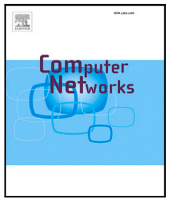
Computer Networks 240 (2024) 110189

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Measuring congestion-induced performance imbalance in Internet load balancing at scale

Yibo Pi [a,*], Sugih Jamin [b], Yichen Wei [a]

[a] University of Michigan - Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China
[b] Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, USA

ARTICLE INFO

Keywords:
Network measurement
Internet load balancing
Congestion imbalance

ABSTRACT

Internet load balancers typically use congestion-oblivious hashing algorithms to distribute traffic across load-balanced (LB) paths, resulting in imbalanced load distribution. When congestion occurs, LB paths could experience different levels of congestion, called *congestion imbalance*. Congestion imbalance has been extensively studied in datacenters (DCs). It is, however, still under-explored in the Internet. In this paper, we take a first step towards measuring congestion imbalance among Internet LB paths at scale. We present *Congi*, our lightweight prober, that leverages support vector machine (SVM) classifiers to efficiently detect congestion imbalance using latency samples. Our experiments show that Congi is capable of detecting congestion imbalance between uncongested and congested LB paths: on average, the uncongested path has 3x greater throughput and 0.4% lower packet loss rate than the congested one. To measure congestion imbalance at scale, we then use Congi to conduct measurement campaigns from worldwide DCs to millions of/24s. We find that most DCs experience significant congestion imbalance to 36%–43% of end hosts. Lastly, we use Congi to direct web page downloads under congestion imbalance from a campus network, and show that the download time can be reduced by 44% on average for Alexa top sites for clients experiencing network congestion under congestion imbalance.

1. Introduction

When a load balancer learns of multiple paths to a destination, it may update its routing table and distribute traffic to the destination across these *load-balanced (LB) paths*. Load balancers are prevalent in the Internet: nearly 74% of IPv4 paths traverse at least one load balancer [1]. Load balancers typically select paths of equal cost for load balancing and distribute traffic among these paths with simple hashing algorithms. However, since these hashing algorithms are congestion-oblivious [2,3], traffic could be directed towards a congested path, even when there are less- or uncongested alternatives. We refer to this problem as *congestion imbalance*, where LB paths experience different levels of congestion and differ significantly in path performance.

Congestion imbalance has been mostly studied in datacenters (DCs) [3,4], where paths between servers can be carefully planned to share minimal common bottleneck links such that not all of the paths would be congested at the same time. Congestion-aware load balancing thus can better utilize bandwidth in DC networks. In contrast, despite the prevalence of load balancers in the Internet, Internet LB paths are likely to overlap [5,6]. We are not aware of any study reporting on the prevalence of congestion imbalance in the Internet. In this work, we take the first step to measure congestion imbalance in the Internet at scale. We want to understand the prevalence of congestion imbalance and the extent to which it causes LB paths to differ in performance. Considering that congestion affects all performance metrics (throughput, latency, and packet loss) of a path, we expect that harnessing Internet congestion imbalance would benefit a wide range of Internet applications.

Broadly speaking, congestion imbalance exists as long as LB paths experience different levels of congestion. As a first step, we limit our focus on recognizing a congested path from an uncongested one, without differentiating the congestion levels. To measure congestion imbalance at scale, we need to address two major challenges: (1) how to detect congestion, and (2) once a congested path is detected, how to quickly search among a pool of LB paths for an uncongested path while the congestion persists? To detect congestion, we probe paths from a single vantage point (VP) to the public Internet. Network probing infers path performance towards a destination based on its responses to the probes. Due to rate limiting, probes sent to the same destination must be controlled at a very low rate, making us unable to detect transient congestion. We focus on congestion lasting from seconds to minutes,

* Corresponding author.
E-mail addresses: yibo.pi@sjtu.edu.cn (Y. Pi), sugih@umich.edu (S. Jamin).

https://doi.org/10.1016/j.comnet.2024.110189
Received 21 May 2023; Received in revised form 21 October 2023; Accepted 8 January 2024
Available online 9 January 2024

which affects many applications, e.g., web page load and VoIP. Our approach can also be easily extended to detect the diurnal congestion in [7].

Our approach uses latency inflation to infer congestion similar to the Time Series Latency Probing (TSLP) technique [8]. However, to detect congestion at scale (the *first* challenge), we cannot use latency time series collected in the long term as in TSLP. Instead, we develop *Congi*, our network prober that leverages a support vector machine (SVM) classifier to detect congestion with a small number of latency samples. As Congi is designed to detect congestion imbalance, not just congestion, it also includes two SVM classifiers, focusing on the *speed* and *accuracy* of detection respectively, to detect uncongested paths. The *speed-focused* classifier aims to fast search among LB paths for potential uncongested paths, which are further verified by the *accuracy-focused* classifier. We train these SVM classifiers on a ground-truth dataset that we build from path performance data between a large number of source–destination pairs.

We verify Congi's performance with both trace-driven simulation and real-world experiments, and find that Congi excels at detecting significant congestion imbalance: the uncongested paths are on average 3x greater in throughput and 0.4% lower in packet loss rate than the congested ones. In our large-scale experiments, to avoid affecting end hosts, we ran Congi from 12 DCs around the globe to the last-hop routers of end hosts. We found that most DCs experienced long imbalance lasting 120 s to 9%–13% of the last-hop routers and short imbalance lasting 24 s to 36%–43% of the routers (see Section 6 for limitations). We further evaluated the impact of congestion imbalance on web page load by downloading web resources using the congested and uncongested paths respectively. We found that half of the download times could be reduced by at least 53% using the uncongested paths. In summary, this paper makes the following contributions:

- We show that it is feasible to detect the performance imbalance between congested and uncongested paths with latency measurements and conduct a systematic comparative study to choose the best latency-based metric for congestion imbalance detection.
- We present Congi, a network prober leveraging SVM classifiers to efficiently detect congestion imbalance at scale. We verify that Congi is capable of detecting significant congestion imbalance between LB paths.
- We run Congi to conduct large-scale measurement campaigns and find that short-term congestion imbalance is prevalent in the Internet. We also demonstrate that congestion imbalance greatly affects web page load time.

2. Measurement methodology

2.1. Our goal

Congestion imbalance occurs between two addresses when the connecting LB paths experience different levels of congestion. In this work, we simply categorize paths as either congested or uncongested, with no further distinction of congestion levels. Congestion imbalance occurs when an uncongested path exists concurrently with a congested one. Our goal is to measure congestion imbalance at scale from a single VP with network probing, assuming no direct access to a large number of instrumented machines. We focus on measuring short-term congestion imbalance lasting from seconds to minutes, which affects many applications like web page load and video streaming. Nonetheless, our approach can also be used to measure transient congestion imbalance within sub-seconds, if probe rate limiting is not a concern, and to measure long-term congestion imbalance.

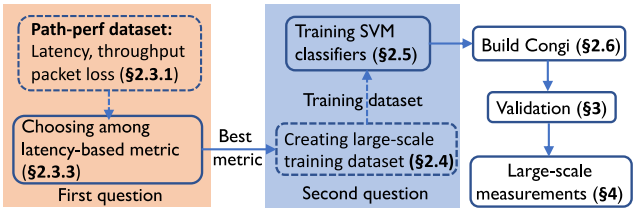


Fig. 1. Overview of measurement methodology.

2.2. Overview of our methodology

We want to design *Congi* (short for *Congestion Imbalance*), a network prober that collects latency samples to detect congestion imbalance at scale. Fig. 1 shows the high-level overview of our measurement methodology. We must address two key questions: (1) *Can we differentiate between congested and uncongested paths from their latency samples?*; (2) *Can we detect congestion imbalance with a small number of latency samples?* To answer the first question, we conduct a systematic study of latency-based metrics to select the best metric, such that we can detect congestion-induced performance imbalance between LB paths with latency time series. Using the best latency-based metric, we answer the second question by collecting large-scale datasets with well-designed network probing techniques and validate that it is possible to leverage lightweight machine learning techniques to efficiently detect congestion imbalance detection even with a small number of latency samples.

A positive answer to the *first question* requires the existence of a *latency-based metric* capable of such differentiation. Specifically, we want to build a dataset of path performance and use it to check if the latency-based metric can classify paths such that paths identified as "congested" and "uncongested" differ significantly in performance. This dataset can be collected by measuring performance of all LB paths between source–destination pairs, which is a very resource intensive process. Instead, we collect the performance of one path continuously over a long period of time and check if a metric can differentiate between congested and uncongested periods of the path. This not only simplifies data collection, but also ensures that congestion would be the reason for performance degradation with all other factors being equal. An intuitive way to build such a dataset is to use publicly available datasets with validated congested and uncongested periods. However, to the best of our knowledge, the most related public datasets only provide databases of time series together with analysis scripts to infer congestion and no associated throughput and packet loss measurements are made available [7]. We thus collect a dataset, referred to as the *path-perf dataset*, including the real performance data (latency, throughput, and packet loss) of paths (Section 2.3.1), and experiment on several latency-based metrics (*latency elevation*, *latency deviation*, and *latency inflation*). We find that *latency elevation* (increase in mean latency) is the best metric, capable of differentiating congested and uncongested periods of a path (Section 2.3.3), and also excels at detecting congestion imbalance across multiple paths (Section 3.2).

To answer the *second* question, we want to find classifiers capable of accomplishing the same task above but with a very small number of latency samples. Although the latency-based metric enables us to detect congestion imbalance without actually measuring throughput and packet loss rates of LB paths, it relies on long running latency time series to be effective. To find proper classifiers, we create a large dataset consisting of latency time series, each labeled by the latency-based metric as being of a congested or uncongested path (Section 2.5), and use this dataset to train classifiers. These lightweight classifiers enable us to scale the detection of congestion imbalance. We combine them to build Congi, verify its ability to detect congestion imbalance (Section 3.2), and conduct measurement campaigns from VPs around the globe to Internet-wide addresses (Section 4).

2.3. Choosing the latency-based metric

The intuition behind a latency-based metric is that when a path is congested, its capacity is over-utilized, causing packets to be buffered and resulting in inflated measured latency. Dhamdhere et al. [7] have shown that inflation in latency time series can be used to detect congested periods. However, we are not aware of any study to compare different latency-based metrics and to systematically explore their use in detecting congestion periods. In this section, we will compare alternative metrics for describing latency inflation and determine thresholds that properly separate congested periods from uncongested ones. We start by the collection and processing of the *path-perf dataset* that includes the performance data of a diverse range of paths.

2.3.1. Data collection: The path-perf dataset

We use this dataset to evaluate several latency-based metrics, which involves correlating path latency with throughput and packet loss. In this dataset, the path metrics (latency, throughput, and packet loss) are collected for each path included. As public clouds enable us to easily access computing resources in DCs around the globe, we create virtual machines (VMs) in DCs as our VPs. For throughput measurement, we run the Network Diagnostic Tool (NDT) test from our VPs to M-Lab NDT servers [9]. Since we use latency to infer congestion, our latency and throughput measurements do not overlap in time, to avoid self-induced latency inflation by throughput tests. The TCP trace made public by M-Lab only provides latency during TCP sessions and thus does not fit our needs [9].

Measurement Methodology We want to measure latency and throughput for the same path between a source–destination pair, including both the forward and return directions. As 98% of load-balanced paths include only per-flow and per-destination load balancers, traffic with the same flow identifier (source IP, destination IP, source port, destination port, protocol ID) will follow the same path when there is no path change [5]. Given a source–destination pair, we enforce all measurement traffic to use the same source and destination ports. Since NDT tests use TCP, we measure latency to the NDT hosts by sending TCP ACK to the port numbers used in NDT tests. The measured latency of the path is the elapsed time between sending the TCP ACK and receiving the corresponding TCP RST. Measured packet loss rate is the percentage of TCP ACK probes without responses.

To understand how latency is correlated with throughput, we want latency and throughput to be measured under various network conditions, including both congested and uncongested periods. Since collecting one latency sample only takes one probe, we periodically measure path latency to all NDT servers in parallel. In contrast, each NDT test measures upload and download TCP throughput separately, each taking about 10 s. We only run one NDT test from each VP to a target server at the same time to avoid multiple NDT tests competing for bandwidth. Considering the overheads of throughput measurements and that congested periods are typically less common than the uncongested ones, it is impractical and unnecessary to periodically measure throughput to all NDT servers. We instead drive NDT tests by latency variation: we trigger a NDT test to measure a path when the path latency is seen varying, and follow up with another NDT test when the path latency returns back to stable. In this way we distribute throughput measurements more evenly between paths of varying and stable latencies, and still covers paths within the full range of latency variations. Moreover, we want to focus on latency variation due to congestion rather than path changes, where path latency during congestion is constantly varying while a path change causes path latency to suddenly increase or decrease and remain stable afterwards. We thus want *latency variation* to be robust to sudden latency changes and define it as the average absolute difference between neighboring latency samples, i.e., $\sum_{i=1}^{n} |r_i - r_{i-1}| / (n-1)$, where n is the number of samples and r_i is the ith latency sample. We further mitigate the impact of path changes with data processing (Section 2.3.2) and analysis (Section 2.3.3).

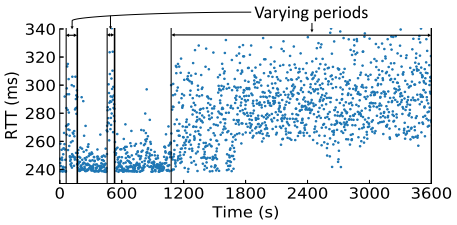


Fig. 2. An example of time series segmentation.

Experiment Setup We used 12 geographically-distributed VPs (2 in Europe, 5 in Asia, 2 in North America, 1 in South America, 1 in Africa, and 1 in Oceania) and measured path performance from these VPs to 471 NDT-7 (NDT version 7) servers [10] to create this dataset. As cloud providers may differ in how they route traffic from their DCs to the public Internet [11], these 12 VPs are further distributed evenly among 3 cloud providers (Google Cloud, Microsoft Azure, and Alibaba Cloud).[1] Each VP sent TCP ACK probes to all NDT-7 servers every 2 s to measure path latency and packet loss. Latency variation was calculated every 60 s with the most recent 90 latency samples, where samples below the 10th percentile and above the 90th percentile were not used, to mitigate the impact of outliers. For each source–destination pair, we triggered a NDT test when latency variation was greater than 2 ms, a relatively small threshold to capture small latency changes. When a NDT test was done, we waited at least 15 min before checking if the latency variation returned back to stable (below the threshold) and we could conduct a follow-up NDT test.

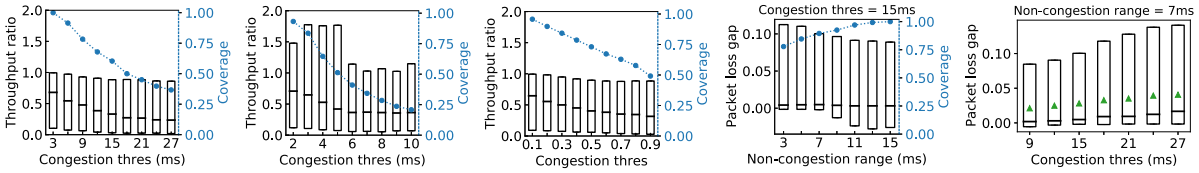
2.3.2. Data processing: Time series segmentation

We want to prepare the dataset for correlating latency with throughput and packet loss. Specifically, given a latency time series between a pair of VP and NDT server, we want to (1) segment it into periods of varying latency (or *varying periods*) and periods of stable latency (or *stable periods*) such that latencies in the same period reflect the same state of the path, and (2) match each period with the throughput samples measured in it. Note that a varying period is not necessarily a congested one because apart from congestion, other factors, e.g., delayed responses, could also cause varying latency [12]. A congested period is a varying period that actually causes performance degradation. We will find proper thresholds for concluding if a path is congested or uncongested in Section 2.3.3. In the following, we first discuss the current method of detecting varying periods and its drawbacks, and then introduce a more accurate method for time series segmentation.

Drawbacks of the Current Detection Method Observing that latency samples in varying periods are typically inflated and have higher means than those in stable periods, Dhamdhere et al. [7] proposed to use latency mean shifts in time series to identify varying periods. Their method uses a moving window to sequentially scan the latency time series and, with a statistical test, checks if there is a mean shift in the moving window. This method relies on a moving window, including only a partial view of the entire time series, to make decisions and is thus prone to false positives when the entire moving window is in a varying period. Moreover, the statistical test can detect level shifts but is not capable of telling the exact boundary between varying and stable periods, which is critical for segmenting time series.

An Improved Method: Changepoint Detection To address the above issues, we introduce a new method called *changepoint detection*, which detects mean shifts by observing the entire time series [13]. Specifically, the changepoint detection method segments the time series

[1] We did not have VPs in Amazon AWS for this dataset because the download TCP throughput from the NDT-7 servers to our VPs in Amazon DCs was throttled to a low level, which decorrelated latency and throughput.



(a) Latency elevation (b) Standard deviation (c) Inflation level (d) Threshold tuning (e) Packet loss gap

Fig. 3. Comparing latency-based metrics and tuning thresholds.

Table 1
The path-perf dataset summary.

Cloud providers	#VP-Server pairs	#Throughput samples	#Latency samples
Google Cloud	342	1,903	22,239,395
Alibaba Cloud	629	2,683	30,464,281
Microsoft Azure	627	2,789	40,717,975

into time periods with different means. Let P_i be the ith period, y_{ij} be the jth sample in P_i, and m be the number of periods. The optimal segmentation is obtained by minimizing the loss function

$$\min_{\{\bar{y}_1,\ldots,\bar{y}_m\}} \sum_{i=1}^{m} \sum_{y_{ij}\in P_i} |y_{ij} - \bar{y}_i|^2 + \beta m,$$

where $\bar{y}_i$ is the latency mean of the ith period and βm is a penalty term to avoid overfitting. We use one of the common choices for the penalty term, i.e., the Bayesian Information Criterion penalty by setting β to $\sigma^2 \log(n)$, where n is the total number of samples and σ^2 is the variance of samples [13]. The `ruptures` package [14] is used to segment time series in our dataset. The minimum period length is set to 15 samples to obtain periods lasting for at least 30 s (inter-sample time is 2 s). The minimum period length is chosen to be relatively small such that short varying periods can still be detected.[2] Fig. 2 shows an example of time series segmentation with the changepoint detection method, where we can clearly see the boundaries between neighboring periods. As a result of path changes, neighboring periods could be both stable. We treat individual periods separately as each period presents a different state of the path.

Matching Segmented Periods with Throughput Samples After time series segmentation, it is straightforward to associate each segmented period with the throughput samples measured in it by time. We need to handle some edge cases where throughput samples are measured close to the boundaries. To avoid associating throughput samples with the wrong periods, we only use throughput samples measured at least 10 s away from both the start and end times of their associated periods.

2.3.3. Comparative study of latency-based metrics

Table 1 summarizes the *path-perf* dataset, which only counts the VP-server pairs (pairs of VP and NDT server) that have at least one varying and one stable periods, both with throughput samples. Google Cloud has the least number of VP-server pairs, likely because probes from VPs in Google Cloud to NDT servers spend most of their times traversing Google's well-provisioned private WANs [11,15]. Since we run only one NDT test at a time from each VP, when the paths to multiple NDT hosts have varying latencies, only one will be measured—the others would have to wait for future rounds of measurement. This case happens more often for the VPs in Alibaba Cloud and Microsoft Azure, leaving them with fewer throughput samples for each pair, on average. To compensate for this difference, we run experiments longer for VPs in Alibaba Cloud and Microsoft Azure. We next use this dataset to compare different latency-based metrics.

We want to find a latency-based metric that can differentiate between congested and uncongested periods of a path by the gap in path performance during these periods. We will compare three metrics describing latency inflation from different aspects: (1) *latency elevation*, the difference between latency mean and the minimum path latency, (2) *latency deviation*, the standard deviation of latency samples, and (3) *inflation level*, the percentage of inflated latency samples.

Defining the Performance Gap We define the performance gap between the congested and uncongested periods for two path metrics, throughput and packet loss. For each source–destination pair, the throughput gap is defined as the ratio of the average throughput during congested periods to that during uncongested ones, also referred to as the *throughput ratio*. Similarly, the *packet loss gap* is defined as the difference between the average packet loss rate during congested periods and that during uncongested ones. We use the average path performance to alleviate the impact of measurement errors that are infrequent but difficult to exclude. For example, we could not tell if low throughput during a stable period is because the bottleneck link, though uncongested, has high utilization [16] or because the NDT server is under heavy load, which we want to exclude as it is not network related. Path change is also one source of such measurement errors, where changes, if not occurring on the bottleneck link or not incurring a new bottleneck link, do not affect throughput and packet loss.

Ability to Maximize Throughput Gap To compare the alternative metrics, we use each metric to classify periods in our dataset as congested or uncongested, and calculate the throughput gap between the congested and uncongested periods. The best metric is the one that achieves the maximum throughput gap. Given a metric, we consider a period *congested* if the metric calculated from the latency samples during the period is above a pre-defined *congestion threshold*; otherwise, we consider the period *uncongested*. Per the definition of throughput gap, it can only be calculated for pairs having at least a congested period and an uncongested one. This means that the number of pairs with a throughput gap is determined by the choice of the metric and the congestion threshold. In our results, when we use the *latency elevation* metric with a congestion threshold of 3 ms, the number of pairs with a throughput gap reaches the maximum, and is about 47% of the total VP-server pairs in our dataset. We consider this the total number of available pairs and calculate the *coverage* (i.e., the percentage of pairs with a throughput gap) at other congestion thresholds with respect to this number.

To mitigate the impact of outliers and path changes, we estimate the minimum path latency at time t as the 10th percentile of samples covered in a 30-minute interval centered at t. Fig. 3(a) shows the throughput gap/ratio and coverage under different congestion thresholds when latency elevation is used as the metric, where each box represents the distribution (10th, 50th and 90th percentiles) of throughput ratios for the given threshold. We can see that as the congestion threshold increases, the throughput ratio decreases, indicating a larger throughput gap. Meanwhile, the coverage also decreases, which implies that a higher throughput gap is achieved at the cost of coverage, a tradeoff between throughput gap and coverage. When the congestion threshold is 15 ms, we can achieve a median throughput ratio of

[2] A long minimum period forces the method to consider short varying periods as noise and may result in long stable periods with short varying periods inside them. A very short minimum period makes the method over-reactive to outliers.

0.33 and cover 60% of available pairs. The average throughput ratio between the 10th and 90th percentile is only 0.37.

Figs. 3(b) and 3(c) show the throughput ratios when latency deviation and inflation level are used as the metrics. We calculate standard deviation with the samples between the 10th and 90th percentiles and consider a sample inflated if it is larger than the minimum latency for 10 ms. Using standard deviation with a threshold of 6 ms can achieve a median throughput ratio of 0.37 but covers only 45% of pairs, which is much less coverage than using latency elevation. Using an inflation level of 0.7 achieves both similar coverage and throughput gap as using latency elevation with a threshold of 15 ms, but it underperforms latency elevation in terms of the smallest throughout ratio that can be achieved. We use latency elevation as the best latency-based metric and 15 ms as the congestion threshold to obtain a good tradeoff between throughput gap and coverage. We next discuss the ability of latency elevation to describe the packet loss gap.

One Threshold or Two? Given that the congestion threshold is 15 ms, we calculate the packet loss gaps for available pairs and find that 11% of pairs have a negative packet loss gap. This implies that some pairs have their congested periods misidentified as uncongested, resulting in the packet loss rates in uncongested periods higher than those in congested ones. We revisit our way to differentiate between congested and uncongested periods, and find that the culprit is that since only one threshold is used to separate the congested periods from the uncongested ones, there is a sudden switch in categorization. We thus add a second threshold to identify uncongested periods and enforce a buffer region between the two thresholds. A period is *uncongested* if it has latency elevation below the second threshold. We refer to the range from zero to the second threshold as the *non-congestion range*. Fig. 3(d) shows the packet loss gaps under different non-congestion ranges, where the 10th percentile packet loss gap approaches zero when the non-congestion range is 7 ms or less. The use of a buffer region may reduce the coverage. We calculate the coverage at different non-congestion ranges with respect to that when the buffer region is zero, i.e., both the non-congestion range and the congestion threshold are 15 ms. We use 7 m as the non-congestion range, which reduces coverage by 15%. Note that the two thresholds are chosen to maximize the performance difference between LB paths in a collective way such that we can characterize congestion imbalance at scale. No definitive conclusion can be made on the actual cause of degraded performance at the path level.

Relating Latency Elevation with the Packet Loss Gap After the non-congestion range is determined, we can calculate the packet loss gaps under different congestion thresholds. As shown in Fig. 3(e), both the median and mean packet loss gaps increase with the congestion threshold. This provides additional validation that latency elevation is an effective latency-based metric in separating congested periods from uncongested ones. Nonetheless, it is noticeable that compared with the throughput gap, the packet loss gap is in general very small, with the median being as low as 0.6% when the congestion threshold is 15 ms. It is more interesting to look at the packet loss gaps in the high percentiles, where the 75th and 90th percentiles are about 3% and 10% respectively. We refer to this method of using latency elevation and thresholds to identify congested and uncongested periods as the *metric-based method*.

2.4. Building a ground-truth dataset

The main drawback of the metric-based method is that it relies on long latency time series. We want lightweight classifiers that can achieve a similar accuracy in detecting congested and uncongested paths as the metric-based method but does so with much fewer samples. Specifically, we (1) want a dataset consisting of the latency time series of paths, where each time series is labeled either belonging to a congested or uncongested path, and (2) train the classifiers to determine if a path is congested or uncongested with a small portion of samples from its latency time series.

Large-Scale Ground-Truth Dataset Instead of using the path-perf dataset, we want the ground-truth dataset to be of a larger scale and include a more diverse range of paths such that the classifiers can be accurately trained and perform well when used in real-world scenarios. To create such a large-scale dataset, we collect latency time series from VPs in DCs around the globe to a large amount of random IPv4 Internet addresses. Since we want the classifiers to achieve similar performance as the metric-based method, we apply the metric-based method on the collected data to construct the ground truth.

Data Collection We randomly select addresses from the IPv4 Internet address space and probe from 12 geographically-diverse VMs to the last-hop routers of these addresses to avoid affecting the end hosts. Latency is measured as the elapsed time between sending a UDP probe with the TTL expiring at the last-hop router and receiving the ICMP response from it. For each target, we force all UDP probes and ICMP responses to take the same path by sending UDP probes with the same source and destination ports [11]. The rate of UDP probes sent to each router is controlled at 0.5 packet/s, unlikely to cause packet drop due to ICMP rate limiting on routers [17]. To capture short-term congestion of various durations, we periodically measure each last-hop router for 1 h and stop immediately once there is a path change, which is detected by path discovery to the last-hop router every 5 min with TTL-limited probes. After 7 days of measurements in October 2020, we collected latency and path information from our VPs to 342,139 Internet addresses, where each target has 1,800 latency samples.

Data Labeling We first segment latency time series into periods of different latency means with the changepoint detection method. For each period, we calculate the latency elevation and label those with latency elevation greater than 15 ms as "congested", less than 7 ms as "uncongested", and the rest as "undetermined".

2.5. Training Congi's SVM classifiers

Given a target, Congi first collects data with its probing module and then uses its SVM classifiers to detect congestion imbalance with the collected data.

Accuracy- and Speed-Focused SVM Classifiers Congi includes two types of SVM classifiers: the *accuracy-focused* and *speed-focused* classifiers. The *accuracy-focused classifiers* aim to detect both congested and uncongested paths with high accuracy, while the *speed-focused classifiers* aim to detect uncongested paths with a reasonable accuracy but with much fewer samples. For each target host, Congi starts the probing by detecting congestion with an accuracy-focused classifier, because if an uncongested path is misclassified as congested, all subsequent measurements, e.g., the search of an uncongested path, will be wasted. Congi also ends the probing by reconfirming congestion imbalance with the accuracy-focused classifiers. The speed-focused classifiers are used to search for an uncongested path once a congested path is detected. Between a source–destination pair, there could be tens of LB paths [18]. It is crucial to check each path fast such that an uncongested path can be found while congestion persists. We choose SVM as our classifiers because the separating hyperplane between classes is determined only by the data points near the hyperplane (a.k.a. support vectors). This makes it more robust to outliers compared to logistic regression, which is crucial to mitigate the impact of occasional spikes in latency time series. Moreover, if needed, we can separate non-linearly separable classes by mapping them to higher dimensional spaces with non-linear kernels [19]. Lastly, it is simple to integrate SVM with our network prober written in C++. More efficient machine learning classification algorithms are subject to future research.[3]

[3] We have compared SVM with several other common classification algorithms. SVM outperforms logistic regression, random forest, and decision tree by 2 to 5% in F1 score when the sample size is small, but the performance gap diminishes as the sample size increases.

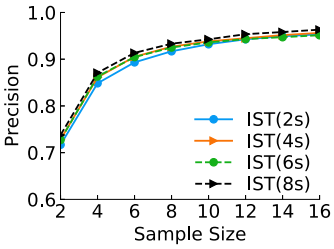
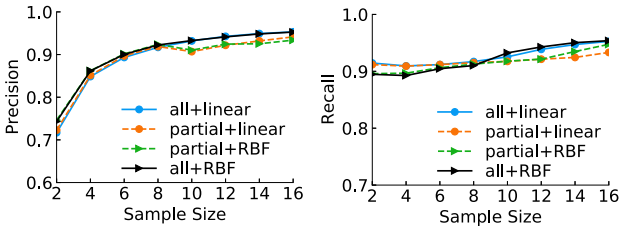


Fig. 4. Tuning inter-sample time (IST).



(a) Precision (IST = 2s) (b) Recall (IST = 2s)

Fig. 5. Performance in detecting congested paths.

Creating Training and Testing Datasets from the Ground Truth The ground-truth dataset above includes a collection of labeled latency time series. For each time series, Congi's classifiers only take a small portion of samples as input. The process of picking these samples from the time series simulates how probing works. There are two design parameters for probing, the number of samples to collect (denoted as m) and the inter-sample time (denoted as d). As different probing schemes lead to different data being collected, we will train the SVM classifiers on datasets collected under different d's and m's to choose the best scheme.

Consider a probing process with $m = 3$ and $d = 4$s. To simulate this process on a ground-truth time series (inter-sample time is 2 s), we first randomly select a sample in the time series as the start point (the 1st sample) and then select the 3rd and 5th samples as input to the classifiers. As probing could begin from any sample within the time series, we repeat the probing process at different start points to average out randomness. Let d^* and m^* denote the upper bounds of m's and d's in the design space. To simulate a probing scheme with $m = m^*$ and $d = d^*$, we need latency time series of length at least equal to $l^* = d^* \times (m^* - 1) + 1$. We require each time series in the dataset to support simulating the probing scheme with $m = m^*$ and $d = d^*$ such that all probing schemes with d's and m's within the upper bounds can be simulated on the same dataset for fair comparison. Another issue of this dataset is that we have much more uncongested paths than congested ones. We handle this imbalanced class problem by randomly selecting the same amount of uncongested and congested paths in the dataset before simulating the probing process. For each pair of d and m, this random selection is repeated to average out randomness. We use two thirds of the data for training and the rest for testing.

Classifier Training for Detecting Congested Paths As mentioned in Section 2.4, our dataset includes three classes of paths (congested, uncongested, and undetermined), which implies that this is a multi-class classification problem. As we are not interested in detecting the undetermined paths, we take a one-versus-rest approach to detect the congested and uncongested paths respectively [20]. That is, when our goal is to detect one class, we consider it as the positive class and the rest two together as the negative class. This requires us to have separate classifiers for detecting the congested and uncongested paths. Considering the number of samples as input to the classifiers is limited, we generate a feature from each sample and form a feature vector of dimension equal to the number of samples. Specifically, given m samples, we form a m-dimensional feature vector by (1) considering the minimum sample as the minimum latency, (2) subtracting it from each sample to obtain the inflated part, and (3) sorting the inflated parts of all samples in an increasing order.

Fig. 4 shows the precision in detecting congested paths under different m's and d's, where a linear kernel is used and the precision is the fraction of true positives among all claimed positives. We can see that the precision increases gradually with the sample size, but the improvement becomes very marginal when the sample size is greater than 12. As the sample size becomes larger, samples are more likely to include outliers far away from other samples, which causes more uncongested paths to be misclassified as congested and counteracts the improvement by using a higher dimension. We can also see a slight improvement in precision by using a larger inter-sample time (IST), because a larger IST makes samples more independent from each other and is thus more likely to result in larger difference between latency samples for congested paths. Considering that the improvement for using a large inter-sample time is limited and that congestion is typically short-lived, we want probing to be done fast and thus choose the inter-sample time to be 2 s.

In Fig. 4, all m samples are used to construct the feature vector. We wonder if excluding the extreme samples that might be outliers could improve precision and that how kernel selection affects precision. With the inter-sample time being 2 s, Figs. 5(a) and 5(b) show how feature and kernel selection affects the precision and recall in detecting congested paths, where "all" means that all samples are used for prediction and "partial" means that only samples between the 10th and 90th percentiles are used, to exclude outliers. When a percentile lies between two samples, the smaller one is used for the 10th percentile and the larger one is used for the 90th percentile. In Fig. 5(a), the scheme of using partial samples begins to drop samples when the sample size reaches 10, which results in a lower-dimensional feature vector and causes a sharp decrease in precision. This indicates that the precision gain by excluding outliers cannot cover the precision loss due to using a lower-dimensional feature vector. We therefore use all samples as input for prediction. Comparing the two schemes of using all samples in Figs. 5(a) and 5(b), we find almost no difference in precision between using a linear and a radial basic function (RBF) kernels and a slight 1% difference in recall when the sample size is 10. When the sample size is 12, using both kernels achieves a high precision of 95% and a high recall of 94%.

Classifier Training for Detecting Uncongested Paths We want to train two types of classifiers for detecting uncongested paths: an accuracy-focused classifier and a speed-focused classifier. The uncongested paths constitute the positive class and the other two classes together constitute the negative class. We train these classifiers for detecting uncongested paths similarly to the way we train the classifiers for detecting congested paths described in the previous subsection. We find that for both classifiers with a linear and a RBF kernels, the precision and recall increase with the sample size and the increase rate begins to slow when the sample size reaches 8 (not shown). When the sample size is 8, using a RBF kernel results in a precision of 94%, almost the same as using a linear kernel, and a recall of 87%, about 2% higher than using a linear kernel. For the speed-focused classifier, we want a small sample size that can provide a reasonable precision and recall such that each LB path can be checked quickly. When the sample size is 4, using a RBF kernel achieves a precision of 88% and a recall of 81%, while using a linear kernel achieves a precision of 91% and a recall of 76%. Since a high recall is desired while searching for uncongested paths, we choose the RBF kernel for the speed-focused classifier.

2.6. Putting it all together to build Congi

Congi probes targets in parallel and the total sending rate depends on the number of targets being probed simultaneously. The probing to each target is a three-stage process.

Stage I: Detecting Congestion Given a target, Congi first detects if congestion exists in a path to the target. To avoid affecting the end hosts, we can use the last responding router along the path to the target as a proxy and measure congestion imbalance from the source to the router. This means that all probes are still sent to the target with TTLs expiring at its last responding router such that congestion imbalance seen by the router will also be seen by the target. To force all probes to take the same path, Congi sends UDP probes with the same source and destination ports. To avoid triggering ICMP rate limiting, Congi sends one UDP probe to the router every 2 s until the 12 samples are collected. Congi then feeds the 12 samples into its accuracy-focused classifier to detect congestion. If no congestion is detected, Congi moves on to the next target; otherwise, Congi enters into stage II. It is possible that the target router has a very low budget of ICMP responses and drops most of our probes. Congi allows a packet loss rate less than 30% for each target and stops probing a target if the packet loss rate is above the threshold. Note that although UDP probes are used above, Congi can also use other types of probes.

Stage II: Searching for an Uncongested Path Once a congested path is detected, Congi begins to search for a new path to the router and check if it is uncongested. This process is repeated until an uncongested path is found or failed to be found after 15 trials. The discovery of new paths is done by sending UDP probes to random destination port numbers. Although the paths of UDP probes are passively selected by load balancers based on their port numbers, with a sufficiently large number of trials, we can discover an uncongested path, if one exists, with a high probability. For instance, assuming that LB paths have equal probability of being taken by UDP probes with random port numbers, even when the portion of uncongested paths is less than 10% among all LB paths, Congi can still find one with a probability of $1 - 0.9^{15} = 79.4\%$ with 15 trials. This is more efficient than finding new paths by hop-by-hop comparison with the previously probed ones. Note that although UDP probes may take different forward paths due to having different port numbers, all ICMP responses take the same return path because Congi manipulates UDP probes such that ICMP responses can have the same checksum, the factor that load balancers use to determine the paths ICMP packets take [11]. For each new path, Congi has to estimate the TTL from the source to the target router. Congi uses the TTL of probes in the congested path as reference and search neighboring TTLs (± 2) for the new path. Once a new path is found, Congi collects 4 samples and use the speed-focused classifier to check if this path is uncongested. If so, Congi continues to collect 4 more samples and verify this with the accuracy-focused classifier. Congi either enters into Stage III after a successful verification or returns back to finding a new path if the verification fails.

Stage III: Collecting More Samples Congi collects more samples for both the congested and uncongested paths for two reasons. First, as it takes time to find the uncongested path, during which the congestion may have ended, Congi confirms congestion imbalance with the most recent 12 samples of each path at the end of the probing. Second, we want to collect 30 samples for each path to estimate their mean path latencies for later study of the latency imbalance between LB paths (Section 4.3). Based on the central limit theorem, when the sample size is 30 or more, we can use the sample mean to estimate the population mean regardless of the population distribution [21].

3. How Congi actually performs?

We verify Congi's ability to detect congestion imbalance with trace-driven simulation and real-world experiments.

3.1. Trace-driven simulation

We use the path-perf dataset to simulate how Congi works and expect it to detect similar throughput and packet loss imbalance as the latency-based metric does in Section 2.3.3. Recall that in the path-perf dataset, each throughput sample is associated with a segmented period of latency samples. We use Congi to determine if a period is congested or uncongested based on its latency samples and compute the throughput and packet loss gaps as in Section 2.3.3. To simulate how Congi works, for each throughput sample, we feed the 12 latency samples measured right before starting the throughput test into Congi's accuracy-focused classifier to detect congested paths. If the result comes back positive, we then feed the 12 latency samples right after the completion of the throughput test into the same classifier. We consider the associated period congested if both the results are positive. We check congestion both before and after the throughput test to ensure that the throughput sample is measured while congestion persists. Similarly, we consider the period uncongested if both the results come back positive from Congi's accuracy-focused classifier for detecting uncongested paths. Table 2 compares the throughput ratios and packet loss gaps under Congi and the metric-based method. We can see that Congi achieves similar throughput ratios and slightly better packet loss gaps at different percentiles compared to the metric-based method. This verifies that we can detect congestion imbalance with a small number of latency samples.

Table 2
Trace-driven simulation.

Method	Throughput ratio				Packet loss gap			
	25th	50th	75th	Mean[a]	25th	50th	75th	Mean[a]
Metric-based	0.11	0.32	0.58	0.35	0	0.4%	3.3%	1.6%
Congi	0.09	0.28	0.61	0.34	0	0.5%	5.5%	2.0%

[a] Mean of the samples between the 10th and 90th percentiles.

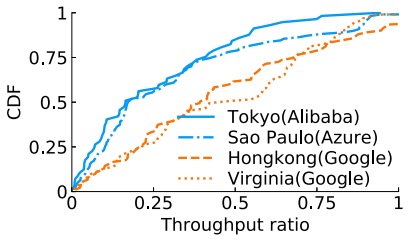


Fig. 6. Ability to detect throughput imbalance.

3.2. Real-world experiments

We next verify Congi's ability to detect the throughput and packet loss imbalance with real-world experiments.

Detecting Throughput Imbalance We ran Congi to measure congestion imbalance from VPs in 9 geographically-diverse DCs (3 DCs per cloud provider) to 471 NDT-7 servers, where NDT servers are used as destinations because we want to measure path throughput with NDT tests. Each VP periodically detects congestion imbalance to all NDT servers every 3 min. Once congestion imbalance is detected between a congested and an uncongested paths to a NDT server, we immediately stop Congi and measure throughput for each path respectively with a NDT test. Since two back-to-back NDT tests take at least 40 s, it is possible that congestion imbalance ceases during the NDT tests. After the NDT tests are done, we resume Congi to collect more samples and double check if congestion imbalance still exists. This ensures that congestion imbalance persists when throughput samples are measured.

Among all congestion imbalance events detected by Congi, the uncongested paths on average have throughput 3x greater than the congested ones. The 25th, 50th, and 75th percentiles of throughput ratios between congested and uncongested paths are 0.10, 0.28, and 0.60 respectively. This implies that Congi is capable of differentiating between the congested and uncongested paths in real-world settings.

Table 3
Prevalence of long imbalance.

	Google Cloud			Amazon AWS			Microsoft Azure			Alibaba Cloud		
	London	Tokyo	HK[a]	London	Tokyo	Cali	London	Tokyo	SP[a]	London	Tokyo	Shanghai
Congestion	251K	171K	172K	331K	349K	339K	283K	295K	318K	362K	392K	1,693K
Long imbl.	15K	15K	16K	37K	48K	35K	36K	32K	39K	59K	43K	32K
%	6.0%	8.8%	9.3%	11.1%	13.7%	10.3%	12.7%	10.8%	12.3%	16.3%	11.0%	1.9%

[a] HK = Hongkong, SP = Sao Paulo.

Moreover, by comparing the throughput gaps in the real-world experiment and the trace-driven simulation, we find that Congi, though trained on the congested and uncongested periods of the same path, can achieve similar throughput gaps between different LB paths in the real-world experiment. This validates our design choice of using the performance data of the same path to train Congi's classifiers in Section 2.5.

Fig. 6 shows the distribution of throughput ratios for four DCs. The figure shows that Alibaba Tokyo's and Microsoft Sao Paulo's DCs have the two smallest mean throughput ratios of 0.22 and 0.26, and that Google's DCs in Hongkong and North Virginia have the two largest mean throughput ratios of 0.43 and 0.46. We can see that although throughput imbalance varies across DCs, all DCs experience significant congestion imbalance on their paths to NDT servers. The average throughput of uncongested paths from Alibaba Tokyo's DC was almost 5x that of the congested paths. For the simplicity of design, Congi uses the same set of classifiers for all DCs and may achieve larger throughput imbalance when customized for each DC.

Detecting Packet Loss Imbalance Packet loss rate is a statistic calculated from a large number of collected samples. To obtain the packet loss gap between two paths, we want to find a long time period wherein both paths remain in the same state (having either constantly varying or stable latency), and calculate the packet loss rate for each path in the period. This implies that we can create a dataset comprising latency time series of LB paths, segmented into varying and stable periods, to evaluate Congi's ability to detect packet loss imbalance. We build this dataset by alternately probing two random LB paths for each selected VP-destination pair, where three VPs (each from a different cloud provider) are used and destinations are the 471 NDT servers. We measure only two LB paths at a time for each VP-destination pair to control the probing rate to a destination at a low level. To mitigate packet drops due to de-prioritization of ICMP responses, we use TCP ACKs rather than ICMPs as probes. Upon completion of probing a pair of LB paths, we segment the resulting latency time series using the changepoint detection method in Section 2.4 and pair the segmented periods of the two LB paths that overlap in time. The packet loss gap is calculated for pairs where each period includes at least 200 samples and congestion imbalance is detected.

We first use Congi to detect congestion imbalance for each pair, where Congi picks a random starting point and takes samples from each period. The 25th, 50th, and 75th percentiles of the resulting packet loss gaps are 0, 0.2% and 0.8% respectively, with a mean of 0.4%. As this dataset includes latency time series of paths, we can also apply the metric-based method to detect packet loss imbalance, which achieves a mean packet loss gap of 0.3%. This verifies that Congi can detect packet loss imbalance as the latency-based method does. Nonetheless, we notice that the packet loss imbalance in this dataset is much lower than that in the path-perf dataset (Table 2). This is because the path-perf dataset includes periods that have a higher packet loss rate, with a mean of 3.7% for all involved periods (the mean is 1.4% for this dataset). Looking at the error rate, we find that Congi only mis-identifies congestion imbalance for 5.6% of pairs (with a negative packet loss gap) in this dataset. The result aligns with our design of Congi, which focuses on differentiating between congested and uncongested paths, but makes no further distinction of congestion levels.

4. Congestion imbalance: A cloud-centric view

We ran Congi from 12 DCs of 4 cloud providers to 10.8M routable /24 s in the CAIDA's IP-to-Prefix dataset [22]. The DCs are selected from major cloud providers and are geographically-diverse to cover different continents. This experiment took about 10 days in January of 2021. The average probing rate of Congi was controlled at 500 packets/second or approximately 110 Kbps to avoid self-induced queueing delay. For each /24, we randomly select an address and probe to the first responsive router closest to the address, so as not to affect the end host.

4.1. Does Congi measure short or long congestion imbalance?

Recall that Congi checks congestion imbalance twice (in Stage II and III respectively) in a complete cycle of probing a destination. We refer to the initial congestion imbalance detection as *short imbalance* and secondary detection as *long imbalance*. The initial detection differs from the secondary detection in that the initial detection uses samples collected in sequence for both the congested and uncongested paths, whereas the secondary detection uses interleaved samples collected alternately between the two paths. This results in the initial detection being prone to false positives: an uncongested path is found because congestion ceases early, causing all paths to be uncongested. We do not use the sample interleaving method for initial detection because it is inefficient when there is no congested path.

We first focus on the long imbalance measured by Congi and design separate experiments to measure short imbalance later in Section 4.4. Another detailed discussion of short imbalance can be found in Section 5, where we use short imbalance as a guide for real applications. In a complete cycle of probing, Congi collects 30 samples for each of the two paths, which means that long imbalance lasts for at least 120 s (inter-sample time is 2 s).

4.2. Prevalence of long imbalance

Table 3 summarizes the number of congestion events and long imbalances seen by each DC in our experiments. Most DCs see long imbalances to 9%–13% of probed addresses. Google Cloud observes much less congestion than other cloud providers. This is due to Google using its own, well-provisioned private WANs to route traffic to egress points closest to the destinations [11]. For Alibaba Cloud, DCs in London and Tokyo observe similar amounts of congestion as Amazon's and Microsoft's DCs. This may be because unlike Google Cloud, they rely on the public Internet to route traffic to destinations [11]. Alibaba Shanghai observes a much higher level of congestion events than other DCs, which indicates the presence of persistent under-provisioning of network resources between China and the rest of the world. Despite that congestion events are frequent, Alibaba Shanghai has the lowest percentage of congestion imbalance among all DCs, because when congestion occurs, almost all LB paths are congested.

Table 4
Prevalence of short imbalance.

Detection method	Google Cloud			Amazon AWS			Microsoft Azure			Alibaba Cloud		
	London	Tokyo	HK[a]	London	Tokyo	Cali	London	Tokyo	SP[a]	London	Tokyo	Shanghai
Interleaved	24%	36%	40%	30%	38%	37%	43%	42%	39%	36%	36%	6.6%
Non-interleaved	43%	62%	71%	64%	59%	58%	65%	62%	63%	58%	57%	29%

[a] HK = Hongkong, SP = Sao Paulo.

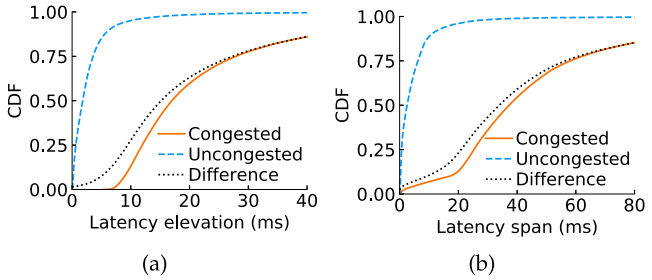


Fig. 7. Latency imbalance.

4.3. Impact on latency imbalance

Congestion imbalance causes LB paths to differ significantly in latency. We analyze the latency imbalance between LB paths under congestion imbalance from several aspects.

Imbalance in Latency Elevation We first look at how much latency is elevated under congestion imbalance in our large-scale experiments. One caveat is that we only have 30 samples for each path and simply use the minimum sample as the minimum latency to roughly estimate latency elevation, which may underestimate latency elevation when all 30 samples are inflated. Fig. 7(a) shows the distribution of latency elevation for the congested and uncongested paths separately and the difference in latency elevation between them. We can see that the congested paths have much more significant latency elevation than the uncongested ones. The average latency elevation is 24 ms for the congested paths and 3 ms for the uncongested ones. This confirms that Congi can detect significant latency imbalance between LB paths. Moreover, about 15% of congested paths have latency elevation higher than 40 ms. We would expect that these congested paths experience severe throughput drop and have much less throughput than the uncongested ones.

Imbalance in Latency Variation We next want to understand the imbalance in latency variation, critical to interactive applications [23]. Specifically, we focus on latency spikes, which may be transient but of significant magnitude. We define *latency span* as in [24], which is the difference between the 90th percentile sample and the minimum. Fig. 7(b) shows the distribution of latency span for all the congested and uncongested paths. The uncongested paths have much more stable latency than the congested ones. About 16% of congested paths have latency span greater than 80 ms, meaning that for 16% of congested paths, 10% of samples could experience latency spikes 80 ms higher than the minimum path latency. The latency spikes are unlikely due to ICMP responses traversing a slow path, because they occur mostly on the congested paths and that Congi ensures that the congested and uncongested paths use the same return path (Section 2.6). We also compare the standard deviation of latency samples between congested and uncongested paths. The average standard deviation is 14 ms for the congested paths and only 1.3 ms for the uncongested paths.

Imbalance in Latency Mean We further look at the distribution of the difference in mean latency between congested and uncongested paths (not shown). About half of VP-destination pairs have a mean latency difference greater than 17 ms, and 10% of pairs have a mean latency difference greater than 80 ms. We also notice that about 10% of pairs have an uncongested path with a higher mean latency than the congested one. This implies that a path with lower latency is not always the better choice as it could be congested.

4.4. Prevalence of short imbalance

As mentioned in Section 4.1, Congi achieves fast search of short imbalance at the cost of accuracy. To better characterize short imbalance, we design a separate experiment as follows. The method used in this experiment is not very efficient. We employ it here to better understand short imbalance but do not otherwise use it as part of Congi.

Experiment Design To reduce the false positive rate of short imbalance detection, we use the interleaved sampling method originally employed in Section 4.1 to detect long imbalance. Given a fixed probe sending rate and a minimum number of samples required for each path, the more LB paths we probe under this method, the longer the detection will take. To detect short imbalance before it abates, we have to keep the number of LB paths probed to the minimum. We determine the number of LB paths to probe by considering the number of LB paths probed in our previous experiment on measuring long imbalance in Section 4.2. We find that short imbalance can be detected in nearly half the cases by probing only two LB paths and if a third path is added, short imbalance can be detected in about two thirds of the cases. Unfortunately, adding a third path also makes the method less sensitive to short imbalance, where only imbalance lasting 36 s or longer can be detected (suppose the probing rate to a destination is 1 packet/s). In this experiment, we probe two LB paths in parallel to detect short imbalances lasting at least 24 s.

Data Collection and Experiment Results From each of our VPs (used in Section 4.2), we alternately probe two LB paths in parallel to 1M random IPv4 addresses and collect 12 samples for each LB path. Then, for each pair, we feed the samples of LB paths to Congi's classifiers to detect if short imbalance exists. Table 4 shows the percentage of short imbalance seen by our VP in each DC. We can see that congestion imbalance detected using interleaved sampling is much smaller than that using non-interleaved sampling, and should reflect more closely the actual percentage of short imbalance. Overall, except for Alibaba Shanghai, short imbalance is significant and prevalent for all DCs.

5. Impact on applications

Since congestion affects all performance metrics (throughput, latency, and packet loss) of a path, we would expect that congestion imbalance has an impact on a wide range of applications. We use web page load as an example to show how application performance would be affected. We simply measure the performance of web page load with the page download time. More sophisticated performance metrics are out of the scope of this work.

5.1. Web page load

When loading a web page, a client browser resolves the dependency between objects and issues a series of HTTP requests to download the objects following the dependency chain. The total download time is determined by each individual request on the longest dependency

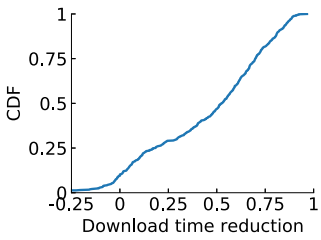


Fig. 8. Impact on web page load.

chain. To understand how congestion imbalance impacts the completion time of individual requests, we use Congi to monitor congestion imbalance to servers hosting the Alexa top-1000 websites. We find the largest object in a web page using the Chrome web browser's Remote Debugging Protocol (RDP). Since congestion imbalance may not persist, we download the target object immediately after short imbalance is detected. The object is fetched with HTTP requests along the congested and uncongested paths, and is downloaded twice for each path, where the first download is to warm the cache and the second download is used to obtain the download time. All downloads are done serially, to avoid self-interference.

We denote t_1 and t_2 as the respective download time for using the congested and uncongested paths, and measure the reduction in download time as $(t_1 - t_2)/t_1$, indicating the possible performance improvement for clients using the congested paths under congestion imbalance. Fig. 8 shows the distribution of download time reduction for 500 downloads from a node in the campus network, where there is sufficient bandwidth for the access link. We can see that the median download time reduction is as high as 53% under congestion imbalance. About 9% of downloads have a negative reduction in download time, which indicates that using the uncongested path results in a larger download time than using the congested path. This aligns with our observation in Section 4.3 that uncongested paths may have longer latency than the congested ones.

6. Discussion

Limitations. Although we have verified the congestion imbalance between our VPs and NDT servers, we cannot do the same for our large-scale experiments measuring imbalance to random Internet addresses. We thus interpret our large-scale measurement results as providing a coarse estimation of congestion imbalance. A more accurate large-scale experiment can be done with user study or by content providers that can directly infer throughput and packet loss imbalance from the traffic between their servers and clients.

Implications. Our experimental results show that performance imbalance between load-balanced paths is both significant and prevalent in the Internet. The problem of congestion imbalance can be mitigated at different layers of the network stack. At the network layer, congestion-aware load balancing algorithms can be designed. Instead of relying on congestion-oblivious hashing algorithms, a closed-loop control can be applied for load balancing, e.g., integrating the early congestion notification (ECN) in the load balancing algorithm. At the application layer, applications can carefully choose routing paths by manipulating the port numbers, as shown in Section 5.

7. Related work

Topology and Latency Difference Between LB Paths. Since *Paris Traceroute* was developed by Augustin et al. in 2006 [25], the topology difference between Internet LB paths has been extensively studied [18,26,27]. However, there are a very limited number of measurement studies about the performance difference between LB paths. Augustin et al. [5] first studied latency imbalance in 2007, which is the difference in the minimum latency between paths. In 2020, Pi et al. [11] revisited the difference in the minimum latency between load-balanced paths, which excludes latency inflation due to queueing. This work complements the previous work by studying the difference in the inflated latency between load-balanced paths, as well as the difference in throughput and packet loss. As congestion events last temporarily, it is hard to capture.

Latency-based Congestion Detection. The first step to detect congestion imbalance is to find an effective way of detecting congestion in the Internet. Latency inflation during TCP sessions has been used to infer congestion for TCP congestion control since CARD [28], DUAL [29], and Vegas [30] decades ago. In latency-based congestion control, the estimated path latency is updated almost every round-trip time to infer transient congestion [31,32]. Although our interest is not to detect transient congestion that requires high-frequency latency samples, the idea of using latency inflation as an indicator to congestion is also applicable to detecting long-term congestion. In 2014, Luckie et al. [8] proposed the time-series latency probes (TSLP) method that detects persistent inter-domain congestion by observing elevated latency in latency time series collected in the long term. After that, the TSLP method was used to study the impact of congestion on the African IXPs in [33] and was further improved and validated in [7]. Our work also uses latency probes to detect congestion, but differs in three respects. First, we conduct a comparative study of several latency-based metrics for congestion detection to choose the best metric. Second, we propose to detect congestion with a very small number of samples rather than long-term latency time series, which enables congestion detection at scale. Third, we conduct large-scale measurement campaigns from VPs in DCs to Internet-wide addresses to show the prevalence and significance of congestion imbalance. Congestion imbalance has also been studied in DC networks [3,4,34] with the goals of promoting congestion-aware forwarding rather than characterizing it.

Throughput Estimation and Modeling. Our work is also related with those that use network probes to estimate available bandwidth or throughput. Available bandwidth estimation techniques send a sequence of probes and use the change either in inter-probe time or between the sending and receiving rates to estimate the available bandwidth [35–37]. They generally have at least one of the following drawbacks preventing them from being used to measure available bandwidth at scale: (1) control is required at both the sender and the receiver sides, e.g., Pathload [36] and Spruce [38]; (2) performance varies under different network settings and there is no guarantee of accuracy [39] and that (3) estimation may incur high measurement overhead or require high resolution on inter-probe time [39]. Similar to available bandwidth estimation, there are extensive research efforts modeling TCP throughput with latency and packet loss [40,41]. He et al. [42] show that using latency and packet loss measured before a TCP session to predict TCP throughput could result in large errors and provide significantly less accuracy than using those metrics measured during the TCP session. A recent work [43] greatly improved the modeling accuracy by incorporating the congestion window size, which however limits its usage to throughput prediction during TCP sessions. Considering these drawbacks and limitations, none of the works above provides a viable light-weight solution to characterize throughput imbalance between LB paths at scale with network probing.

8. Conclusion

In this paper, we took the first step towards measuring congestion imbalance at scale. We presented Congi, a network prober that uses SVM classifiers to detect congestion imbalance with a very small number of samples, which enables us to conduct large-scale measurements. Congi was verified to be capable of detecting the imbalance in throughput, packet loss and latency between LB paths. We used Congi to measure congestion imbalance from our VPs around the

globe to Internet-wide addresses, and found that short-term congestion imbalance is prevalent and significant in the Internet. Taking web page load as an example, we demonstrated that congestion imbalance greatly affected application performance. For future work, we want to pinpoint the networks responsible for congestion imbalance in the Internet and characterize congestion imbalance when various access networks are considered.

CRediT authorship contribution statement

Yibo Pi: Conceptualization, Methodology, Software, Writing – original draft. **Sugih Jamin:** Writing – review & editing. **Yichen Wei:** Validation, Software, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

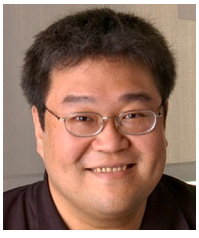
Data availability

Data will be made available on request.

References

[1] Rafael Almeida, Ítalo Cunha, Renata Teixeira, Darryl Veitch, Christophe Diot, Classification of load balancing in the internet, in: INFOCOM, 2019.
[2] Srikanth Kandula, et al., Dynamic load balancing without packet reordering, in: SIGCOMM CCR, 2007.
[3] Mohammad Alizadeh, et al., CONGA: Distributed congestion-aware load balancing for datacenters, in: SIGCOMM, 2014.
[4] Naga Katta, et al., Clove: Congestion-aware load balancing at the virtual edge, in: CoNEXT, 2017.
[5] Brice Augustin, Timur Friedman, Renata Teixeira, Measuring load-balanced paths in the internet, in: IMC, 2007.
[6] Youndo Lee, Neil Spring, Identifying and aggregating homogeneous IPv4/24 blocks with hobbit, in: IMC, 2016.
[7] Amogh Dhamdhere, et al., Inferring persistent interdomain congestion, in: SIGCOMM, 2018.
[8] Matthew Luckie, et al., Challenges in inferring internet interdomain congestion, in: IMC, 2014.
[9] M-Lab, NDT (Network Diagnostic Tool), https://www.measurementlab.net/tests/ndt/.
[10] M-Lab, ndt7 Protocol - NDT (Network Diagnostic Tool), https://www.measurementlab.net/tests/ndt/ndt7/.
[11] Yibo Pi, et al., Latency imbalance among internet load-balanced paths: A cloud-centric view, in: SIGMETRICS, 2020.
[12] Bob Briscoe, et al., Reducing internet latency: A survey of techniques and their merits, IEEE Commun. Surv. Tutor. (2014).
[13] Charles Truong, Laurent Oudre, Nicolas Vayatis, Selective review of offline change point detection methods, Signal Process. (2020).
[14] Charles Truong, Laurent Oudre, Nicolas Vayatis, ruptures: change point detection in Python, 2018, arXiv preprint arXiv:1801.00826.
[15] Sushant Jain, et al., B4: Experience with a globally-deployed software defined WAN, in: SIGCOMM, 2013, pp. 3–14.
[16] Srikanth Sundaresan, et al., Challenges in inferring internet congestion using throughput measurements, in: IMC, 2017.
[17] Hang Guo, John Heidemann, Detecting ICMP rate limiting in the internet, in: PAM, 2018, pp. 3–17.
[18] Kevin Vermeulen, Stephen D. Strowes, Olivier Fourmaux, Timur Friedman, Multilevel MDA-Lite Paris traceroute, in: IMC, 2018.
[19] Chih-Chung Chang, Chih-Jen Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. (2011).
[20] Ganesh R. Naik, Dinesh Kant Kumar, Twin SVM for gesture classification using the surface electromyogram, IEEE Trans. Inf. Technol. Biomed. (2009).
[21] Carolyn J. Anderson, Central limit theorem, in: The Corsini Encyclopedia of Psychology, 2010.
[22] CAIDA, Routeviews prefix to AS mappings dataset for IPv4 and IPv6, 2018, https://www.caida.org/data/routing/routeviews-prefix2as.xml.
[23] Chi-Yao Hong, et al., Achieving high utilization with software-driven WAN, in: SIGCOMM CCR, 2013.
[24] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, Anna Brunstrom, Measuring latency variation in the internet, in: CoNEXT, 2016.
[25] Brice Augustin, et al., Avoiding traceroute anomalies with Paris traceroute, in: IMC, 2006.
[26] Brice Augustin, Timur Friedman, Renata Teixeira, Measuring multipath routing in the internet, IEEE/ACM Trans. Netw. (2010).
[27] Rafael Almeida, et al., A characterization of load balancing on the IPv6 internet, in: PAM, 2017.
[28] Raj Jain, A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks, in: SIGCOMM CCR, 1989.
[29] Zheng Wang, Jon Crowcroft, A new congestion control scheme: Slow start and search (Tri-S), in: SIGCOMM CCR, 1991.
[30] Lawrence S. Brakmo, Sean W. O'Malley, Larry L. Peterson, TCP vegas: New techniques for congestion detection and avoidance, in: Proceedings of the Conference on Communications Architectures, Protocols and Applications, 1994.
[31] Venkat Arun, Hari Balakrishnan, Copa: Practical delay-based congestion control for the internet, in: NSDI, 2018.
[32] Belma Turkovic, Fernando A. Kuipers, Steve Uhlig, Fifty shades of congestion control: A performance and interactions evaluation, 2019, ArXiv Preprint arXiv:1903.03852.
[33] Rodérick Fanou, Francisco Valera, Amogh Dhamdhere, Investigating the causes of congestion on the African IXP substrate, in: IMC, 2017.
[34] Zixi Cui, Yuxiang Hu, Julong Lan, Yu Wang, Load balancing based on flow classification for datacenter network, Acta Electonica Sinica 49 (3) (2021).
[35] Vinay Joseph Ribeiro, et al., Pathchirp: Efficient available bandwidth estimation for network paths, in: Passive and Active Measurement Workshop, 2003.
[36] Manish Jain, Constantinos Dovrolis, End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput, in: SIGCOMM CCR, 2002.
[37] Ningning Hu, Peter Steenkiste, Evaluation and characterization of available bandwidth probing techniques, IEEE JSAC (2003).
[38] Jacob Strauss, Dina Katabi, Frans Kaashoek, A measurement study of available bandwidth estimation tools, in: IMC, 2003.
[39] Alok Shriram, et al., Comparison of public end-to-end bandwidth estimation tools on high-speed links, in: International Workshop on Passive and Active Network Measurement, 2005.
[40] Jitendra Padhye, et al., Modeling TCP throughput: A simple model and its empirical validation, in: SIGCOMM, 1998.
[41] Mariyam Mirza, et al., A machine learning approach to TCP throughput prediction, in: SIGMETRICS, 2007.
[42] Qi He, Constantine Dovrolis, Mostafa Ammar, On the predictability of large transfer TCP throughput, in: SIGCOMM, 2005.
[43] Yi Cao, et al., Econ: Modeling the network to improve application performance, in: IMC, 2019.

Yibo Pi received the B.S. and M.S. degrees in ECE from the Shanghai Jiao Tong University in 2012 and 2015. He received his Ph.D. degree in Computer Science from the University of Michigan in 2021. He is currently an assistant professor with the UM-SJTU Joint Institute, Shanghai Jiao Tong University. His current research interests include network measurement, wireless networking, and network telemetry.



Sugih Jamin is an Associate Professor in the Department of Electrical Engineering and Computer Science at the University of Michigan. He received his Ph.D. in Computer Science from the University of Southern California, Los Angeles in 1996. He received the National Science Foundation (NSF) CAREER Award in 1998, the Presidential Early Career Award for Scientists and Engineers (PECASE) in 1999, and the Alfred P. Sloan Research Fellowship in 2001. He co-founded a live streaming Internet-TV company, Zattoo Inc., in 2005.

Yichen Wei is currently an undergraduate student in ECE at the University of Michigan - Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University. His current research interests include network measurement and wireless networking.