Zooming in on Wide-area Latencies to a Global Cloud Provider

Yuchen Jin
Microsoft/University of Washington
yuchenj@cs.washington.edu

Sundararajan Renganathan
Microsoft
t-sur@microsoft.com

Ganesh Ananthanarayanan
Microsoft
ga@microsoft.com

Junchen Jiang
University of Chicago
junchenj@uchicago.edu

Venkata N. Padmanabhan
Microsoft
padmanab@microsoft.com

Manuel Schroder
Microsoft
manscho@microsoft.com

Matt Calder
Microsoft
mcalder@microsoft.com

Arvind Krishnamurthy
University of Washington
arvind@cs.washington.edu

ABSTRACT

The network communications between the cloud and the client have become the weak link for global cloud services that aim to provide low latency services to their clients. In this paper, we first characterize WAN latency from the viewpoint of a large cloud provider Azure, whose network edges serve hundreds of billions of TCP connections a day across hundreds of locations worldwide. In particular, we focus on instances of latency degradation and design a tool, BlameIt, that enables cloud operators to localize the cause (i.e., faulty AS) of such degradation. BlameIt uses passive diagnosis, using measurements of existing connections between clients and the cloud locations, to localize the cause to one of cloud, middle, or client segments. Then it invokes selective active probing (within a probing budget) to localize the cause more precisely. We validate BlameIt by comparing its automatic fault localization results with that arrived at by network engineers manually, and observe that BlameIt correctly localized the problem in all the 88 incidents. Further, BlameIt issues 72× fewer active probes than a solution relying on active probing alone, and is deployed in production at Azure.

CCS CONCEPTS

- **Networks** → **Network performance analysis**; **Network measurement**; **Error detection and error correction**; **Network monitoring**; **Public Internet**;

KEYWORDS

Wide-area network, Network diagnosis, Internet latency measurement, Network fault localization, Tomography, Active network probes

ACM Reference Format:
Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N. Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. 2019. Zooming in on Wide-area Latencies to a Global Cloud Provider. In *Proceedings of ACM SIGCOMM 2019 Conference, Beijing, China, August 19–23, 2019 (SIGCOMM '19),* 13 pages.
https://doi.org/10.1145/3341302.3342073

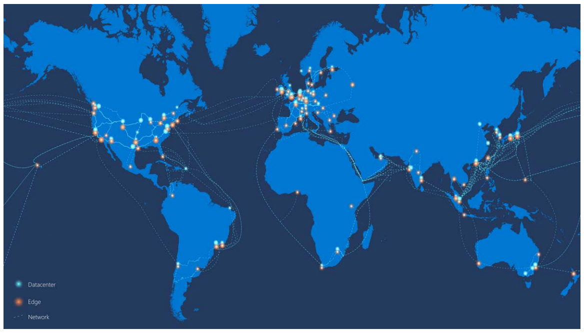


Figure 1: Map of Azure locations worldwide.

1 INTRODUCTION

Global cloud services rely on three types of communication: intra-data center, inter-data center, and the public Internet. Intra-DC and inter-DC communications, which are under a single administrative domain, have seen rapid improvement in their performance and reliability over the years. On the other hand, public Internet communication, which connects clients to the cloud locations, has seen less progress because it typically spans multiple administrative domains, which limits visibility into the network and the velocity of change. As a result, the public Internet communication become the weak link for cloud services. This *cloud to client communication* is the focus of this paper.

We focus on a global-scale cloud service, Azure, that hosts a range of interactive services. Azure's clients access their content, over TCP, from the hundreds of its network edge locations worldwide (see Figure 1). The hundreds of billions of TCP connections from the clients per day provides a measure of latency — the handshake RTT from TCP connection setup.

Measurement Characterization: Using a few trillion RTT measurements from a month, we first characterize poor (or "bad") latency for the cloud-client connections; we define badness based on Azure's region-specific RTT targets. Instances of bad latency are

SIGCOMM '19, August 19–23, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5956-6/19/09...$15.00
https://doi.org/10.1145/3341302.3342073

Desired Property	BlameIt	Tomography [9]	EdgeFabric [32]	PlanetSeer [38]	iPlane [26]	Trinocular [30]	Odin [8]	WhyHigh [22]
Latency degradation	✓	✓	✓	✗	✓	✗	✓	✓
Internet scale	✓	✗	✓	✗	✗	✓	✓	✓
Work with insufficient coverage	✓	✗	✓	✓	✗	✓	✓	✓
Automated root-cause diagnosis	✓	✓	✗	✓	✓	✓	✓	✗
Diagnosis with low latency	✓	✗	✓	✗	✗	✓	✓	✗
Triggered timely probes	✓	✗	✗	✓	✗	✗	✗	✗
Impact-prioritized probes	✓	✗	✗	✗	✗	✗	✗	✗

Table 1: Comparison with prior network diagnosis solutions on the desired properties for scalable fault localization.

not concentrated in a few locations but rather widely prevalent. The duration of badness (persistence) has a long-tailed distribution: most instances of badness are fleeting (<5 minutes) while a small number of them are long lasting. Finally, despite the widespread prevalence of badness among many IP-/24's, the *affected clients* themselves are concentrated only in a small number of IP prefixes.

Localization of faulty AS: The administrators of Azure have a strong need for a tool for *fault localization* when there is RTT degradation (i.e., the RTT breaches the target), specifically, a tool that ascribes blame to the AS(es) that caused the spike in RTT. The faulty AS could be Azure's cloud network itself or one or more of the AS'es in the cloud-client path. In our study of actual incidents (§6.3), we see various reasons for RTT degradation, including overloaded cloud servers to congested cloud networks, maintenance issues in the client's ISP, and path updates inside a transit AS. A tool to localize the faulty AS, as quickly as possible, will help Azure's operators trigger remedial actions such as switching egress peers, or investigating server-side issues, thereby minimizing the duration of user impact. In the absence of such a tool, the current practice is to investigate a handful of incidents from the previous day, often chosen in an untargeted manner, which could lead to wasted effort on issues that are not fixable by the cloud operators (e.g., client ISP fault) as well as the ignoring of severe issues (e.g., cloud's fault with many affected clients). Hence, the need for an accurate tool to localize the faulty AS(es). [1]

Fault localization in the network is a long-standing problem. Table 1 summarizes the main prior solutions, along with the desired properties of a fault localization solution. Our work borrows some elements from prior solutions but distills an overall approach that localizes high latency faults *while* maintaining a frugal budget for measurement probes and prioritizing the high-impact issues. In particular, prior solutions can be bucketed into "passive" techniques that analyze end-to-end RTT data alone [9, 32, 38] or "active" techniques that rely on issuing probes [26, 30]. The former is often intractable because of insufficient coverage of paths in the measurements, while the overhead of probing with the latter is often prohibitive. Even when techniques combine passive analysis with active probing [8, 22], they do not opportunistically use the passive data for localization (whenever possible), they do not trigger probes in a timely fashion (during the issue), nor do they prioritize the probes (for high impact issues). This leads to many unnecessary probes as well as probing after the incidents. Our work addresses the above issues and intelligently mixes passive analysis with prioritized active probing to achieve accurate fault localization at only a fraction of their probing costs.

Two-phased Design: Our solution BlameIt proceeds in two phases. The first is a *passive* phase, where BlameIt uses the RTT data from the existing TCP connections between clients and the cloud to *coarsely* assign blames to the "client", "cloud", or "middle" segments. We define the middle segment as the "BGP path" (set of middle AS'es between the client and cloud). We avoid the intractability of classical tomography solutions by leveraging two strong empirical traits: (1) typically, only one of the AS'es in a client-cloud path is at fault, and (2) a smaller "failure" set is more likely than a larger set, i.e., increases in RTT for *all* the clients connecting to a cloud location is likely due to a fault at the cloud end and not due to faults at each of the client AS'es. These traits allow for a hierarchical elimination approach, starting with the cloud as the potential culprit and going down to the middle and client segments.

To resolve the location of the "middle" segment problems to the granularity of an AS (as desired by Azure's administrators), BlameIt resorts to an *active* phase. BlameIt issues traceroutes from relevant Azure locations to a targeted set of client IPs while the latency issue is ongoing. The results of these traceroutes are compared with a baseline obtained from "background" traceroutes, to localize the specific AS(es) responsible for the increase in latency. BlameIt is judicious in its use of traceroutes by *prioritizing* their use on those issues that are expected to last for longer and impact a larger number of clients (both, predicted from historical data). It also heavily optimizes the overheads of background probes by trading it for a small drop in accuracy. Overall, we adopt a systematic measurement-based approach to designing the various aspects in BlameIt's two-phased fault localization.

Production Deployment: BlameIt is in production deployment, its passive component since Nov 2017 and its active component is beginning to get widely rolled out. Its outputs are used by network operators for their investigations on a daily basis. We compare the accuracy of BlameIt's result (i.e., the blamed AS) to 88 incidents that had manual reports and find that it correctly localizes the fault in *all* the incidents; §6.3 covers a subset of the incidents. In addition, BlameIt's selective active probing results in 72× fewer traceroutes than a solution that relies on active probing alone, and 20× fewer traceroutes than Trinocular that optimizes probing for WAN unreachability [30] .

Contributions: BlameIt makes the following contributions.

(1) We present a large-scale study of cloud-client latencies using trillions of RTTs from a global cloud provider; §2.
(2) We design a practical solution using only passive measurements for coarse-grained fault localization; §4.
(3) We judiciously issue active probes to localize middle segment issues, while prioritizing the high-impact issues; §5.

[1] We use the terms "RTT degradation", "fault", "issue", "incident" etc. interchangeably.

# RTT measurements	Many trillions
# client IP	$\mathcal{O}$(100 million)
# client IP /24's	Many millions
# BGP prefixes	$\mathcal{O}$(100,000)
# client AS'es	$\mathcal{O}$(10,000)
# client metros	$\mathcal{O}$(100)

Table 2: Details of the dataset analyzed (one month in 2018).

2 CHARACTERIZING WORLDWIDE LATENCY

We present a panoramic view of Azure's client latency. We first characterize the general patterns of high latency – prevalence, persistence and long-tailed duration of issues – (§2.2 and §2.3), and then identify spatial aggregates that account for much of the latency issues (§2.4).

2.1 Dataset and methodology

Azure's infrastructure consists of hundreds of network edge locations, with tens of services hosted at each location. The services are interactive (latency-sensitive) and cater to consumer and enterprise clients covering a broad set of products around productivity, search, communications and storage. Hundreds of millions of clients use the services on Azure every day.

Each client connection is over TCP to one of the nearest cloud locations. [2] For each connection, Azure records the TCP handshake RTT at the cloud server. Across all the cloud locations, Azure records hundreds of billions of RTTs each day. Table 2 lists the details of our dataset. We use Azure's targets as the latency "badness" thresholds and it varies according to the region and the device connectivity type. [3] The targets are in keeping with the variation in RTTs across regions and are set such that no client prefix's RTT is consistently above the threshold.

Good/Bad Quartet: To understand the structural patterns of high latencies, we analyze our global-scale dataset through the lens of *quartets*. Each quartet is a 4-tuple consisting of ⟨client IP /24 prefix, cloud location, mobile (or) non-mobile device, 5 minute time bucket⟩. This definition helps us bundle together the measurements from clients that are geographically nearby using similar AS-level network paths [25], and connecting to the same cloud location at similar times. Despite the fine granularity of slicing, we typically still have many tens of RTT samples in each quartet.

We classify each quartet as "good" or "bad" depending on whether the average RTT in that quartet is below or above the corresponding badness thresholds. Note that we require each quartet to have at least 10 RTT samples for confidence in our estimates. We also verified that when the RTT samples in a quartet were randomly divided in to two sets, the Kolmogorov-Smirnov test showed that the samples in both these sets were from the same distribution.

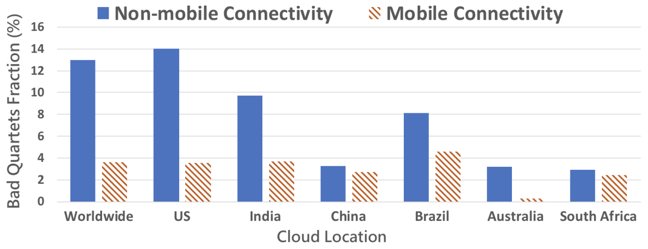


Figure 2: Fraction (%) of quartets whose average RTT was deemed to be bad. Badness thresholds are set based on Azure's region-specific RTT targets.

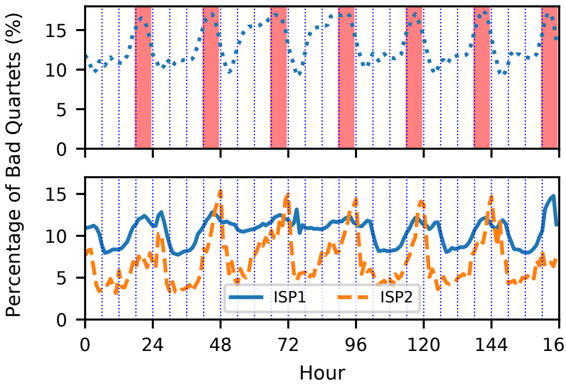


Figure 3: Bad quartets (%) by the hour for 1 week in USA (top) and for two ISPs (bottom). Night hours are marked. Weekend is between the 48th and 96th hours.

2.2 Spatial & temporal distribution of latencies

Our first set of findings show that the the occurrence of high (or "bad") WAN latency is widely spread, both in time and in space.

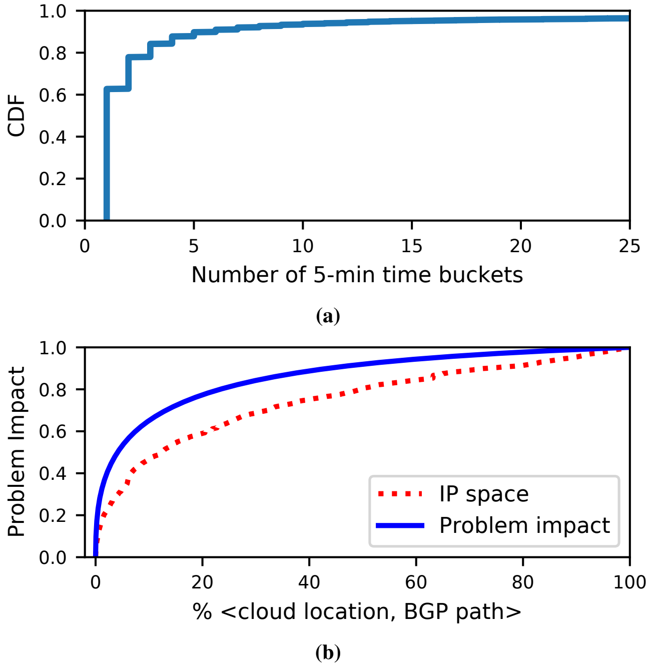
How prevalent is bad connectivity? Figure 2 plots the fraction of quartets whose RTT was bad, split by region. We see that high latency issues are widely distributed, with a substantial fraction of bad quartets for both mobile and non-mobile connections in all regions. While the trend on bad quartets generally decreases with the advancement in network infrastructure of the region (e.g., China and Australia), we see that surprisingly the USA has a higher fraction of bad quartets, most likely due to aggressive thresholds for the USA.

We next analyze the prevalence of latency issues from the perspective of cloud locations. We see that *one-third of the cloud locations have at least 13% bad quartets*.

How does badness vary over time? We also study the effect of time-of-day in the incidents of bad RTTs. Figure 3 (top) plots the fraction of bad quartets, bucketed by hours, over the course of a week. While there is an unsurprising diurnal pattern to badness, an interesting aspect is that the *fraction* of bad quartets is consistently *higher* in the nights than during work hours. We speculate that this is because the connections during off-work hours would tend to be from home ISPs compared to the well-provisioned enterprise networks during the day. Indeed, we see that BlameIt's fault localization often lays the blame for the instances of bad RTTs at night on the client ISP.

[2] The cloud servers are organized into multiple anycast rings and clients connect to the ring that corresponds to their location and the service accessed, leaving it to BGP to direct them to the "nearest" server.

[3] Typically, mobile clients use cellular connectivity while non-mobile clients use a broadband network. Going forward, we plan to distinguish Wi-Fi connections as well.



(a)

(b)

Figure 4: (a) Persistence of bad RTT incidents in a day (in consecutive 5-min buckets). (b) CDF of problem impact when ⟨Cloud location, BGP path⟩ are ranked by two orders.

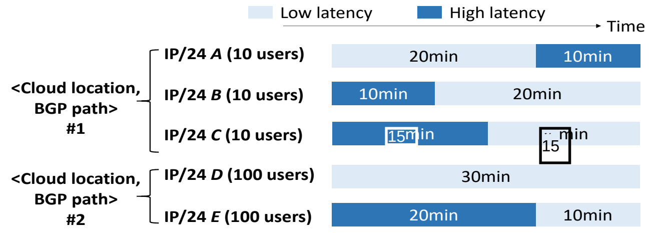
Figure 3 (bottom) zooms in to show that the magnitude of temporal variance can be different; the fraction of bad quartets in ISP1 varies within a range of 5%, while that of ISP2 is over 10%. Moreover, ISP1 shows a different pattern during the weekend (between the 48th and the 96th hour), in which the diurnal pattern becomes less obvious than the workdays. All of this suggests that the design of `BlameIt` should not assume general temporal predictability to badness.

2.3 Long-tailed distribution of badness durations

Incidents of bad latency last for different durations and we analyze the distribution of their persistence. We count the *consecutive* 5-minute time windows for which a ⟨IP /24, cloud location, device-type⟩ tuple suffers poor latency (§2.1). As shown in Figure 4a, over 60% of the issues last for ≤ 5 minutes while only 8% of the issues last for over 2 hours (with the distribution being long-tailed). In other words, most of the issues are fleeting rather than long-lived. Our solution aims to identify the serious long-lived issues and alert `BlameIt`'s cloud operators. We also prioritize and issue traceroute probes *during these incidents* that help with their investigations.

2.4 Distribution of performance impact

Of relevance to performance diagnosis solutions at global scale (like `BlameIt`) is the spatial concentration of issues. For each ⟨cloud location, BGP path⟩ tuple, we calculate its "impact": defined as the number of affected users (distinct IP addresses) multiplied by the duration of the RTT degradation. BGP path is the set of middle AS'es between the client and cloud. Measuring impact at the granularity of ⟨cloud location, BGP path⟩ naturally aligns with our coarse-grained segments (covered in §3).



	Weighted by # of affected prefixes	Weighted by actual problem impact ∑(duration) X (# of clients)
#1	3	350
#2	1	2000

Figure 5: Illustrative example of how ⟨Cloud location, BGP path⟩ tuples can be ranked in two different orderings.

We sort the ⟨cloud location, BGP path⟩ tuples by two metrics (Figure 5). The simplest is by the number of problematic IP-/24s contained in the ⟨cloud location, BGP path⟩ tuple, which is similar to how prior works rank the importance of a spatial aggregate [22]. We could also rank the tuples by their actual impact, as we defined above. Figure 4b plots the CDF of impact of the tuples when sorted in these two orderings.

When ranked by the IP-/24s, the top 60% of the ⟨cloud location, BGP path⟩ tuples cover nearly 80% of the cumulative problem impact (red line). However, to achieve the same coverage in problem impact, we only need 20% of the ⟨cloud location, BGP path⟩ tuples, if we rank them by their problem impact (blue line), or a 3× lower value.

Thus, our analysis shows that although there is a skewed distribution of the impact of latency problems in the IP space (as also seen by prior works [22]), there is a *much higher skew in the problem's impact when viewed jointly in space and time*, *i.e.,* the number of affected users and the problem's duration. It can be inefficient to simply use the number of IP prefixes to measure the impact of RTT degradations. This observation will be central to `BlameIt`'s budgeted active probing in §5.

2.5 Summary of observations

The takeaways from our measurement study are as follows:

(1) *Spatial and temporal spread of high latencies.* Many cloud locations and client-side prefixes have experienced latency regression for a non-negligible amount of time.

(2) *Long-tailed distribution of badness durations.* Most incidents of bad latencies are fleeting with only a small number of them being long-lasting and of interest for investigations.

(3) *Uneven distribution of the impact of high latencies.* Despite many IP prefixes having high latencies, a substantial fraction of the affected clients are in a small number of IP prefixes.

3 OVERVIEW OF `BLAMEIT`

The primary purpose of `BlameIt` is to help network administrators investigate reports of poor network performance, i.e., RTTs between the clients and the cloud locations being above the badness threshold.

BlameIt attributes such *path-level* latency degradations to localize the faults at the *AS-granularity*. In this section, we explain the high-level intuitions behind our solution before elaborating on the details in §4 and §5.

3.1 Two-level blame assignment

Modeling the paths between the cloud locations and client at the AS-granularity, as per traditional approaches [9, 38], is problematic. First, the coverage of the measurements is skewed and therefore there are often not sufficient measurements to identify the contributions by each AS in the path. Second, modeling the graph at the AS granularity introduces ambiguities, e.g., a large AS like Comcast might have a problem along certain paths but not all. The impact of both these problems is compounded when paths in the Internet change.

Instead of modeling a path as a concatenation of AS'es, BlameIt views each path in two granularities—a *coarse-grained* segmentation of the path into three segments of "client" (the client AS), "cloud" (the cloud AS), and "middle" (all AS'es between the cloud and the client), and then a *fine-grained* AS-level view of only the "middle" segment. Correspondingly, BlameIt localizes the fault in two steps.

(1) Attribute the blame for latency degradation on one of the three coarse segments, using *only* "passive" data.

(2) If a fault is attributed to a middle segment, BlameIt (optionally) moves to the "active" phase to trigger probes for AS-level fault localization.

Note that modeling *each* path into three segments (albeit at a coarse granularity) allows BlameIt to identify localized issues in an AS that occur only in that path (but not elsewhere in that AS, thus avoiding the ambiguities explained above).

Coarse Segmentation: The three-way segmentation of each path into client, middle, and cloud segments is borne out of Azure's operational requirements. Localizing the fault to the cloud or the client AS is already actionable. A *cloud-induced* latency degradation would directly lead to personnel being deployed for further investigation. When the problem lies with the *client* network (e.g., the client's ISP), the cloud operator typically cannot fix the problem other than informing the client about their ISP choices (e.g., [1]). Thus, our segmentation proved to be pragmatically beneficial and allowed for a phased deployment with our coarse-grained fault localization solution (based on passive measurements) being deployed in production much earlier.

3.2 Impact-proportional budgeted probing

BlameIt uses active probes (traceroutes) only for fine-grained localization of "middle" segment blames, as these segments may contain multiple AS'es. Since probing all the paths for full coverage is prohibitive given the sheer number of paths (nearly 200 million per day; §6.5), BlameIt sets a *budget* on the active probes and allocates the budget to probing bad middle segments based on their expected *impact* on clients (§2.4), i.e., by estimating *how many clients will be affected* and *for how long*. BlameIt's budgeted probing is unique in two aspects compared to prior work (e.g., [22, 26, 38]).

Spatially, BlameIt defines the importance of a segment in proportion to its impact on actual *number of clients*, rather than just the addresses in the IP block (BGP-announced prefix) [22]. In Azure's operations, large IP address blocks often have fewer active clients than smaller IP blocks, thus making it desirable to predict and prioritize issues affecting the most active clients.

Temporally, by *estimating* the timespan of each latency degradation, BlameIt can focus on long-lived problems rather than fleeting problems that may end shortly after we probe it, thus wasting our limited budget for probing. Note that we do not need very precise estimate on the timespan of a problem because of the long tail distribution of problem durations (§2.3). It would suffice if we only separate the few long-lived problems from the many short-lived problems.

3.3 End-to-end workflow

BlameIt's workflow is as follows. RTT data is passively collected at the different cloud locations and sent to a central data analytics cluster. A data analytics job periodically makes coarse-grained blame assignments to all bad RTT instances; §4. These blame assignments trigger two sets of actions. The middle segment issues are ordered based on their impact and active probes are issued from the appropriate cloud servers for finer localization; §5. All the latency inflations are prioritized based on their impact, and the top few are sent to network administrators for investigation (details in §6.1).

4 FAULT LOCALIZATION WITH PASSIVE MEASUREMENTS

In this section, we describe BlameIt's fault localization method using passive RTT measurements only. We begin by laying down a set of empirical insights (§4.1) that leads to the practical fault localization method (§4.2 − §4.3).

4.1 Empirical insights

At first glance, it is tempting to localize faults using standard network tomography techniques, especially given the coarse three-way segmentation (§3.1) of the network topology. Unfortunately, the client-middle-cloud segmentation still lacks the topological properties needed for a standard network tomography formulation. To see a concrete example, let us consider two cloud locations c_1 and c_2 serving k client prefixes $p_1 \ldots p_k$ in two distinct geographical regions using two middle segments, m_1 and m_2. Using a drastically simplified setting where there is no noise in the measurements (i.e., the latency values are not obtained from a distribution but are fixed unknown quantities), we can express the delay measurements (d_{ij}) as linear constraints of the following form: $l_{c_i} + l_{m_i} + l_{p_j} = d_{ij}$. However, even though there are $2k$ equations on $k+4$ variables, we cannot infer the individual latency values (e.g., l_{c_i}, l_{m_i} or l_{p_j}). Instead, it is easy to show that we can only solve for the following composite expressions: $l_{c_1} + l_{m_1} - l_{c_2} - l_{m_2}$ and $l_{p_s} - l_{p_t}$ (for $s,t \in 1 \ldots k$); see footnote [4]. As a consequence, we cannot simply solve for the latency parameters (even at the coarse level of cloud, middle and client segments) with the observed latency measurements. Solving linear equations is impractical, even with "boolean" tomography [13, 15, 16] (each client/middle/cloud segment is either "good" or "bad", and a path is good only if all its segments are good).

[4] There are k equations of the form, $\left(l_{c1} + l_{m1} + l_{p_1} = d_{11}\right) \ldots \left(l_{c1} + l_{m1} + l_{p_k} = d_{1k}\right)$ and k equations likewise for $\left(l_{c2} + l_{m2} + l_{p_1} = d_{21}\right) \ldots \left(l_{c2} + l_{m2} + l_{p_k} = d_{2k}\right)$. Subtracting within the first k equations will solve for $l_{p_s} - l_{p_t}$ (for $s,t \in 1 \ldots k$). Subtracting among the first and second set of equations will solve for $l_{c_1} + l_{m_1} - l_{c_2} - l_{m_2}$.

We overcome the infeasibility of the above problem formulation using two key empirical insights. We derive them from the results of manually-labeled investigations of 88 incidents over many months. In addition, we also looked at extensive traceroutes collected from 22 `Azure` locations every ten minutes for 14 days (described in §6.5) and focused on the RTTs above the badness thresholds.

Insight-1: Typically, *only one* of the cloud, middle, or client network segments causes the inflation. For example, inflated latency between a `Azure` location and the client might be due to the middle segment or the client segment but not due to moderate increases in both. All of the manual reports support this observation. In the traceroutes, 93% of the instances of RTT inflation is due to just one of the segments contributing the dominant inflation ($\geq$ 80%) in the RTT.

Insight-2: A smaller failure "set" is more likely than a larger failure set, e.g., when all the RTTs to a cloud location are bad, it is highly likely due to the cloud segment and not due to *all* the middle segments or clients experiencing an issue simultaneously. In our dataset, over 98% of incidents support this observation. Typically, each cloud location is reached via many middle segments, each of which in turn, connects clients from many client IP-/24s.

These two insights allow us to overcome the insufficiency of measurements by investigating blame assignment sequentially *starting from the cloud* segment and stopping when we have sufficient confidence to blame *one* of the segments. Opportunistically using the passive data for coarse-segmentation is a key differentiation of `BlameIt`: it allows us to overcome the intractability faced by prior passive techniques [9, 38] while also limiting the need for active probes [22, 26, 30].

4.2 Coarse-grained Fault Localization

Recall the definition of a "quartet" from §2 that aggregates RTT values from an IP-/24 block to a cloud location within a 5 minute window, with each quartet being "good" or "bad" based on the badness thresholds. Algorithm 1 shows the pseudocode to localize the cause of the fault in each bad quartet into one of three categories: the cloud, middle, client segments; it also calls out when the data is "insufficient" or "ambiguous" to classify a bad quartet.

1) Blaming the cloud: `BlameIt` starts its blame assignment sequentially beginning with the cloud's network as the potential culprit. It takes an aggregate view of the quartets that correspond to a cloud location in the same time period but spanning a wide variety of client locations. If a considerable majority ($\geq \tau$) of the IP /24's connecting to a cloud location see bad RTT values, then `BlameIt` concludes that the cloud's network is at fault (lines 5 and 12-13 in Algorithm 1). In our deployment, we set $\tau = 0.8$ and it works well in practice.

Starting the assignment of blames from the cloud segment of the network (instead of the client) gives us more diverse RTT samples. Each cloud location (in a 5 minute window) has clients connecting to it from hundreds of thousands of /24 IP blocks across thousands of client AS'es. Bad performance across this broad spectrum makes it more likely that the cloud's network is the problem than independent problems afflicting the various clients (insight-2 in §4.1). In contrast, the client side does not have the same richness of measurements. Hence, we start blame assignment in Algorithm 1 from the cloud segment.

Algorithm 1: `BlameIt` **using passive measurements.**

```
   Input  : 1) Quartets, Q:
              ⟨IP-/24, cloud-node, time, mobile, BGP-path, RTT⟩
            2) List of cloud locations, C
            3) List of BGP-Paths, B
   Output: Assign Blame to each "bad" quartet
1  Dictionary<Segment, float> Bad-Fraction → $\phi$
2  Dictionary<Segment, int> Num-Quartets → $\phi$
   /* Populate dictionaries for cloud nodes                    */
3  foreach cloud-node c in C do
4      Num-Quartets[c] → CalcNumQuartets(c, Q)
5      Bad-Fraction[c] → CalcBadFraction(c, Q, c.expected-RTT)
   /* Populate dictionaries for middle BGP-paths               */
6  foreach BGP-path b in B do
7      Num-Quartets[m] → CalcNumQuartets(m, Q)
8      Bad-Fraction[b] → CalcBadFraction(b, Q, b.expected-RTT)
   /* Blame assignment for each quartet                        */
9  foreach quartet q in Q with q.RTT $\geq$ Threshold_RTT do
       // Customized badness threshold for RTT
10     if Num-Quartets[q.cloud-node] $\leq$ 5 then
11         q.Blame ← "insufficient"
12     else if Bad-Fraction[q.cloud-node] $\geq \tau$ then
13         q.Blame ← "cloud"
14     else if Num-Quartets[q.BGP-path] $\leq$ 5 then
15         q.Blame ← "insufficient"
16     else if Bad-Fraction[q.BGP-path] $\geq \tau$ then
17         q.Blame ← "middle"
18     else if q.IP-/24 has "good" RTT to another cloud-node then
19         q.Blame ← "ambiguous"
20     else
21         q.Blame ← "client"
```

A subtle aspect to note is that the CalcBadFraction() method in Algorithm 1 does *not* weight the quartets by the number of RTT samples contained in them. Even though different IP-/24s may have a varying number of connections (and hence, RTT samples) to a cloud location, weighting has the undesirable effect of a small handful of "good" IP-/24s with a large number of RTT samples masking the bad performance seen by many IP-/24s connecting to the same cloud location.

While bad quartets are identified using the RTT thresholds (§2), `BlameIt` also learns the typical RTT value of clients connecting to each cloud location, c.expected-RTT. It uses deviation in this expected value for blame attribution. §4.3 provides the justification and details of this learning.

2) Blaming the middle: If the fault does not lie with the cloud network, the next candidate is the middle section of the network. As defined earlier, this is all the AS'es *in between* the cloud network and the client prefix.

`BlameIt` groups all the quartets *sharing the same set of AS'es in the BGP routing path* between the client IP-/24 and the cloud location (we refer to this set of middle AS'es as "BGP path") to identify middle segment faults. If all these quartets sharing the same middle AS'es have bad RTT values, we attribute blame to the middle segment (lines 8 and 16-17 in Algorithm 1). As before, we learn the expected RTT of each middle segment (b.expected-RTT) and look for deviations.

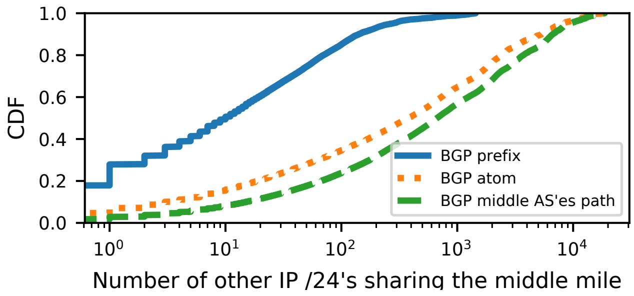


Figure 6: CDF of the number of IP /24's sharing the same "middle segment" (different definitions) within 5 minutes.

We arrived at the decision to group clients by the BGP-path after exploring multiple options. While we considered grouping by the client AS and metro area, based on prior studies [25], we notably found on analyzing `Azure`'s BGP tables that only 47% of clients in the same ⟨AS, Metro⟩ clients see a single consistent path from the `Azure` location even within a 5-minute window. Thus, we conclude that the granularity of ⟨AS, Metro⟩ is too coarse-grained.

We also considered two other subtly different options. Let the BGP tables at `Azure` contain two entries to client prefixes C1 and C2, (G – X_1 – X_2 – C1) and (G – X_1 – X_2 – C2); note that both C1 and C2 prefixes could be coarser than /24 addresses and in different AS'es. To analyze a bad quartet from C1, we could look at all the RTT values traversing either the path (X_1 – X_2 – C1) or the path (X_1 – X_2 – C), where C is the AS of the prefix C1. The former (called "BGP prefix") is fine-grained while the latter (called "BGP atom" [6]) is coarser. As Figure 6 compares, using our solution of "BGP path" provides us with more RTT samples than both these options, and hence gives us more confidence while still being accurate. Thus, we decide to group by the BGP path, i.e., all the AS'es *in between* the cloud network and the client prefix.

3) Blaming the client: Finally, we ascribe blame to the client segment of the network path (lines 20-21). However, if the client IP-/24 has connected to *other* cloud locations in the same time period but experienced good RTT performance, we label the quartet as "ambiguous" (lines 18-19) because there is not a way to conclusively assign blame.

Finally, in any step, if the number of RTT samples is too small, `BlameIt` outputs "insufficient" (lines 10-11, 14-15).

4.3 Learning RTT thresholds

Central to the cloud and middle segment blame assignments in Algorithm 1 are two RTT thresholds c.expected-RTT and b.expected-RTT. These thresholds compare against the expected RTTs of clients connecting to *each of the cloud locations* and traversing *each middle segment* of the network, and are also learned separately for mobile and non-mobile clients.

Cloud Segment c.expected-RTT. Learning the RTT values separately for each cloud location helps us identify inflations in RTT at each cloud location compared to *its own* historical values. In general, c.expected-RTT is less than the (region-specific) RTT badness thresholds. We provide a simple example to show how using c.expected-RTT instead of the badness threshold in Algorithm 1 (lines 5 and 12-13) disambiguates cloud faults. Consider the case when the RTT between a client IP-/24 and cloud location X is 55ms, with the RTT threshold being 50ms. Say, this is only due to a fault inside `Azure`, so assuming everything else is unchanged, the cloud should be blamed.

Let us assume that the RTTs of client IP-/24s connecting to X uniformly varied between [35ms, 45ms] historically, thus c.expected-RTT is learned to be 40ms. After the fault, due to X's increased contribution, the range changes to uniformly vary between [40ms, 70ms]. If we were to use the RTT threshold itself (of 50ms) in Algorithm 1, only $\frac{1}{3}$ of the quartets connecting to X will be bad (in between [40ms, 70ms]) and the blame will not fall on the cloud network (since $\tau = 0.8$). However, using c.expected-RTT = 40ms leads to the correct blame assignment as the RTTs are all above this learned value.

We use the following simple approach to learn c.expected-RTT for each cloud location. We use the median of RTT values from the last 14 days as the c.expected-RTT of connections to that location. (Using the median of historical values and $\tau = 0.8$ essentially means that we check if the distribution has shifted "left" by 30%; we picked these parameters by extensive analysis of our incidents and can vary them adaptively.) While we considered other approaches like comparing the RTT distributions, our simple approach works well in practice.

Middle Segment b.expected-RTT. We also observe considerable difference between the RTTS traversing the BGP paths (10× difference between the 10th and 90th percentile RTTs). As before, we set the m.expected-RTT thresholds for each middle segment (BGP path), learned as the median from RTT values of the past 14 days, and look for deviations.

5 FINE-GRAINED LOCALIZATION WITH ACTIVE PROBES

We build on the techniques in §4 to judiciously employ *active* measurement for fine-grained fault localization of middle segment faults. We begin by explaining the rationale behind selective active probing (§5.1), present an overview of our approach (§5.2), and then explain how we optimize both our on-demand as well as background traceroutes (§5.3 and §5.4).

5.1 Rationale for selective active probing

Algorithm 1 blames the incidence of high RTT to one of three network segments: cloud, middle, or client. While attributing blame to the cloud or client segment automatically identifies the specific AS responsible for the latency degradation, attributing blame to the middle segment of the network could be due to any of the AS'es in the BGP path between the cloud and client could be causing the degradation.

We considered extending the approach of "hierarchical elimination" of Algorithm 1 to the AS-level graph, with the blame being attributed to the AS common to the paths of the majority of connections that are experiencing high RTT. In practice, however, the data density is insufficient for such AS-level analysis. While we considered coarsening the definition of a quartet (e.g., larger client IP aggregates than /24's or wider timer buckets than 5 minutes), we decided against it as it would hurt the accuracy of the fault localization.

Active measurements – issuing continuous traceroutes from the different `Azure` locations to client locations – is a natural approach to localize faults. [22, 26, 30] However, for accurate localization of the middle AS, we need continuous traceroutes so that we have data to compare *before and after an incident*. As our calculations in §6.5 will show, that works out to nearly 200 million per day, an amount that is prohibitive and infeasible in Azure's production deployments. Besides the overhead, such volumes could also trigger security alarms in the various AS'es since the traceroute packets and their ICMP responses could be mistaken for probing attacks [24].

Due to routing asymmetries [19], the "forward" (cloud-to-client) and "reverse" (client-to-cloud) Internet paths can be different. Our current solution only uses traceroutes issued from the cloud locations (for ease of deployment) but we believe that reverse traceroute techniques [20] can be incorporated into BlameIt's active phase. Azure already has many users with rich clients [8] that can be coordinated to issue traceroutes to measure the client-to-cloud paths.

5.2 Approach for fine-grained localization

Our approach for fine-grained fault localization is as follows: judiciously issue "on-demand" traceroutes based the passive analysis of RTTs (§4), and then compare these measurements with the *baseline* established through infrequent "background" traceroutes to obtain the picture prior to the fault. We illustrate how these traceroutes are compared using an simple example.

Illustrative Example: Consider the following example, modeled on a real-world investigation in India, on how the background and on-demand traceroutes are compared. The AS-level path between cloud X and client c is X - m1 - m2 - c. The RTT observed by the background traceroutes from X to the *last* hop in each of these four AS'es was 4ms, 6ms, 8ms and 9ms, respectively. However, after the latency degradation, the corresponding values became 4ms, 60ms, 62ms and 64ms. Specifically, the individual contribution of the middle AS m1 went up from $(6-4)$ = 2ms to $(60-4)$ = 56ms, thus pointing to m1 as the reason behind the performance degradation. [5]

There are two key challenges to realizing the above approach at scale. 1) First, we have to be judicious about when to issue "on-demand" traceroutes since fine-grained localization might be of little interest for issues that are ephemeral and/or only affect a small number of clients. 2) Second, background measurements have to be optimized so that they are not wasteful or cause a significant increase in network traffic. We next present BlameIt's solution to both issues – prioritized on-demand measurements (§5.3) and optimized background measurements (§5.4).

5.3 Prioritized On-Demand Measurements

BlameIt prioritizes on-demand traceroutes for latency degradations that, (1) have lasted for a long duration, (2) are expected to persist for longer durations, and (3) would potentially impact many clients. Prioritization helps BlameIt focus on issues of most impact to clients while avoiding the high probing overheads of prior solutions [22, 26, 30]. As §2.3 and §2.4 showed, there is considerable variation across the middle-segment issues. They may impact just 10 clients or all the way up to 4 million client IPs. While some issues are ephemeral, others can last multiple hours.

Client-time Product: To prioritize the middle segment issues, we use "client-time product" as our metric. Client-time product is simply the product of (a) the predicted duration of bad performance of the middle segment, multiplied by, (b) the number of clients that are expected to access the cloud via that middle segment in that period (and hence, will be impacted). This is akin to the metric in §2.4 but with estimated values.

Budgeted Prioritization: BlameIt operates under a "budget" for traceroutes defined in the number of traceroutes permissible out of each of Azure's cloud locations per time window (say, every 5 minutes or every day). For simplicity, we avoid setting budgets per AS and instead employ a larger budget for traceroutes emanating from a cloud location.

Our prioritization takes all the quartets with middle segment issue (from Algorithm 1), and groups them by their AS-level BGP path (m1 - m2 in §5.2's example). This grouped set P contains all the paths with at least one problematic AS. For each path p in P, we *predict* the two metrics ((a) and (b)) that are needed to calculate the client-time product. We sort paths in descending order of client-time product and issue traceroutes within the budget (one per middle segment issue). We next explain how the two two metrics are estimated.

(a) Predicting the duration of degradation. BlameIt estimates the probability of a problem persisting *given that* the problem has lasted thus far. Specifically, if an issue has lasted for a duration t, it estimates the probability of it lasting for an *additional* duration T, $\mathrm{P}(T|t)$. Using estimates of $\mathrm{P}(T|t)$ for different values of T, we can calculate the expected duration, $\Sigma_{T=1}^{T_{\max}} P\left(T|t\right) * T$, for which the issue will persist. We increase T in increments of 5 minutes. We obtain $\mathrm{P}(T|t)$ for various T values based on historical fault durations in *each* BGP path. Given the long-tailed distribution of problem durations (§2.3) we only need to separate out the long-lived problems rather than precisely estimate the duration of each problem.

(b) Predicting the number of impacted clients. BlameIt uses historical data to also predict the number of clients that are likely to connect (and hence, be impacted) during an issue. We empirically observe that predicting the number of *active* clients in a BGP-path using the same 5-minute time window in the previous days is more accurate than using recent history (e.g., a few prior time windows in the same day). Hence, we use the average number of clients that connected via the same middle BGP-path in the same time window in the past 3 days.

The above approaches work well in practice with our prioritization of traceroutes closely matching an oracle (§6.5).

5.4 Optimized Background Measurements

Recall from the example in §5.2 that BlameIt uses traceroutes from *before* an incident of RTT degradation to localize the faulty AS. These "background" traceroutes are in addition to those issued for Algorithm 1's middle segment faults. Background traceroutes are issued *periodically* to each BGP path as well as *triggered* based on BGP path change at Azure's border routers.

To keep the overhead manageable, the periodic traceroutes are performed infrequently, two times a day to each BGP path from each Azure location. As we report in §6.5, the relative stability of Internet paths in normal times means that the above low frequency is a good "sweet spot" between traceroute overheads and accuracy of fault localization.

[5] While the latency measurements to a later hop in traceroutes could be smaller than to an earlier hop [5, 28, 35], this issue is less prevalent when we look at latencies at the level of AS'es.

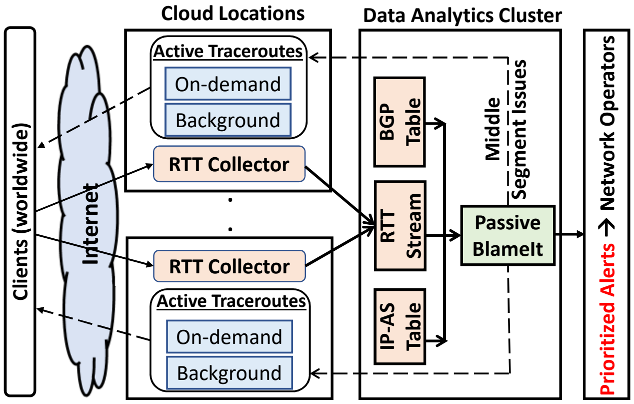


Figure 7: `BlameIt`'s key components in production.

In addition, we use information from `Azure`'s BGP listener, which connects to all of the border routers over IBGP, to determine if the AS level path to a client prefix has changed at a border router or a route has been withdrawn. In either case, we issue a traceroute to that client prefix from the cloud location which connects to the border router in question. Favorably for us, analysis of `BlameIt`'s BGP messages show that nearly two-thirds of the BGP paths at the routers do not see any churn in an entire day, thus limiting our overheads.

The combination of periodic (but infrequent) traceroutes and traceroutes triggered by BGP churn enables `BlameIt` to maintain an accurate picture of paths from the cloud to clients in various locations with low overhead.

6 EVALUATION

`BlameIt` is in production deployment at `Azure` (§6.1 and §6.2); here are the highlights.

(1) `BlameIt`'s fault localization matches the results from manual investigations in 88 real-world incidents. §6.3
(2) As further validation, we also use large-scale traceroutes to corroborate `BlameIt`'s accuracy. §6.4
(3) Active probing with `BlameIt` prioritizes the right middle-segment issues and is 72× cheaper. §6.5

6.1 Implementation and Deployment Details

`BlameIt` is in production deployment, its passive component since Nov 2017 and its active component beginning to get widely rolled out. Figure 7 shows all its relevant components. RTTs (from TCP handshakes) are continuously collected as clients connect to `Azure` and are aggregated at an analytics cluster where the `BlameIt` script runs periodically. Its outputs trigger prioritized alerts to operators, and targeted traceroutes to clients.

RTT Collector Stream: `Azure`'s cloud locations generate two data streams, one containing the client's IP and the other the RTT (in addition to many other fields). These two streams had to be joined (via a "request id") for `BlameIt`'s Algorithm 1 to be executed. Since these streams were large in size, their joining happened only once every day to limit the load on the data analytics cluster. As part of `BlameIt`'s deployment, we modified the RTT stream to also include the client IP field, thus not needing to wait for the joining.

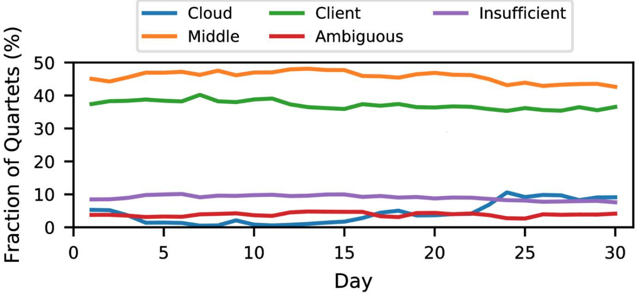


Figure 8: Blame fractions in a one-month period.

Periodic execution: `BlameIt`'s Algorithm 1 is scheduled to execute every 15 minutes (which is sufficient given the long-tailed distribution we saw in §2.3). We encountered one tricky issue in running `BlameIt` every 15 minutes. Every hour, a few hundred "storage buckets" are created afresh and each RTT tuple is written into a *randomly* chosen bucket. This leads to a loss of temporal ordering *within the hour*, so each run of `BlameIt`, even though it needs only the last 15 minutes of data, has to read all the buckets filled thus far in that hour (and filter out RTTs). We are currently working on creating finer buckets.

Active traceroutes: `BlameIt`'s outputs are used to prioritize middle segment issues and issue traceroutes to the clients. This module periodically fetches the destinations from the analytics cluster to issue on-demand traceroutes (§5.3) . Finally, it also issues background traceroutes (§5.4) at a regular cadence. For issuing traceroutes, we use the native Windows tracert command.

Network Operators: `BlameIt` helps prioritize the issues based on their business impact and the top few are automatically "ticketed" for investigation. The detailed outputs of `BlameIt` are auto-included in these tickets for ease of investigation. `BlameIt`'s coarse segmentation of the issues helps route the tickets to the appropriate teams to investigate server issues, networking issues, peering relationships, etc. Crucially, they only investigate high-impact and relevant issues.

6.2 Blame Assignments in Production

Each day, `BlameIt` assigns blames for millions of "bad" quartets whose RTT are above the badness thresholds. We present a flavor of the results of `BlameIt`'s Algorithm 1 – the blame fractions and durations of badness.

Figure 8 plots the fraction of blame categories worldwide for a one month period. We notice a general stability of the fractions accounted by each blame category, with middle segment issues slightly higher than client issues. Even though cloud segment issues generally account for less than 4% of bad quartets, these are investigated with high priority. Their increase around day-24 in Figure 8 is due to a scheduled maintenance.

Figure 9 takes one of the days and splits the blame fractions by cloud regions. One notable aspect is that middle segment issues dominate in India, China and Brazil. This is likely due to the still-evolving transit networks in these regions, compared to relatively mature regions like the US. Another aspect we notice is that the "insufficient" and "ambiguous" categories constitute a high fraction, but this presents an interesting quandary: we can relax our minimum RTT samples and use coarser grouping for middle prefixes (like AS,

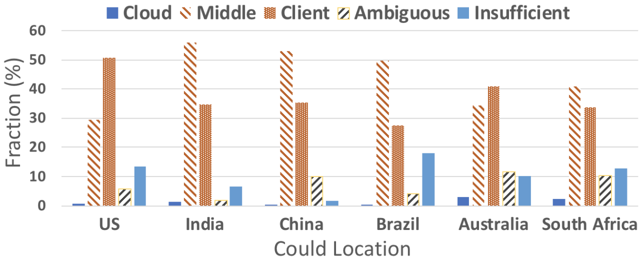
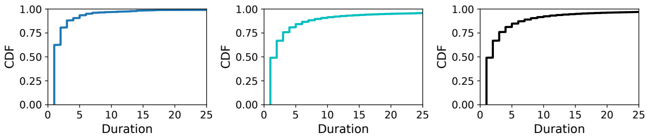


Figure 9: Blame fractions for one day in six cloud locations. (In each location, the blame fractions sum to 100%**.)**



(a) Cloud **(b) Middle** **(c) Client**

Figure 10: Duration of cloud, middle, client segment issues in the units of consecutive 5-min time buckets.

Metro) to reduce these categories, but it would likely come at the expense of accuracy of our fault localization. We could also issue traceroutes to the clients in these two categories, but that would increase the overheads of active probing.

Finally, how long do the different badness incidents last? Back in §2.3 we had observed a long-tailed distribution of all badness incidents, and Figure 10 breaks their durations by blame categories; similar distributions persist across the categories. Cloud issues generally last for lesser durations than middle or client segment issues, possibly explained by Azure dedicating a team to fix them at the earliest.

6.3 Real-world Case Studies

We analyzed investigation reports from 88 incidents of latency degradation in production that were investigated by Azure's network administrators. As part of their investigations, they look at performance logs, network captures, as well as communicate with administrators in other AS'es. Their reports document the cause behind the degradation and identify the faulty AS. An encouraging result is that BlameIt's localization of the faulty AS matched the conclusions of the network administrators for *all* the incidents. We now discuss a few in detail to present a flavor of the faults, including some where BlameIt helped with the investigation.

1) Maintenance in Brazil: Azure's cloud network in one of the locations in Brazil had internal routing issues due to an unfinished maintenance operation that considerably increased the latency of the southern American clients that were connecting to the location. Since the incomplete maintenance was not detected, clients were not rerouted to a different location and investigations focused on other segments of the network including peering AS'es. The issue was finally fixed after a couple days. This incident was before the deployment of BlameIt and *post facto* analysis of the RTT logs shows the correct localization of the fault on the cloud segment.

2) Peering fault: There was a high-priority issue where many customers of Azure's services experienced high latencies. This was a widespread issue affecting many clients in the USA in the east coast, west coast, and central regions. BlameIt correctly identified it to be a "middle segment" issue, thus avoiding the unnecessary involvement of Azure's internal-networking teams. Finer localization revealed that the issue was due to changes inside a peering AS, with which BlameIt peers at multiple locations.

This incident provided an interesting comparison with other monitoring systems. One system was based on periodic traceroutes *from a small fraction* of Azure's clients, but these clients happened to not be impacted much by this issue and hence did not detect the problem (in other words, lack of coverage of client "vantage points", a problem that BlameIt does not face). Another system made web users download a small web object to measure the latency, but these active measurement were conservatively deployed to limit overheads (since they were not triggered in a targeted manner, like BlameIt does using analysis of passive data). Finally, the system to monitor RTTs in each AS, metro also did not raise an alarm because no single AS, metro was excessively affected even though there were many affected clients countrywide (speaks to the value of BlameIt's fine-grained analysis using BGP-paths and client IP-/24s). Combining large-scale aggregated measurements, we can detect and localize even slight but widespread increases in latency.

3) Cloud overload in Australia: Recently clients of Azure connecting to a cloud location in Australia experienced higher RTTs than the RTT targets. During this incident the median RTT went up from 25ms usually to 82ms. An interesting aspect in the investigation of this incident is that many clients sharing the BGP paths to reach this specific location (in Australia) saw increases in their RTTs. However, BlameIt's approach of starting the blame assignment from the cloud (Algorithm 1) ensured that the blame was correctly pinpointed to the cloud segment (and not the middle BGP paths). As a validation, even though the same BGP paths were also used to connect to other nearby cloud locations in Australia, those clients did not experience bad RTTs (Insight-2 in §4.1). Investigations tracked the issue to an increase in CPU usage of the servers (overload) that was leading to the spike in RTTs.

4) Traffic shift from East Asia to US West coast: Due to some unforeseen side-effects of changes in BGP announcements, Azure clients in east Asia were starting to get routed to Azure's locations in the US west coast instead of the locations in east Asia. This substantially increased their latencies and BlameIt blamed the middle segment. The ISPs of east Asian clients did not have good peers and connections to route them to the US west coast, since traffic rarely gets routed that direction, and thus, the middle segments contributed to the substantial increase. The issue was fixed with the east Asian clients being redirected back

5) Client ISP issues in Italy: The median RTT of Azure users in a major city in Italy increased from the usual value of 9ms to 161ms. Given the substantial increase in RTT, the persistence of the issue, and the high number of users that were affected, an investigation was launched that concluded that the increase was due to a maintenance inside the client ISP, for which no advance notice was provided.

This incident was prior to BlameIt's deployment, but our analysis shows that BlameIt would have blamed the client AS with a high confidence of 93% (confidence is obtained by calculating the proportion of quartets blamed in each category of Algorithm 1). It

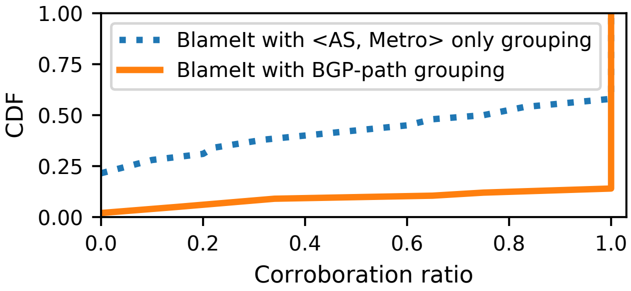


Figure 11: Large scale traceroute based validation. Corroboration ratios of `BlameIt` to clients in $1,000$ BGP paths.

would have avoided this wasted effort as there was little that `Azure` could do in fixing the issue.

6.4 Large-scale Corroboration

The case studies above are encouraging and notes by network operators help us see the value provided by `BlameIt` in their diagnoses. However, we also corroborate the results of `BlameIt` at a larger scale by comparing its results with those from continuous traceroutes. We continuously issue traceroutes (every minute) from different `Azure` locations to clients in $1,000$ select BGP paths. We treat the latency contributions (of each AS) from the traceroutes as the "ground truth". When latency goes beyond the targets, we compare each AS'es "normal" contribution to the end-to-end latency with the contributions *just after* the incident. We call the AS with the most increase in its contribution as the culprit AS. We compare this result to `BlameIt`'s identification of the culprit AS. Given the overhead of such large-scale traceroutes, we had to restrict this experiment to only three `Azure` locations in the US east coast for one day.

For each BGP path, we measure its *corroboration ratio*, defined as the fraction of latency issues where `BlameIt`'s diagnosis (of the culprit AS) matched the traceroutes. Note that this can include any of the AS'es in the middle or the client AS or `Azure`; recall from §4 that the blame is ascribed to only one AS. Figure 11 plots the CDF of corroboration ratios of the $1,000$ BGP paths. We observe near-perfect corroboration (ratio of 1.0) for nearly 88% of the paths (orange line), thus showing that despite the two-level approach used by `BlameIt`, it does not lead to loss in accuracy. The figure also vindicates our decision to use BGP paths to group clients in the same middle segment as opposed to the traditional practice of grouping them by $\langle$AS, Metro$\rangle$ [25], which significantly lowers the corroboration ratio (blue line).

To reiterate, `BlameIt`'s high accuracy is without the high overhead of continuous traceroutes (which we deployed only for obtaining the ground truth for large-scale corroboration). We next analyze the aspects of our selective probing from §5.

6.5 Active Probes for Middle Segment Issues

We dig deeper into `BlameIt`'s active traceroutes for fine-grained localization of middle segment issues.

For extensive data-driven evaluations, we issued traceroutes from 22 `Azure` cloud locations to $85,100$ client IP /24's in $23,000$ BGP paths, once every 10 minutes for a period of 14 days. As in §6.4, we use the traceroutes as ground truth for contributions by each AS.

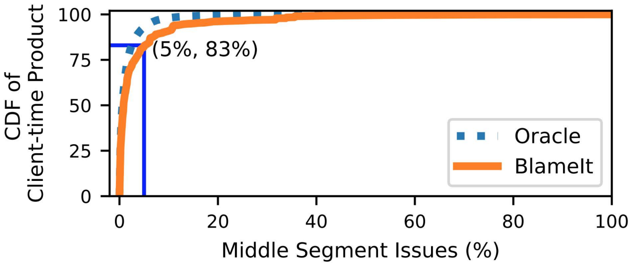


Figure 12: CDF of client-time product of middle segment issues ranked by the Oracle.

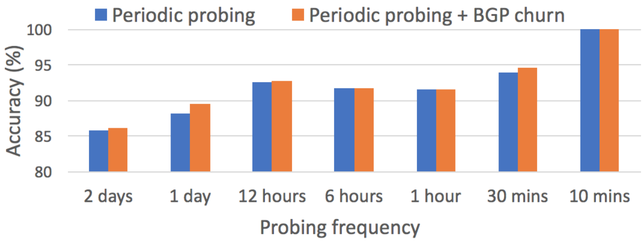


Figure 13: Accuracy of active `BlameIt` under different probing frequencies with/without probes triggered by BGP churn.

We identify middle segment issues, and identify the faulty AS in the middle by comparing against its historical contributions to the end-to-end latency.

Client-time product: Recall from §5.3 that `BlameIt` prioritizes on-demand traceroutes for middle segment issues based on an estimate of their client-time product. We had settled on simple approaches for our estimations and Figure 12 illustrates that as a result of our accurate estimates, we are able to prioritize the traceroutes as good as an oracle; Figure 12 sorts the middle segment issues (on the x-axis) by their client-time product as calculated by the oracle. Note that 5% of the middle segment issues cover over 83% of the cumulative client-time product impact. Thus, a 5% budget would suffice for diagnosing the high-impact middle segment issues.

Background probing frequency vs. Accuracy: There is an intrinsic trade-off between the frequency of background traceroutes (§5.4) and the accuracy of the fault localization. Too fine a frequency increases the accuracy but at the expense of overheads. Sending a probe every ten minutes to cover all the BGP paths achieves high accuracy, but to cover all the BGP paths seen across all the `Azure` locations, this amounts to nearly 200 million traceroutes out of `Azure` each day, an unacceptably high overhead. Figure 13 plots how the accuracy drops at lower probing frequencies. We notice a "sweet spot": with a frequency of once per 12 hours (along with BGP churn triggered probes), we still obtain an accuracy of 93%. This frequency represents a traceroute overhead that is 72× lower, and this is feasible for `Azure`.

We also compare `BlameIt` to Trinocular [30] that diagnoses WAN unreachability (not latency inflation) using an optimized network model to trigger active probes. Compared to Trinocular, `BlameIt` issues 20× fewer active probes.

7 RELATED WORK

We briefly survey several strands of work related to BlameIt.

Diagnosis by passive/active measurements: Diagnosing anomalies in network performance has been studied extensively (e.g., [8, 21, 23, 27]); see Table 1. Network tomography based solutions [3, 9, 20] passively deconstruct end-to-end performance, but run into intractable formulations at scale. The passive diagnosis approach in BlameIt is closest to NetProfiler [29]. However, BlameIt operates at much larger scale and its selective active probing triggered by passive analyses.

Other works [12, 17, 22] combine passive measurements and active measurements to identify problems, while Odin [8] probes randomly selected clients to maintain visibility to alternate network paths. But these works do not use passive data to minimize the active probes, do not trigger probes during a latency degradation (are essentially “offline”), and waste many probes due to lack of impact prioritization.

Many works make optimal use of a measurement budget for probing, e.g., Trinocular [30], Sibyl [11], iPlane [26]. However, BlameIt is distinguished from this prior work in its approach to predict the *impact* of latency degradation (e.g., how many clients would likely be affected) in deciding which problems to investigate. Finally, solutions that rely on rich information (e.g., TCP logs) collected at end-hosts for diagnosis [4, 31] are feasible in datacenters but harder to deploy in the inter-domain wide-area.

Measurement of WAN latency: WAN latencies have been studied from the viewpoint of the cloud networks. The most relevant studies are those on network path inflation (e.g., [33, 34]), alternate paths (e.g., [22]), deployment of CDN servers (e.g., [7]), IXP performance (e.g., [2]), application-level overlay paths (e.g., [18]). While these studies provide valuable insights on WAN performance in the wild, such offline analysis is not an integrated part in the online operations for fault diagnosis. Partly inspired by the prior measurement studies, BlameIt seeks to understand the typical latency degradations from the viewpoint of Azure, and more importantly, it provides a measurement-based solution to automate the process of root-causing the degradations.

Others seek to obtain a comprehensive view of the network conditions by redirecting existing traffic flows (e.g., Edge Fabric [32], Espresso [37]) or by performing active measurements (e.g., Entact [39]). Although network performance is explicitly measured in these schemes, such measurements are purely end-to-end.

CDN traffic engineering: There has been substantial work on server selection in CDNs, with a range of approaches, including IP anycast [8], DNS-based redirection (including EDNS extensions to help better identify the client [10]), or anycast-based DNS server selection with co-located proxies [14]. Anycast-based selection, in particular, only has a loose connection to network performance since it depends on BGP routing to direct the client to a server. CDN performance can be significantly improved by choosing the network path and ingress point a client should be directed to as a function of path performance (e.g., [8, 32, 36, 37, 39]). BlameIt is complementary to these techniques, as it can serve as a common underlay that provides insights on network performance for CDN traffic engineering.

8 CONCLUSION

We presented BlameIt, a tool that automatically localizes the faulty AS when there is latency degradation between clients and cloud locations, using a combination of analysis of passive measurements (TCP handshake RTTs) and selective active measurements (traceroutes). Such a tool is highly valuable for global cloud providers like Azure. BlameIt smartly leverages the passively collected measurements and a small amount of active probes for its fault localization. In doing so, BlameIt avoids the problems of intractability that stifled prior tomography based solutions and prohibitively high overhead that plagued probing solutions based on global vantage-points. BlameIt is in production deployment at Azure and produces results with high accuracy at low overheads.

Ethics Statement: This work does not raise ethical issues.

ACKNOWLEDGMENTS

We would like to thank the anonymous SIGCOMM reviewers and our shepherd Theophilus Benson for their valuable feedback. We also would like to call out the invaluable support of the Azure Front Door team members, especially Minerva Chen and Madhura Phadke, for their role in testing and deploying BlameIt. Sorabh Gandhi provided feedback in the early stages of the work.

REFERENCES

[1] Google Video Quality Report. https://support.google.com/youtube/answer/6013340?hl=en.

[2] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger. Anatomy of a large european ixp. *ACM SIGCOMM Computer Communication Review*, 42(4):163–174, 2012.

[3] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. H. Liu, J. Padhye, B. T. Loo, and G. Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, Renton, WA, 2018. USENIX Association.

[4] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 440–453. ACM, 2016.

[5] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 153–158. ACM, 2006.

[6] A. Broido and k. claffy. Analysis of RouteViews BGP data: policy atoms. In *Network Resource Data Management Workshop*, Santa Barbara, CA, May 2001.

[7] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan. Mapping the expansion of google’s serving infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 313–326. ACM, 2013.

[8] M. Calder, R. Gao, M. Schröder, R. Stewart, J. Padhye, R. Mahajan, G. Ananthanarayanan, and E. Katz-Bassett. Odin: Microsoft’s scalable fault-tolerant CDN measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Renton, WA, 2018. USENIX Association.

[9] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: Recent developments. *Statistical science*, pages 499–517, 2004.

[10] F. Chen, R. K. Sitaraman, and M. Torres. End-user mapping: Next generation request routing for content delivery. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 167–181. ACM, 2015.

[11] Í. Cunha, P. Marchetta, M. Calder, Y.-C. Chiu, B. Schlinker, B. V. Machado, A. Pescapè, V. Giotsas, H. V. Madhyastha, and E. Katz-Bassett. Sibyl: A practical internet route oracle. In *NSDI*, pages 325–344, 2016.

[12] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. Luckie, R. K. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 1–15. ACM, 2018.

[13] N. Duffield. Network tomography of binary network performance characteristics. *IEEE Transactions on Information Theory*, 52(12):5373–5388, 2006.

[14] A. Flavel, P. Mani, D. A. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. *connections*, 27:19, 2015.

[15] D. Ghita, K. Argyraki, and P. Thiran. Network tomography on correlated links. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*,

pages 225–238. ACM, 2010.

[16] D. Ghita, C. Karakus, K. Argyraki, and P. Thiran. Shifting network tomography toward a practical goal. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 24:1–24:12, New York, NY, USA, 2011. ACM.

[17] V. Giotsas, C. Dietzel, G. Smaragdakis, A. Feldmann, A. Berger, and E. Aben. Detecting peering infrastructure outages in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 446–459. ACM, 2017.

[18] O. Haq, M. Raja, and F. R. Dogar. Measuring and improving the reliability of wide-area cloud paths. In *Proceedings of the 26th International Conference on World Wide Web*, pages 253–262. International World Wide Web Conferences Steering Committee, 2017.

[19] Y. He, M. Faloutsos, S. Krishnamurthy, and B. Huffaker. On routing asymmetry in the internet. In *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.*, volume 2, pages 6–pp. IEEE, 2005.

[20] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, et al. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 286–299. ACM, 2016.

[21] P. Kanuparthy and C. Dovrolis. Pythia: Diagnosing performance problems in wide area providers. In *USENIX Annual Technical Conference*, pages 371–382, 2014.

[22] R. Krishnan, H. V. Madhyastha, S. Jain, S. Srinivasan, A. Krishnamurthy, T. Anderson, and J. Gao. Moving beyond end-to-end path information to optimize cdn performance. In *Internet Measurement Conference (IMC)*, pages 190–201, Chicago, IL, 2009.

[23] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[24] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic. Distributed denial of service attacks. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 2275–2280. IEEE, 2000.

[25] Y. Lee and N. Spring. Identifying and aggregating homogeneous ipv4 /24 blocks with hobbit. In *Internet Measurement Conference (IMC)*, Santa Monica, CA, 2016.

[26] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 367–380. USENIX Association, 2006.

[27] A. A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards automated performance diagnosis in a large iptv network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 231–242. ACM, 2009.

[28] M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an accurate as-level traceroute tool. In *ACM SIGCOMM*, 2003.

[29] V. N. Padmanabhan, S. Ramabhadran, and J. Padhye. Netprofiler: Profiling wide-area networks using peer cooperation. In *International Workshop on Peer-to-Peer Systems*, pages 80–92. Springer, 2005.

[30] L. Quan, J. Heidemann, and Y. Pradkin. Trinocular: Understanding internet reliability through adaptive probing. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 255–266. ACM, 2013.

[31] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren. Passive realtime datacenter fault detection and localization. In *NSDI*, pages 595–612, 2017.

[32] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 418–431. ACM, 2017.

[33] A. Singla, B. Chandrasekaran, P. Godfrey, and B. Maggs. The internet at the speed of light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 1. ACM, 2014.

[34] N. Spring, R. Mahajan, and T. Anderson. The causes of path inflation. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 113–124. ACM, 2003.

[35] R. Steenbergen. A practical guide to (correctly) a practical guide to (correctly) troubleshooting with traceroute. In *NANOG*, 2017.

[36] V. Valancius, B. Ravi, N. Feamster, and A. C. Snoeren. Quantifying the benefits of joint content and network routing. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 243–254. ACM, 2013.

[37] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445. ACM, 2017.

[38] M. Zhang, C. Zhang, V. S. Pai, L. L. Peterson, and R. Y. Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, volume 4, pages 12–12, 2004.

[39] Z. Zhang, M. Zhang, A. G. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, pages 33–48, 2010.