Public Review for

Path Persistence in the Cloud: A Study of the Effects of Inter-Region Traffic Engineering in a Large Cloud Provider's Network

Waleed Reda, Kirill Bogdanov, Alexandros Milolidakis, Hamid Ghasemirahni, Marco Chiesa, Gerald Q. Maguire Jr., and Dejan Kostić

This paper presents a detailed analysis of path persistence across the data centers of Amazon Web Services. The authors deployed experiments using virtual machines in all AWS availability zones, and use them to probe latency between them. They designed a specific heuristics to measure path changes. Results show that paths between Amazon data centers change frequently, and leads to a latency penalty up to 32%. Overall, the paper shed an interesting light to disentangle cloud infrastructure and performance. While end-users are interested in the latency from the edge to the datacenter, showing path inconsistencies between datacenters become crucial for applications that route traffic within the same AWS network. For instance, in these days when people are forced to resort to online collaborations, the end-to-end delay becomes one of the uttermost indices to let multiparty applications work properly. The results and methodologies presented in this paper allows the community to better understand these implications.

Public review written by
Marco Mellia
Politecnico di Torino, Italy

Path Persistence in the Cloud: A Study of the Effects of Inter-Region Traffic Engineering in a Large Cloud Provider's Network

Waleed Reda
Université catholique de Louvain
KTH Royal Institute of Technology
wfhsr@kth.se

Kirill L. Bogdanov
KTH Royal Institute of Technology
kirillb@kth.se

Alexandros Milolidakis
KTH Royal Institute of Technology
miloli@kth.se

Hamid Ghasemirahni
KTH Royal Institute of Technology
hamidgr@kth.se

Marco Chiesa
KTH Royal Institute of Technology
mchiesa@kth.se

Gerald Q. Maguire Jr.
KTH Royal Institute of Technology
maguire@kth.se

Dejan Kostić
KTH Royal Institute of Technology
dmk@kth.se

A commonly held belief is that traffic engineering and routing changes are infrequent. However, based on our measurements over a number of years of traffic between data centers in one of the largest cloud provider's networks, we found that it is common for flows to change paths at ten-second intervals or even faster. These frequent path and, consequently, latency variations can negatively impact the performance of cloud applications, specifically, latency-sensitive and geo-distributed applications.

Our recent measurements and analysis focused on observing path changes and latency variations between different Amazon *aws* regions. To this end, we devised a path change detector that we validated using both *ad hoc* experiments and feedback from cloud networking experts. The results provide three main insights: (1) Traffic Engineering (TE) frequently moves (TCP and UDP) flows among network paths of different latency, (2) Flows experience unfair performance, where a subset of flows between two machines can suffer large latency penalties (up to 32% at the 95th percentile) or excessive number of latency changes, and (3) Tenants may have incentives to selfishly move traffic to low latency classes (to boost the performance of their applications). We showcase this third insight with an example using *rsync* synchronization. To the best of our knowledge, this is the first paper to reveal the high frequency of TE activity within a large cloud provider's network. Based on these observations, we expect our paper to spur discussions and future research on how cloud providers and their tenants can ultimately reconcile their independent and possibly conflicting objectives. Our data is publicly available for reproducibility and further analysis at http://goo.gl/25BKte.

1 INTRODUCTION

Cloud provider networks play an essential role in guaranteeing Quality-of-Service (QoS) of tenant applications; however, little is known about how traffic is routed in practice across such networks. Network operators have long relied on Traffic Engineering (TE) tools to optimize the flow of traffic within their networks, with varying degrees of success. At the beginning of the 2000s, shortest-path routing protocols, such as OSPF [28] and RIP, were prevalent despite being ill-suited for TE operation [16]. Since these protocols lack fine-grained routing control, TE optimization was infrequent and suboptimal. TE optimization became prevalent with the advent of MPLS technology [32] and later with the introduction of the Software-Defined Networking [11] paradigm. Since then, Microsoft and Google have reported performing TE optimizations of their wide area networks with a 5-minute period [22, 29].

Are today's TE optimizations hindering network traffic performance? Despite the crucial role played by TE tools, the impact of frequent TE optimization on the network traffic's performance has not been adequately explored. In this paper, we take a first step in this direction by looking at the impact of TE on cloud application's traffic.

Cloud providers perform TE to efficiently utilize their network resources. However, this goal may not align with the needs of tenants' applications and their associated traffic. Fig. 1 shows the Round Trip Times (RTTs) of three different TCP connections established between two machines located in two different Amazon *aws* regions — namely Oregon and Virginia. Each TCP connection uses a distinct TCP source port and periodically generates a ping message to measure the RTT. We observe that each flow experiences different RTT values. Moreover, these RTTs are relatively stable around one value for a period of time before suddenly "jumping" to a different value.

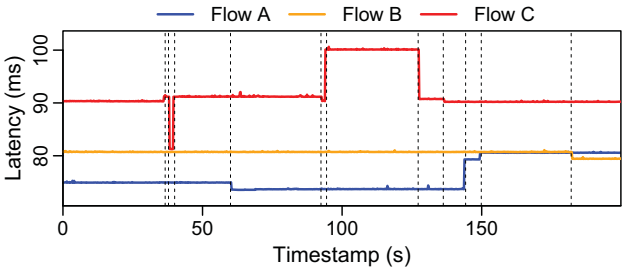


Figure 1: TCP RTT measured between two Amazon VMs deployed in Oregon & Virginia. The black dotted vertical lines highlight moments when RTT latency changed and indicate potential TE activity (e.g., path changes).

This simple experiment leads to three observations: *(1)* TE is operating at a time scale of seconds, *(2)* traffic flows may experience frequent latency variations, and *(3)* TE may be unfair as some flows consistently experience higher latencies (*e.g.* flow C) and/or experience more frequent latency changes (*e.g.* flows A and C) than others.

Highly-variable flow latencies can be cumbersome for application developers. First, high-frequency oscillations in inter-region latencies adds yet another source of uncertainty for application developers — as Microsoft Bing's [7] developers observed [29]. Consequently, guaranteeing latency-related Service Level Agreements (SLAs) for latency-sensitive geo-distributed services [20, 23] can be difficult. Secondly, the performance of congestion control algorithms, both loss-based and delay-based ones, degrade under frequent latency changes, e.g., due to packet re-ordering [8, 25, 35, 38]. Additionally. applications relying on UDP can be negatively affected by packet reordering (e.g., VoIP [24]). Finally, our results show that a fraction of "unlucky" flows can suffer from high RTTs over prolonged periods (hours and days), further exacerbating application performance problems. Our prior work [9] shows that these traffic characteristics have existed consistently for at least a few years (since January 2015) and our continued measurements have shown that they are not transient. This prompted us to take a deep dive and study this phenomenon more comprehensively.

Measuring TE effects in a large-scale Cloud network. In this paper, we study the effects of TE on flow latencies measured among all 16 currently available Amazon *aws* regions, all of which are connected via Amazon's privately-owned infrastructure [17]. We conducted measurements between up to 105 unique region pairs to identify latency characteristics of traffic flows and thoroughly study the three observations presented above.

Results and implications for TE & the design of applications. Our measurements for this paper encompass data gathered over a 2-week span. We have also conducted several other measurement snapshots for AWS from 2015 to 2019. All our data indicates that flows frequently jump between paths that can have as high as 20% greater latency than the lowest observed latency path. We corroborated our findings in discussions with cloud networking experts.

Our analysis shows that the changes in latency are caused by frequent routing changes and are potentially problematic for delivering traffic that desires specific performance guarantees. Moreover, we also found that, despite these frequent changes, there is unfairness in the observed flow latencies — where some flows get the lion's share of *low and consistent* latencies while others are penalized over long periods of time. Flow unfairness can be undesirable, especially for latency-sensitive services deployed across several geographic regions [37]. Last but not least, tenants have incentives to monitor the latency observed per source port and then *selfishly* move latency-sensitive traffic to low latency paths. This could, in turn, overload certain paths and trigger a counter-response from the TE mechanisms, a classic problem known as "Selfish Routing" [33]. This paper does not aim to solve these problems but rather exposes the current effects of TE. The hope is that quantifying these effects would motivate the community to tackle the challenges of designing TE mechanisms suitable for cloud environments.

2 BACKGROUND

This section provides the necessary background on both cloud network architectures and TE techniques to optimize the flow of traffic in cloud networks.

Architecture of a cloud provider network. The three largest cloud providers worldwide (e.g., Amazon AWS, Microsoft Azure, Google Cloud Platform) deploy their networks using a similar hierarchical structure. Within each cloud provider, a set of typically more than 10 *regions* are interconnected by a globally-deployed, singly administered, cloud backbone network infrastructure. All of the regions are subdivided into availability zones, each consisting of at least one datacenter (DC) network. Cloud tenants buy computing, storage, and network resources within or across regions where their services are deployed.

Traffic engineering basics. Network operators perform TE to optimize the flow of traffic in their network, e.g., reducing communication latency, ensuring fairness, and providing traffic isolation without increasing packet drops. TE consists of a closed control loop between a *monitoring* component and a route *computation* component. This loop spans both the data- and control-planes. The data-plane both forwards packets according to the installed monitoring & forwarding rules and collects traffic statistics. The control-plane fetches the collected traffic statistics and feeds them as input to the route computation engine, and then updates the data-plane with newly computed routes (in the form of per-switch forwarding rules).

Multipath forwarding mechanisms. Modern TE mechanisms make extensive use of multipath forwarding [22] to split an aggregate of traffic flows across different paths. Specifically, for each aggregate of flows between two nodes in a network, the operator specifies the *splitting ratio* of this traffic aggregate that should be routed on each path [39]. The operator also specifies whether the splitting of the traffic aggregate should be performed at the *per-flow* granularity (i.e., packets with a common set of header fields traverse the same path) or at the *per-packet* granularity (i.e., packets are routed using some sort of weighted round-robin scheduler). Network operators have traditionally operated networks using per-flow splitting mechanisms, e.g., based on hash calculations on the packet header (i.e., specific IP and TCP/UDP fields). The reason for using per-flow mechanisms stems from the way TCP congestion control (such as, NewReno and Cubic) *reacts* to the presence of packet reordering by assuming network congestion, which may not always be the case. Specifically, when the receiver of a TCP flow receives packets out of order, it sends duplicate acknowledgments (dupACKs) for the first missing packet in the received TCP stream. When the sender receives three dupACKs, the sender assumes that the network is congested and reduces its congestion window, and consequently its sending rate. Packet reordering may arise when a routing (re-)configuration takes place and a flow is moved from a high latency path to a low latency path. This can happen, for

example, when network operators (or automated TE tools) update any hash-based traffic splitting ratios in the network, which is an operation known to cause flow rehashing [21, 39].

Evolution of TE in cloud networks. Traditional TE tools and mechanisms (e.g., OSPF [28] and RIP [18]) are ill-suited for supporting the ever-growing inter-region traffic of large cloud networks. Such tools are tailored to shortest path routing and only support uniform splitting ratios, thus they lack the fine-grained routing control needed to effectively utilize network resources. Consequently, wide-area network operators have long relied on MPLS-based OSPF extensions for improved TE [15], by periodically re-optimizing network routes using preconfigured route computation algorithms (e.g., Constrained Shortest Path First (CSPF)) according to the measured traffic volumes. Recently, the three largest cloud providers (*aws*, *azure*, and *gcp*) transitioned to SDN-based networks [17, 20, 22], in which network operators have full control of the monitoring and route computation processes using the well-defined interfaces between the control- and data-planes.

3 MEASUREMENT METHODOLOGY

To understand the extent to which a cloud provider's TE operations affect the latency of traffic in their networks, we performed a study across Amazon *aws* [17], the largest worldwide cloud provider network in terms of market share [27].

Goals of the study. In Fig. 1, when our application established three TCP connections to another region, we noticed that the latencies were not only different among the three connections but also changed over time and in a step-wise manner. Many questions arise from this measurement: *(i)* are path changes the cause of the steps in the RTT trace? if so, *(ii)* how and what is the persistence of each path? *(iii)* what is the difference between the minimum and maximum observed latency, *(iv)* are there source ports that experience long-term better RTTs (and One Way Delay (OWD) latencies) with respect to other ports? *(v)* do we observe the same behaviour across all region pairs and to what extent?

In order to answer these questions, we performed a systematic set of measurements of the *aws* inter-region network. We note that, while we suspect that Amazon *aws* uses hash-based traffic splitting mechanisms, our measurements and conclusions do *not* rely on any specific assumptions of how the traffic splitting mechanisms are implemented. Our first goal was to identify whether the step-wise latency changes correspond to actual path changes (possibly due to TE operation) and analyzing the frequency and extent of latency changes both *spatially* — across different flows — and *temporally*.

3.1 Detecting TE activity

To detect latency changes that occur as a result of path changes, we introduce a methodology for filtering out congestion noise.

Measuring RTT and OWD flow latencies. Before delving into the intricacies of the algorithm, we first describe our measurement setup. We performed both RTT and OWD measurements using TCP and UDP flows. On all Virtual Machines (VMs) we use `chrony` configured to access the Amazon Time Sync Service [6], which provides high precision time synchronization. As we do not have visibility inside the Amazon *aws* network or to its routing, we assume flows can use different forward and backward paths for their traffic. As such, from here on, we use the term "path" to simply refer to the joint forward and backward communication paths used by a flow for the RTT measurements and the forward path only for OWD measurements.

Decoupling propagation delays from congestion. In Fig. 1, one can easily observe that the latency experienced by a flow consists of a *base propagation* delay (due to traversing a certain geographical distance over a given routing path) and spurious congestion delay. In this figure, we identified several moments (each denoted by a black dashed vertical line) where the minimum latency observed during a certain time window (i.e., the base propagation latency) suddenly changes by at least 0.5 ms.

One way to infer TE activity in the cloud's backbone network is to detect these sudden changes in the base propagation latencies while filtering out the congestion component — which otherwise might be falsely interpreted as path changes resulting in an overestimation of TE activity. This problem is well-known and several techniques have previously been proposed to extract base propagation delay from latency measurements when possible (i.e., when congestion is limited and buffers periodically drain) [2, 10, 26]. Roughly speaking, these techniques are based on computing a rolling minimum of the last k observed latencies. Using this approach, we were unable to achieve a 0% rate of false positive, i.e., avoiding non-existing path changes. The minimum false positive rate that we achieved was 7.69% with a rolling window of $k = 20$. Therefore, we enhanced the rolling minimum technique as we will explain later in this section. It is also worth noting that using traceroutes to detect paths [3] requires cooperation of the cloud provider and can fail to correctly detect paths when routers do not respond with ICMP errors.

We first present our technique to extract base propagation latencies from *aws* measurements while removing congestion. Then we validate the accuracy of this technique to detect TE activity (i.e., path changes). We want to stress that we do not claim our technique is general but rather we tailored our path change detection to the *aws* network and its observable congestion profile. We further discuss this aspect in Section 6.

Our approach to detect path changes. We use a conservative approach that computes the "mode" in a sliding window (see Fig. 2 for an example). For each flow, we first aggregate the measured latencies (top-left part of the figure) across 1 second intervals by computing the minimum latencies across 5 probes[1]. We then apply our sliding window mode-based function using a window size of 4 seconds. While sliding our window we maintain an estimate of the current base propagation latency. If 3 out of 4 observations in a window are within 0.5 ms of the minimum value in the window, we say that the minimum is *stable*. Two cases are possible: (a) a window has a stable minimum or (b) not. In case (a), i.e., the minimum is stable, if the minimum is also within ± 0.5ms from the currently estimated base propagation latency, we assume that the estimated base propagation latency is stable and do not modify it. Otherwise

[1] Probes are sent at 200 ms intervals

(i.e., the new minimum is not within ± 0.5ms from the estimated propagation latency), we update the propagation latency to the new minimum. In case (b), i.e., the minimum is unstable, hence we ignore the value and keep the estimated base propagation latency unchanged.

In Fig. 2, the minimum of 7.9 ms in window w_n is stable since at least 3 latency samples in the window are within 7.9 ± 0.5 ms. Therefore, we update the estimated propagation latency with this stable minimum. Window w_{n+1} is stable and the minimum of 8.2 is within 7.9 ± 0.5 ms. Therefore, we do not update the base propagation estimate. Both windows w_{n+2} and w_{n+3} are unstable because of the three samples 6.8, 8.2, and 9.8ms. For this reason, we keep the previously computed base propagation latency as our estimate. Window w_{n+4} is stable and the minimum of 6.8 ms is not within 7.9 ± 0.5 ms; therefore, we update the new base propagation latency with this new value.

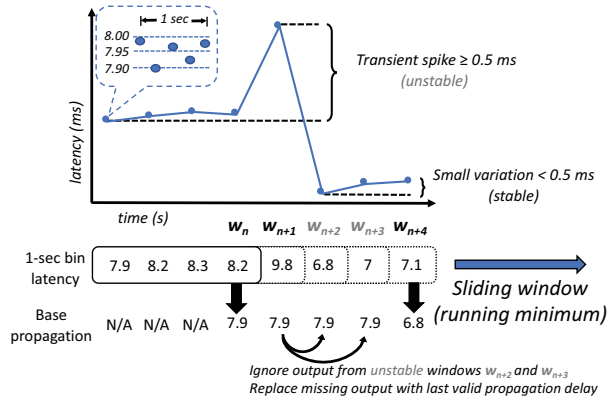


Figure 2: An overview of path detection using a mode-based sliding window that computes a running minimum. We depict an example where transient spikes (≥ 0.5 ms) are filtered out, whereas actual base propagation changes (with stable latencies) are eventually detected.

To further remove congestion noise, we cluster our latency classes into 1-millisecond bins, by rounding to the nearest millisecond. We then compute the churn — or number of flows admitted — to all observed clusters. If a cluster has less than 0.1% of the sum of all cluster admissions, we conservatively consider it as congestion noise and remove it from our measurements. We presume that, in these cases, such events are likely to occur if there is persistent congestion — with minimal latency oscillations which can result in false positives. As such, given that Amazon offers 99.9% availability for most of their inter-region services, we opt to remove clusters receiving less than 0.1% activity. We refer to the sequential execution of the mode sliding window followed by the above filtering technique as the *path change detector*.

Note that if a flow is moved to a path with the *same* latency, we are unable to detect this change, thus our conservative approach has false negatives. In the next subsection, we show that our approach can accurately distinguish path changes from congestion events in the *aws* network. This eliminates false positives in path change detection.

3.2 Accuracy of path change detector

To verify that the latency changes observed by our measurements (e.g., Fig. 1) are due to routing changes in the network (and *not* caused by our measurement technique, congestion, cloud interference, etc.), we performed three sets of experiments: (i) we measured RTTs among VMs *within* one availability zone between two different DCs, (ii) identified and correlated packet reordering across the *aws* WAN with the changes in the extracted base propagation latencies from RTT measurements, and (iii) identified and correlated changes in network paths using traceroute with measured OWD. Additionally, we discussed our findings with cloud networking experts, who had read an early draft of our paper, and they supported these findings.

Latency measurements within the same region. First, we measured network RTT between different VMs located in the same region but in different availability zones. Our results show that intra-region latencies are stable and do not experience changes such as those seen in Fig. 1. We conclude that latency fluctuations are caused by traversing the inter-region cloud backbone network.[2]

Packet reordering correlation. Next, we deployed a pair of `c5.large` VMs in the Oregon and Virginia *aws* regions. Using our custom traffic generator (deployed in Oregon and Virginia) we measured RTT by sending UDP packets every 0.5 ms continuously for 2 hours. Each packet had a unique ID and a monotonically increasing sequence number. We note that no packet was dropped during the experiment. By identifying inconsistencies in the sequence numbers at the receiver, we could detect when packets arrived out of order. Next, we correlated the instances of out-of-order arrivals with the observed RTT between VMs in Oregon and Virginia. The top graph in Fig. 3 shows a 2-hour snapshot of RTTs between Oregon and Virginia. The red vertical lines indicate when a sequence of packets arrived out of order. Note that such reordering events are strongly correlated with *decreases* in measured RTT. In some cases, possibly due to asynchronous network updates, we observe that the base propagation latency may quickly oscillate between two values before stabilizing to one of them. These events may mistakenly give the impression that, in Fig. 3, we observed packet reordering events when the base latency increased, as the path latency decreases are indistinguishable.

As discussed in Section 2, when a network flow is moved (e.g., due to TE) to a path with lower latency, subsequent packets can overtake the in-flight packets on the higher latency paths; thus, they arrive out of order. In contrast, moving to a path with a higher latency does not affect the order of packet arrivals. Therefore, packet reordering events can only be used to detect path changes to lower latency paths. Furthermore, routing changes that cause RTT decreases of less than 0.5 ms (e.g., at 14:00 on the x-axis) may not be detected by our measurement as packet probes are transmitted exactly every 0.5 ms. We recall that since we do not observe any packet drops, packet reordering can only occur due to path changes or some specific switch's packet schedulers; hence, they are not due to congestion in the network. By observing a high correlation between packet reordering events and latency decreases, we exclude those reordering events due to specific packet schedulers at the switches.

[2]For brevity, we do not show these results, but all samples can be found in [1].

Increasing the frequency of probe generation could increase the sensitivity of our measurement.

To evaluate whether our path change detector detects all the path changes associated with packet reordering (and thus changes to lower latency paths), we fed as input the packet trace measurements to our path change detector and plotted the output in the bottom subplot of Fig. 3. We plotted a dotted red line when our path detector detects a negative/positive latency path change. We observe that our path detector accurately detect the vast majority of the negative latency path changes corresponding to packet reordering events (some of them may not be detected due to noise in the latency signal). More importantly, out path change detector *never* reports a non-existing latency-decrease path change, thus achieving zero false positive rate. We note that the packet reordering correlation cannot be used to verify the accuracy of latency-increase path change detection.

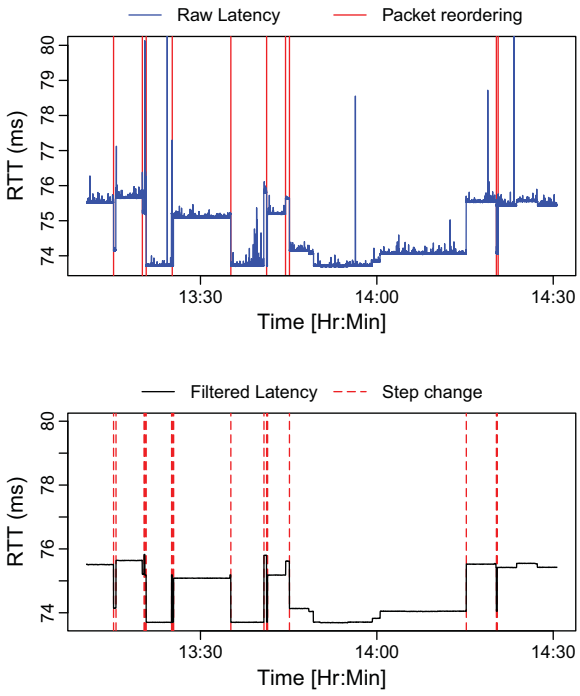


Figure 3: Measuring path detection accuracy using reordering events. The top plot shows observed RTTs and packet reordering occurrences whereas the bottom plot shows the filtered RTTs computed via our path change detector as well as the detected path change events.

Traceroute correlation. Finally, we identified and correlate changes in network paths (using traceroute) for a sample TCP flow between VMs in Oregon and Virginia with the measured OWD.

We established a TCP flow between these two regions and sent ping probes at a rate of 5 per second to measure RTT and OWD. In parallel, we ran traceroute measurements that were obtained using our custom traceroute (based on the principles described in [3]). The traceroute crafted network packets that matched the 5 tuples of the sample TCP flow to guarantee that both packets from the traceroute and the TCP flow would follow the same network path.[3]

We ran traceroute every 20 seconds and recorded the observed network paths. By analyzing the sequence of measured paths we identified the moments when changes in routing happened. These events were correlated with the OWD latency measured using TCP flows. Fig. 4 shows the OWD network latency with routing changes shown as vertical red lines. This figure illustrates that every change in OWD is accompanied by a change in the forward network path (unless the duration of such a change was less than 20 seconds as the traceroute would miss such event). Note that the opposite is not always true; as two distinct network paths may have nearly identical network latency, thus resulting in no network latency change.

As with packet reordering, we show in Fig. 4 that our path change detector identifies all the traceroute path change events that incur a latency change of at least 0.5 ms.

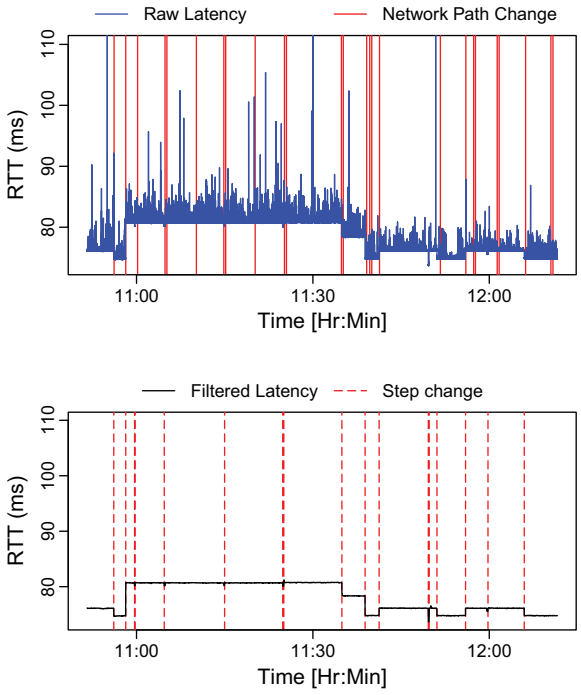


Figure 4: Measuring path detection accuracy using inflow traceroutes. Top plot shows path changes captured via traceroutes and bottom plot shows same output as Fig. 3.

3.3 Measurements description

We have so far established that it is feasible to rely on changes in the base propagation latency to accurately detect path changes in the *aws* network. We now discuss in detail the set of long-term measurements performed to answer the measurements goals stated at the beginning of this section.

[3]This is the only experiment where we assume the presence of per-flow load balancing in the network.

Our measurements can be broken into two sets: (i) *Macro-scale* measurements that collect all the observable base propagation latencies between 120 unique Amazon *aws* region pairs and (ii) *Micro-scale* measurements that are used for analyzing in deeper detail the performance of the individual flows (e.g., number of path changes experienced) and compare among flows (e.g., to examine fairness) during the complete duration of the measurement. In the first set of measurements, for each pair of regions, we select two machines in two different regions and create a large number of flows using many different source ports. We randomly pick a set of 512 ports every 30 seconds. The goal is to compute the maximum difference between observed base propagation latencies among all the DC pairs. The second set of measurements targets a limited number of selected region pairs, but generates a greater number of probes per second in order to detect latency changes at a fine-grained temporal resolution. To gather per-flow statistics, we randomly select 512 ports and keep them unchanged for the entire duration of the measurement. We describe the specifics of our experimental configurations below and summarize them in Table 1.

Table 1: Testbed configuration for measurements.

Config.	Macro-scale	Micro-scale
# of DC Pairs	120	4
# of Flows	512	512
Probing Rate	10 probes every 30s	5 probes/s
Flow Generation	Dynamic (every 30s)	Static
Duration	2 days	1 week
Ping Mechanism	Raw Sockets	TCP Ping
Results	Fig. 5	all except Fig. 5 & 15

Testbed. We use two distinct setups to conduct our macro and micro-scale measurements on *aws*. Our macro-scale testbed probes paths between 16 regions located in 16 different Amazon *aws* regions. We create `c5.large` EC2 instances in each region and connect them all-to-all, for a total of 120 unique region pairs. In contrast, our micro-scale measurements are deployed in a more limited setting and focus on only 4 region pairs.

Probing. In our macro-scale experiments, we use raw sockets to perform an exhaustive search of base propagation latencies by emulating TCP connections. This allows us to efficiently probe paths by creating custom TCP packets with different source ports *without* being bound by the maximum number of TCP connections that can be created on a given VM instance. We perform a search by picking 512 random ports every 30 seconds. For our micro-scale measurements, we focus on only four region pairs — selected among the top 10% pairs in terms of greatest differences in the previously observed propagation latencies — and no longer emulate TCP connections, but instead rely on hash-consistent TCP ping, where we maintain 512 static TCP connections between each regions pair. We verified that our probes are never retransmitted twice during our measurements, e.g., due to packet drops.

4 ANALYSIS OF THE RESULTS

We showed in Section 3 that we can use our path change detector to extract the base propagation latency of the path through which a flow in being routed as well as detect when the flow is rerouted over a different path. We refer to *flow latency* as the base propagation latency of the path through which the flow is being routed. We say that a flow *changed path* if the flow latency changes by at least ±0.5 ms, as described and validated in Section 3.

In this section, we evaluate the frequency and magnitude of the flow latency changes observed between *aws* regions using the measurements obtained from both the macro- and micro-scale experiments described in Section 3. These results shed light on the extent to which TE operation is performed within the *aws* cloud backbone network and the assumptions that can be made by cloud application developers. Our main findings can be summarized as follows: *(i)* between two machines there may exist many different path with diverse latencies (Section 4.1), *(ii)* some paths may suddenly become unavailable (Section 4.2), *(iii)* half of the observed paths persist for 10 seconds or less (Section 4.3), *(iv)* the lowest latency paths have the highest flow churn (Section 4.3), and *(v)* some flows may *consistently* experience worse propagation latencies than others (Section 4.4). Results (ii-v) are based on the micro-experiments, while result (i) is based on the macro-experiment.

4.1 Flow latencies vary greatly across regions

To understand the spectrum of possible base propagation path latencies experienced by a flow, we first measure the highest (max) and lowest (min) flow latency for each of the unique region pairs as described in the macro-scale paragraph in Section 3. Fig. 5(a) shows a CDF of the distribution of $max - min$ latency differences across all DC pairs. We see that the median of these differences is at 10 ms, but can increase to roughly 35 ms at the 95$^{\text{th}}$ percentile.

While these differences appear to be non-negligible, it is hard to reason about their significance in isolation since the minimum inter-region latencies can be 10s to 100s of milliseconds. In order to put these numbers into perspective, Fig. 5 (b) shows a CDF of the *relative* $\frac{max}{min}$ flow latency percentages; where the latency change is computed as the percentage increase from the lowest (min) to the highest (max) flow latency observed for each region pair. We can clearly see a heavy-tailed distribution, with changes of up to 32% at the 95$^{\text{th}}$ percentile. When such *large jumps in latency occur* this can negatively affect services that require consistent inter-region request-response latencies.

4.2 Availability of paths

While it is clear that the maximum latency change can be quite profound, this value is computed based on the best and worst flow latencies observed during a period of 2 days. However, it is not immediately clear whether these latency classes are available universally or only during certain time periods. To evaluate this, we took a closer look at four region pairs, namely: “Oregon-Virginia”, “Sydney-Tokyo”, “São Paulo-Montreal”, and “Singapore-Paris”. We chose these pairs to represent different continent combinations (with the exception of “Oregon-Virginia”, where both are in the US) from the top 50$^{\text{th}}$ percentile of regions pairs that showed the largest absolute latency differences. We plot the latency differences

for these regions pairs in Fig. 6 where "Oregon-Virginia" has the highest percentage change at 40% and "Sydney-Tokyo" being the lowest at roughly 10%.

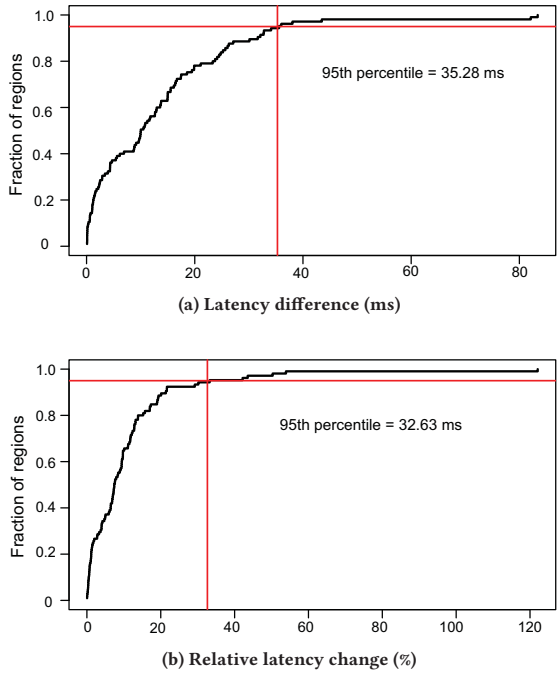


Figure 5: CDF of the (a) absolute differences ($max - min$) and (b) relative percentages ($\frac{max}{min}$) flow latencies across all the *aws* region pairs.

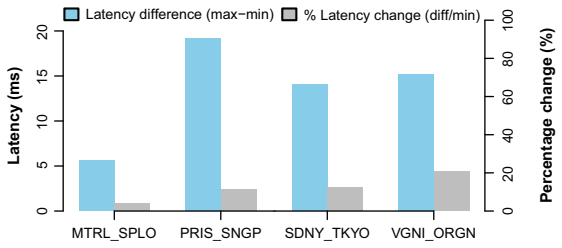


Figure 6: Absolute and relative latency changes for four different Amazon *aws* DC pairs.

To show the availability of paths across time, in Fig. 7 we plot the flow latency percentiles of 512 flows observed for each region pair over time. The blue area corresponds to the inter-quartile, the bottom (upper) dark gray area corresponds to the 5th to 25th and the 75'th to 95th interpercentile range, and the bottom (upper) light gray area corresponds to the 0th to 5'th and the 95th to 100th interpercentile range. The plotted latencies are the average base propagation latencies obtained from the path change detector (fed with the raw measurements) binned in 1 hour intervals, and thus do not include congestion-induced spikes. We can clearly see in all subfigures (with the exception of (a)) that the latencies experienced by the flows can change dramatically with time. The inter-quartile ranges (highlighted in blue) can shift sporadically over time, indicating that certain paths become less prevalent or harder to reach (e.g., due to increased load or failures). Moreover, looking at the minimum observed latency we observe that lowest-latency paths become unavailable over time. For instance, in Fig. 7 (c), we saw on Day 3 a new low latency path that lasts for only a few hours and then disappears for the rest of the week. This could be caused by TE operations that reallocate the paths across different region pairs or perhaps because of different types of customers.

Summary. Flow latencies between a single region pair can change dramatically (by up to 32% at the 95th percentile). Moreover, some latency classes are unavailable during specific hours of the day, suggesting that specific paths become unavailable.

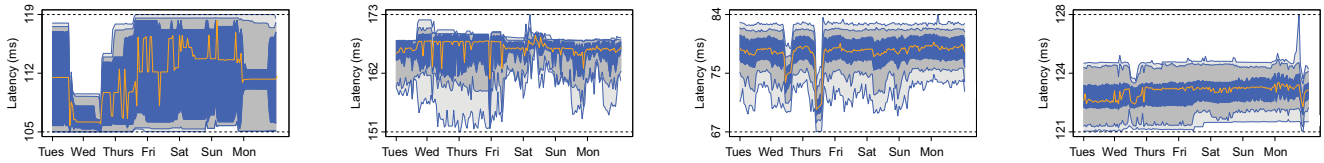
4.3 Flow latency persistence

In the previous section, we explored differences between flow latencies and how flows are distributed across paths varies over time. However, it is desirable to know *how often* these flow latencies change. In fact, frequent routing changes can be a problem both for cloud network operators as well as for a cloud's customers since frequent changes can have a negative impact on TCP flows due to packet re-ordering and inaccurate RTT estimation [35].[4]

We study these changes by first looking into how long a flow persists in a latency class. Fig. 8(a) shows a CDF of flow latency durations for all flows for each region-pair. The y-axis represents the fraction of events in which a flow moved to a new path and persisted on that path for less than x seconds before being rerouted to a different path. Surprisingly, from this plot we find that in "São Paulo-Montreal" roughly 40% of flows moving to a new path continue to use this path less than 10 seconds before moving to another path. This can lead to packet re-ordering for flows longer than 10 seconds — which, to put into context, is more than 75% of the inter-region flows in Facebook's caching clusters [34]. In contrast, for "Sydney-Tokyo" flows change their paths rather infrequently and 50% of those flows moving to a new path continue to use that path for more than 280 seconds. As previously seen in Fig. 7, this region-pair rarely exhibits any changes across its flow latency distributions, suggesting that its paths are unlikely to be subject to frequent TE changes. In Fig. 9, we break down the flow persistence graph into the forward and reverse paths.

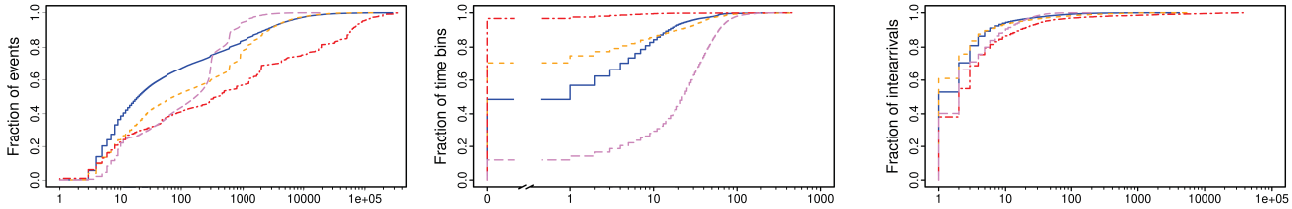
Secondly, we examine whether flows change their paths in tandem, i.e., whether flows move to different paths at the same time. Fig. 8(b) plots the number of flows (out of 512) that changed their path at least once during a 20-second interval. The y-axis presents the fraction of 20 second time interval bins within which at most x flows change their path in a single bin. We find that, "Singapore-Paris" and "São Paulo-Montreal" exhibit an *all-or-nothing* property where either the majority of flows are affected or none of them are affected. This indicates that TE events, when triggered, can dramatically impact a large population of flows, perhaps due to failures in the network, planned topology

[4]Evaluating the impact of latency instability on different congestion control mechanisms is outside the scope of this paper.



(a) Sydney-Tokyo (b) Singapore-Paris (c) Oregon-Virginia (d) São Paulo-Montreal

Figure 7: Changes in the distribution of latencies across 512 different flows binned in 1 hour intervals.



(a) Flow persistence (s) (b) # of flows changing paths (c) Flow path change interarrival time (s)

Figure 8: CDFs showing characteristics of traffic class changes across all the monitored flows. Subfigure (a) shows every path change event of a flow, (b) shows the # of flows that experienced at least 1 path change in each 20 second time interval bin, and (c) shows the interarrival time between any two path change events observed in the network. Legend: São Paulo-Montreal ———; Paris-Singapore - - - - -; Sydney-Tokyo -·-·-·-; and Oregon-Virginia - - - - -.

augmentation, an unwanted cascading effect — where changes in the traffic matrix caused by the initial TE events trigger successive waves of routing changes — or even asynchronous network updates. In contrast, "Oregon-Virginia" ("Sydney-Tokyo") exhibits many small TE operations during (50% of) all the 20-second intervals. To further investigate these cases, Fig. 8(c) shows the inter-arrival time between flow path changes. In this figure, the y-axis shows the fraction of path changes events across all flows such that the time between that path change event and the next path change event (possibly of a different flow) is x. Similar to the previous plot, "Oregon-Virginia" also demonstrates unique characteristics. On average, its flows change paths more frequently — with interarrival times of inter-flow path changes of less than 5 seconds at the median. Moreover, this rate of changes can increase leading to many path changes in 150 ms. This may suggest different subclasses of TE events — with major events (likely in response to significant traffic changes) occurring less frequently but affecting more flows. Minor TE events might be triggered in reaction to smaller congestion events and consequently only need to reroute a small subset of the flows (e.g., by slightly modifying traffic splitting ratios).

Different TE mechanisms impact flow persistence. Based on discussions with cloud networking experts, we decided to verify how the persistence of the flows would vary during different times of the year. We measured the *aws* network for several weeks, starting from December 2018 until the end of February 2019, for the critical pair of Oregon and Virginia regions. We observed a significant change during the month of February. Conversations with cloud networking experts suggested that this could have been due to a change of the *aws* configuration to a less reactive TE after the high-load period of the Christmas holidays and January. We plot the CDF of the flow persistence (similarly to Fig. 8 (a)) in Fig. 10 for two distinct weeks of February when we observed the change. While the median flow persistence clearly decreased by roughly a factor of 1.5, one still observes a very low path persistence for ~30% of the flow path change events. However, a thorough investigation of these results is outside the scope of this paper.

Are flow latencies correlated with flow churn? While flows can change paths quite frequently (in the order of seconds), we have not yet studied the frequency of path changes of a flow with respect to its flow latency. Specifically, do flows on low latency paths experience higher *flow churn*, i.e., the number of flow rerouting events on a specific path. To investigate this we group paths by rounding the path latency and call the rounded latency a *latency class*. We choose the relatively stable region pair of São Paulo-Montreal to see how flows compete for paths in steady state. We show in Fig. 11 a CDF (blue dotted line) of the flow churn of each latency class. The x-axis shows all the observed latency classes, ordered from left to right by increasing number of flow churns. The corresponding latency of each latency class is shown by the red solid line. We can clearly observe a negative correlation, where paths belonging to low latency classes exhibit higher flow churn (with only a few outliers). We reason that this could occur due to flows being opportunistically routed to low latency paths (provided they are available) with the possibility of subsequently being preempted by higher priority flows. Therefore, given the increased competitiveness for low latency paths, flows experience more churn on such paths, while the reverse is true for higher latency paths. This type of greedy shortest-path TE mechanisms

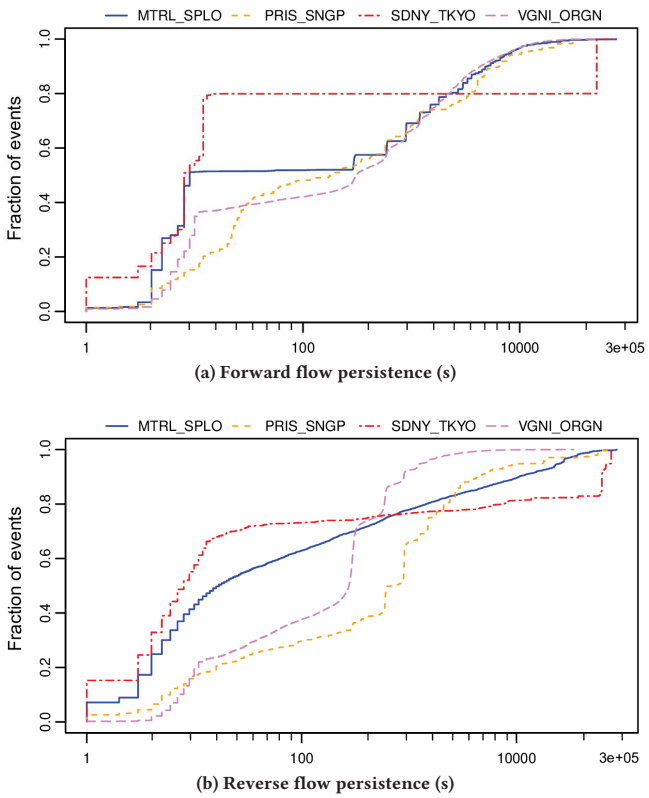


Figure 9: CDFs showing asymmetry of flow latency class persistence across all 512 monitored flows. Subfigures (a) & (b) report results for the latency classes on the forward and reverse paths, respectively.

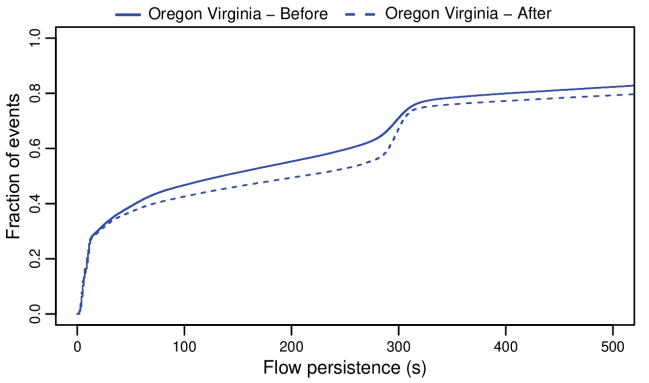


Figure 10: CDF of class durations before and after a TE policy change by Amazon.

have been widely used in MPLS-TE [29, 32] and B4 [22] networks. Our results suggest that the Amazon *aws* network uses similar TE mechanisms, although we did not get confirmation of this from the *aws* team.

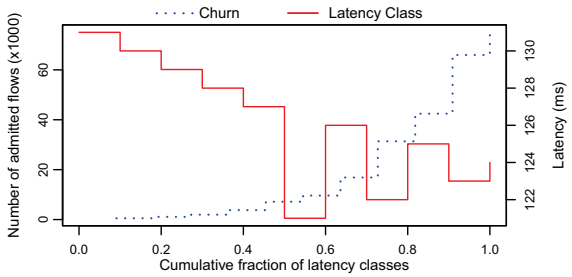


Figure 11: Flow churn observed for latency classes found on São Paulo-Montreal. The blue dotted line is a CDF of the number of admitted flows reported by each latency class. The red line is the latency of the associated class.

Summary. We have shown that flows change their latency classes frequently and more than 50% of the flows moving to a new path change paths within 10 seconds. We have also established that these changes often happen in tandem in certain region pairs. We presented our results to cloud networking experts and they confirmed that low-latency paths tend to experiences higher flow churn, suggesting TE tends to reactively move flows to low latency paths whenever capacity is available. However, as noted earlier these frequent path changes can negatively impact TCP's congestion control algorithm and ultimately negatively affect tenant applications.

4.4 Flow latency fairness

The previous sections highlighted issues due to short persistence of flows on paths. However, one might assume that frequent latency class changes means that flows are unlikely to have unfavorable path assignments for prolonged durations. Unfortunately, our results show that this is not the case.

Fig. 12 shows a CDF of the median, 95th, and 99th percentile base propagation latencies of all the flows over a one week timespan. We see that these plots resemble a step function, but with significant differences among flows. For instance, in Fig. 12(d), the median latency changes from 105 to 118 ms — a 12% increase. The same applies for the higher percentiles as well.

To further study unfairness, we also look at path changes. Fig. 13 shows a CDF of the total number of path changes (over the same one week interval) experienced by different flows. Similarly, we see that certain flows are more likely to exhibit more frequent path changes, thus further exacerbating the level of unfairness among flows.

Summary. These results indicate that *despite* frequent path changes, flows are treated unfairly, with some persistently experiencing poor performance. One reason for this unfairness may lie in the specific implementation of weighted traffic-splitting mechanisms. For instance, implementations based on tables and buckets [39] may tend to rarely move some of the flows, thus leaving them in a poor latency class for a prolonged time.

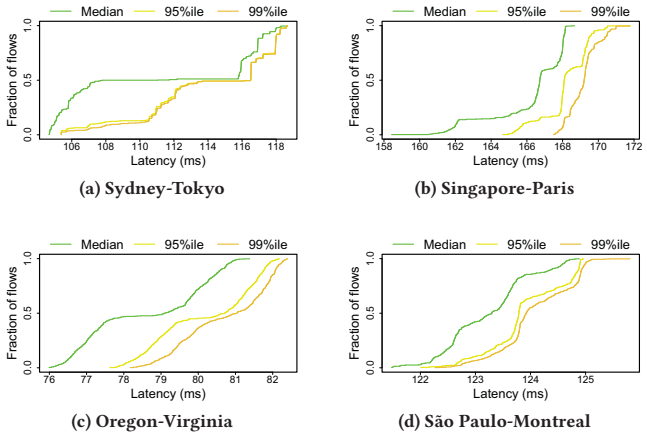


Figure 12: Distribution of flow latency percentiles on 512 monitored flows. Flow latencies can differ, even over two days. This indicates that some flows are consistently unfairly penalized *despite* frequent path changes.

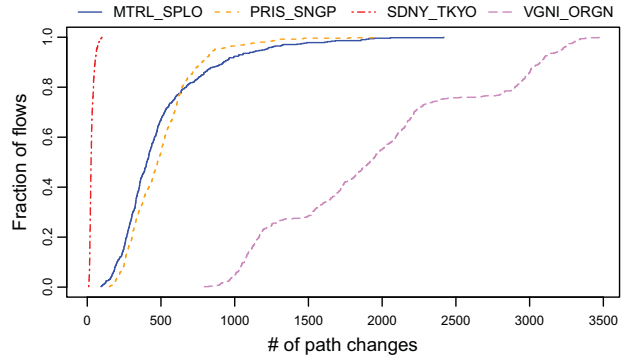


Figure 13: CDF of path changes experienced by different flows.

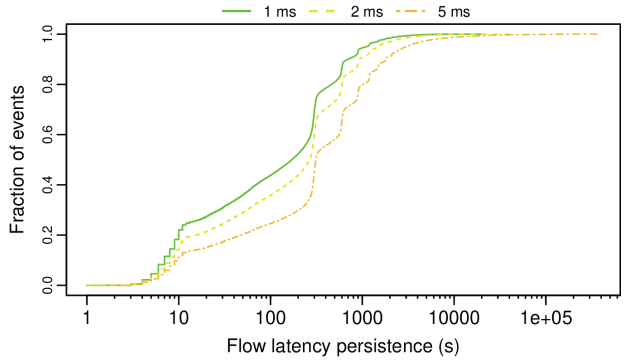


Figure 14: Flow latency persistence expressed in terms of a given "spike" threshold.

4.5 Impact on application performance

We also conducted application benchmarks between instances deployed in Oregon and Virginia. Before discussing these results, it is relevant to look into path change frequency for a given latency change threshold, which we refer to as *spike*, since some applications may not be affected by smaller latency spikes (i.e. < 1 ms). Fig. 14 shows a CDF of the latency persistence for spikes of at least 1 ms, 2 ms, and 5 ms for pings between Oregon and Virginia. We can see that the median flow persistence at 1 ms is ~170 seconds, and this increases up to ~300 seconds at a 5 ms spike threshold. This shows that flows running for more than a few minutes can experience up to 5 ms spikes — almost a 10% latency increase. Spikes less than 5 ms occur infrequently and are not shown in this graph.

For our application benchmarks, we used the *rsync* utility — which is popularly used for mirroring and performing incremental snapshots across storage nodes. We configured our benchmark to transfer 100K files between instances running in both regions, with each file having a constant size of 1 KB. For each rsync run, we use two distinct strategies to choose the source port for the rsync connection: (a) *Random*: This is the default strategy, where the source port is picked by the OS from the ephemeral range of available ports. (b) *Minimum RTT*: This strategy probes 128 different TCP ports for 10 seconds prior to each rsync run. The port with the lowest average RTT is chosen and a NAT rule is inserted to force the next rsync call to utilize the selected port. We repeated these experiments 500 times and collected the runtimes of rsync calls. Fig. 15 shows a CDF of the rsync runtimes for both configurations. We can see that, even when applying a simple minimum RTT strategy for preferential port selection, we observe a ~7% reduction at the median. In addition, the resulting rsync performance is more predictable — notably, the 99th percentile for Min. RTT is lower than the median for Random and the standard deviation drops by a factor of 2.8x, *i.e.*, from 259 ms to 90 ms. By analyzing TCP dumps of this traffic, we observed that these differences are mainly due to the RTT of the assigned path — which directly impacts the throughput of TCP. We also note that this is not an artifact of TCP slow start, since rsync maintains the same TCP connection across an entire run (spanning 100K files). The impact on shorter flows should be even more noticeable and we leave this for future work.

Cloud provider tenants can arguably use similar strategies to reduce latency for their geo-distributed applications. However, this could increase unfairness, as tenants that do not utilize this information would be prone to using unfavorable path assignments. Moreover, this behavior might be at odds with the TE of the cloud provider — which could be trying to move flows away from congested paths. This raises the important question as to whether cloud providers should make path allocations more transparent and/or allow tenants to control such allocations (perhaps for a price). A game theoretic analysis could aid in answering such questions and we leave this investigation as future work.

Summary. Application performance can be hampered by unfavorable path assignments. Tenants might be incentivized to actively pick ports with the lowest observed latencies in order to achieve better performance. However, this could also exacerbate unfairness among tenants and work antagonistically with the cloud provider's TE policies.

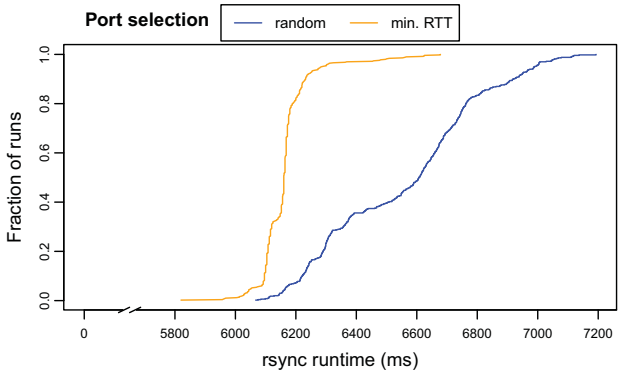


Figure 15: CDF of rsync runtimes for transferring 100K files between Oregon and Virginia. Two different strategies for port selection are shown.

5 RELATED WORK

There has been a great amount of work concerning measurements of Internet routes across a variety of performance dimensions. In this section, we discuss only the most closely related work. Several previous works investigated the impact of TE mechanisms on flow performance [4, 12]. Some of these results showed how the performance of a flow can be affected during reconvergence of routing protocols in the wider Internet, possibly including paths across different domains [31]. In contrast, our work focused on *intra*-domain routing where the TE modification is controlled by a single entity. Others have shown how MPLS-based TE can potentially move flows of traffic to high-latency paths in response to routing reoptimization computation [29]. This is to be expected as TE has to move flows of traffic away from congested shortest paths. However, the authors of this previous work do not discuss whether some source ports experience much worse latencies (i.e., flow unfairness). Additionally, we observed a much higher frequency of path changes than expected, possibly due to SDN control of the network.

The techniques to identify base propagation latency are not novel and the concept of identifying base propagation latency by tracking a minimum latency sample has been previously used in [2, 10, 26].

Another large body of literature focuses on inferring the topology of load-balanced networks. The Multipath Detection Algorithm [5] together with Paris [3] and Tokyo [30] traceroutes improve upon traditional traceroute by avoiding measurement anomalies due to load balancing in the network. DTRACK [13] is a probing tool that predicts routing changes and decides on the number of probes necessary to identify a new network path. In contrast, our work does not aim to inferring the network's topology, rather our goal is to observe the *impact* of TE on flow latency persistence and unfairness across different source ports over time.

Using source port manipulation to send traffic via the highest-performance paths has received some attention in the context of datacenter networks [14] and MPTCP [19]. However, our work differs in several ways. First, we do not claim any novelty in using source ports to route traffic along different paths. Second, MPTCP periodically opens connections on new source ports to discover better paths. We believe MPTCP — and MultiPath protocols in general, could use dynamic port search techniques to *proactively* select the best performing source ports. Finally, these prior works do not investigate the impact of TE on flow latency.

6 DISCUSSION

Comparisons with other Cloud Provider networks. This paper does not aim to compare different cloud providers' networks. We merely highlight the presence and effects of very reactive TE mechanisms in the largest cloud provider's network. However, we also ran packet reordering experiments on Google Cloud for its corresponding Oregon-Virginia pair. Our results indicated fewer path changes, only occurring on the order of tens of minutes, leading us to believe that they utilize less reactive TE policies for this particular pair. This is possibly due to over-provisioning and/or more predictable traffic demands as most applications in the Google network are controlled by Google and not tenants. We leave the study of other cloud providers' networks as future work.

Tuning the path change detector. We tuned our path change detector specifically for the Amazon *aws* network by setting the path change threshold to 0.5ms and validated this value with additional measurements (see Section 3). We note that the path detector must be carefully tuned and validated for each network, using the techniques described in Section 3.2. We leave this task as future work.

Impact of VM instances. One may wonder whether the latency variations are due to delays in the aws VM instances. As mentioned in Section 3.2, we performed latency measurements within the same region and observed stable latencies. Furthermore, our packet reordering measurements further confirmed the observed latency decreases were due to path changes.

Impact of different packet types (e.g., TCP, UDP). Given the inherent blackbox measurements of this paper, we asked ourselves whether some of our traffic was affected by the fact that we used carefully crafted TCP and UDP packets. We shared the results with cloud networking experts and were able to corroborate that what we observed was due to TE activity and not simply an artifact of our probing technique.

Use of flowlet switching. We have conducted packet train experiments that transmit UDP packets at up to 5000 packets per second to see whether the *aws* network uses flowlet switching [36] — a load balancing technique that aims to preserve flow path assignments for larger batches of packets. Even at these higher packet rates, we still observe packet reordering, which leads us to believe that the network does *not* employ flowlet switching.

7 CONCLUSIONS

The growth of Internet applications with low-latency and high-bandwidth requirements places tremendous challenges on network operators. To investigate our observations of latency variations, we performed a large-scale measurement of the Amazon *aws* network and devised techniques to accurately detect path changes. Our results unveiled some surprising results. TE mechanisms in this network seem to operate around at approximately 10-second

intervals, well below previously reported time scales. Consequently, flows of traffic may be subject to frequent and sharp latency changes as well as persistent unfair treatment. Flow latencies between a single region pair can change dramatically (by up to 32% at the 95th percentile) and expose traffic to greater unfairness across flows. Finally, tenants have incentives to move their traffic to low-latency paths as demonstrated in our *rsync* use case. We believe this paper will spur discussions on the impact of high-frequency TE on the design of congestion control mechanisms and cloud applications.

ACKNOWLEDGEMENTS

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 770889). This work was also funded by the Swedish Foundation for Strategic Research (SSF).

REFERENCES

[1] All measurement data used in this paper can be found via this link, goo.gl/25BKte
[2] Arun, V., Balakrishnan, H.: Copa: Practical delay-based congestion control for the internet. In: 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). USENIX Association (2018)
[3] Augustin, B., Cuvellier, X., Orgogozo, B., Viger, F., Friedman, T., Latapy, M., Magnien, C., Teixeira, R.: Avoiding traceroute anomalies with paris traceroute. In: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. pp. 153–158. ACM (2006)
[4] Augustin, B., Friedman, T., Teixeira, R.: Measuring load-balanced paths in the internet. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. pp. 149–160. ACM (2007)
[5] Augustin, B., Friedman, T., Teixeira, R.: Multipath tracing with paris traceroute. In: End-to-End Monitoring Techniques and Services, 2007. E2EMON'07. Workshop on. pp. 1–8. IEEE (2007)
[6] AWS, A.: Setting the Time for Your Linux Instance, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/set-time.html Accessed 2-Feb-2019
[7] Bing: Microsoft Bing, https://www.bing.com Accessed 1-Oct-2018
[8] Blanton, E., Allman, M.: On making TCP more robust to packet reordering. ACM SIGCOMM Computer Communication Review **32**(1), 20–30 (2002)
[9] Bogdanov, K., Peón-Quirós, M., Maguire Jr, G.Q., Kostić, D.: The nearest replica can be farther than you think. In: Proceedings of the Sixth ACM Symposium on Cloud Computing. pp. 16–29. ACM (2015)
[10] Bogdanov, K.L., Reda, W., Maguire Jr, G.Q., Kostić, D., Canini, M.: Fast and accurate load balancing for geo-distributed storage systems. In: Proceedings of the ACM Symposium on Cloud Computing. pp. 386–400. ACM (2018)
[11] Casado, M., Freedman, M.J., Pettit, J., Luo, J., McKeown, N., Shenker, S.: Ethane: Taking control of the enterprise. In: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. pp. 1–12. SIGCOMM '07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1282380.1282382
[12] Cunha, Í., Teixeira, R., Diot, C.: Measuring and characterizing end-to-end route dynamics in the presence of load balancing. In: PAM. vol. 11, pp. 235–244. Springer (2011)
[13] Cunha, I., Teixeira, R., Veitch, D., Diot, C.: Predicting and tracking internet path changes. ACM SIGCOMM Computer Communication Review **41**(4), 122–133 (2011)
[14] Detal, G., Paasch, C., van der Linden, S., Mérindol, P., Avoine, G., Bonaventure, O.: Revisiting flow-based load balancing: Stateless path selection in data center networks. Computer Networks **57**(5), 1204–1216 (2013)
[15] Dhody, D., Palle, U., Singh, R., Gandhi, R.: PCEP Extensions for MPLS-TE LSP Automatic Bandwidth Adjustment with Stateful PCE. Internet Drafts **PCE Working Group** (Nov 2018), https://tools.ietf.org/html/draft-ietf-pce-stateful-pce-auto-bandwidth-08
[16] Fortz, B., Thorup, M.: Optimizing OSPF/IS-IS Weights in a Changing World. IEEE Journal on Selected Areas in Communications **20**(4), 756–767 (2002)
[17] Hamilton, J.: Aws re:invent 2016: Tuesday night live, https://www.youtube.com/watch?v=AyOAjFNPAbA Accessed 17-Jun-2018
[18] Hedrick, C.L.: Routing Information Protocol. Internet Request for Comments **RFC 1058 (Historic)** (Jun 1988). https://doi.org/10.17487/RFC1058, http://www.rfc-editor.org/rfc/rfc1058.txt
[19] Hesmans, B., Detal, G., Barré, S., Bauduin, R., Bonaventure, O.: SMAPP: Towards smart multipath TCP-enabled applications. In: Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies. pp. 28:1–28:7. CoNEXT '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2716281.2836113
[20] Hong, C.Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., Wattenhofer, R.: Achieving high utilization with software-driven wan. SIGCOMM Comput. Commun. Rev. **43**(4), 15–26 (Aug 2013). https://doi.org/10.1145/2534169.2486012
[21] Hopps, C.: Analysis of an Equal-Cost Multi-Path Algorithm. Internet Request for Comments **RFC 2992 (Informational)** (Nov 2000). https://doi.org/10.17487/RFC2992, http://www.rfc-editor.org/rfc/rfc2992.txt
[22] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A.: B4: Experience with a Globally-Deployed Software Defined WAN. In: SIGCOMM (2013)
[23] Jalaparti, V., Bliznets, I., Kandula, S., Lucier, B., Menache, I.: Dynamic pricing and traffic engineering for timely inter-datacenter transfers. In: Proceedings of the 2016 ACM SIGCOMM Conference. pp. 73–86 (2016)
[24] Laor, M., Gendel, L.: The effect of packet reordering in a backbone link on application throughput. IEEE network **16**(5), 28–36 (2002)
[25] Ludwig, R., Katz, R.H.: The Eifel algorithm: making TCP robust against spurious retransmissions. ACM SIGCOMM Computer Communication Review **30**(1), 30–36 (2000)
[26] Madhyastha, H.V., Isdal, T., Piatek, M., Dixon, C., Anderson, T., Krishnamurthy, A., Venkataramani, A.: iPlane: An Information Plane for Distributed Services. In: Proceedings of the 7th symposium on Operating systems design and implementation. pp. 367–380. USENIX Association (2006)
[27] Meyer, D.: AWS Remains Dominant Player in Growing Cloud Market, SRG Reports, https://www.sdxcentral.com/articles/news/aws-remains-dominant-player-in-growing-cloud-market-srg-reports/
[28] Moy, J.: OSPF Version 2. Internet Request for Comments **RFC 2328 (INTERNET STANDARD)** (Apr 1998), http://www.rfc-editor.org/rfc/rfc2328.txt, updated by RFCs 5709, 6549, 6845, 6860, 7474
[29] Pathak, A., Zhang, M., Hu, Y.C., Mahajan, R., Maltz, D.A.: Latency inflation with MPLS-based traffic engineering. In: IMC (2011)
[30] Pelsser, C., Cittadini, L., Vissicchio, S., Bush, R.: From Paris to Tokyo: on the suitability of ping to measure latency. In: IMC (2013)
[31] Pucha, H., Zhang, Y., Mao, Z.M., Hu, Y.C.: Understanding Network Delay Changes Caused by Routing Events. ACM SIGMETRICS Performance Evaluation Review - SIGMETRICS '07 Conference Proceedings **35**(1), 73–84 (2007)
[32] Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. Internet Request for Comments **RFC 3031 (Standards Track)** (Jan 2001), http://www.rfc-editor.org/rfc/rfc3031.txt
[33] Roughgarden, T., Tardos, É.: How bad is selfish routing? Journal of the ACM (JACM) **49**(2), 236–259 (2002)
[34] Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C.: Inside the social network's (datacenter) network. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. pp. 123–137 (2015)
[35] Takács, A., Császár, A., Bíró, J., Szabó, R., Henk, T.: Path integrity aware traffic engineering [telecom traffic]. In: Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE. vol. 2, pp. 692–696. IEEE (2004)
[36] Vanini, E., Pan, R., Alizadeh, M., Taheri, P., Edsall, T.: Let it flow: Resilient asymmetric load balancing with flowlet switching. In: 14th USENIX Symposium on Networked Systems Design and Implementation NSDI'17). pp. 407–420 (2017)
[37] Venkataramani, V., Amsden, Z., Bronson, N., Cabrera III, G., Chakka, P., Dimov, P., Ding, H., Ferris, J., Giardullo, A., Hoon, J., et al.: TAO: How Facebook Serves the Social Graph. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 791–792. ACM (2012)
[38] Zhang, M., Karp, B., Floyd, S., Peterson, L.: RR-TCP: a reordering-robust TCP with DSACK. In: 11th IEEE International Conference on Network Protocols, 2003. Proceedings. pp. 95–106. IEEE (2003)
[39] Zhiruo Cao, Zheng Wang, Zegura, E.: Performance of hashing-based schemes for internet load balancing. In: Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. vol. 1, pp. 332–341 (March 2000). https://doi.org/10.1109/INFCOM.2000.832203