of the latency-inducing components between an application to the *wire*. We hope that other researchers can make use of this work to calibrate their design goals and results, and provide a better understanding of the key components of overall latency. The results are generalizable also to other platforms and other Linux kernel versions[8].

Four observations summarize the lessons learned. First, there is no single source of latency: using ultra low latency switches or NICs alone are insufficient even when using sophisticated kernel bypass options. It is only the combination of each of these efforts which may satisfactorily reduce latency experienced in a network system. Second, tail events are no longer negligible and result in two side effects: (1) latency-sensitive transactions may experience delays far worse than any performance guarantee or design for resilience (e.g. if the event is longer than retransmission timeout (RTO)) and (2) the "noise" – events well beyond the 99.9th percentile – potentially consume far more than 0.01% of the time. This calls for a change of paradigm: instead of qualifying a system

[8] Based on evaluation on Xeon E5-2637 v3, i7-6700K and i7-4770 based platforms, and Linux kernels ranging from 3.18.42 to 4.4.0-42.

by its 99.9th percentile, it may be that a new evaluation is called for; for example a system might need to meet a certain signal-to-noise ratio (SNR) (i.e. events below 99.9th percentile divided by events above it), as in other aspects of engineered systems.

Finally, in large scale distributed systems (e.g., hyper data center) the impact of the speed of light increases. When a data center uses hundreds of meters long fibers [14] and the RTT on every 100m is 1μs, the aggregated latency is of the order 10μs to 20μs. Consequently, the topology used in the network and the locality of the data become important, leading to approaches that increase networking locality, e.g. rack-scale computing. While hundred-meter long fibers can not be completely avoided within hyper-data center, such traversals should be minimized.

5.1 The Good, The Bad and The Ugly

The obtained results can be categorized into three groups: the "Good", the "Bad", and the "Ugly".
The Good are the latency contributors whose 99.9th percentile is below 1μs. This group includes the simple operations in kernel and user space, PCIe and a single switch latency.
The Bad are the latency contributors whose 99.9th percentile is above 1μs, but less than 100μs. This includes the latency of sending packets over user space+OS, entire host latency, client-server latency, RTT over 100m fibers and multi-stage network topology.
The Ugly are the large latency contributors at the far end of the tail, i.e. the "noise", contributing more than 100μs. These happen mostly on the user space and within a VM. "Ugly" events will increasingly overshadow all other events, thereby reducing the SNR. Some events outside the scope of this paper, such as network congestion, also fall within this category [7].

5.2 Limitations

This paper focuses upon the **unavoidable** latency components within a system. It thus does not take into account aspects such as congestion, queueing or scheduling effects. No attempt is made to consider the impact of protocols, such as TCP, and their effect on latency and resource contention within the host is also outside the scope.

This work has focused on commodity hardware and standard networking practices and on PCIe interconnect and Ethernet-based networking, rather than, e.g., RDMA and RCoE, reserved for future work.

6 Conclusion

Computer users hate to wait – this paper reports on some of the reasons for latency in a network-based computer system. Using a decompositional analysis, the contribution of the different components to the overall latency is quantified, and we show that there is no single overwhelming contributor to saving the end-to-end latency challenge in data centers. Further we conclude that more and more latency components, such as the interconnect and cabling, will become significant as the latency of other components continues to improve. We also conclude that the long tail of events, beyond the 99.9th percentile, is far more significant than its frequency might suggest and we go some way to quantify this contribution.

"Good","Bad", and "Ugly" classes are applied to a range of latency-contributors. While many of the "Bad" latency contributors are the focus of existing effort, the "Ugly" require new attention, otherwise performance cannot be reasonably guaranteed. Giving the "Ugly" latencies attention will require concerted effort to improve the state of instrumentation, ultimately permitting end-to-end understanding.

7 Acknowledgments

We would like to thank the many people who contributed to this paper. We would like to thank Salvator Galea and Robert N Watson, who contributed to early work on this paper. This work has received funding from the EPSRC grant EP/K034723/1, Leverhulme Trust Early Career Fellowship ECF-2016-289, European Union's Horizon 2020 research and innovation programme 2014-2018 under the SSICLOPS (grant agreement No. 644866), ENDEAVOUR (grant agreement No. 644960) and EU FP7 Marie Curie ITN METRICS (grant agreement No. 607728).

Dataset A reproduction environment of the experiments, and the experimental results, are both available at `http://www.cl.cam.ac.uk/research/srg/netos/projects/latency/pam2017/`.

References

1. Barroso, L.A.: Landheld Computing. In: IEEE International Solid State Circuits Conference (ISSCC) (2014), keynote
2. Cheshire, S.: It's the Latency, Stupid (may 1996), `http://www.stuartcheshire.org/rants/Latency.html`, [Online; accessed July 2016]
3. Guo, C., *et al.*: RDMA over commodity ethernet at scale. In: SIGCOMM '16 (2016)
4. Hemminger, S.: NetEm - Network Emulator. `http://man7.org/linux/man-pages/man8/tc-netem.8.html`, [Online; accessed July 2016]
5. Kalia, A., *et al.*: Design guidelines for high performance RDMA systems. In: USENIX ATC 16. pp. 437–450 (2016)
6. Mayer, M.: What Google Knows. In: Web 2.0 Summit (2006)
7. Mittal, R., *et al.*: TIMELY: RTT-based congestion control for the datacenter. In: SIGCOMM Comput. Commun. Rev. vol. 45, pp. 537–550. ACM (2015)
8. Nussbaum, L., Richard, O.: A comparative study of network link emulators. In: SpringSim '09. pp. 85:1–85:8 (2009)
9. Oracle: Oracle VM VirtualBox, `https://www.virtualbox.org/`, [Online; accessed October 2016]
10. Paoloni, G.: How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures. Tech. Rep. 324264-001, Intel (2010)
11. Patterson, D.A.: Latency lags bandwidth. Commun. ACM 47(10) (Oct 2004)
12. Rumble, S.M., *et al.*: It's time for low latency. In: HotOS'13. pp. 11–11. USENIX Association (2011)
13. SAP: Big Data and Smart Trading (2012)
14. Singh, A., *et al.*: Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network. SIGCOMM Comput. Commun. Rev. 45(4), 183–197 (2015)
15. Tolly Enterprises: Mellanox Spectrum vs. Broadcom StrataXGS Tomahawk 25GbE & 100GbE Performance Evaluation - Evaluating Consistency & Predictability. Tech. Rep. 216112 (2016)
16. Zilberman, N., *et al.*: NetFPGA SUME: Toward 100 Gbps as Research Commodity. IEEE Micro 34(5), 32–41 (September 2014)