CLOUDY FORECAST:
HOW PREDICTABLE IS COMMUNICATION LATENCY IN THE CLOUD?

Anonymous authors
Paper under double-blind review

Abstract

Many systems and services rely on timing assumptions for performance and availability to perform critical aspects of their operation, such as various timeouts for failure detectors or optimizations to concurrency control mechanisms. Many such assumptions rely on the ability of different components to communicate on time—a delay in communication may trigger the failure detector or cause the system to enter a less-optimized execution mode. Unfortunately, these timing assumptions are often set with little regard to actual communication guarantees of the underlying infrastructure – in particular, the variability of communication delays between components. The higher communication variability holds especially true for systems deployed in the public cloud since the cloud is a utility shared by many users and organizations, making it prone to higher performance variance due to noisy neighbor syndrome. In this work, we present StormCloud, a simple tool that can help measure the variability of communication delays between nodes to help engineers set proper values for their timing assumptions. We also provide our observational analysis of running StormCloud in three major cloud providers and share the lessons we learned.

1 Introduction

The reliability and robustness of cloud systems and services are critical to businesses and consumers. Recently, several high-profile cloud outages impacted various aspects of daily life. For instance, a December 2021 AWS outage impacted Amazon delivery service, Roomba smart vacuum cleaners, and streaming services [12, 39]. The financial impact of the majority of such outages far exceeds $100,000 [71].

The software systems deployed in the cloud are designed to be fault-tolerant to prevent smaller failures and outages from getting bigger and impacting users. A typical fault-tolerance mechanism relies on redundancy – systems store many copies of data and have spare compute capacity for processing that data. However, systems also rely on timing assumptions to use this redundancy. For example, a failure of some component needs to be detected before a system may reconfigure to switch to a redundant copy. Such detection usually relies on some timeout or lease mechanism, configured with some timing assumption about expected message delivery. Timing assumptions also play an important role in performance-related optimizations of many protocols and algorithms [19, 24, 52, 55, 67]. Whenever the timing assumption holds, the system may operate in a faster mode. Violating the assumption may cause the system to degrade into a less optimized mode. As a result, failures of timing assumptions present a significant obstacle to reliability, as systems may have trouble detecting failures, experience degraded performance/liveness, or both.

Recently, many algorithms started to rely on increasingly tighter timing assumptions for message delivery. For instance, Copilots [52] relies on 1-millisecond and 10-millisecond timeouts to perform reconfiguration actions. The protocol needs such timeouts to detect slow machines and reconfigure to make progress despite the slow node, ensuring a more predictable performance in case of slowdowns and gray failures [28, 34, 78]. Many other algorithms and systems rely on tight timing assumptions for better concurrency control. The idea for such time-augmented concurrency control is to delay some operation(s) to either ensure safety [19] in the presence of potentially conflicting/concurrent operations or to avoid doing a more expensive conflict resolution [24, 55, 67]. For instance, EPaxos Revisited [67] state machine replication protocol expects that nodes see most messages for a replication round by some deadline; if communication takes longer, the protocol may experience a conflict situation when it is uncertain about the order of replicated commands, causing an expensive conflict resolution procedure to kick in. Naturally, these systems cannot afford to wait for longer than strictly necessary and must resort to the tightest possible assumptions on message delivery latency.

Cloud is a shared environment, with many tenants running their systems and applications in isolated virtualized environments. This isolation often makes it look like each tenant operates with their dedicated resources provided by the cloud provider. However, in reality, tenants share the physical resources of the cloud provider, ranging from servers to networking, storage, and more. Such sharing leads to noisy neighbor syndrome [44, 46, 56], a situation when one tenant may monopolize some shared resource, even if for a very brief time, starving other tenants and negatively impacting their performance. Modern cloud manages the problem decently well at larger time frames – tenants' ability to burst above their nominal resource allocation is often constrained in time, while various schedulers and cluster management systems [15, 30, 75] work hard to achieve a balanced placement of tenants and work within the cluster or datacenter.

However, with tight expectations on the timing and delivery of messages, even short-lived performance variations of the underlying network or host servers can have a significant negative impact on systems, especially if such variations are frequent. Consider the example of a 10-millisecond timeout used in Copilots [52] in a network whose latency spikes above 10 ms once every minute. In such a scenario, the system will likely enter a once-per-minute reconfiguration loop to transition from two co-leaders down to one, paying the associated price in excess resource usage and possibly performance degradation. However, a less tight timeout could have avoided this reconfiguration loop. Similarly, systems that rely on message delivery assumptions for concurrency management will go through cycles of high and low performance when communication timing assumptions get violated. Whenever a system uses resources to reconfigure or run in a less-optimized mode, it loses the opportunity to do useful work. These behaviors can take variations in one resource (i.e., constrained network) and transform and amplify them into variations in another resource (i.e., increased CPU and memory usage to perform reconfiguration or go through a non-optimized execution path). This kind of transformation may even create a feedback loop and cause metastable failure [16, 33]. When the system enters a less performant state, it can cause work to queue up, timeout, and retry, creating even more work and exacerbating performance problems.

Unfortunately, publicly available information on communication latencies in the cloud is scarce, impeding the decision-making process regarding critical timing assumptions. There are a plethora of websites and services that provide some information about average or median latency between cloud regions [5–7], but these services often do not provide region-local latencies or expose finer statistics on latency distributions to help engineers estimate how often, and by how much the communication latency may deviate from the average or median. As a result, most literature pulls these communication latency timing assumptions out of thin air. For example, the aforementioned Copilots [52] work runs on a dedicated cluster with average inter-node communication latency of 0.25ms, making 1 ms timeout a plausible assumption.

In this work, we study the predictability of communication in the cloud to understand and empirically justify the timing assumptions engineers and designers make when working on cloud-native services and applications. To that extent, our contributions are two-fold. First, we present StormCloud, a simple open-source tool to collect communication latency data across many cloud VMs. Second, we use StormCloud to study the communication latency patterns between VMs in three large cloud providers: Amazon Web Services (AWS), Google Compute Platform (GCP), and Microsoft Azure.

Our tool, StormCloud, is a simple echo-like application that can deploy to many VMs in different parts of the cloud, such as different placement groups, availability zones, or regions. The tool runs TCP traffic of configurable payload size and frequency between the VMs and collects the round-trip latency between all VM pairs. StormClud also includes tools and scripts to process raw data and extract valuable statistics for desired pairs or sub-clusters of VMs (i.e., creating a latency histogram for all node pairs in the same availability zone).

We used StormCloud to collect communication latency data in three large cloud providers. In particular, we look at the communication RTT between VMs in different deployment configurations, such as VMs in the same availability zone (AZ) or across AZs and regions. This data gives a lot of insights into the predictability of cloud communication in several common situations. For instance, we observe the potential for significant tail latency applications may experience in the cloud. The 99.99^{th} percentile tail latency for VMs in the same subnet of the same AZ is as much as 36× higher than the average latency, while maximum RTT reaches as much as 2900× the average. Considering that the 99.99^{th} percentile is not that rare (roughly every $10{,}000^{\text{th}}$ round-trip communication exchange), such high-tail latency can significantly impact applications relying on tight timing and communication latency assumptions. We also notice significant latency variations (as much as 7% change in 10 minutes) throughout the day across all tested clouds, suggesting a substantial impact from other cloud workloads/tenants on latency.

2 Background

2.1 Latency in Cloud Systems

The end-to-end communication latency in networked applications and systems consists of more than just the network latency between the nodes. Server hardware, virtualization stack, operating system, and the application itself may introduce additional overheads and jitter, as shown in Figure 1. Moreover, the relative contribution of these different components may change depending on the system's load and corresponding queuing effects [32], operating system, and hypervisor choice. For instance, the Linux Kernel TCP stack can add between 20 to 110,000 microseconds to packet processing compared to kernel-bypass technologies; furthermore, the kernel's TCP stack is significantly more variable, with a latency standard deviation of as much as two orders of magnitude higher than that of kernel bypass solutions [29, 38, 40]. Similarly, virtual machine hypervisors can also reduce performance [8] across the board, with TCP latency degrading as much as 28% [25]. The compute-intensive performance can also decrease substantially; for example, in [22] authors observe as much as 50% performance penalty for dedup benchmark. Naturally, systems that use more compute capacity can experience longer queuing delays [32], impacting the observed end-to-end communication latency.

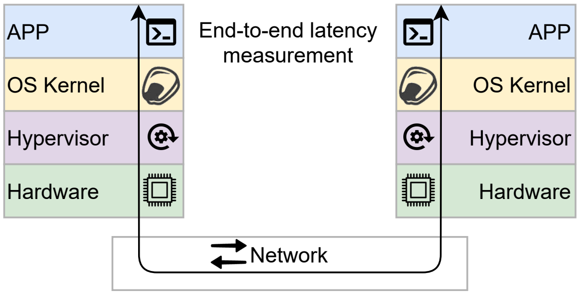


Figure 1: End-to-end latency in cloud systems is impacted by network, hardware, hypervisors, OS kernel, and system itself.

2.2 Noisy Neighbor Syndrome

Within cloud computing, multiple users or tenants often share the same physical resources partitioned into smaller units such as virtual machines, containers, functions of serverless computing, or even subscriptions to shared services. Ideally, the resources should be distributed fairly among all tenants to ensure smooth and consistent performance—those who pay or subscribe for more resources should be able to get them when needed. However, such fair isolation of tenants may not always work, especially at the smaller time scales, leading to the noisy neighbor syndrome [14, 46]. The noisy neighbor syndrome refers to the impact some tenants may have on others through the variability of their work by occasionally hogging more resources, such as CPU, disk IOPS or network bandwidth. For example, many services allow bursting—a short-term increase in capacity over some baseline. If too many tenants need to burst at the same time, they may ask to consume more resources than available, impacting each other and potentially other non-bursting tenants.

Cloud providers have made significant efforts to mitigate the noisy neighbor problem using various approaches, such as resource isolation [59], fair resource allocation [64, 69], and resource scaling out. Moreover, researchers have invested considerable resources in detecting [44, 54, 80] or reducing [31, 70] the impact of noisy neighbors. Despite the efforts to mitigate the noisy neighbor problem, it may continue to impact the cloud, especially at smaller time scales before the mitigations and isolations can kick in.

3 StormCloud

In this section, we describe StormCloud, a simple tool to measure and study the inter-VM end-to-end communication latency (Figure 1) in the cloud. StormCloud can deploy on some arbitrary number of VMs in the cloud, covering a desired topology, such as VMs in the same AZ and across different AZs. After deployment, the tool starts a continuous echo-like message exchange in the cluster with messages of configurable size and frequency. StormCloud records the observed round-trip latencies at each node for further analysis. We implemented StormCloud in Go [26], a popular language for distributed applications and services.

The main goal we want to achieve with StormCloud is realistic testing of end-to-end communication latency in the cloud environment. This goal partly influenced our language choice and the overall design of the tool. Our tool uses a common garbage-collected language that relies on an OS-provided network stack. While a more precise measurement of network latency may be possible by using a non-GC language, such as C++ or Rust, deploying on bare-metal instances to avoid hypervisor, and using kernel-bypass technologies, such as DPDK [36], we believe that such comparison would be less representative of an average app or service.

We design StormCloud to run continuously for prolonged periods. The system operates with many nodes, and each node gets deployed on a separate VM in the cloud. A node has two roles: a sender and a receiver. The sender operates in rounds and broadcasts a message consisting of a round number, some configurable-size payload, and its identity to all the receiver nodes. Upon receiving such a message, the receiver nodes echo back the payload to the sender. The sender ultimately receives the echoes from all receivers and records the round's latency for each receiver. As such, in each round, the sender records multiple latency observations, one for each node in the StormCloud cluster. Each node, serves both sender and receiver roles, allowing us to record detailed latency observations between all node pairs in the cluster. We use TCP for message exchange as one of the most common communication standards. We use the TCP_NODELAY option to disable Nagle's algorithm [20, 50] and obtain more accurate round-trip latency.

Each node contains a measurement component to record the round-trip latency for each observation. The challenge here is to isolate the data recording from the workload that runs the message exchange—we do not want the act of recording measurements to impact the communication latency. To that order, our measurement component collects data in memory before periodically flushing it to local storage. Specifically, we flush data to a CSV file in a separate goroutine from recording the new data, such that data collection can work in parallel with flushing already collected data while having little impact on each other, given a sufficient number of available cores in the underlying VM. The tool can also completely bypass flushing data to disk while collecting measurements—each observation needs only 52 bytes of memory, allowing collecting over 20,000,000 latency observations with just 1 GB of memory. In this case, the collected measurements are kept in memory until after the end of the experiment, at which point they get written to local storage.

StormCloud also includes helper scripts to deploy, start, stop nodes, and retrieve the data from the VMs in the cluster. Furthermore, we provide rudimentary data-processing capability to convert raw data into histograms and compute

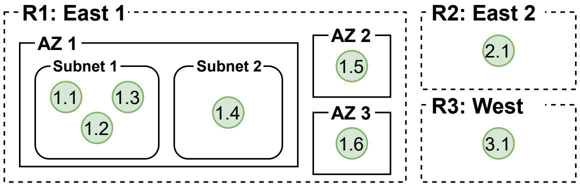


Figure 2: StormCloud deployment in the cloud.

statistics, such as average, median, and latency percentiles.

4 Evaluation of Communication in the Cloud

With the help of StormCloud, we evaluate the round-trip communication latency and its reliability/predictability in three public clouds: Amazon Web Services (AWS), Google Compute Platform (GCP), and Microsoft Azure. The cloud offers great flexibility in deploying, running, and managing virtualized infrastructure, making it impossible to test all conceivable deployment scenarios. As a result, our goal was to provide as much information relevant to a typical deployment.

4.1 Testing Setup

For our experiments, we used a similar setup in each of the three clouds, illustrated in Figure 2. In all three clouds, we used a US East region as the main region (East1). In this region, we deployed 6 VMs in 3 different availability zones (AZs), such that AZ1 had 4 VMs in two different subnetworks with cross-subnet communication setups. Two other nodes were in the remaining AZs. This regional setup allowed us to test the communication within the same subnet in the same AZ, across different subnets in the same AZ, and across 3 AZs, representing a few standard deployment configurations [23, 74]. Having 3 nodes in the same AZ and across 3 AZs also enabled us to study the quorum latency, as quorums are typically used in systems to mask the impact of slow or failed nodes. We also deployed two more nodes in two remote regions, referred to as East2 and West.

We deployed StormCloud on VMs running under a fresh install of Ubuntu 22.04 LTS, one of the more popular Linux distributions. We compiled the tool with Go 1.20.1. We used a common type of x86-based VMs on each cloud. All VMs ran on comparable Intel CPUs and were configured with 2 vCPUs and 8GB of memory. Specifically, we used m5.large, e2-standard-2, and Standard D2s v3 VMs from AWS, GCP, and Azure respectively. In all cases, dedicated instances were used to avoid any potential issues with spot instances. These VMs are popular choices in the cloud, and for example, D2s v3 appears in a "most used by Azure users" section of the Azure portal at the time of our study. The selected VMs also come with a comparable level of networking performance, at least on paper, with AWS offering potentially being the most limited. AWS advertises up to 10 Gbps bandwidth in a burst for its m5.large VM, while providing a base-level bandwidth of only 0.75 Gbps [9, 10], GCP e2-standard-2 VM provides "maximum egress bandwidth" of 4 Gbps [27], and Azure D2s V3 has "expected network bandwidth" of 1000 Mbps [47].

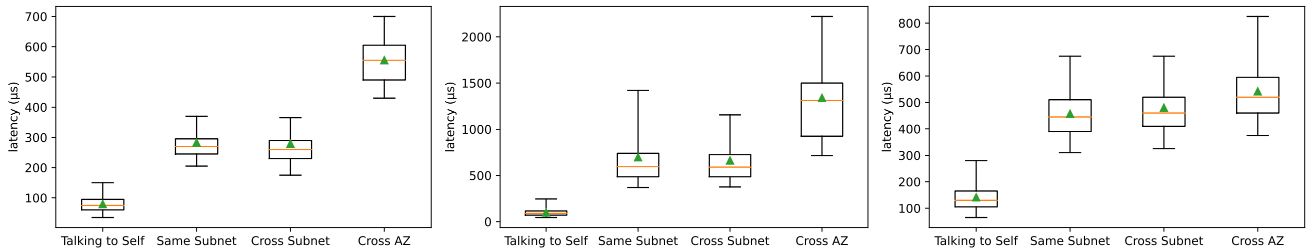
For our base workload, we configured StormCloud to perform 100 measurement rounds per second at each node in the cluster shown in Figure 2. The default payload size was set to 1024 bytes, with 8 nodes in the cluster, resulting in an overall traffic of 1.6 MB/s in each direction (13 Mbps) at each node. These testing parameters also resulted in roughly 15%-17% CPU utilization in our VMs, leaving plenty of headroom. We picked this workload to ensure we collect data at sufficient resolution without straining the VMs, as we want to observe the best possible scenario for end-to-end round-trip communication delays. Increasing the load on VMs, for example by collecting more samples each second, can increase queuing effects for the compute and network resources, resulting in higher variability. While real systems are expected to use the most out of their resource, the load characteristics of each system are unique due to the nature of the system and desired SLOs. As such, we want to collect the best-case measurement with the expectation that adding more load to the distributed system can worsen the end-to-end latency.

Since StormCloud operates in rounds, we can collect point-to-point communication latency between any two nodes in the cluster for each round. We can also collect the quorum latency for each round. Quorums are powerful abstractions to mask slow nodes [51, 77], with many systems and services [17, 19, 21, 35, 48, 63, 66] relying on them. In our observation study, we collect data on two quorums – a majority quorum of nodes in the same AZ (i.e., a majority quorum formed from nodes 1.1, 1.2, and 1.3) and a majority quorum for nodes across 3 AZs (i.e., nodes 1.1, 1.5, and 1.6 as one possible group of such cross-AZ nodes). Unless otherwise stated, most of the data we present comes from a 6-hour run taking place on a weekday between roughly 2 pm and 8 pm in April.

Public cloud vendors use different hypervisors, hardware, and network topologies inside their data centers. Since we cannot control or often directly observe those components, we treat the entire cloud as a black box. Such a lack of observability makes it impossible to tell which components introduce latency and to what extent. As a result, any explanations for the observed behaviors are, at best, educated guesses that we keep to a minimum. This, however, does not diminish the importance of the observations for latency-sensitive applications and systems since observed latency variations and behaviors may impact their performance.

4.2 Threats to Validity

Despite our utmost efforts to obtain the most comprehensive, accurate, and reliable data, we still face considerable challenges and limitations that can impact the validity of our



(a) AWS

(b) Azure

(c) GCP

Figure 3: The box-plot summary for round-trip latency over a 6-hour interval, starting at 2 pm EST on a weekday. The box denotes the IQR, with the middle line designating the median, while the whiskers represent the 5^{th} and 95^{th} percentiles. The green triangle shows the average latency. Note that we are not comparing clouds, so the subfigures have different scales to better show data for each cloud provider.

observations. While we think that our results can serve as a baseline for discussing the variability of communication in the cloud, we refrain from drawing larger conclusions and generalizations and comparing different cloud providers due to the limited scope of our observations—we used specific VM types during a relatively short period and in a limited number of regions and availability zones.

State of underlying infrastructure: The underlying infrastructure serves as the foundation for the performance and reliability of cloud service providers' systems. This includes hardware infrastructure, such as networks and servers, and the software infrastructure to support and manage these facilities. The state of the infrastructure may be affected by various factors, such as hardware failures [76] and resource contention [58]. These issues can impact our observations, limiting our ability to draw definitive conclusions when evaluating cloud services. Furthermore, while we make the best efforts to monitor publicly available outage data, some smaller issues that may impact communication latencies and their variations may not be reported to the public.

Sample size: Sample size directly affects the significance and reliability of research results. Ideally, we would collect many weeks' worth of observations across all regions, AZs, and VM sizes. However, this is prohibitively expensive and time-consuming. Instead, we focused on providing a sufficiently long baseline and ensuring its reproducibility over multiple runs. While most data we report comes from a single 6-hour run (138 million datapoints), we conducted multiple such runs and a few smaller ones to ensure reproducibility. Longer continuous runs, however, can differ substantially from our observations due to the diurnal patterns (we capture a hint of such possibility), business patterns, such as the difference between weekdays and weekends, or special events, such as Black Friday. Additionally, our experiments are limited to VMs and do not reflect the performance and reliability of communication in other cloud services, such as serverless compute platforms or container offerings.

Representation of communication patterns: In cloud systems, communication patterns refer to the ways components interact and collaborate. Communication patterns in production systems and services can be highly complex and varied and may not be properly represented in our simplified observation study. Our study focuses on point-to-point communication between pairs of nodes. We also explore quorum-style communication, in which a sender node expects some threshold of replies before considering the round complete. While these two communication styles represent some basic building blocks, large systems with many components interacting can add further complexity, delays, and variability.

4.3 Regional Communication

In the first set of observations, we examine the round-trip communication latency in a regional setup. Referring to Figure 2, we look at communication between nodes 1.1, 1.2, and 1.3 for latency within a single availability zone and subnet (referred to as "Same Subnet" later in the paper). The latency between node pairs <1.1, 1.4>, <1.2, 1.4>, and <1.3, 1.4> provide insight into communication delays when two nodes are in different subnets, but still in the same AZ, referred to as "Cross Subnet". Finally, node pairs <1.1, 1.5>, <1.1, 1.6>, <1.2, 1.5>, <1.2, 1.6>, <1.3, 1.5>, <1.3, 1.6>, <1.4, 1.5>, <1.4, 1.6> give us communication latency between nodes located in different AZs of the same region.

We illustrate the summary of our regional experiment in Figure 3. The figure shows the boxplots for four observations in each cloud. "Talking to Self" refers to a node using TCP to send a message to itself, "Same Subnet" designates the communication within the same subnet of an AZ, while "Cross Subnet" shows the communication latency between nodes in different subnets of the same AZ. Finally, "Cross AZ" represents the communication between nodes in different AZs of the same region. The figure shows round-trip latency, with the box showing IQR (25th to 75th percentile range) with the median line in between. The whiskers show the 5th and 95th percentiles, and the triangle designates the mean.

The figure illustrates a handful of important points about the reliability of communication latency in the cloud. One of our initial hunches was that having nodes in different subnets of an AZ may have a noticeable impact on latency, but this is not entirely the case in all clouds. Similarly, we expected a significant difference between communication latency within the same AZ and across AZs. While this is the case, GCP's cross-AZ round-trip communication latency appears very close to its same AZ latency. Note that while the latency distributions have substantial overlap, there is still a statistically significant difference between the "Same Subnet" (which has nodes in the same AZ) and "Cross AZ" samples (two-sample student's t-test with $t = 668.952$ and $p = 0$) in GCP. Furthermore, even comparing "Same Subnet" with "Cross Subnet" shows a significant difference in means for all clouds! However, as the difference between means does not exceed 32 microseconds, it is negligible in practice.

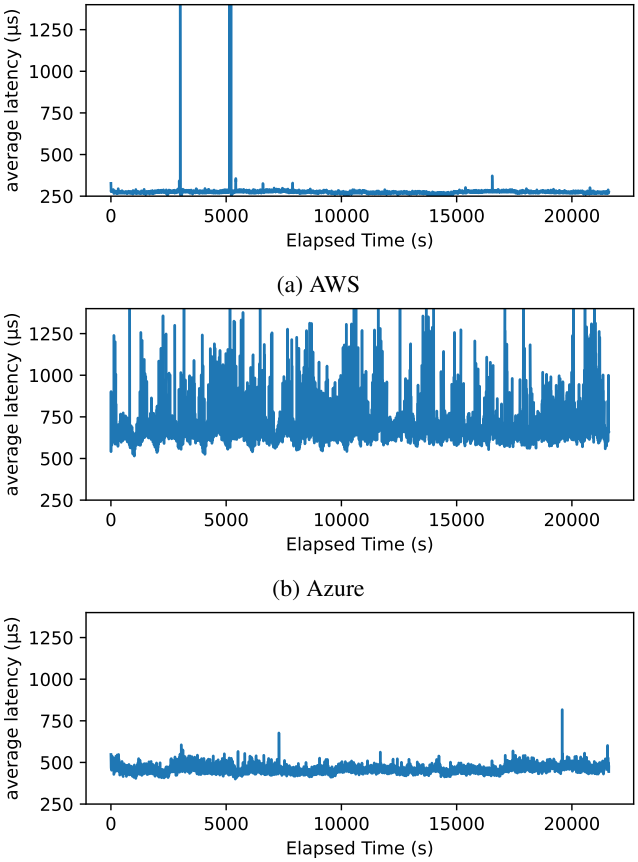
Lesson 1: Nodes in different subnets of the same availability zone are likely to communicate as fast as nodes within the same subnet.

Another important observation we make from the overview data (Figure 3) is about the latency of self-loop when a node uses its TCP interface to send messages to itself. While sending messages to self is substantially faster than talking to any other node, the time delay is not insubstantial. In fact, at the tail end, all clouds produced observations as high as the latency between nodes in the same subnet/AZ.

Lesson 2: Talking to self is substantially faster than talking to other nodes, but this is not a guarantee, as in 2 out 3 clouds (GCP and Azure) we observed that 99.99th percentile round-trip latency of talking to self exceeds the mean latency across nodes in the same subnet/AZ. This finding confirms an observation from a few years ago [42].

4.3.1 Same AZ Round-Trip Latency

Figure 4 provides a high-level view of the communication latency in the same subnet across a single AZ for all three clouds over an entire 6-hour run starting at roughly 2 pm EST and ending at roughly 8 pm EST. The figure shows the average latency aggregated over 30-second windows. Such aggregation hides some finer latency fluctuations but can capture larger details on the stability of communication latency. This figure excludes the communication latency of nodes talking to themselves and only counts the latency between two distinct nodes in the same subnet of the same AZ. Over this 6-hour interval, we have collected roughly 13 million data points on communication in the same subnet of an AZ.



(a) AWS

(b) Azure

(c) GCP

Figure 4: Round-trip latency between nodes in the same subnet of the same AZ in East1 region over a 6-hour interval, starting at 2 pm EST on the weekday.

AWS 30-second averages are relatively stable, as seen in Figure 4a. However, there are a couple of notable exceptions, as we observe spikes that push the average 30-second window round-trip latency above 1000 microseconds. The spike at roughly 5200 seconds of elapsed time is an especially interesting one. This latency spike is due to one node (1.3) experiencing some networking trouble. This node recorded the same latency of 832.83 ms for all its peers in one communication round, including the nodes in other regions. The same node also experienced a three-fold increase in average latency while talking to itself around that time and an elevated latency (≥800 ms) when talking to some peers in other rounds within a few seconds around the big spike, after which the latency returned to normal. This may be an example of some networking or hypervisor problem, largely affecting the ability to receive incoming messages.

When we look at latency between specific node pairs in Figure 8a, we can see some fluctuations over 6 hours. While these changes seem to follow the same pattern for AWS, the magnitude of changes is different for distinct node pairs. The GCP data (Figure 8c) shows that similar latency shifts can affect some, but not all nodes.

Lesson 3: Substantial, frequent, and rapid round-trip latency fluctuations are common in the cloud. Some latency spikes can reach 2,900× the average latency, and can appear and disappear quickly.

The Azure observation in Figure 4b shows several interesting patterns. For one, Azure experiences substantial and frequent latency spikes. Furthermore, Azure latency exhibits noticeable cyclical patterns, as evident in the first hours of the 6-hour run, suggesting a more pronounced instance of noisy-neighbor syndrome during this evaluation run. Figure 5b shows one of such cyclical "arches" visible in the first half of Figure 4b in more detail. Figure 8b depicts the same subnet/AZ data with round-trip latency between each node pair. We can see the cyclical patterns coming from each node pair (this is especially pronounced for nodes 1.2 and 1.3).

Lesson 4: Cloud may experience cyclical latency fluctuations.

Interestingly enough, the <1.2, 1.3> pair seem to show fewer extreme and random fluctuations that show up in pairs <1.1, 1.2> and <1.1, 1.3>, leading to our next lesson:

Lesson 5: Not all nodes of the same type and running identical work behave the same. Communicating with some nodes may innately be more noisy and less predictable than with others.

A high-level overview of GCP latency (Figure 4c) shows mostly stable performance with some noticeable spikes throughout the run. More interestingly, GCP round-trip latency in the same subnet of the same AZ seems to sometimes switch between lower and higher latency modes. While this may be difficult to spot in the high-level view Figure 5c shows a zoomed-in view on a 10-minute interval with one of such transitions. There, the round-trip latency appears to gradually increase from a median latency of 439 microseconds to 468 microseconds at the end of that 10-minute interval, as shown in Figure 6 that depicts box plots for the first and last two minutes of the 10-minute interval from Figure 5c. The cause of this shift can be seen in Figure 8c, where node pairs <1.1, 1.3> and <1.2, 1.3> transition from lower round-trip latency to higher latency in a period of 10 minutes starting roughly at the 16,800-second mark.

Lesson 6: The round-trip latency between any two nodes may change substantially and for the long-term in a matter of minutes and without any warning signs.

To further study the latency distribution in the same subnet of an AZ, we plot the latency distribution histogram and CDF in Figure 7. The distributions for two clouds, AWS and GCP, appear largely Normal. Azure latency, on the other hand, has a much larger tail toward higher latency. This high tail shows in the latency CDF Figure 7b. A zoomed version of the CDF, showing the top 5% of latency observations, indicates that Azure's 99th percentile latency exceeds 2.5 ms. Azure had a 99th percentile of 3.14 ms and 99.99th percentile latency of 21.2 ms. AWS and GCP fare much better, with 99.99th percentile latency of 0.77 ms and 2.79 ms, respectively.

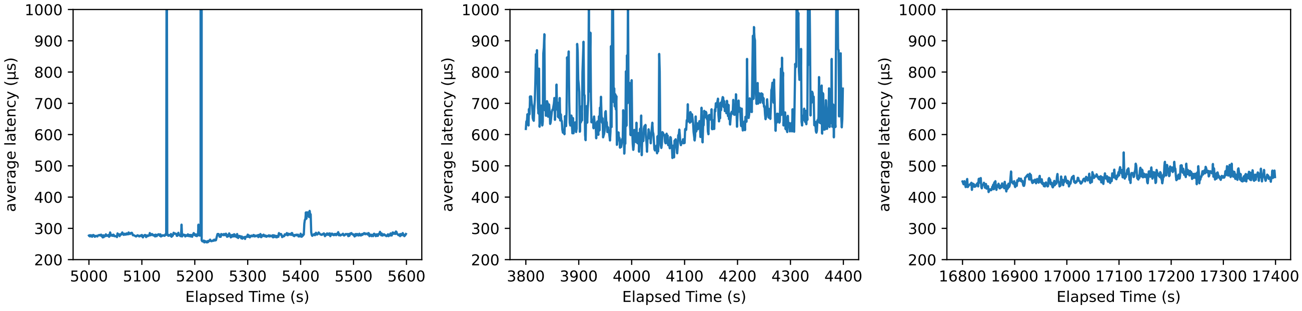
Lesson 7: Expect high tail latency, with 99th percentile often twice the average latency and 99.99th percentile reaching 25× the average.

4.3.2 Cross-AZ Round-Trip Latency

Many applications and services place their components across different networks and AZs of a region to improve fault tolerance and resilience by ensuring that nodes reside in independent failure domains [23, 63, 74]. Such design choices may have an impact on the observed communication latency. The latency distribution for nodes in the same AZ but across different subnets looks nearly indistinguishable from the same-subnet data (Figure 7), as can be seen in the high-level overview of our same-region data (Figure 3). However, the communication latency changes more drastically when placing nodes across different AZs. Figure 9 shows the round-trip latency distribution for nodes in different AZs over the same 6-hour run as in previous experiments. The most noticeable feature is the bi-modal latency distributions for AWS and Azure. GCP does not experience such bi-modal behavior, but its round-trip latency across AZs and within AZ are much closer together than in other clouds.

The bi-modal nature of AWS and Azure cross-AZ distributions may be the result of different latencies between data centers that support distinct AZs. For example, AZ1 may be closer to AZ2 than AZ3, resulting in slightly lower latency between AZ1 and AZ2 (it takes light 3.3 seconds to travel 1km in a vacuum). Additionally, it is worth noting that for AWS, a single AZ may be supported by multiple data centers [41], adding a degree of luck to the picture for latency even within the same AZ, as we discuss in § 4.3.3. The overview data presented in the Figure 3, the multi-modal nature of cross-AZ latency distributions, and the potential for multi-modal latency for the same AZ due to cross-datacenter communication § 4.3.3, leads us to the following lesson:

Lesson 8: Communication across AZs has a noticeable penalty compared to communication within the AZ, but the degree of such penalty varies depending on the cloud and factors such as VM placement in the data center or AZ.



(a) AWS (b) Azure (c) GCP

Figure 5: Zoom in at some 10-minute interval from Figure 4. The interval is different for each cloud and shows important latency changes in more detail. (a) focuses on a large latency spike in AWS. (b) shows a latency drop between cyclical features in Azure. (c) shows a 7% latency increase in GCP.

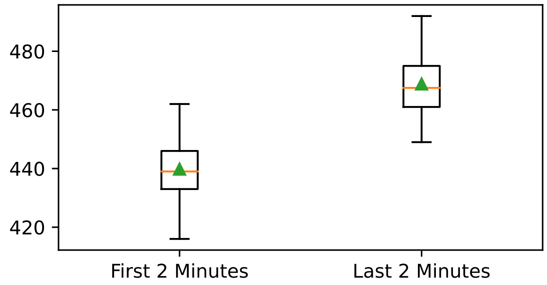


Figure 6: GCP latency for the start and end of Figure 5c. The average latency increases by 7% in 10 minutes.

4.3.3 Impact of VM Placement

Figure 8 also shows that different nodes may have substantially different latency throughout their lifetime. This is evident in every cloud we tested. For example, Figure 8a illustrate the AWS with a clear difference in communication latency between node pairs $< 1.1, 1.3 >$ and $< 1.1, 1.2 >$. While the difference, on average, does not exceed 20 microseconds, it is apparent during the entire 6-hour run. Moreover, we can observe the latency of all node pairs fluctuating similarly during that experiment. Azure, on Figure 8b has similar behavior, with differences between fast and slow node pairs of over 100 microseconds.

As part of our reproducibility effort, we performed several data collection runs. Figure 10 we aggregate the data from three 1-hour runs conducted on three different weekdays during business hours. We ensure that each run produced the same amount of data to ensure equal weight in the aggregated result. For each run, the VMs were brought up from the shutdown state. We did not use any advanced VM placement configurations besides specifying the AZ to allow the cloud to pick the VM placement within the AZ [3, 4, 45].

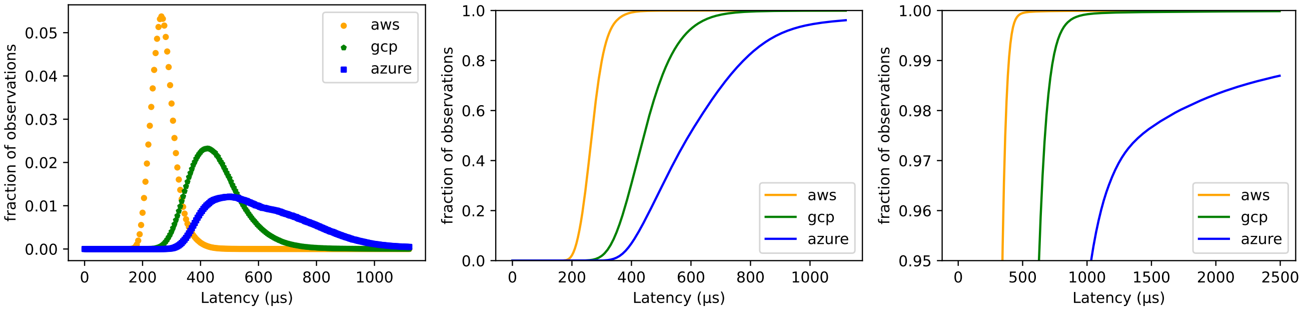
Figure 10a shows the aggregated latency distribution across the three 1-hour runs. The most noticeable difference between this experiment and the 6-hour data (Figure 7a) is the multi-modal latency in AWS for same-AZ communication. We further inspected this phenomenon, and observed a very pronounced difference between several node pairs, in fact, with different clusters of node pairs corresponding to the peaks in the aggregated histogram data. In our observations, this difference can reach nearly 200 microseconds, a very substantial variation considering the initial run's same AZ average latency of only 260 microseconds. We believe the answer for such variability lies in the AZ structure at AWS. In particular, many distinct data centers may support each AZ [41]. If the allocated VMs in the same AZ happen to reside in different data centers, we expect to incur a latency penalty. Unfortunately, this may present an element of luck for users if they do not resort to tools such as placement groups, representing a serious problem for many academic system evaluations in the cloud, as very little thought is typically given to VM placement in literature.

Lesson 9: Round-trip communication latency depends on VM placement within the AZ and datacenter, however, tools exist to influence placement [3, 4, 45].

4.3.4 Payload Size

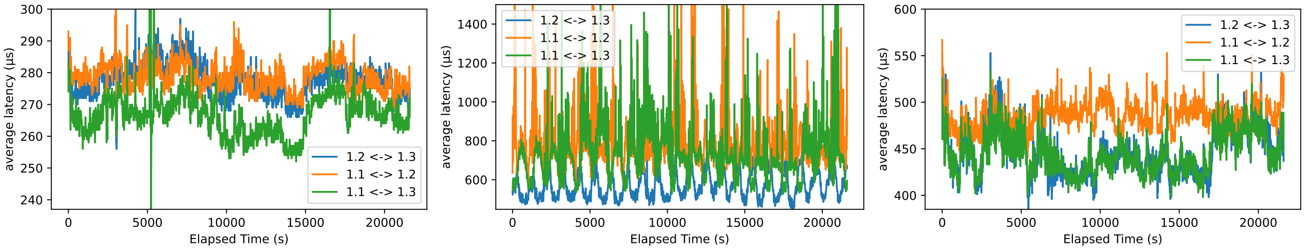
While our default test used a 1024-byte payload in each message going back and forth, the message size can impact the performance. Bigger payloads may exceed the Network's maximum transmission unit (MTU), requiring the message to split into multiple packets [11]. For all of our clouds, we used the default MTU settings. For AWS nodes in the same regions, the MTU is 1300 bytes [11], so a 1024-byte payload should fit into a single packet, while larger payloads will be fragmented. Similarly, GCP's MTU between our nodes was 1460 bytes, while Azure's MTU was 1500 bytes.

Figure 11 shows the effect of payload size on round-trip latency between VMs in the same subnet/AZ. In this experiment, we ran StormCloud with each payload size for one hour during business hours on a weekday. We used the same



(a) Same Subnet Latency Histogram (b) Same Subnet CDF (c) Same Subnet 95%-100% CDF

Figure 7: Same subnet/AZ round-trip latency distribution over a 6-hour interval.



(a) AWS (b) Azure (c) GCP

Figure 8: Same Subnet/AZ round-trip latency over a 6-hour interval for individual pairs of nodes.

VMs for all payload sizes to avoid the VM-allocation lottery described in § 4.3.3. The round-trip latency for GCP (Figure 11c) gradually increases as the payload size grows. Interestingly, AWS shows the lowest average latency for a 1024-byte payload and increased latency for larger messages. The 512-byte message, while comparable to 1024-bytes has more variability. Please note that the difference between 512 bytes and 1024 is small, and falls within the scope of fluctuations we observed over the 6 hours (Figure 8). On Azure, we also see a small growth in average and mean latency as the payload increases, although the difference is less pronounced due to higher variation in observed latency.

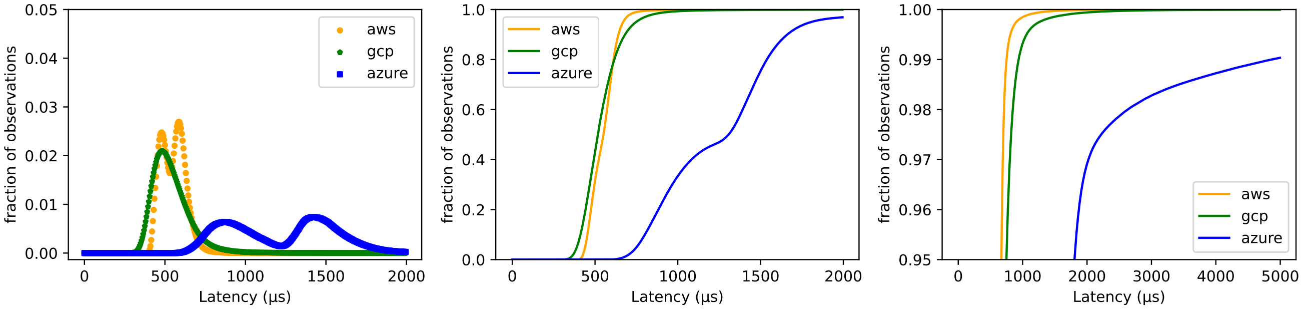
Lesson 10: Payload size will likely increase the round-trip latency, especially if the payload exceeds MTU and must be fragmented. While in some cases, the impact on latency may be very pronounced (nearly 200 microseconds median latency difference between 512 and 4096 bytes!), in other cases, it can be masked by daily latency variations.

4.3.5 Latency in a Quorum

Quorum-style communication is a popular method of masking failures and slow nodes in many data-driven systems. In quorum systems, the same operation or request completes multiple times at different nodes for reliability purposes. Typically, one node will send the message/request to its peers, and wait for a sufficient number of replies before considering the operation complete in the quorum. The completion latency, therefore, is determined by the quorum of the fastest nodes. The most popular quorum system is majority quorums, requiring a majority from the set of all nodes to complete each request. The latency of communication rounds in such a quorum system is determined by the fastest majority. For instance, many database systems rely on three-way replication by default [19, 49, 53, 65] and need a quorum of two nodes, allowing the slowest node to still receive and process requests, but masking its "slowness" from the users.

StormCloud operates in rounds and can measure quorum latency by only looking at the fastest replies in each round. Figure 12 shows the impact of quorums on latency. In this figure, we use our default 6-hour dataset and compute the majority quorum (2 out of 3 nodes) round-trip latency, marked as Q-2/3, and all nodes quorum (Q-3/3) for both the same subnet/AZ and cross-AZ communication.

The majority quorum can filter the slowest node in each round, resulting in better overall latency than waiting for all nodes in a round (i.e., a quorum of all nodes, Q-3/3). The majority quorum also has better latency distribution than simple round-trip node-to-node communication—the node-to-node latency spikes impact the distribution and drive the average higher, while quorums "forget" the slowest node in each round, lowering the average. Such masking effect is

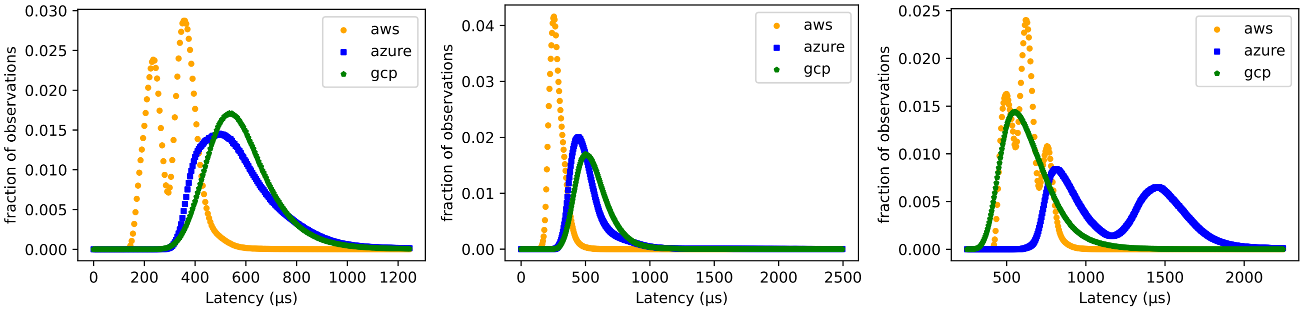


(a) Cross AZ Latency Histogram

(b) Cross AZ CDF

(c) Cross AZ 95%-100% CDF

Figure 9: Cross-AZ round-trip latency distribution over a 6-hour interval.



(a) Same Subnet

(b) Cross Subnet

(c) Cross AZ

Figure 10: Comparison for Three 1-hour Runs

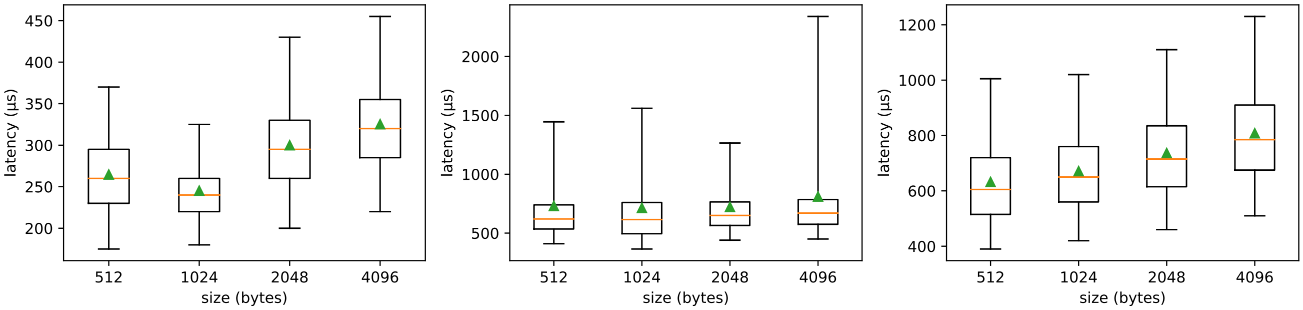
especially noticeable at extreme tail latency: the 99.999th percentile latency for quorums at AWS was 0.85ms in the same AZ, while the same percentile latency for individual node-to-node observations was 490.72 ms, a 577× difference. Similar improvements occur across all cloud providers and AZs.

4.4 Cross-region Communication

Our 6-hour dataset includes nodes at 3 different regions, allowing us to examine the variability of cross-region communication. Figure 13 shows the cross-region latency between two regions in the eastern United States (East1 and East2). These regions are relatively close to each other and have smaller latencies. Please note that due to the different geographical placement, the latencies between the two East regions in different clouds are very distinct. Instead of comparing latencies, we focus on the stability of the cross-region latency. The AWS cross-region latency, shown in Figure 13a appears “jagged” with sudden and abrupt shifts up and down. Excluding a few occasional spikes, the difference between the lowest and highest sufficiently stable and prolonged periods of operation is around 1.5 ms or roughly 10% of the average latency between the two regions. Both Azure and GCP do not have the “jagged” latency and instead show frequent fluctuations up and down with some longer-term trends.

Looking at larger distances between regions, Figure 14 illustrates the latency between the East (East1) and West regions of our cloud deployments. In this case, all three clouds exhibit the “jagged” step-like latency pattern with abrupt shifts up or down. Similarly, there is a noticeable difference between high- and low-latency modes. For instance, in Azure, it is around 3ms, as latency largely shifted from 52ms to 49 ms, while AWS shows a difference of roughly 0.75 ms, excluding spikes.

The jagged patterns for cross-region communication may indicate the different routes packets predominantly take at different times depending on load and congestion in the cloud provider’s cross-region network. Nevertheless, similar to latency within the region and within the same AZ, tenants need to expect substantial variation. During one of our reproducibility runs, we encountered a severe issue on the Azure network, impacting a single East1 node’s ability to communicate with the West region. In that run, the round-trip latency for communication between nodes 1.1 and 3.1 went above 5 seconds(!) and remained high (above 4 seconds round-trip) for several minutes. At the same time, no other nodes in the East1 region were affected; node 1.1 continued normal message exchange with all peers except 3.1, while other nodes could communicate with node 3.1 without apparent problems.

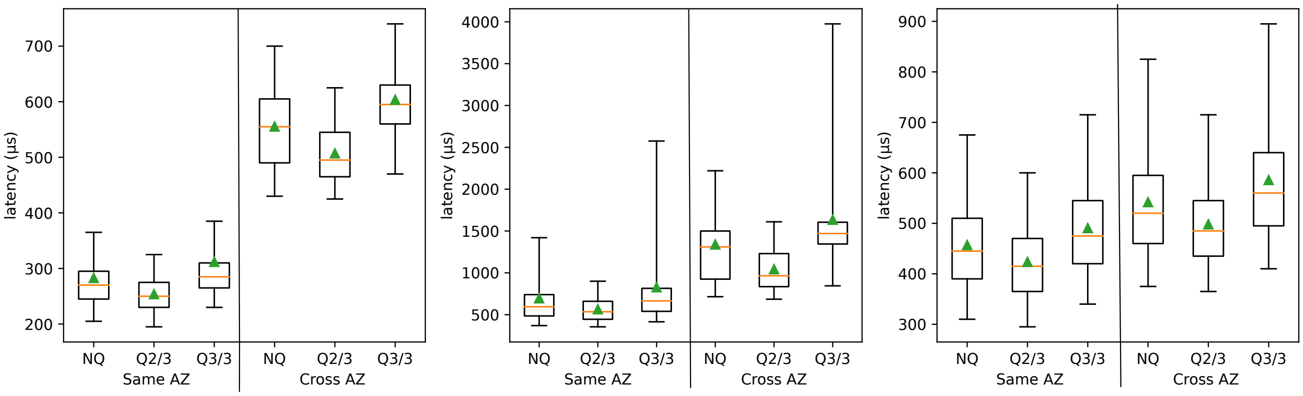


(a) AWS

(b) Azure

(c) GCP

Figure 11: Round-trip latency over a 1-hour interval for different payload sizes. The data were obtained over 4 hours running back-to-back and are subject to possible daily fluctuations shown in Figure 8.



(a) AWS

(b) Azure

(c) GCP

Figure 12: Box plots for round-trip latency over a 6-hour interval with quorums. NQ stands for no-quorum distribution, representing a simple node-to-node communication. Q2/3 is a majority quorum of two nodes out of three for each communication round. Q3/3 is an all-quorum, requiring a response from all nodes in each round.

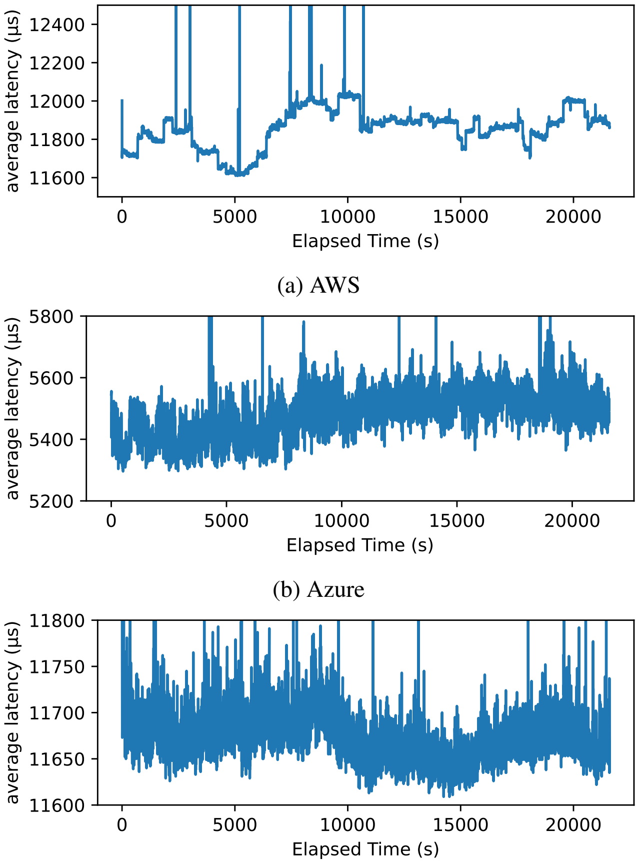
Lesson 11: Cross-regional communication often exhibits a "jagged" pattern with abrupt changes up and down that can nearly instantaneously change average round-trip latency by as much as a few milliseconds.

5 Related Work

Network Simulators. Simulation is a popular methodology for analyzing network performance [79]. However, it does not necessarily apply as meaningfully when trying to replicate the behavior in the cloud. Various tools such as the ns-3 simulator [60], OMNeT++ [72], and NetSim [73] offer the ability to simulate reproducible results for a network system, given the network topology and some information about the system, such as network nodes, channels, protocols, and traffic (data flow) [61]. These tools allow engineers who know the infrastructure of existing systems to model the behavior of communication and analyze the results for performance metrics. However, such simulations have limitations compared to real-time observations.

First, the simulations rely on knowledge of the network topology and specification. For a system that the user has full knowledge about and control in implementation, this is not a problem. However, this is not likely in the public cloud, as users are not aware of the network and the exact placement of nodes in the system. Therefore, these types of simulations can only provide realistic estimates within a margin and are less accurate depending on the amount of information known about how the system is structured. In contrast to this, we perform real-world measurements in the cloud, and while our data has limitations (§ 4.2), it provides a reasonable snapshot of cloud behavior.

Second, the simulations assume a controlled environment. In contrast, cloud systems operate as a global hive mind in which thousands of users contribute to traffic in ways that we have limited control over. We can make predictions about usage, which we can include in simulations, and we can mitigate these hypothesized impacts to the best of our abilities, but ultimately we can only use our best estimations to simulate "real" or "average" cloud usage. Even then, creating a simulation that accurately replicates the amount of traffic that

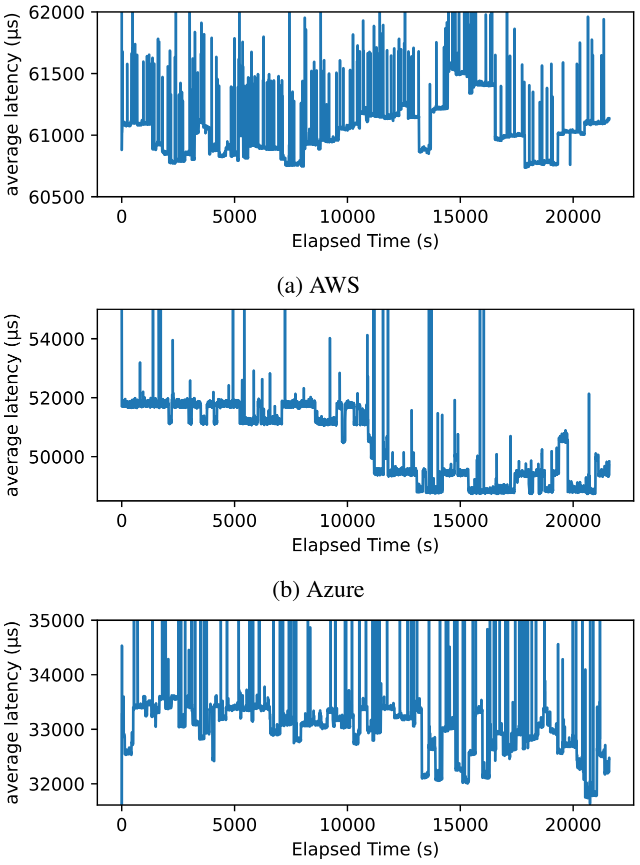


(a) AWS

(b) Azure

(c) GCP

Figure 13: Cross-region East US to East US (East1—East2) round-trip latency over the 6-hour interval.



(a) AWS

(b) Azure

(c) GCP

Figure 14: Cross-region East US to West US round-trip latency over the 6-hour interval.

cloud systems handle is expensive in terms of resources and processing time, making it impractical for analyzing longer periods.

Similar Systems. Communication latency is an important metric concerning networks and distributed systems that rely on them [57]. As a result, plenty of tools exist for the sole purpose of pinging various cloud service machines across different regions and countries and recording the latency [5–7]. These tools are helpful for testing the response of these services at a given moment, typically focusing on pinging anywhere from five to over 40 different machines per service across the world from a single vantage point, often the user's location. Rather than measuring communication from a static point or end-user to a given region, we focus on observing the latency variability between nodes within the cloud.

A few studies look at the latency in the cloud. For instance, Tomanek et al., [68] examines the latency observed from multiple outside vantage points. That study also focuses on the temporal component, comparing the data from 2013 and 2016. In contrast, our work looks at fine-grained latency changes and predictability within the cloud. Other systems looked at some application-specific aspects of latency in the cloud, such as gaming [18]. Some studies take a more holistic approach and compare cloud providers from multiple relevant performance metrics, including latency [43]. Others focus on only the network component of end-to-end latency by measuring the delays on relevant packets at the smart switch [62]. The reliance on smart switches also makes such an approach infeasible for cloud deployment by tenants. Other work, such as Jain et al., [37] focus on bandwidth. In this work we assume sufficient bandwidth because cloud providers typically provide a way to purchase more bandwidth if it becomes necessary, making bandwidth a matter of how much money one is willing to spend [1, 2, 13].

6 Conclusion

Communication timing plays a critical role in many systems, with many systems using timing assumptions for operation- or performance-critical tasks. However, these timing assumptions are often estimated without a proper understanding of the environment in which systems may be deployed. In this paper, we design and develop a simple tool to study communication latency between nodes in the cloud environment. With our tool, we showed the fickle nature of the cloud- -latency between nodes or VMs can change abruptly and without warning. These changes can be very substantial, as much as thousands of times the average latency. However, they can also be less dramatic (i.e., 20% increase or decrease) but long-term, lasting for hours or more. While we cannot possibly study all imaginable deployment scenarios for systems in the cloud, we believe our work provides a substantial introduction to the variability of cloud communication.

Such communication latency variability may impact systems and applications running in the cloud, especially the ones designed for low latency. Such systems often have knobs and settings controlling the expected communications delays and variations. Our work sets the baseline for such timing parameters and presents a step towards empirically setting them for the most efficient operation in a particular cloud or network.

Artifacts

Source code is available at https://anonymous.4open.science/r/CLOUDY-FORECAST-225D. Additionally, HashiCorp's Terraform was used for deployment to AWS and Azure, although due to issues with Linux VMs on Azure the deployment process is more manual there. A makefile, compatible with GNU Make, is provided in each directory to handle a multi-region deployment similar to the one used in our experiments, with the exception that they default to the cheapest VM we could find on each cloud to minimize any accidental costs. The TF_VAR_instance_type environment variable may be set to change the instance type to one more suitable for testing.

References

[1] Amazon EC2 instance network bandwidth - Amazon Elastic Compute Cloud.

[2] Network bandwidth | Compute Engine Documentation.

[3] Placement groups - Amazon Elastic Compute Cloud.

[4] Use VM instance placement policies | Compute Engine Documentation.

[5] Aws latency monitoring. https://www.cloudping.co/grid, 2023.

[6] Cloud ping test (latency) for different providers like aws, azure, gcp, digital ocean from your web browser. https://cloudpingtest.com/, 2023.

[7] Simultaneous ping test for all popular cloud providers. https://webping.cloud/, 2023.

[8] Luca Abeni and Dario Faggioli. Using Xen and KVM as real-time hypervisors. 106:101709.

[9] Amazon Web Services. General purpose instances. https://aws.amazon.com/message/12721/, 2021.

[10] Amazon Web Services. Amazon ec2 m5 instances: Balanced compute, memory, and networking resources for general purpose workloads. https://aws.amazon.com/ec2/instance-types/m5/, 2023.

[11] Amazon Web Services. Network maximum transmission unit (MTU) for your EC2 instance. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/network_mtu.html, 2023.

[12] Amazon Web Services. Summary of the aws service event in the northern virginia (US-EAST-1) region. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/general-purpose-instances.html#general-purpose-network-performance, 2023.

[13] asudbring. Azure virtual machine network throughput.

[14] Anthony O. Ayodele, Jia Rao, and Terrance E. Boult. Performance measurement and interference profiling in multi-tenant clouds. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 941–949, 2015.

[15] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(7), 2013.

[16] Nathan Bronson, Abutalib Aghayev, Aleksey Charapko, and Timothy Zhu. Metastable failures in distributed systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '21, pages 221–227, New York, NY, USA, 2021. Association for Computing Machinery.

[17] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

[18] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM International Conference on Multimedia*, MM '11, pages 1269–1272, New York, NY, USA, 2011. Association for Computing Machinery.

[19] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter

Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3), August 2013.

[20] Russ Cox. preliminary network - just dial for now. https://github.com/golang/go/blob/e8a02230f215efb075cccd4146b3d0d1ada4870e/src/lib/net/net.go#L398, 2008.

[21] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[22] Xiaoning Ding and Jianchen Shan. Diagnosing Virtualization Overhead for Multi-threaded Computation on Multicore Platforms. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 226–233.

[23] Mostafa Elhemali, Niall Gallagher, Nick Gordon, Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somasundaram Perianayagam, Tim Rath, Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, and Akshat Vig. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 1037–1048, Carlsbad, CA, July 2022. USENIX Association.

[24] Benedict Elliott Smith, Tony Zhang, Blake Eggleston, and Scott Andreas. Cep-15: Fast general purpose transactions. https://cwiki.apache.org/confluence/display/CASSANDRA/CEP-15%3A+General+Purpose+Transactions?preview=/188744725/188744736/Accord.pdf, 2021.

[25] Pekka Enberg. A Performance Evaluation of Hypervisor, Unikernel, and Container Network I/O Virtualization.

[26] Google. The go programming language, 2018. https://go.dev/.

[27] Google Cloud Platform. Compute engine general-purpose machine family. https://cloud.google.com/compute/docs/general-purpose-machines, 2023.

[28] Haryadi S. Gunawi, Riza O. Suminto, Russell Sears, Casey Golliher, Swaminathan Sundararaman, Xing Lin, Tim Emami, Weiguang Sheng, Nematollah Bidokhti, Caitie McCaffrey, Deepthi Srinivasan, Biswaranjan Panda, Andrew Baptist, Gary Grider, Parks M. Fields, Kevin Harms, Robert B. Ross, Andree Jacobson, Robert Ricci, Kirk Webb, Peter Alvaro, H. Birali Runesha, Mingzhe Hao, and Huaicheng Li. Fail-slow at scale: Evidence of hardware performance faults in large production systems. *ACM Trans. Storage*, 14(3), October 2018.

[29] Dániel Géhberger, Dávid Balla, Markosz Maliosz, and Csaba Simon. Performance Evaluation of Low Latency Communication Alternatives in a Containerized Cloud Environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 9–16.

[30] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, David Dion, Esaias E Greeff, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. Protean: Vm allocation service at scale. In *OSDI*. USENIX, October 2020.

[31] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. Mittos: Supporting millisecond tail tolerance with fast rejecting slo-aware os interface. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 168–183, New York, NY, USA, 2017. Association for Computing Machinery.

[32] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[33] Lexiang Huang, Matthew Magnusson, Abishek Bangalore Muralikrishna, Salman Estyak, Rebecca Isaacs, Abutalib Aghayev, Timothy Zhu, and Aleksey Charapko. Metastable failures in the wild. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 73–90, Carlsbad, CA, July 2022. USENIX Association.

[34] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R. Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 150–155, New York, NY, USA, 2017. Association for Computing Machinery.

[35] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010*

USENIX annual technical conference (ATC 2010), pages 11–11. USENIX Association, 2010.

[36] Intel. DPDK: Data plane development kit. https://www.dpdk.org/, 2014.

[37] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica. Skyplane: Optimizing Transfer Cost and Throughput Using {Cloud-Aware} Overlays. pages 1375–1389.

[38] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems. pages 489–502.

[39] Karen Weise. Amazon's cloud computing outage disrupts its warehouse operations. *The New York Times*.

[40] Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr. Sharma, Arvind Krishnamurthy, and Thomas Anderson. TAS: TCP Acceleration as an OS Service. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, pages 1–16. Association for Computing Machinery.

[41] Hui Kenneth. Aws 101: Regions and availability zones. https://www.rackspace.com/blog/aws-101-regions-availability-zones.

[42] Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu, and Haryadi S. Gunawi. TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems. *SIGPLAN Not.*, 51(4):517–530, March 2016.

[43] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 1–14, New York, NY, USA, 2010. Association for Computing Machinery.

[44] Tania Lorido-Botran, Sergio Huerta, Luis Tomás, Johan Tordsson, and Borja Sanz. An unsupervised approach to online noisy-neighbor detection in cloud data centers. *Expert Systems with Applications*, 89:188–204, 2017.

[45] mattmcinnes. Proximity placement groups - Azure Virtual Machines.

[46] Microsoft. Noisy neighbor antipattern. https://learn.microsoft.com/en-us/azure/architecture/antipatterns/noisy-neighbor/noisy-neighbor, 2000.

[47] Microsoft Azure. Dv3 and dsv3-series. https://learn.microsoft.com/en-us/azure/virtual-machines/dv3-dsv3-series, 2022.

[48] MongoDB Inc. The MongoDB 4.2 manual.

[49] Iulian Moraru, David G Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 358–372. ACM, 2013.

[50] John Nagle. Congestion control in ip/tcp internetworks. https://datatracker.ietf.org/doc/html/rfc896, 1984.

[51] Moni Naor and Avishai Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, 1998.

[52] Khiem Ngo, Siddhartha Sen, and Wyatt Lloyd. *Tolerating Slowdowns in Replicated State Machines using Copilots*, pages 583–598. USENIX Association, November 2020.

[53] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 2014)*, pages 305–319. USENIX Association, 2014.

[54] Bruno Ordozgoiti, Alberto Mozo, Sandra Gómez Canaval, Udi Margolin, Elisha J. Rosensweig, and Itai Segall. Deep convolutional neural networks for detecting noisy neighbours in cloud infrastructure. In *The European Symposium on Artificial Neural Networks*, 2017.

[55] Haochen Pan, Jesse Tuglu, Neo Zhou, Tianshu Wang, Yicheng Shen, Xiong Zheng, Joseph Tassarotti, Lewis Tseng, and Roberto Palmieri. Rabia: Simplifying state-machine replication through randomization. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, pages 472–487, New York, NY, USA, 2021. Association for Computing Machinery.

[56] Pat Helland. Fail-fast is failing... fast! *ACM Queue*.

[57] Diana Popescu, Noa Zilberman, and Andrew Moore. Characterizing the impact of network latency on cloud-based applications' performance. 2017.

[58] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu. Understanding performance interference of i/o workload in virtualized cloud environments. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 51–58, 2010.

[59] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced

cloud services. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, page 77–84, New York, NY, USA, 2009. Association for Computing Machinery.

[60] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[61] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[62] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. Continuous in-network round-trip time monitoring. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, pages 473–485, New York, NY, USA, 2022. Association for Computing Machinery.

[63] Dharma Shukla, Shireesh Thota, Karthik Raman, Madhan Gajendran, Ankur Shah, Sergii Ziuzin, Krishnan Sundaram, Miguel Gonzalez Guajardo, Anna Wawrzyniak, Samer Boshra, Renato Ferreira, Mohamed Nassar, Michael Koltachev, Ji Huang, Sudipta Sengupta, Justin Levandoski, and David Lomet. Schema-Agnostic Indexing with Azure DocumentDB. *Proc. VLDB Endow.*, 8(12):1668–1679, August 2015.

[64] Stavros Souravlas and Stefanos Katsavounis. Scheduling fair resource allocation policies for cloud computing through flow control. *Electronics*, 8(11), 2019.

[65] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD 2020)*, SIGMOD '20, pages 1493–1509, New York, NY, USA, 2020. Association for Computing Machinery.

[66] The Apache Software Foundation. Apache Cassandra. http://cassandra.apache.org, 2019.

[67] Sarah Tollman, Seo Jin Park, and John Ousterhout. EPaxos Revisited. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021.

[68] Ondrej Tomanek, Pavol Mulinka, and Lukas Kencl. Multidimensional cloud latency monitoring and evaluation. *Computer Networks*, 107:104–120, 2016. Machine learning, data mining and Big Data frameworks for network monitoring and troubleshooting.

[69] Takuro Tomita and Shin-ichi Kuribayashi. Congestion control method with fair resource allocation for cloud computing environments. In *Proceedings of 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 1–6, 2011.

[70] Luis Tomás, Carlos Vázquez, Johan Tordsson, and Ginés Moreno. Reducing noisy-neighbor impact with a fuzzy affinity-aware scheduler. In *2015 International Conference on Cloud and Autonomic Computing*, pages 33–44, 2015.

[71] Uptime Institute. Uptime institute's 2022 outage analysis finds downtime costs and consequences worsening as industry efforts to curb outage frequency fall short. https://uptimeinstitute.com/about-ui/press-releases/2022-outage-analysis-finds-downtime-costs-and-consequences-worsening, 2022.

[72] Andras Varga. *OMNeT++*, pages 35–59. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[73] Tamie L. Veith, John E. Kobza, and C.Patrick Koelling. Netsim: Java™-based simulation for the world wide web. *Computers & Operations Research*, 26(6):607–621, 1999.

[74] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1041–1052. ACM, 2017.

[75] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[76] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 193–204, New York, NY, USA, 2010. Association for Computing Machinery.

[77] Michael Whittaker, Aleksey Charapko, Joseph M. Hellerstein, Heidi Howard, and Ion Stoica. *Read-Write Quorum Systems Made Practical*. Association for Computing Machinery, New York, NY, USA, 2021.

[78] Andrew Yoo, Yuanli Wang, Ritesh Sinha, Shuai Mu, and Tianyin Xu. Fail-slow fault tolerance needs programming support. In *Proceedings of the Workshop*

on Hot Topics in Operating Systems, HotOS '21, page 228–235, New York, NY, USA, 2021. Association for Computing Machinery.

[79] Kevin Zhao, Prateesh Goyal, Mohammad Alizadeh, and Thomas E. Anderson. Scalable tail latency estimation for data center networks, 2022.

[80] Yuxuan Zhao, Dmitry Duplyakin, Robert Ricci, and Alexandru Uta. Cloud performance variability prediction. In *Companion of the ACM/SPEC International Conference on Performance Engineering*, ICPE '21, page 35–40, New York, NY, USA, 2021. Association for Computing Machinery.

7 Appendix

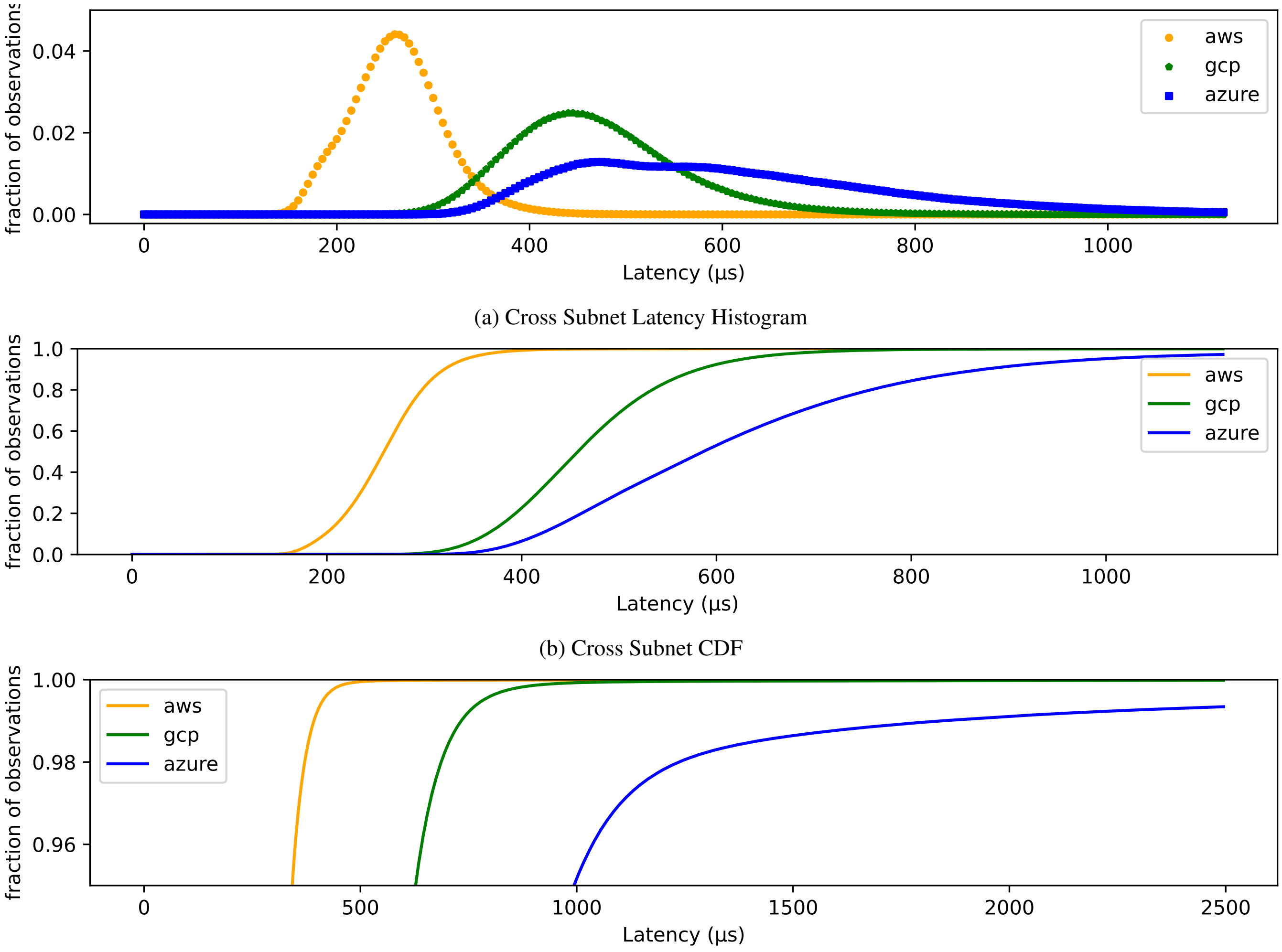
This appendix contains supplemental figures which we thought were interesting or potentially helpful, but were not important enough for the main body of work.

Figure 15 shows various aggregate measures of latency over the six hours. These are grouped by cloud, but we would like to reinforce that comparing between clouds as far as latency numbers are concerned is specific to our particular run, using the same regions and VM sizes, at a given time. However, Figure 15a shows some interesting behavior, as Azure is showing a slightly bimodal distribution, whereas AWS and GCP are much more normally distributed.

Figure 16a displays clear stratification, with all pairs involving 1.5 having noticeably lower latency than those involving 1.6. Since this is cross-AZ communication we hypothesize that this results from the network topology and the physical distance between the nodes. Figure 16b exhibits similar behavior to Figure 16c, except pairs with 1.6 are lower latency than pairs with 1.5. Figure 16c is less clear visually, but there is a 40 μs difference in mean latency.

Lesson 12: Different pairs AZs in the same region may have different latencies between them. It may be worthwhile to try several combinations of AZs to find the one with the lowest latency for your usecase.

Figure 17a depicts the large latency spike at 5200 seconds for AWS Same Subnet, also depicted in Figure 5a, showing that 1.1 to 1.2 was unaffected and that 1.3 was likely the culprit. Figure 17b shows part of one of the "arches" that are present in Figure 4b, showing that the slight increase in 1.2 to 1.3 combined with the increasing 1.1 to 1.3 likely led to the observed "arch" shape for this area. Figure 17c shows that the gradual increase depicted in Figure 5c was the result of the pairs involving 1.3 increasing in latency. As an aside, we also determined that the GCP AZ containing 1.5 was at absolute most 43 miles away from the primary AZ assuming instant network equipment processing, no OS network stack latency and the speed of light in fiber-optics being the same as in a vacuum, so it is likely much closer.



(a) Cross Subnet Latency Histogram

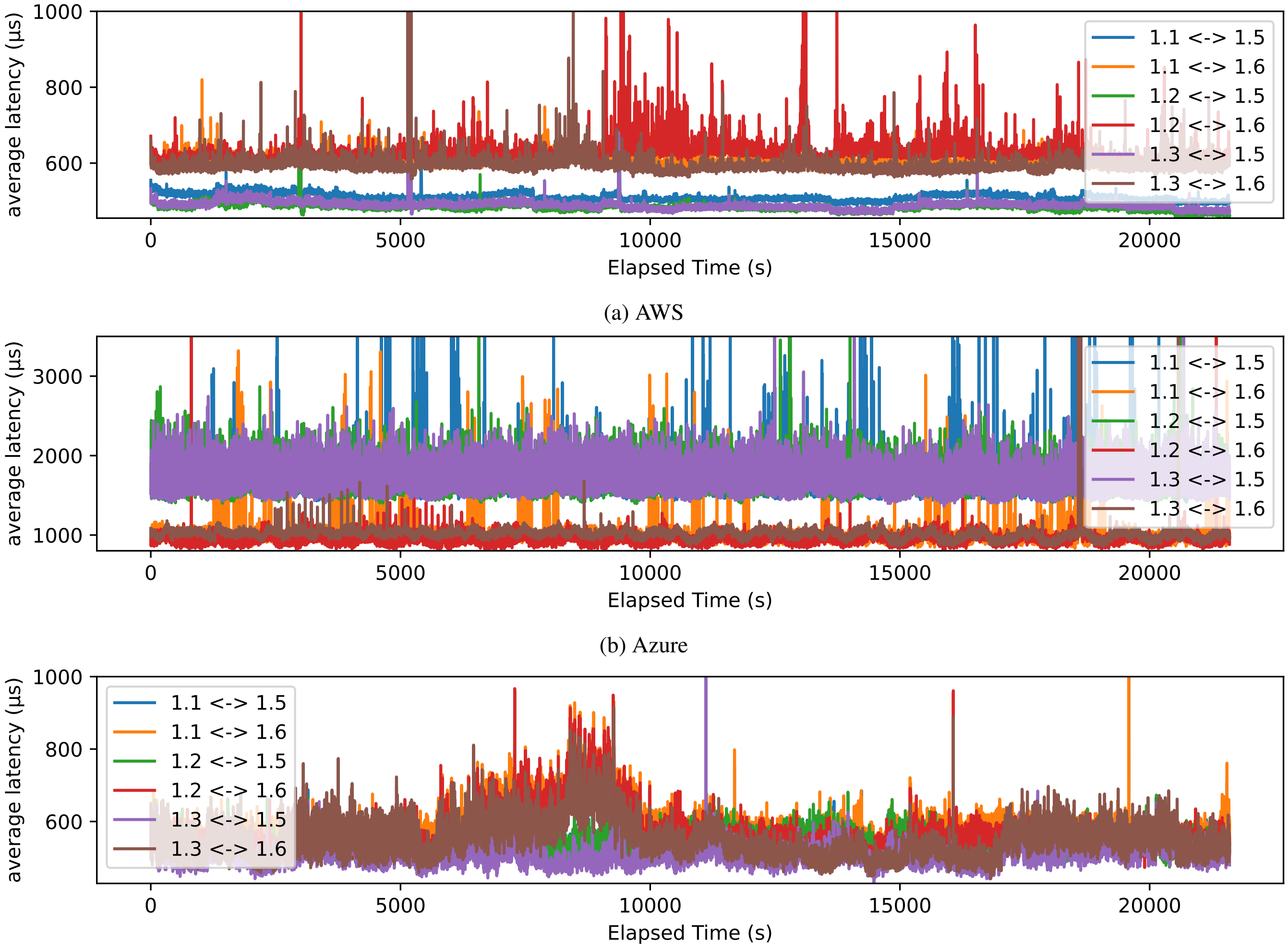
(b) Cross Subnet CDF

(c) Cross Subnet 95%-100% CDF

Figure 15: Cross-subnet round-trip latency over the 6-hour interval.

Cloud	AWS			Azure			GCP		
	Same Subnet	Cross Subnet	Cross Az	Same Subnet	Cross Subnet	Cross Az	Same Subnet	Cross Subnet	Cross Az
median (μs)	270.0	260.0	555.0	595.0	590.0	1310.0	445.0	460.0	520.0
p5 (μs)	215.0	185.0	445.0	390.0	395.0	755.0	325.0	345.0	395.0
p25 (μs)	245.0	230.0	490.0	485.0	485.0	925.0	390.0	410.0	460.0
mean (μs)	283.0	279.0	555.4	696.7	661.9	1339.5	457.2	480.1	542.0
p90 (μs)	325.0	325.0	645.0	890.0	875.0	1665.0	580.0	585.0	685.0
p95 (μs)	345.0	345.0	670.0	1035.0	995.0	1810.0	630.0	630.0	750.0
p99 (μs)	395.0	395.0	755.0	3135.0	1845.0	4865.0	740.0	735.0	940.0
p999 (μs)	470.0	470.0	1105.0	9795.0	8925.0	16670.0	955.0	945.0	1690.0
p9999 (μs)	760.0	745.0	1905.0	21200.0	21705.0	23085.0	2780.0	3210.0	3390.0
p99999 (μs)	486136.2	619828.0	11080.3	26797.0	35115.0	50710.0	6227.0	390171.4	10375.0
max (μs)	832830.0	849470.0	832830.0	119145.0	125500.0	137960.0	28335.0	776095.0	328685.0

Table 1: Latency statistics by cloud and network group.

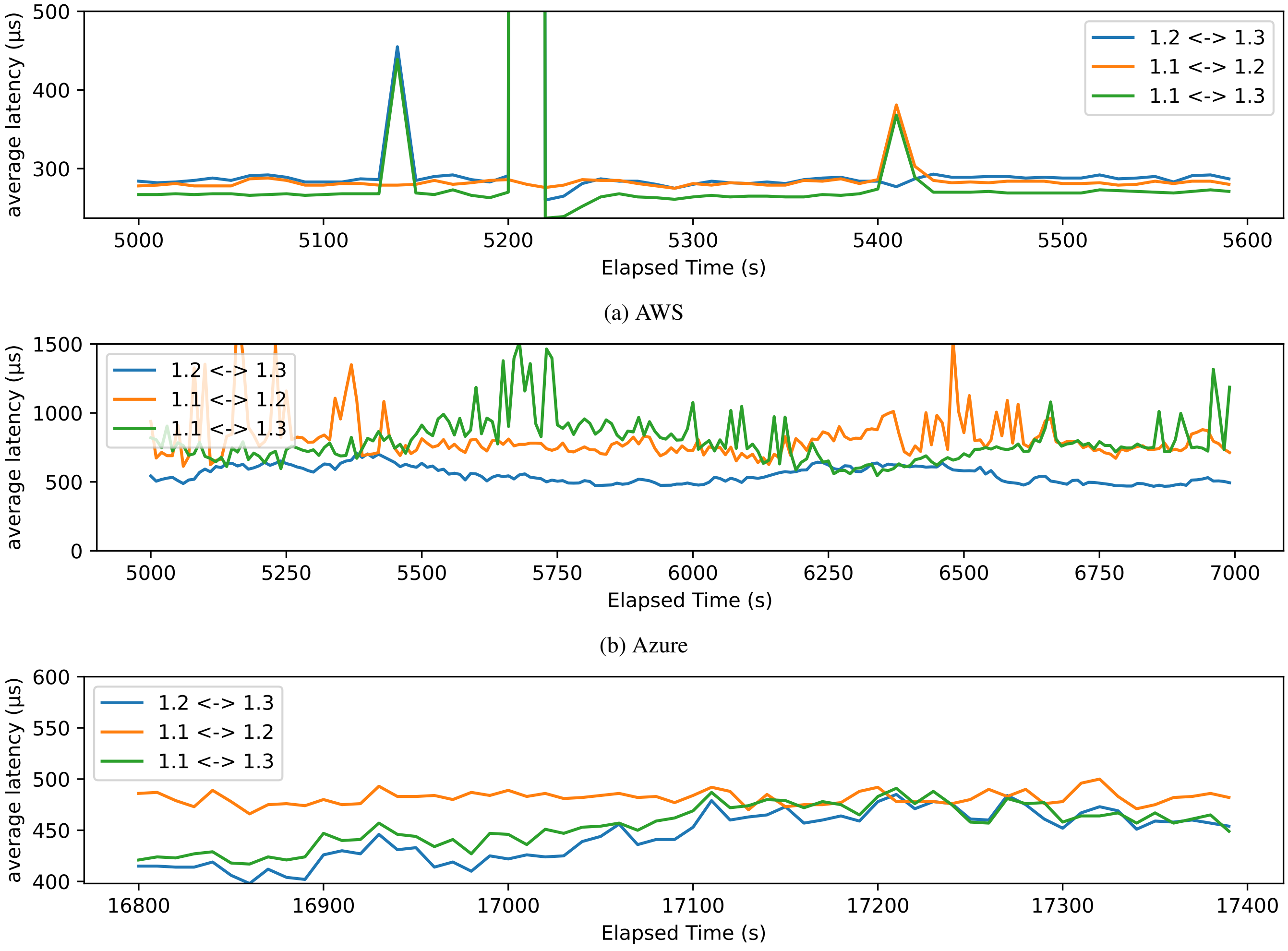


(a) AWS

(b) Azure

(c) GCP

Figure 16: Pairwise Cross-AZ round-trip latency over the 6-hour interval.



(a) AWS

(b) Azure

(c) GCP

Figure 17: Pairwise Same Subnet round-trip latency over the 10-minute interval.