Public Review for

Explaining Packet Delays under Virtualization

Jon Whiteaker, Fabian Schneider, and Renata Teixeira

This paper studies the impact of virtualization on round-trip time (RTT) measurements by conducting controlled experiments on Linux-VServer and Xen, two popular platforms for virtualization. This is an important and timely topic given the increasing use of virtualized machines in both production networks (e.g. Amazon EC2) and research testbeds (e.g. PlanetLab). The measurements are carefully designed and insightful. The results not only reinforce similar previous results but also shed more light on the potential root causes. Some interesting new findings include: (i) heavy network traffic from competing virtual machines can introduce significant delay to RTT measurements, (ii) most delay is introduced while sending packets (as opposed to receiving packets). The paper also discusses the implications of these findings and proposes a feedback based mechanism to avoid measurement bias in virtualized environment. While some of these findings may require further investigation to fully understand their root causes (e.g. by more heavily instrumenting the virtualization platforms), they are clearly useful results to keep in mind when performing RTT measurements in virtualized environment.

Public review written by
Yin Zhang
The University of Texas at Austin, USA

acm sigcomm

Explaining Packet Delays under Virtualization

Jon Whiteaker
jbw@berkeley.edu

Fabian Schneider
fabian@ieee.org

Renata Teixeira
renata.teixeira@lip6.fr

UPMC Sorbonne Universités and CNRS

ABSTRACT

This paper performs controlled experiments with two popular virtualization techniques, Linux-VServer and Xen, to examine the effects of virtualization on packet sending and receiving delays. Using a controlled setting allows us to independently investigate the influence on delay measurements when competing virtual machines (VMs) perform tasks that consume CPU, memory, I/O, hard disk, and network bandwidth. Our results indicate that heavy network usage from competing VMs can introduce delays as high as 100 ms to round-trip times. Furthermore, virtualization adds most of this delay when sending packets, whereas packet reception introduces little extra delay. Based on our findings, we discuss guidelines and propose a feedback mechanism to avoid measurement bias under virtualization.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring; D.4.8 [**Performance**]: Measurement

General Terms

Measurement, Performance

Keywords

Virtualization, Xen, VServer, Packet Delay, Timestamping

1. INTRODUCTION

Virtualization has become a popular method to share system resources among different users, services, or applications. For example, PlanetLab [25] and Measurement Lab [15] use Linux-VServer [5, 14] to enable researchers to test distributed systems and measure Internet performance. Commercially, virtualization is a popular tool for cloud computing, for instance Amazon's EC2 [1] is based on Xen [31]. In addition to measurements of Internet properties, services running in virtual environments leverage network measurements to obtain better performance. Thus, it is important to understand how performing measurements from a virtual environment can affect the results, e. g., the inferences of Internet measurements might be inaccurate when unexpected delay is added to round trip time (RTT) measurements.

In this study, we focus primarily on two virtualization implementations, Xen and Linux-VServer (called VServer in the following) because both are popular and open source. Xen is known to achieve high performance through paravirtualization [3]. Contrary to Xen, which supports different OSes running in the virtual machines (VMs), VServer only supports Linux-based OSes. Section 2 surveys the two virtualization techniques and describes their design differences.

While the performance of virtualization schemes has been studied extensively with regards to achievable bandwidth, throughput, and delay [3, 7, 13, 29], there is less work on the effects of virtualization on network measurements [27, 28]. Despite many studies reporting increased round trip times (RTT) and packet loss due to virtualization [27–29] the causes are not well understood because the effects have been observed in deployed systems.

In an operational system such as PlanetLab, it is hard to fully control competing VMs, and visibility into other VMs is limited. Therefore we setup a controlled experimentation environment (see Section 3) with full control and visibility over both the system and the usage of the competing VM to identify the causes of increased packet delays under virtualization. We first determine factors that contribute to increased delays by exploring different workload cases (e. g., CPU, disk, or network load) on other VMs competing with our VM performing RTT measurements. The controlled environment allows us to directly attribute variations in the measurement to the individual parameters that we alter. Second, we locate where the delay is added by comparing timestamps from different locations (e. g., user- and kernel-space) along the packet sending and receiving path within the virtualized system. The measured RTTs are composed of several delays, e. g., transmission delay or propagation delay, but also queuing delay at various network elements and end hosts. Network measurements usually include all of these delays. However, for end to end measurements, most tools and researchers assume that local processing and queuing delays are negligible and interpret their results without considering local delays. We show that this assumption does not hold under virtualization, and quantify the error.

As presented in Section 4 we experience extra delays added by the host of up to 100 ms, which is on the order of round trip propagation delays across the Atlantic ocean. The stress cases with the largest impact on delays are network load (via a bulk TCP transmission), full CPU utilization, and context switching (via heavy I/O activity). Pinpointing the source of the delay we find that for both VServer and Vanilla Linux most time is spent for moving packets between kernel and application. For Xen, the most severe delay is added while the packet traverses `dom0`. However, cross-checking our results with OpenVZ [21], another virtualization technique similar to VServer, suggests the cause for Xen's severe delays may be due to the use of virtual interfaces as opposed to strictly the method of virtualization. Moreover, investigating the direction in which the delay is added, we determine that sending packets adds most of the delay, while receiving adds almost no delay.

After reviewing related work in Section 5, we discuss how to mitigate the increased delay in Section 6. We endorse the suggestion of Spring et al. [28] to rely on kernel timestamps and Sommers and Barford's `MAD` tool [27] for sending precisely timed packets.

2. VIRTUALIZATION BACKGROUND

Virtualization at its core provides an intermediate software layer that manages multiple software instances, abstracting away the hardware and other running code. For our purposes, we will only consider virtualization techniques that offer isolation at least at the operating system level. In virtualization, the software layer responsible for providing isolation is called the virtual machine monitor (VMM). The VMM manages multiple virtual machines (VMs), each containing a "guest" OS. Depending on the implementation, the VMM has different responsibilities and handles hardware access in different ways.

2.1 Xen

Xen achieves virtualization via paravirtualization. Paravirtualization requires the guest OS kernel to rewrite privileged system calls to use instead an API offered by the VMM, which significantly increases performance from full virtualization. Xen's VMM, known as the Xen hypervisor, offers complete hardware virtualization for OS guests. Access to the underlying hardware by guest VMs (known as Domain Us or `domUs`) is facilitated through a special guest, known as Domain 0, or `dom0`. A good example of this process is the implementation of networking in Xen. For each interface seen by a guest OS in its `domU`, a virtual interface exists in `dom0` which connects to one of the physical interfaces on the machine. Thus, a packet sent from a `domU` travels first to `dom0` via a virtual interface, then to the hypervisor via the rewritten system calls in the `dom0` kernel, and finally onto the wire.

2.2 VServer

VServer drops the idea of an external VMM, instead it modifies the Linux kernel to support virtualization. This kernel runs at all times, which means that all guests must run Linux. The virtualization mechanism isolates different guests by using an advanced `chroot` like mechanism, switching between different containers, each containing a complete OS environment [26]. This technique is known as OS-level virtualization. While VServer is limited to Linux-based guests, it offers significant performance gains. The only virtualization overhead comes from switching between the containers; once the switch is complete, the guest runs as if it was the host with direct access to hardware. VServer outperforms non OS-level virtualization solutions in achievable bandwidth since it runs essentially without a VMM and offers direct access to the underlying hardware [4].

3. MEASUREMENT METHODOLOGY

This section first describes the testbed we build for our controlled experiments. Secondly, we describe how we infer delays by measuring RTTs. Lastly, we establish a set of stress cases that can potentially increase the RTTs.

3.1 Testbed

The measurement testbed consists primarily of three components: The virtualized system under test (SUT), the measurement target, and a machine containing an Endace DAG Network Monitoring Card [8] (see Figure 1). The SUT and target are connected to each other via Ethernet, and the DAG card is connected in-line. Ideally, we would have installed the card on the virtualized machine; however, the Endace drivers are unable to operate under virtualization (specifically Xen). Furthermore, we found that operating the DAG card under heavy traffic can introduce a significant load to the machine which could influence the results themselves. As such, we installed the card into the remote PC used to respond to active measurements. The DAG has its own clock with 15 nanosecond precision and passively captures packets seen on the wire. The DAG connects in-line with a fixed copper connection between its two RJ45 ports so no processing delays occur as packets pass through.

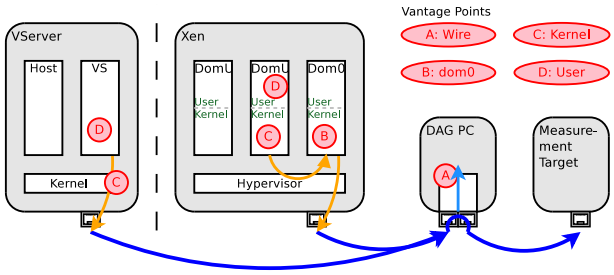

Figure 1: Measurement Setup

The SUT has a 1.8 GHz Pentium 4 processor and 2 GB RAM, similar hardware as used by Sommers and Barford [27]. The SUT has a Gigabit Ethernet card, however we configured it to 100 Mb/s to be compatible with the DAG, which only supports FastEthernet. The rest of the setup consists of Gigabit devices capable of easily sustaining a full duplex 100 Mb/s load. The base system and virtual machines consist of the minimum software base of the GNU Linux based Debian distribution [6] running the 2.6.26 Linux kernel and a handful of additional packages necessary to perform the measurements. The minimal install base ensures that nothing other than the VMM or other VMs can be responsible for differences seen under virtualization.

3.2 Measurements

We base our experiments around a modified version of the standard `ping` tool, which estimates RTTs to a remote host using ICMP packets. This measurement involves sending an ICMP Echo Request packet, receiving an ICMP Echo Reply packet, and recording the elapsed time between the two events. By using `ping` we will measure any delays in sending the first packet, as well as any delays in receiving the second packet.

Instead of reporting the RTT only, our `ping` version reports the send and receive timestamps used to calculate the RTT. We will refer to our modification as `tping` in the remainder of the paper. Furthermore, our version utilizes the `glibc gettimeofday()` system call for timestamping received packets instead of the default behavior of using `ioctl()` and `SIOCGSTAMP` to retrieve kernel timestamps for incoming packets[1]. A `tping` measurement run consists of a series of 200 ICMP Echo Request packets padded to 100 bytes sent from the SUT to the measurement target with 250 ms between each packet. Each ICMP Echo Request received on the target then generates an ICMP Echo Reply back to the SUT. We use a gap of 250 ms because this is sufficiently greater than any delays experienced after the packet arrives at the network card of the SUT.

We record the same events from several different vantage points (see Figure 1) to understand the contribution of each part of the system. Using the DAG, we measure the timestamps of packets on the *wire* as ground truth. This includes the reception and response generation at the measurement target. With `bpf` and `libpcap` we record timestamps in *kernel-space*. Since packets in Xen pass through the privileged domain before reaching the guest OS, we measure kernel-space from both `domU` and `dom0`. Finally, we use the timestamps recorded by `tping` to measure delays in *user-space*. While it is common knowledge that kernel-space timestamps are more accurate than user-space time-stamps, we are interested in how these values fluctuate depending on the configuration of the system. Unfortunately, because the DAG and SUT are

[1]A behavior achieved in the `ping` tool with the `-U` option

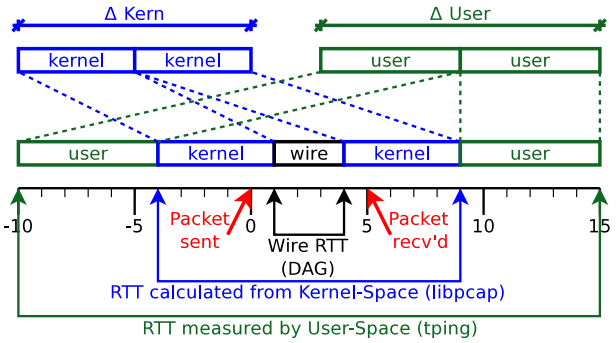

Figure 2: Round Trip Times (RTTs)

in different machines, we cannot directly compare timestamps. We compute the RTT at each vantage point by subtracting the ICMP Echo Request timestamp from the ICMP Echo Reply time-stamp (see Figure 2). Then, by comparing the difference in RTTs computed at each vantage point, we compute bi-directional delay introduced at each layer, called ΔRTT[2] in the following. We use the ΔRTTs computed at each vantage point as our primary source of comparison.

However, when comparing RTTs we cannot distinguish between delays due to sending or receiving. To compare the timestamps directly we need to synchronize the clocks at each vantage point. Initially we attempted to do this by synchronizing on the first packet of each measurement. This approach failed for two reasons: first of all, if we synchronize the clocks based on packet arrival, we are effectively zeroing out part of the delay that we wish to measure; secondly, since no two clocks are identical, we see a drift between the clocks that skews results. The first problem is easy to solve by synchronizing the machine *at rest*, i. e., before we increase hardware utilization. This method only loses a small factor of the delay. The second problem is more difficult to solve. We start by sending a set of packets at the beginning and end of each measurement when the machine is *at rest*. Next, we set all timestamps as relative to the recorded time-stamp of the first *at rest* packet. We then graph the results of the difference between the recorded values on the SUT and the DAG. Since we observe a linear drift, we perform a least squares fit on the leading and trailing *at rest* packets. Finally, we transform the graph on this line, and can now compare the delay for each direction individually.

3.3 Variables

The main metric of our controlled experiments is the OS configuration. We focus on two virtualization solutions: *VServer* , and *Xen*. As a baseline, we first test the system using *Vanilla Linux*, i. e., the pure Linux installation without any virtualization. In all tests, we use the 2.6.26 Linux kernel base with patches for VServer 2.3.0.26 and the Xen 3.2.1 hypervisor respectively. Preliminary experiments with newer versions, notably Xen 4.0, running on Linux kernel 2.6.32 confirm our high-level findings.

The variance in the measurements is much more likely to show up when the system itself is being taxed, so we artificially increase hardware utilization in several cases. Again, as a basis for comparison we perform measurements while the machine is *at rest*. To introduce strain on the hardware, we use the open source tool `stress` [30], which executes particular system calls as fast as it can in a loop until killed. We use `stress` for the following stress cases: *CPU* stress loops on `sqrt()` calls; *HD* stress alternates between `write()` and `unlink()` calls; *I/O* stress loops over the `sync()` system call; and *memory* stress spins on consecutive `malloc()` and `free()` calls. We introduce network activity using the open source tool `iperf` [18]. We use two TCP connections (each sending data in different directions up to the link capacity) to stress the *network* on the same interface used by `tping`. Since the network operates at 100 Mb/s this leads to a maximum load of 100 Mb/s in each direction.

For all experiments, except Vanilla Linux, *four* guest VMs are instantiated. One guest executes the specified stress case, one performs the measurements, and the remaining two are idle. Although not presented here, we performed our experiments using up to four guest VMs executing stress, with each VM executing up to four stress threads. The results of these tests do not deviate significantly from those presented. Therefore, we are confident that our findings are not intrinsic to our methodology.

4. RESULTS

Table 1 presents the median and 99th percentile of the RTT measurements for all three environments and all stress cases at different vantage points. In the Δ columns, we report the contribution of every layer at the host. The delay added by all layers, shown in the total columns, does not necessarily add up to the total because we report percentiles only. Note that for all experiments the time spent on the wire (including the response generation) is already subtracted from the reported delays. We first present our baseline assessment of RTTs in vanilla Linux, followed by that of VServer, and then Xen. After that we use a third virtualization technique, OpenVZ, to decouple the delay overhead of having a VMM versus virtual interfaces. Before we close this section with a discussion of implications of our findings, we dissect the RTTs into send and receive delays for all environments.

4.1 Vanilla Linux

For most stress cases the kernel of Vanilla Linux adds no more than 134 µs to RTTs. The notable exception is *network* stress where the delay of kernel-space RTTs can jump up to 6 ms. RTTs obtained in user-space with `tping` reach up to 120 ms, yielding markedly worse results.

We deduce that user-level timestamping can result in gross RTT overestimation even without virtualization. As expected, this effect is significantly reduced when using kernel-level timestamps, better reflecting the actual RTT on the wire. Nevertheless, kernel-level timestamps still exhibit delays of up to 6 ms, due to queuing at the network card or its driver. Since this delay is not constant, performing a number of tests and selecting the minimum RTT helps to achieve better results. If the machine is running vanilla Linux, experimenters also have more visibility of and control over the state of the system. Using the information on resource utilization of the machine (e. g., *CPU*/*network* load) allows experimenters to infer possible impact on the measured timestamps. This approach is already leveraged by some measurement tools such as `grenouille` [9], which tests network utilization before conducting experiments.

4.2 VServer

VServer shows results in the same order of magnitude as Linux for all stress cases. Interestingly, for *CPU* stress, VServer even performs slightly better than Linux. For heavy network traffic, the kernel-space RTTs show delays of 6 ms and user-space RTTs show delays up to 98 ms. These results agree with the findings of Sommers and Barford [27], who observe similar delays in the RTTs, particularly for the *network* stress case. However, our results differ

[2] ΔKern = kernel RTT − wire RTT; Δ`dom0` = `dom0` RTT − wire RTT; Δ`domU` = `domU` RTT − `dom0` RTT; ΔUser = user RTT − kernel/`domU` RTT;

Table 1: Delays (in ms) under different workloads on systems running vanilla Linux, VServer, and Xen.

	Stress Case	Vanilla Linux			VServer			Xen			
		ΔKern	ΔUser	Total	ΔKern	ΔUser	Total	Δdom0	ΔdomU	ΔUser	Total
median	At Rest	.051	.082	.134	.032	.036	.068	.150	.298	.144	.595
	CPU	.049	.075	.124	.035	.033	.069	.148	.464	.159	.861
	I/O	.049	.081	.129	.039	.059	.098	.140	4.221	.155	4.521
	HD	.055	.099	.154	.062	.121	.184	.137	.294	.141	.584
	Memory	.076	.143	.220	.076	.141	.217	.150	.320	.149	.644
	Network	.373	9.374	14.124	.377	24.633	25.008	5.924	5.079	.222	11.429
99th percentile	At Rest	.063	.243	.293	.048	.174	.214	.267	.584	.304	.889
	CPU	.067	.156	.208	.045	.132	.193	.257	32.204	.342	32.539
	I/O	.060	.178	.233	.050	.114	.160	.267	56.240	.355	56.566
	HD	.134	.435	.501	.192	.349	.454	.284	27.546	.378	27.925
	Memory	.089	.538	.605	.089	.518	.607	.263	.512	.338	.948
	Network	6.039	89.527	92.331	6.042	83.860	84.824	8.486	19.655	.576	26.293

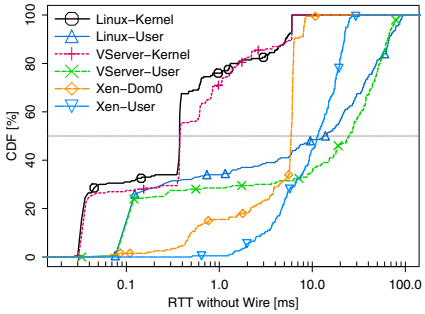

Figure 3: CDF of user- and kernel-space RTTs under network stress.

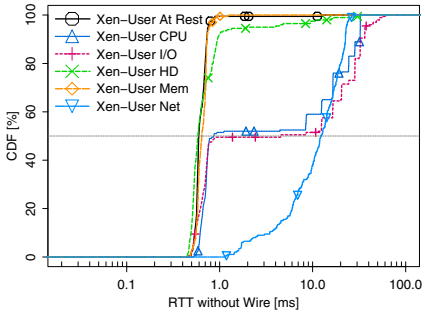

Figure 4: CDF of user-space RTTs for Xen under various stress cases.

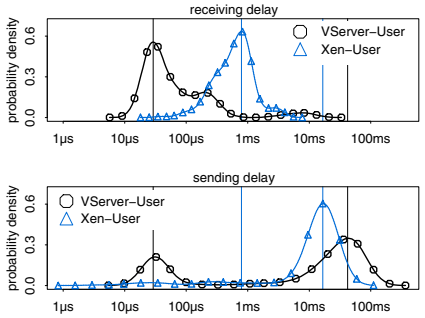

Figure 5: PDFs of send/receive delays for VServer and Xen (network stress).

from those of Spring et al. [28], in that we observe greater differences between kernel and application level timestamps (ΔUser) when *network* stress is introduced. The difference may be because of the load of the machines during their experiments, which is not documented. Figure 3 shows the CDF (cumulative distribution function) of the measured user- and kernel-space RTTs excluding the time on the wire under *network* stress for all environments. Linux and VServer show similar distributions with increased RTTs in VServer's user-space over Linux's user-space for roughly 10% of the probes.

As mentioned earlier with VServer it is not easy to monitor resource consumption or running processes of other guests. Especially with PlanetLab's VNET system [11], where each guest can only capture its own traffic, a guest cannot know that the delay comes from the host.

4.3 Xen

The results in Table 1 show that *at rest*, Xen causes delays three to four times higher than those of vanilla Linux and VServer. The kernel-space RTTs obtained from `dom0` expose delays of up to 284 µs. The timestamps from `domU` kernel- and user-level add a constant of roughly 300 µs and 150 µs, respectively.

Figure 4 shows the CDF of user-space RTTs (corresponds to total column in Table 1) for various stress cases. We can easily identify two classes of delays: those smaller than 1 ms and those larger than 10 ms. This bimodal distribution suggests that VM scheduling occurs in 10 ms time-slices (compare also the staircase effect for *CPU* and *I/O* stress cases) which matches the time-slices used in Xen's default credit scheduler [19]. The stress cases mainly differ in how many probes are affected by being scheduled in another time-slice.

CPU stress increases RTTs by up to 32 ms, and *I/O* stress up to 56 ms. Interestingly, the measurements for `domU` and user-space timestamps are nearly identical in these cases, suggesting that the packet is passed on and both timestamps are generated in the same VM time-slice. Note that these delays are not seen at all (RTTs < 300 µs) when measured in `dom0`, showing that the majority of the delay is introduced as the packet passes through `dom0`.

Once again, the most interesting results occur under heavy TCP traffic. Similar to the previous scenarios, the `domU` and user-level timestamps mirror each other, but unlike before the gap between `dom0` timestamps is much more pronounced. We observe delays of up to 25 ms in this scenario. Network traffic induced delays are larger than kernel-space RTTs of VServer and Linux, but less than their user-space RTTs. Since there is virtually no delay added by the user-space (also compare Figure 3), the delay must originate in `dom0`. As sending/receiving packets is a privileged operation in Xen, the extra delays could come from the overhead of additional context switching and queuing between the `domU`s, `dom0`, and the hypervisor.

Since Amazon uses a heavily modified version of Xen for EC2 [2], it is possible the same effects would not appear there. However, our results align well with Wang et al.'s [29] findings on RTT variance, suggesting that the same underlying issues observed in our experiments apply to EC2.

4.4 Virtual interfaces

Studying the systems under *network* stress revealed strong differences between Xen and VServer. Furthermore *network* stress affects RTTs across all setups. While the VServer user-space RTTs have a heavier tail than those of Xen, the kernel-space RTTs of VServer are much smaller than Xen's. One could easily attribute

this disparity to the type of virtualization: Xen implements *paravirtualization*, causing extra overhead for a VMM (the Xen hypervisor) managing multiple kernels. Implementing *OS-level virtualization*, VServer requires less overhead due to running only a single kernel.

However, it is possible that the disparity results from the method of routing traffic to the guests, which is not coupled to the type of virtualization. VServer routes traffic to guests by modifying the underlying networking stack to mark packets for the appropriate container [26]. This is not changed when using the VNET system [11]. On the other hand, Xen routes traffic by creating virtual interfaces (VIFs) in `dom0` that mirror the interfaces seen by `domUs` [3] in combination with software Ethernet bridges. In these VIFs and bridges additional queuing can occur.

OpenVZ combines OS-level virtualization with bridges and virtual interfaces [12]. This turns OpenVZ into a welcome opportunity to determine if the source of the increased delays is due to overhead from the Xen hypervisor or due to virtualized routing. Therefore, we repeat all tests running the OpenVZ modified 2.6.26 Linux kernel to cross check our results. We find the performance of OpenVZ to be nearly identical to that of VServer, except in the case of *network* stress. The CDF distributions of RTTs for Xen and OpenVZ when captured from VIFs (i.e. `domU`) are nearly identical, with OpenVZ even performing slightly worse than Xen. Considering that OpenVZ experiences this with a single kernel this suggests that in the case of heavy network activity the method of routing traffic to guests is a more important factor than the overhead of the underlying virtualization technique. While this is difficult to confirm under our current instrumentation, as part of our future work we would like to compare the performance of Xen's default networking stack to Open vSwitch [20], a software switch designed explicitly for virtualization and reported to perform better than the default Linux software bridge [24].

4.5 Sending vs. receiving delay

So far, our results focused on complete RTTs and did not dissect the RTTs into delay per direction. For this we apply the synchronization scheme explained in Section 3.2. We show the results of the breakdown for VServer and Xen under *network* stress in Figure 5. We find that the accuracy of *receiving* packets rarely exceeds 300 µs in VServer and Linux, and 2 ms for Xen. The vast majority of the delay occurs when *sending* packets. This observation is in stark contrast to the findings of Sommers and Barford [27] and Spring et al. [28] which both saw receiving delays greater than sending delays. A possible explanation for the differing results is that their observed delays are caused primarily by VM scheduling whereas the *network* stress implies packet queuing as a main reason for delays. It is possible that queuing upon sending causes the packet to miss its time-slot. Figure 5 supports this theory, given that Xen's default credit scheduler allows domains to run between 10-30 ms and the peak of Xen's recorded sending delay lies squarely in this range [19].

It is our understanding that *network* stress is not an uncommon scenario in virtualized environments. Any tool that measures TCP achievable throughput (as `iperf` does) would create the same workload, as could several guests performing network transfers. In fact, such a situation could yield results with even higher variance as the link utilization fluctuates. We also note that because the most significant delays are due to sending, the `ping` tool performs equally poorly under *network* stress when using kernel-level timestamps for ICMP Echo Reply packets (the default behavior in Linux).

4.6 Implications

The increased delay under virtualization has an effect on both applications running in a virtualized environment and measurements to infer Internet properties carried out from virtualized environments.

Imagine a distributed application (e. g., delay sensitive P2P live video streaming) running on virtualized nodes. To provide best performance to its users this service monitors the RTT to all of its peers and decides if a peer is good or bad based on these measurements. Connections to bad peers are exchanged for connections to better peers. Now if the source of an increased RTT is the node itself, all connections will be affected, thus changing to another peer only adds overhead and service interruption/degradation. Even worse considering the variations in the RTTs due to different loads, such a P2P system could suffer from route oscillation.

Another effect of increased delay on applications is on TCP's RTT estimation, as already pointed out by Wang et al. [29]. Yet, the effect is likely not that severe since TCP RTT estimation relies on kernel timestamps, which we found to be fairly accurate except for Xen. Moreover, the resulting congestion window increase from the increased RTTs does not necessarily have a negative effect since higher RTTs will cause less throughput when the congestion window is at limit. TCP's reaction of decreasing the sending rate can actually be desired as it also reduces the major cause of additional delays, namely network load.

Let us now turn to the implications for Internet property measurements. Network measurements often rely on being able to send out accurately spaced packet trains. Sommers and Barford [27] as well as Wang et al. [29] discuss how increased and varying delays can influence tools for bandwidth and loss estimation. Thus, we refer to those papers for details.

5. RELATED WORK

Xen's performance has been well documented since its inception [3, 4, 7, 10, 16, 22] . However, when it comes to network measurements, the focus is usually about maximizing bandwidth. Egi et al. [7] test a number of different Xen configurations to determine the settings for optimal bandwidth. Menon et al. [16] showed how to increase the maximum bandwidth in Xen by making improvements along the network path from a guest to the physical device. Gupta et al. [10] improve isolation and fairness by adding a network scheduler, but still focus on network throughput as an evaluation. The paper that comes closest to evaluating the state of network measurements from Xen is an evaluation of Amazon's EC2 performance by Wang et al. [29]. While bandwidth is still a primary metric, they also measure RTTs and note issues in the results. However, without direct access to the machines and underlying software, it is more difficult to determine the actual cause of the discrepancies.

Chun et al. [5] argued that due to higher performance offered via OS-level virtualization, VServer was chosen for PlanetLab. Some years later Spring et al. [28] performed a thorough analysis of the myths and complaints on PlanetLab. Among other issues, they discussed accuracy of latency measurements and concluded that accurate network delay can be obtained from kernel timestamps. Sommers and Barford [27] expanded on another issue reported by Spring et al. [28], i. e., sending precise packet trains. After confirming the problem by placing an Endace DAG card in line with one of their PlanetLab nodes, they developed and evaluate a Linux kernel module, named `MAD`, that facilitates sending probes at precise intervals. Both studies base their advice on observations of PlanetLab nodes, whose utilization was neither recorded nor influenced.

Yet, monitoring deployed PlanetLab nodes might pose more stress in terms of VM switching, whereas we focus on the impact of a single competing VM.

6. DISCUSSION

The first lesson that our study reinforces is that timestamps acquired via the `gettimeofday()` system call do not capture the effective sending or receiving time of a packet. This inaccuracy can be due to scheduling effects or high system load. If available, we suggest using kernel timestamps, which can be obtained by using `libpcap` for packet capture. This result is fairly self-evident, and despite the differences in our results from those of Spring et al. [28] regarding application-level timestamps, they also recommend using kernel timestamps for the best accuracy. If accessible, monitoring the system to detect concurrent usage and other factors that could affect measurements can help to avoid bad measurement samples. PlanetLab already has such a monitoring system in place with CoMon [23]. Experiments could log CoMon output while performing measurements, then decide on the validity of the results. Enhancing measurement tools to use kernel-level timestamps and accounting for concurrent usage is certainly worthwhile.

However, under network load, even kernel-level timestamps are affected. In VServer, we believe the `MAD` [27] tool would experience the same queuing issues if running as a user process, but might alleviate the delays if running as a kernel module. A possible alternative solution for Xen is the stricter resource scheduling suggested by Gupta et al. [10], where network activity is shared equally between `domUs`.

Unfortunately, using kernel-space timestamps does not solve the problem for systems such as Xen and OpenVZ that can have significant differences between obtained timestamps and effective send/receive times. No system offers any interface to query the sending timestamp as seen by the physical network interface card. Hence, investigating if hardware or network interface drivers can be extended to report such timestamps is logical next step.

7. ACKNOWLEDGMENTS

This work is part of the Nano Data Centers [17] project, which is supported by the European Community's Seventh Framework Programme (FP7/2007-2013) no. 223850.

8. REFERENCES

[1] Amazon ec2. `http://aws.amazon.com/ec2/`.

[2] Amazon web services: Overview of security processes. Tech. rep., Amazon, 2008.

[3] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proc. SOSP '03* (2003), pp. 164–177.

[4] CAMARGOS, F. L., GIRARD, G., AND LIGNERIS, B. D. Virtualization of linux servers: A comparative study. In *Proc. 2008 Linux Symposium* (2008), vol. 1, pp. 63–76.

[5] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., AND BOWMAN, M. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review 33*, 3 (2003), 3–12.

[6] Debian. `http://www.debian.org/`.

[7] EGI, N., GREENHALGH, A., HANDLEY, M., HOERDT, M., MATHY, L., AND SCHOOLEY, T. Evaluating xen for router virtualization. In *Proc. ICCCN* (2007), pp. 1256–1261.

[8] ENDACE MEASUREMENT SYSTEMS. `http://www.endace.com/`.

[9] Grenouille. `http://www.grenouille.com/`.

[10] GUPTA, D., CHERKASOVA, L., GARDNER, R., AND VAHDAT, A. Enforcing performance isolation across virtual machines in xen. In *Proc. Middleware '06* (2006), pp. 342–362.

[11] HUANG, M. Vnet: Planetlab virtualized network access. Tech. Rep. PDN-05-029, PlanetLab Document, 2005.

[12] KOLYSHKIN, K. Virtualization in linux. Tech. rep., 2006.

[13] LEE, S.-J., SHARMA, P., BANERJEE, S., BASU, S., AND FONSECA, R. Measuring bandwidth between planetlab nodes. In *Proc. PAM* (2005), vol. 3431.

[14] Linux-vserver. `http://linux-vserver.org/`.

[15] Measurement lab (m-lab). `http://www.measurementlab.net/`.

[16] MENON, A., COX, A. L., AND ZWAENEPOEL, W. Optimizing network virtualization in xen. In *Proc. USENIX ATC* (2006), pp. 15–28.

[17] Nanodatacenters. `http://www.nanodatacenters.eu/`.

[18] NLANR. Iperf. `http://iperf.sourceforge.net/`.

[19] ONGARO, D., COX, A. L., AND RIXNER, S. Scheduling i/o in virtual machine monitors. In *Proc. VEE '08* (New York, NY, USA, 2008), ACM, pp. 1–10.

[20] Open vswitch. `http://openvswitch.org/`.

[21] Openvz. `http://www.openvz.org/`.

[22] PADALA, P., ZHU, X., WANG, Z., SINGHAL, S., AND SHIN, K. G. Performance evaluation of virtualization technologies for server consolidation. Tech. Rep. HPL-2007-59, HP Laboratories Technical Report, 2007.

[23] PARK, K., AND PAI, V. S. Comon: a mostly-scalable monitoring system for planetlab. *ACM SIGOPS Operating Systems Review 40*, 1 (2006), 65–74.

[24] PFAFF, B., PETTIT, J., KOPONEN, T., AMIDON, K., CASADO, M., AND SHENKER, S. Extending networking into the virtualization layer. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)* (Oct 2009).

[25] Planet-lab. `http://www.planet-lab.org/`.

[26] SOLTESZ, S., PÖTZL, H., FIUCZYNSKI, M. E., BAVIER, A., AND PETERSON, L. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review 41*, 3 (2007), 275–287.

[27] SOMMERS, J., AND BARFORD, P. An active measurement system for shared environments. In *Proc. IMC'07* (2007), pp. 303–314.

[28] SPRING, N., PETERSON, L., BAVIER, A., AND PAI, V. Using planetlab for network research: myths, realities, and best practices. *ACM SIGOPS Operating Systems Review 40*, 1 (2006), 17–24.

[29] WANG, G., AND NG, T. S. E. The impact of virtualization on network performance of amazon EC2 data center. In *INFOCOM* (2010).

[30] WATERLAND, A. stress. `http://weather.ou.edu/~apw/projects/stress/`.

[31] Xen. `http://www.xen.org/`.