Inferring the Network Latency Requirements of Cloud Tenants

Jeffrey C. Mogul and Ramana Rao Kompella – *Google Inc., Mountain View, CA*

Abstract

Cloud IaaS and PaaS tenants rely on cloud providers to provide network infrastructures that make the appropriate tradeoff between cost and performance. This can include mechanisms to help customers understand the performance requirements of their applications. Previous research (e.g., Proteus and Cicada) has shown how to do this for network-bandwidth demands, but cloud tenants may also need to meet latency objectives, which in turn may depend on reliable limits on network latency, and its variance, within the cloud providers infrastructure. On the other hand, if network latency is sufficient for an application, further decreases in latency might add cost without any benefit. Therefore, both tenant and provider have an interest in knowing what network latency is good enough for a given application.

This paper explores several options for a cloud provider to infer a tenants network-latency demands, with varying tradeoffs between requirements for tenant participation, accuracy of inference, and instrumentation overhead. In particular, we explore the feasibility of a hypervisor-only mechanism, which would work without any modifications to tenant code, even in IaaS clouds.

1 Introduction

Tenants of Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) cloud providers rely on these providers to provide network infrastructure to connect the various components of their applications. In such environments, both providers and their tenants face critical tradeoffs between cost and performance. Underprovisioned applications fail to meet their System Level Objectives (SLOs); overprovisioned applications add costs for either or both parties.

While some tenants deeply understand their own network performance needs, many do not. Some lack the technical sophistication to know how their SLO depends on various aspects of infrastructure performance; others just want to spend their efforts focusing on other challenges (e.g., adding new features). The relationship between application performance and network performance can be quite complex [14, 24, 26]. Also, cloud providers usually hide the details of the underlying infrastructure, so that they can evolve it without being locked into outdated design decisions (and to avoid revealing their trade secrets). Enterprises that are accustomed to designing around specific network hardware structures cannot easily transfer these experiences to opaque cloud platforms.

In addition, one oft-stated justification for the use of cloud platforms is that they support rapid flexing of resources in response to varying demands. These variations in demand and in the amount of computational resources allocated to a tenant can complicate the relationship between SLOs and network provisioning.

Given these impediments to client-specified network performance requirements, a provider can do a better job of optimizing both customer happiness and infrastructure utilization if the provider can estimate the customer's actual demand, rather than requiring the customer to specify it explicitly. Accurately and dynamically choosing the right level of network provisioning could become part of the "undifferentiated heavy lifting"[1] that a cloud provider offers to its tenants.

Previous research (e.g., Proteus [25] and Cicada [17]) has shown how to do this for network-bandwidth demands. Proteus profiles the bandwidth demands of MapReduce jobs, with the goal of overlapping their use of a shared network during future runs of the same jobs. Cicada uses VM-to-VM bandwidth measurements of long-running cloud applications to drive a machine-learning predictor of future bandwidth requirements, including time-varying requirements typical of user-facing applications; these predictions can then be used for admission control and/or VM placement decisions, to maximize the number of tenants whose demands can be met on a given infrastructure.

However, many cloud tenants might also need to meet service-level latency objectives, that in turn depend on reliably low network latency within the provider's infrastructure. User-facing applications, in particular, typically must meet "tail-latency" SLOs [10], and can suffer significant business impact from high or highly-varying network latencies. On the other hand latency decreases below a certain threshold might add cost without any substantial application-level benefit. Therefore, both tenant and provider have an interest in knowing what network latency is "good enough" for a given application.

Existing bandwidth-prediction methods cannot directly reveal a cloud application's network latency requirements. In this paper, we explore several options for a cloud provider to infer a tenant's network-latency demands. These options vary in their tradeoffs between requirements for tenant participation, accuracy of inference, and instrumentation overhead. In particular, we explore the feasibility of a hypervisor-only mechanism, which would work without any modifications to tenant code, even in IaaS clouds.

[1] A phrase from Jeff Bezos [18].

2 Motivation and background

The thesis of this paper is that cloud providers should infer the network latency demands of their tenants. This depends on several premises:

1. **Cloud application performance (sometimes) depends on internal network latency**: This premise is well-established, especially for applications where "tail latency" matters [2, 3, 10, 13].
2. **Network latency within a cloud infrastructure can vary significantly**: Several previous papers have reported measurements of cloud-internal network latencies. For example, Wang & Ng reported large variations in EC2 network latencies [23, §IV-B]; Barker & Shenoy also reported EC2 latencies, but without formally quantifying the variations [6]. We are unaware of more detailed or recent studies ([6, 23] are both from 2010), so in §3 we report our own measurements.
3. **Cloud providers can control latency**: A provider can use one or more of several known mechanisms to control latency. For example, High-bandwidth Ultra-Low Latency (HULL) [3] uses a form of active switch-queue management, DCTCP [2], and sender-based packet pacing, trading a small amount of bandwidth for "dramatically" lower average and tail latencies. Silo [15] improves on HULL by combining sender-based packet placing and careful VM placement, avoiding the need for either DCTCP (i.e., guest modifications) or switch modifications. Silo also supports a combination of guarantees for both bandwidth and latency. Providers can also leverage enhanced hardware, such multiple-queue QoS support available in many switches, by setting DSCP headers in packets to assign latency-sensitive traffic to lightly-loaded queues [13]. One might also hope for deployment of NICs, such as SENIC [21], with scalable hardware support for pacing.
4. **Tenant developers do not understand their own network latency requirements**: Systems such as Silo assume that the latency-guarantee requirements are known to the tenant. Similarly, systems such as Oktopus [5] assume that tenants know what bandwidth guarantees to request for their applications. However, our discussions with cloud operators suggest that they do not believe that most tenants actually know what network guarantees to ask for; therefore, most clouds do not provide any way to request network-performance guarantees. (EC2 does support HPC clusters with low-latency networking, but coupling low-latency networking with the other HPC features yields an inflexible tradeoff between resources.)

Based on these premises, we argue that a cloud provider *should* infer network-latency demands for its tenants. What remains to be shown is whether a cloud provider *can* infer latency demands. We address this in §4.

2.1 Who benefits from latency inference?

Who would benefit from inference of latency demands, and how?

Providers presumably want to find a Pareto-optimal operating point, which maximizes both a provider's own profit and that of its tenants (since happy tenants are more likely to continue as customers). Different tenants are likely to have different tradeoffs between latency demands and bandwidth demands, so a provider that knows these demands can allocate its resources so as to balance them appropriately, while making the most efficient use of its infrastructure (i.e., serving as many tenants as possible using a given set of resources).

A provider that knows whether a tenant's performance problems are attributable to network latency can also do better at responding to complaints about performance.

Tenants want to understand if and how they need to improve their applications or purchase more resources. Latency inference can help them understand the relationship between network latency and their application's SLO. (Wu *et al.* have proposed Virtual Network Diagnosis as a Service [24] as a step in this direction).

2.2 Using inferred latency demands

A provider can use several different methods for balancing resource allocations between tenants. It can do admission control, to avoid overloading its infrastructure. It can change VM placement (as in Oktopus and Silo) to improve network locality or reduce interference. It can reduce the sending pace of latency-insensitive VMs, so as to provide better latency to latency-sensitive VMs (as in HULL and Silo). It can change DSCP settings, to shift flows between switch queues. It can use this information for planning infrastructure upgrades and expansions.

A provider may want to to adjust the relative prices of VMs, bandwidth guarantees, and latency guarantees, to optimize the profit it makes from its resources, while offering customers an opportunity to pay for better application performance. Latency-insensitive tenants would not pay for premium service, and so would experience higher network latency – but would not care.

For example, the provider could say "you currently are paying us for 99th %ile network-level latency SLO of x usec, which appears to be causing your application to have a 99th %ile latency SLO of y msec. If you wish to pay us for network latency $x' < x$, we predict your 99th %ile application latency will decline to $y' < y$."

3 A simple latency-measurement study

Since we found no recent studies that quantify network latency variation across multiple providers, we conducted our own small-scale measurement study. Our limited study probably does not reflect either the typical or worst-case latency variations in current clouds. *We did not design this study as a basis for comparing providers.*

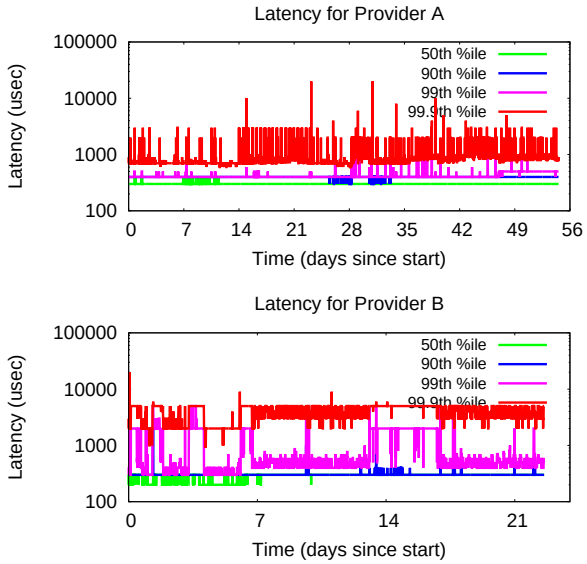

Figure 1: *Latency measurements (log-scaled Y-axis)*

Methodology Due to documented issues with using "ping" (ICMP ECHO) for measuring IaaS latencies [23], we measured TCP latencies using the TCP_RR (request-response) function of the *netperf* tool. We used unmodified netperf-2.6.0, but compiled to enable histogram support for TCP_RR latencies[2]. We used the default 1-byte message length, and ran 60-second trials every 15 minutes over a period of several days.

We ran our tests on several IaaS providers, anonymized as A, B, and C. (We do not wish anyone to use these results to compare between specific providers, since they are hardly definitive.) On each provider, we rented two VMs (the smallest dedicated-core instances available), running stock Ubuntu 14.04 or 14.10.

For each trial, we post-processed the histogram to obtain 50th, 90th, 99th, and 99.9th-%tile latencies. Fig. 1 shows the time series for providers A and B. (We did not try to measure exactly the same period for all providers. The results for C were too boring to include, and most of the high-latency samples for C might be spurious, because we may have failed to obtain dedicated cores.)

Results The results of our simple study suggest that, while 90th-percentile latencies are typically close to the median, the 99th percentile is often significantly larger, and at least for Provider B, can vary tremendously from hour to hour. (Note that since we experimented with only two VMs per provider, this does not mean such variations do not exist for other providers.) For Providers A and B, the 99.9th percentile (which might be significant for applications sensitive to tail latency) is typically on the order of a few msec.

[2]Note that this histogram uses logarithmic decades, each divided into 10 linear buckets, resulting in a factor-of-2 worst-case uncertainty. E.g., one bucket covers all samples between 1 and 2 msec, while another covers all samples between 800 and 900 usec.

The study therefore suggests that tail-latency-sensitive tenants could benefit from a provider that can detect this sensitivity and re-balance resources in response.

4 Using correlation for latency inference

Our goal is to infer the causal relationship between network latency and application-level latency (i.e., the application's latency SLO). In particular, we would like to find the threshold (if any) below which any decreases in network latency would give no further SLO improvements. We might also want to know how dramatically application latency increases when the network latency exceeds this threshold – that is, is application latency highly sensitive to small increases in network latency, or can the application tolerate these increases without user-perceived slowdowns? (For example, an application with poorly-chosen timeouts could seriously magnify a slight change in network latency.)

One could run controlled experiments (similar to what Proteus [25] did for bandwidth) that measure application-level latency, as the network latency is explicitly varied across a chosen range; the shape of the resulting curve would reveal the latency threshold(s). Alternatively, these experiments could vary another parameter under the provider's control, such as packet priorities (set, as described in §2, using DSCP headers). We would then need to apply statistical methods to establish whether an observed correlation between network and application latencies is real (i.e., high-confidence) or accidental, and to extract the latency thresholds from possibly noisy samples. Prior work by Cohen *et al.* [9] suggests that such a correlation-based approach can be made to work.

4.1 Challenges for latency inference

In order to use statistical correlation to discover how network latency affects an application's latency SLO, we need to solve several practical problems. Solving these problems in an IaaS system, without requiring changes to tenant code – that is, primarily within the hypervisor – presents challenges:

- **Measuring network latency**: The hypervisor can record when a guest sends and receives packets, but without additional meta-information, converting these timestamps to round-trip times (RTTs) is tricky. We also need to be able to separate network latencies from service latencies, either at a tenant VM or at a provider's service (such as a storage system).
- **Measuring application-level performance effects**: How can we measure effects on an application's performance? Is there a way for a provider to measure these effects without any modifications to a tenant's code?
- **Perturbing network latency**: If the natural variation in network latency is too small to establish a correlation, how can the hypervisor inject more latency? How

much latency should it inject, and how often?

- **Attributing network latency to the correct SLO-related events**: Assuming that the hypervisor can measure both network latencies and application latencies, for complex applications (with multiple tiers and perhaps multiple entry points), how does the provider connect these measurements?

Figure 2 illustrates how solutions to these problems, and others, fit together into an overall design.

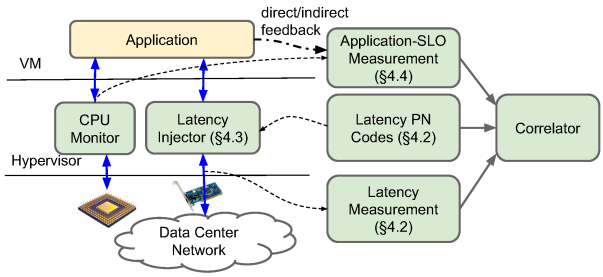

Figure 2: *System design for latency correlation*

We now describe some approaches to these challenges.

4.2 Obtaining network-latency variation data

We need to generate a time series of the network-latency variations experienced by a VM, sampled often enough to detect relatively rare latency spikes, without imposing a lot of overhead. To do this without any tenant cooperation, we considered several options, including:

- **Round-trip timestamps**: Since IaaS providers typically encapsulate tenant packets using protocols such as GRE (RFC 2784) or VXLAN (RFC 7348), one could add a pair of timestamp fields to the encapsulation headers. The sending hypervisor would set a "sent timestamp" field, which would be echoed by the remote hypervisor in a "received timestamp" field on the next packet sent by the same tenant from the remote side. The sender can thus calculate delays without additional per-packet state; however, this does add overhead[3], and if the receiving VM has nothing to send for a while, the measured RTT could be inflated.
- **Monitoring TCP headers**: For tenants using TCP (without IP-level encryption), the hypervisor could monitor TCP headers and run the same RTT estimator as a typical TCP stack. This avoids any on-the-wire overheads, but requires per-flow (rather than per-VM-pair) state in the hypervisor. (Experience with vSnoop [16] suggests that TCP-snooping in the hypervisor actually scales fairly well.) While TCP-snooping measures stack-to-stack RTTs, without any confounding latencies from the tenants or services, it could be tricky to filter out the effects of TCP's delayed ACK.

While simply measuring network delays can yield a time series, the available techniques all have drawbacks – especially if a given VM sees little natural variation.

However, we do not need to restrict ourselves to measurements! Recall that our goal is to find the correlation between network delays and application delays; therefore, one other approach is:

- **Injecting a detectable pattern of latency increases and decreases**: the provider can correlate against this pattern, rather than a measured time series.

Assume that the hypervisor can delay tenant-transmitted packets (as we discuss in §4.3) either by 0 or D msec, and can change between 0 and D every T msec, in a known pattern of length L. This pattern is (in effect) a L-bit binary number; if it is properly chosen, it should be uncorrelated with the actual variations in network delay, and so if the added delay has any effect on application SLO, this effect should be detectable by a correlator that knows the pattern, in much the same way that a GPS receiver detects GPS satellites. (GPS uses pseudo-random codes, also called 'pseudo-noise" or "PN" sequences [1].) The hypervisor might need to vary D and T to discover the threshold at which network latency has an effect on SLO, for any given application.

The PN codes can be chosen, as for GPS, to be unique to each latency injection point, and with minimal correlation between codes (they should be "highly orthogonal")[4]. This solves the problem of how to attribute SLO variations to network latencies, since it allows the correlation-calculator to attribute SLO variations to a specific latency injector. PN-coding also should allow us to separate network latencies from service latencies.

Therefore, we believe that injecting latency changes using PN codes is the most useful way to obtain a known, varying time series of network latencies. It would be useful to couple this with TCP-snooping measurements, which can establish a baseline for latency and its variability between each pair of VMs; this may help with debugging, and to set the values for D and T.

How much delay do we need to inject, and at what "bit-rate"? We do not yet know; we speculate that the rate cannot be higher than the usual 1KHz clock tick rate, and the fall-time for removing X msec of added delay could be as high as X. For applications that act as low-pass filters, the useful bit rate could be even lower.

4.3 Network-latency injection mechanisms

We would like a scalable way for a hypervisor to inject network delays on the packets sent by specific VMs, and with reasonably accuracy. Existing software components, such as *dummynet* [7], can achieve reasonably accurate delays at granularities based on the clock-interrupt resolution (for Linux, typically 1 msec). This does require buffering delayed packets in the hypervisor; at 10 Gbps, delaying all packets by 10 msec requires up to 12.5

[3]On-the-wire overhead could be reduced using MGRP, which "transparently piggybacks application packets inside the often significant amounts of empty padding contained in typical probes" [19].

[4]Techniques exist for generating orthogonal PN codes [11], but we have not yet found a description of their computational costs.

Mbytes of extra buffer space (although we do not expect to delay all packets at any given time).

The provider can also inject latency indirectly, using any of the knobs it can use to favor or dis-favor specific traffic. E.g., we can represent PN codes using one DSCP (priority) setting for "on" bits, and another for "off" bits. Changing priorities, instead of explicitly delaying packets, also allows finer-grained manipulation of network delays, but perhaps with less-predictable values.

4.4 Monitoring application latency

Our most difficult challenge is to measure application latency for IaaS tenants. If we assume that the tenant application and OS cannot be modified, how do we know what events to measure? The options include:

- **Passive network-based measurements**: For example, many cloud customers use provider-supplied **load-balancer** (LB) services (e.g., [20]), at which the provider can measure high-level latencies directly. (However, not all applications use a provider's LB: e.g., those that have only one front-end, or that use Direct Server Return, which bypasses the LB for responses.)

  Another possible approach is to **measure the tenant's network traffic** (bits/sec), since an application waiting for message arrivals might handle fewer requests/sec, reducing its traffic. This is especially true for "chatty" applications [12], but perhaps not as applicable to others (e.g., applications that use several sessions in parallel). Cloud providers already measure network traffic, because they typically bill for it at relatively high prices.
- **Hypervisor-based measurements**: Typically, when an application VM blocks waiting for a network response, the guest OS will issue a reserved instruction (such as HLT or MWAIT [22]) to release the CPU core. The resulting "vmexit" is handled by the hypervisor, which can easily measure the time spent in this state using the cycle-counter. We can therefore use blocked-VM time as a proxy for application-level delay.

  In conjunction with the use of PN-coded delay injection, this approach should work even if the VM that blocks is not the VM where the delay is injected. For example, we can correlate blocked-VM time at the application's front-end VM, even if the delays are being inserted primarily in lower tiers of the application.

  However, this approach will not identify application stalls if the available parallelism in the workload keeps all of the application cores busy, even while end-to-end latency is intolerable.

While passive network-based and hypervisor-based measurements might suffice for some applications, we do not see a zero-modification approach that would work in all cases. Further, while these techniques might allow the provider to measure the correlation between network latency and application latency, they cannot tell the provider what top-level latencies are tolerable (that is, what constitutes an SLO violation).

Therefore, for full generality, we may be forced to use some explicit application modification to support latency inference. This can be quite simple and low-cost; for example, a library linked with the application that provides API calls to record start and end timestamps for application-identified operations. These timestamps can be forwarded to the provider (e.g., via UDP packets to a reserved address) for further processing. Commercial middleware that does a much more sophisticated version of this (for example, AppDynamics [4], which does automatic code injection) is widely used, suggesting that many cloud customers do not just tolerate these modifications, they are willing to pay for them.

With customer approval, but no code modification, the provider can also inject dummy requests into the application to measure its latency. Cloud customers already use probing systems to measure operation latency [8].

Note that some applications that use a partition-aggregate method may set a top-level latency target, and if some underlying RPCs are delayed too long, may return reduced-accuracy results, rather than exceeding their latency SLO. For this applications, we would have to correlate network delays with an accuracy metric, rather than a top-level latency metric; accuracy metrics are obviously application-specific.

4.5 Validation

The next step is to validate our approach. We need to validate some assumptions: to experiment on real cloud applications to learn how significantly network latency affects SLOs; to characterize, using the various options we have discussed, how well a provider can control network latency, and how well it can measure SLOs. We need to understand how to choose PN codes and their amplitudes. Finally, we must validate the overall approach to show that it produces useful insights.

5 Conclusion

We assert that cloud platforms should infer how tenant applications depend on network latency, as part of the "undifferentiated heavy lifting" that makes the cloud valuable to enterprises, and that PN codes may enable this. As with any inference-based system, many challenges remain to be addressed before it is practical and real; prior experience does suggest that it is plausible.

Acknowledgments: We thank Jeremy Sugerman for helping us with details about virtualization.

References

[1] Pseudorandom Noise. http://en.wikipedia.org/wiki/Pseudorandom_noise. Accessed: 2015-01-09; see also http://alumni.cs.ucr.edu/~saha/stuff/cdma_gps.htm for a useful explanation.

[2] Mohammad Alizadeh, Albert Greenberg, David Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *Proc. SIGCOMM*, 2010.

[3] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center. In *Proc. NSDI*, 2012.

[4] AppDynamics. Application Performance Management & Monitoring. http://http://www.appdynamics.com/. Accessed: 2015-01-09.

[5] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards Predictable Datacenter Networks. In *Proc. SIGCOMM*, pages 242–253, 2011.

[6] Sean Kenneth Barker and Prashant Shenoy. Empirical Evaluation of Latency-sensitive Application Performance in the Cloud. In *Proc. MMSys*, pages 35–46, 2010.

[7] Marta Carbone and Luigi Rizzo. Dummynet Revisited. *SIGCOMM Comput. Commun. Rev.*, 40(2):12–20, April 2010.

[8] Cloudify. Using Probes For Monitoring. http://getcloudify.org/guide/2.7/plugins_and_probes/probes.html. Accessed: 2015-01-09.

[9] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *Proc. OSDI*, 2004.

[10] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *CACM*, 56(2):74–80, February 2013.

[11] Helen Donelan and Timothy O'Farrell. A New Method for Generating Sets of Orthogonal Sequences for a Synchronous CDMA System. In Michael Walker, editor, *Cryptography and Coding*, volume 1746 of *Lecture Notes in Computer Science*, pages 56–62. Springer Berlin Heidelberg, 1999.

[12] Kevin Fall and Steve McCanne. You don't know jack about network performance. *Queue*, 3(4):54–59, May 2005.

[13] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. Queues don't matter when you can JUMP them! In *Proc. NSDI*, May 2015.

[14] Keith R. Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J. Wasserman, and Nicholas J. Wright. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Proc. CLOUDCOM*, pages 159–168, 2010.

[15] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. Silo: Predictable Message Completion Time in the Cloud. Technical Report MSR-TR-2013-95, Microsoft Research, 2013.

[16] Ardalan Kangarlou, Sahan Gamage, Ramana Rao Kompella, and Dongyan Xu. vSnoop: Improving TCP Throughput in Virtualized Environments via Acknowledgement Offload. In *Proc. SC*, pages 1–11, 2010.

[17] Katrina LaCurts, Jeffrey C. Mogul, Hari Balakrishnan, and Yoshio Turner. Cicada: Introducing Predictive Guarantees for Cloud Networks. In *Proc. HotCloud*, 2014.

[18] Tim O'Reilly. Transcript of interview with Jeff Bezos. http://archive.oreilly.com/network/2006/12/20/web-20-bezos.html, 2006.

[19] Pavlos Papageorge, Justin McCann, and Michael Hicks. Passive Aggressive Measurement with MGRP. In *Proc. SIGCOMM*, pages 279–290, 2009.

[20] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A. Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, Changhoon Kim, and Naveen Karri. Ananta: Cloud Scale Load Balancing. In *Proc. SIGCOMM*, pages 207–218, 2013.

[21] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. SENIC: Scalable NIC for End-host Rate Limiting. In *Proc. NSDI*, pages 475–488, 2014.

[22] Gabriel L. Somlo. Handling of Guest-Mode MONITOR and MWAIT. http://www.contrib.andrew.cmu.edu/~somlo/OSXKVM/mwait.html, February 2014.

[23] Guohui Wang and T. S. Eugene Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *Proc. INFOCOM*, pages 1163–1171, 2010.

[24] Wenfei Wu, Guohui Wang, Aditya Akella, and Anees Shaikh. Virtual Network Diagnosis As a Service. In *Proc. SOCC*, pages 9:1–9:15, 2013.

[25] Di Xie, Ning Ding, Y. Charlie Hu, and Ramana Kompella. The Only Constant is Change: Incorporating Time-varying Network Reservations in Data Centers. In *Proc. SIGCOMM*, pages 199–210, 2012.

[26] Minlan Yu, Albert Greenberg, Dave Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. Profiling Network Performance for Multi-tier Data Center Applications. In *Proc. NSDI*, pages 57–70, 2011.