Scalable Near Real-Time Failure Localization of Data Center Networks

Herodotos Herodotou
Microsoft Research
herohero@microsoft.com

Bolin Ding
Microsoft Research
bolin.ding@microsoft.com

Shobana Balakrishnan
Microsoft Research
shobanab@microsoft.com

Geoff Outhred
Microsoft
geoffo@microsoft.com

Percy Fitter
Microsoft
percyf@microsoft.com

ABSTRACT

Large-scale data center networks are complex—comprising several thousand network devices and several hundred thousand links—and form the critical infrastructure upon which all higher-level services depend on. Despite the built-in redundancy in data center networks, performance issues and device or link failures in the network can lead to user-perceived service interruptions. Therefore, determining and localizing user-impacting availability and performance issues in the network in near real time is crucial. Traditionally, both passive and active monitoring approaches have been used for failure localization. However, data from passive monitoring is often too noisy and does not effectively capture silent or gray failures, whereas active monitoring is potent in detecting faults but limited in its ability to isolate the exact fault location depending on its scale and granularity.

Our key idea is to use statistical data mining techniques on large-scale active monitoring data to determine a ranked list of suspect causes, which we refine with passive monitoring signals. In particular, we compute a failure probability for devices and links in near real time using data from active monitoring, and look for statistically significant increases in the failure probability. We also correlate the probabilistic output with other failure signals from passive monitoring to increase the confidence of the probabilistic analysis. We have implemented our approach in the Windows Azure production environment and have validated its effectiveness in terms of localization accuracy, precision, and time to localization using known network incidents over the past three months. The correlated ranked list of devices and links is surfaced as a report that is used by network operators to investigate current issues and identify probable root causes.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management; Network monitoring*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD'14, August 24–27, 2014, New York, NY, USA.
Copyright 2014 ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623365.

General Terms

Algorithms; Management

Keywords

Failure localization; Data center networks

1. INTRODUCTION

Communications in data centers today are based on large-scale complex networks that concurrently support a variety of distinct services (e.g., search, email, cloud storage and compute). As an example, the Windows Azure network consists of over 7 thousand network devices (e.g., routers) and over 220 thousand links, which connect together all the servers in all Windows Azure data centers worldwide. All Windows Azure Services [14] depend on a healthy network for providing highly-available and scalable offerings. Despite the built-in redundancy in data center networks, performance issues or failures in the network can lead to *user-perceived* service interruptions. Therefore, determining and localizing user-impacting availability and performance issues in the network in near real time is crucial.

Localizing network faults is done by knowledgeable network operations teams who work with associated on-call engineers to resolve problems in real time with the help of monitoring data. Such an approach can be time consuming, tedious, and is further exacerbated by monitoring noise and the increasing size of the network. As the scale of the network grows, automated fault localization becomes increasingly important since it can reduce mean-time-to-recovery and service disruption.

Network monitoring is at the heart of failure localization and is divided into two categories: *passive* and *active* monitoring. The passive approach typically involves polling the network devices periodically to collect various telemetry data about their health and the traffic that passes by. The system will then analyze the local telemetry data and raise availability and performance alerts at the level of individual devices and links when it detects any abnormalities [4, 5, 13]. While these alerts are often useful in localizing failures, they are also noisy and not a direct signal for user-perceived failures[1], which turns troubleshooting a particular

[1]User-perceived network failures refer to network failures that have a direct effect on user traffic, where users are primarily the services running on top of the network.

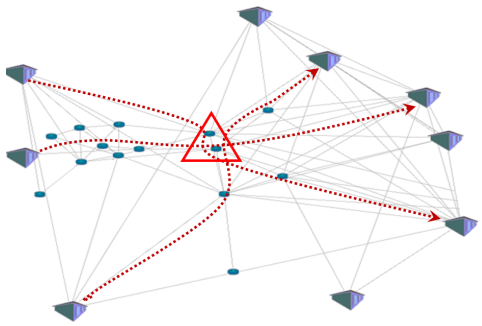

Figure 1: Failure localization approach: The successful, failed, and delayed ping data are overlaid on top of the network topology, and a statistical data mining approach is used to triangulate (i.e., localize) issues in the network.

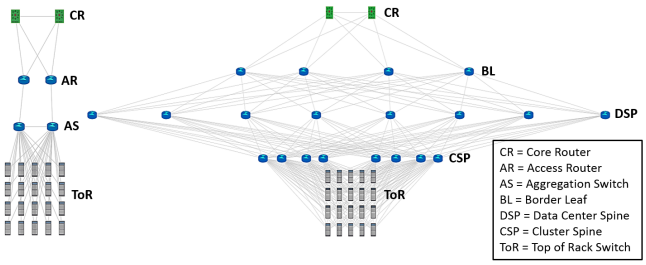

Figure 2: Common network topologies for data centers.

network issue into a time-consuming process that could take anywhere from minutes, to hours, to even days to resolve.

The active approach relies on the capability to inject test traffic into the network and to monitor the flow of that traffic. In particular, data centers typically run a *ping service* that generates large amounts of pings between pairs of end hosts in the network and acts as a proxy for user-perceived network availability and latency [15, 16]. In this setting, ping failures provide a strong signal that there is indeed an issue in the underlying network and, more specifically, that there is an issue somewhere along the path from the source to the destination of the ping. However, pings do not pinpoint the exact device or link that has caused the pings to fail since actual network routes are typically unknown.

Even though a single ping failure cannot help in identifying the culprit, the conjecture is that the combination of ping data between multiple sources and destinations will allow us to triangulate the location of an existing issue. Figure 1 provides a simple visual representation of the proposed approach, which uses statistical data mining techniques to localize user-perceived network failures based on ping data. Specifically, we (i) introduce a probabilistic model for describing the ping routes and failures, (ii) formulate the failure localization problem as a fitting problem to compute failure probabilities for devices and links in near real time, and (iii) detect statistically significant increases in the computed failure probabilities.

Our algorithm produces a short ranked list of devices and links that best explain the observed ping data. Finally, we correlate this list with passive monitoring signals, such as device and link alerts, in order to increase the confidence of the probabilistic analysis. The correlated ranked list of devices and links forms a starting point for investigating current issues in the network and can drastically reduce the time it takes for network operators to identify probable root causes and resolve network issues. Our experimental evaluation on real network incidents validates both the effectiveness and efficiency of our approach to localize network failures. In summary, our key contribution is the novel and successful application of statistical data mining techniques to localize user-impacting availability and performance issues in a worldwide data center network in near real time.

The rest of the paper is organized as follows. Section 2 presents background information for data center networks. Section 3 provides an overview of our approach, while the details are described in Section 4. The implementation and deployment details are discussed in Section 5 and the experimental evaluation is presented in Section 6. Finally, Section 7 discusses related work and Section 8 concludes the paper.

2. DATA CENTER NETWORKS BASICS

Data centers hosting Internet-scale services comprise of servers, direct attached storage, and networking infrastructure. Over the recent years, data center networks have been on the path to commoditization [1]. For example, [7] demonstrates the performance and scale that can be achieved from a network built using low-cost Ethernet switches arranged into a Clos topology that provides multiple paths between servers. The overall network consists of a densely connected mesh network within each data center as well as a core network spanning the geographies. With mega-scale data centers, the size of the network within each data center can be large connecting over 50-100 thousand servers.

Given the types of workloads that run in these data centers, the authors in [3] present the need to treat the data center as one massive warehouse-scale computer and make the appropriate design, operation, and cost tradeoffs that best support the workloads. In this paper, we have only looked at a single aspect—namely the operational aspects of the network—but from a holistic perspective of end-to-end user-perceived network performance and at the scale of these large data center networks.

2.1 Network Topology

Modern data center networks use hierarchical architectures reaching from a layer of servers in racks at the bottom to a layer of core routers at the top [1, 7]. We consider two common architectures within data centers, shown in Figure 2. In both cases, there are typically 20 to 40 servers per rack, each singly connected to a Top of Rack (ToR) switch. The data center backbone in the first topology consists of a layer of Aggregation switches (AS), whereas in the second topology it consists of two layers, namely the Cluster Spines (CSPs) and the Data Center Spines (DSPs). Here, each ToR is connected to a layer of CSPs, which in turn is connected to a layer of DSPs forming two bipartite graphs. The ASs and DSPs are then connected to the core of the network via Access Routers (ARs) and Border Leafs (BLs), respectively. Finally, ARs and BLs are connected to Core Routers (CRs).

The core network connects all data centers together forming an asymmetric wide-area network. It consists of core routers in the data centers as well as intermediate forwarding devices between the data centers. Figure 1 shows the core network between multiple data centers and forwarding stations. For this paper, we only consider the network topol-

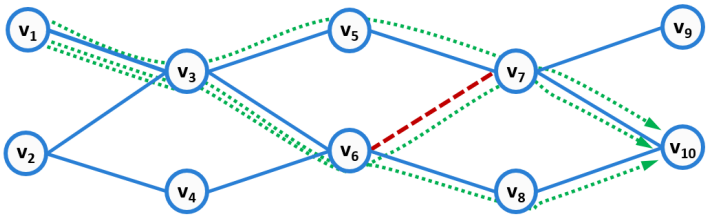

Figure 3: Example network topology. The dotted green arrows show the three possible paths between v_1 and v_{10}. The dashed red edge (v_6, v_7) represents a faulty link.

Table 1: Example ping data that lists ping successes and failures between pairs of vertices in the graph.

Source Vertex	Destination Vertex	Ping Successes	Ping Failures
v_1	v_9	45	5
v_1	v_{10}	38	2
v_2	v_8	46	0
v_2	v_9	34	6
v_2	v_{10}	37	3

ogy that is within the control of Microsoft and, therefore, do not consider peering devices or other ISP networks that connect with the core network. However, the approach can be extended beyond the Microsoft network provided we have an accurate characterization of the topology.

2.2 Routing Protocols

Large-scale data center networks typically employ *multi-path routing*, which involves using multiple alternative paths through the network in order to improve fault tolerance, bandwidth, and security. In this work, we consider *Equal-Cost MultiPath (ECMP)* routing used within data centers and *Border Gateway Protocol (BGP)* used in the core network between data centers. Routers within data centers use ECMP routing to spread the traffic along multiple paths by making per-hop decisions based on routing metric calculations. For routing paths, we assume that all shortest paths are equally likely and have the same cost metric, which closely resembles the ECMP protocol.

BGP is the routing protocol that determines the routes taken in the core network. BGP is responsible for setting up Multiprotocol Label Switching (MPLS) that directs traffic from one network node to the next based on short path labels rather than long network addresses. These network paths are termed Label-Switched Paths (LSPs). For this work, we assume that all shortest paths across the core are equally likely and do not take into account the LSP information available in the core network. We plan to refine the routes in the core with this information in the future.

2.3 Network Monitoring

In passive monitoring, devices are polled periodically using various technologies (e.g., SNMP, NetFlow, RMON) and information is collected to assess network performance and status. Unfortunately, passive monitoring is known to be noisy as the generated alerts are largely threshold based and, in some cases, it suffers from a lack of signal when a device is actually faulty (*silent failures*) [6]. Hence, we leverage these signals to only increase the confidence and improve the pinpointing of the localization from our statistical analysis performed on active monitoring signals.

Our main active monitoring signal comes from the *ping service*, which runs on servers in most racks in all data centers. This service sends pings every minute to other randomly selected end hosts within the cluster, across clusters, and across data centers. Pings can fail or be delayed due to performance issues in the network or hard failures. A significant number of failed or delayed pings is an indication that there may be an issue in the physical network along the path(s) of the pings. Since the ping service runs on the same servers as the other data center services, it is a good proxy for user-perceived network availability and latency.

Another active monitoring tool is *traceroute*, which is used for displaying the path and measuring transit delays of packets across a network. A traceroute is significantly costlier than a ping since the entire history of the route is recorded. Hence, generating traceroutes to cover large-scale data center networks is prohibitively expensive and are primarily used for targeted testing [9].

3. APPROACH OVERVIEW

Our localization approach uses statistical data mining techniques on ping data to identify the links or devices that are responsible for ping failures (or significant ping delays), which typically translate into user-perceived network interruptions. This section will describe the problem formulation and provide an overview of our approach that will guide the description of our solution in Section 4.

Problem Formulation: There are two main inputs to our problem. The first input is the *network topology*, which we represent using a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$. The vertices $\mathbb{V}$ in the graph correspond to network devices, whereas edges $\mathbb{E}$ correspond to network links. The graph contains the links and devices for all compute and storage clusters within each data center, as well as the links and devices that form the global backbone network connecting all data centers together. Figure 3 shows a simple graph with 10 vertices and 12 edges that we will use as a running example.

The second input is the *ping data*, which specifies how many pings succeeded and how many failed (or had high latency) between multiple source and destination vertices. High latency of a ping typically arises for two reasons: (a) some network device had to re-try sending the ping multiple times before succeeding or (b) the ping took a longer path than usual. In either case, significant high latency indicates an underlying issue with the network that we want to capture and localize. In our case, we consider ping latency to be significant if it belongs in the 95^{th} percentile of the latency distribution. Table 1 shows a small example of ping data.

Our goal is to analyze the ping data overlaid on top of the topology in order to generate a short ranked list of edges and vertices associated with *failure scores*, whose potential failure would best explain the observed ping data. This list is then combined with alerts from link- and device-level telemetry data to further help network operators pinpoint a fault.

Solution Phases: The proposed failure localization approach works in 4 phases, each addressing 1 major challenge.

1. *Probabilistic Routing:* The exact ping routes through the network are unknown. Therefore, we model the underlying network protocol typically used within the data centers and then compute the most likely ping routes and their probabilities (Section 4.1).

2. *Probabilistic Failure Modeling:* Network failures are often partial. For example, a faulty link may be dropping only a small fraction of its traffic. For this reason, we model failures using a probabilistic model and formulate a corresponding data fitting problem. We solve this problem and generate a ranked list of edges and vertices associated with failure scores (Section 4.2).

3. *Deviation Detection:* There exists background noise in both the input data and the generated failure scores. To overcome this challenge, the key idea is to first look at the failure scores over time in order to establish a score baseline. We then use statistical hypothesis testing to detect significant deviations from this baseline, filtering out the noise (Section 4.3).

4. *Correlation Analysis:* In certain cases, some devices or links can be indistinguishable from each other, like in the case where multiple physical links are connecting two devices. By intersecting our probabilistic results with the current network alerts produced from passive monitoring, we can pinpoint the specific link or device causing the issue (Section 4.4)

4. FAILURE LOCALIZATION

4.1 Probabilistic Routing

Multipath routing (e.g., ECMP), which is prevalent in almost all large networks, implies that the network route of any given ping (or packet in general) is determined by the routers using local per-hop decisions and is, therefore, unknown (recall Section 2.2). Instead, only the ping source and destination devices are known.

Given a source vertex v_s and a destination vertex v_d in the network graph, the goal of probabilistic routing is to compute the most likely ping routes along with their probabilities. The computation is based on the following two principles that govern multipath routing:

- Only shortest paths (in terms of number of hops) between v_s and v_d are valid routes.
- All next hops from a single vertex are equally probable.

Consider a ping from v_1 to v_{10} in the example network graph in Figure 3. There are three possible routes indicated by the three dotted green arrows, each requiring a total of four hops. All pings from v_1 must first go through v_3. At v_3, there are two equally-likely options: v_5 and v_6. Pings that are routed to v_5 will then traverse v_7 before reaching the final destination v_{10}. On the other hand, pings that reach v_6 will be routed to either v_7 or v_8 with equal probability before reaching v_{10}. In summary, the three routes and the corresponding probabilities are:

$$Pr[v_1 \to v_3 \to v_5 \to v_7 \to v_{10}] = 0.50$$

$$Pr[v_1 \to v_3 \to v_6 \to v_7 \to v_{10}] = 0.25$$

$$Pr[v_1 \to v_3 \to v_6 \to v_8 \to v_{10}] = 0.25$$

Table 2: List of notations.

Notation	Explanation
v_i	Vertex i in the network graph
$e = (v_i, v_j)$	Edge e connecting vertices v_i and v_j
$v_i \to v_j$	Ping from vertex v_i to v_j
$P_{i,j}(e)$	The probability that a $v_i \to v_j$ ping will pass through edge e
$N_{i,j}$	The total number of pings from v_i to v_j
$F_{i,j}$	The number of failed pings from v_i to v_j
X_e	Failure score for edge e

We use the Floyd-Warshall algorithm to compute the shortest distances from all vertices to all other vertices in the graph with worse case complexity of $\mathcal{O}(|\mathbb{V}|^3)$. We then build the routes of each ping opportunistically based on the shortest distances and compute the route probabilities based on the number of forwarding edges of each vertex in each route.

We use the computed routes and probabilities to also calculate the probability $P_{s,d}(e)$ that a given $v_s \to v_d$ ping will pass through a particular edge e. Consider a $v_1 \to v_{10}$ ping and edge $e = (v_7, v_{10})$ in the example graph in Figure 3. This edge is part of two different routes, namely $v_1 \to v_3 \to v_5 \to v_7 \to v_{10}$ and $v_1 \to v_3 \to v_6 \to v_7 \to v_{10}$, with route probabilities 0.5 and 0.25, respectively. The ping will pass through edge e if it takes one of the two aforementioned routes. Therefore, the probability that the ping will pass through edge e equals the sum of the two route probabilities, i.e., $P_{1,10}(e) = 0.5 + 0.25 = 0.75$. To simplify the discussion, we present the model in terms of network edges. A similar analysis applies to vertices.

4.2 Probabilistic Failure Modeling

Network devices and links that are experiencing a failure often drop only a fraction of their overall traffic, depending on the nature of the failure. In particular, issues caused during maintenance or by device malfunctions are fairly common and typically cause small performance degradation or a small packet loss rate (less than 5%). On the other hand, catastrophic events like fiber cuts and complete hardware failures are fairly rare. We have seen only 5 such events out of the 73 real network incidents discussed in Section 6.

In order to address the challenge of partial failures, we employ a probabilistic failure model that assigns a failure probability value to each vertex and edge based on the observed ping data. We will start the discussion using a simple example before we formalize our model. Consider the network graph shown in Figure 3. Suppose edge $e = (v_6, v_7)$ is experiencing an issue and is dropping some packets. Further, suppose the total number of pings from v_1 to v_{10} is $N_{1,10} = 50$. Based on the probabilistic routing discussed in Section 4.1, we calculate the probability that any ping will go through the problematic edge e. Specifically, $P_{1,10}(e) = 0.25$. Finally, assume that this edge is dropping 20% of its traffic. In other words, the probability that any ping going through e will fail is $X_e = 0.2$. Therefore, out of the 50 pings, 25% of them will go through the problematic edge, out of which 20% will fail. Multiplying the 3 numbers together will give us the expected number of ping failures, $F_{1,10} = 5$. Table 2 summarizes the notation used in this section.

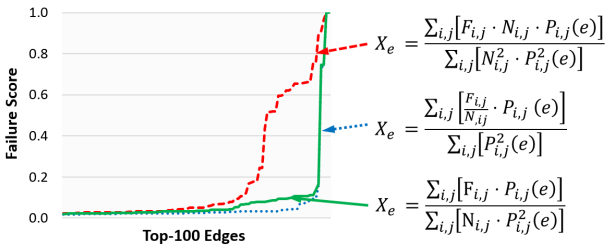

Figure 4: Distribution of failure scores for the top-100 edges for three different least squares formulations.

Equation 1 provides the basic unit of our model:

$$F_{i,j} = N_{i,j} \cdot P_{i,j}(e) \cdot X_e \tag{1}$$

The number of failed pings $F_{i,j}$ and total pings $N_{i,j}$ between vertices v_i and v_j are observed. The probability $P_{i,j}(e)$ that a $v_i \rightarrow v_j$ ping will go through a particular edge e is computed by the routing model. Finally, X_e, the failure score of edge e, is the unknown value we need to compute. This model assumes there is at most one failed edge (or vertex) on the paths from v_i to v_j, which is common in practice. However, concurrent failures on independent network paths are captured by the model. The fact that we have a long list of pings between multiple sources and destinations results in the creation of an overdetermined system of equations that can be numerically solved.

Consequently, we need to solve a data fitting problem by computing the failure score X_e for each edge e that best fits the observed data. There are several well-known techniques for solving such problems and we have chosen to use the method of *Least Squares* because it produces a linear unbiased estimator and it is computationally efficient to solve.

Least squares formulation: The best fit in the least-squares sense minimizes the sum of squared residuals; a residual being the difference between the observed value and the fitted value provided by the model. The model from Equation 1 results in the following formulation:

$$Y_e = \sum_{(\forall v_i, v_j \in V)} \left(F_{i,j} - N_{i,j} \cdot P_{i,j}(e) \cdot X_e\right)^2 \tag{2}$$

Solving the above least squares problem yields the equation:

$$X_e = \frac{\sum_{(\forall v_i, v_j \in V)} \left(F_{i,j} \cdot N_{i,j} \cdot P_{i,j}(e)\right)}{\sum_{(\forall v_i, v_j \in V)} \left(N_{i,j}^2 \cdot P_{i,j}^2(e)\right)} \tag{3}$$

The main drawback of this solution is that it typically assigns a wide range of failure scores to multiple edges during a network issue. In layman's terms, it is spreading the blame around, increasing the size of the output list that a network operator would have to investigate. Figure 4 shows the distribution of failure scores for the top-100 edges computed by three different least squares solutions during a real network incident. The solution from Equation 3 assigns a failure score higher than 0.5 to 25% of the edges. This behavior is partly due to the presence of the $N_{i,j}^2$ term in the denominator; the failure score is thus very sensitive to the number of total pings between each pair of vertices, which can vary between tens to hundreds of pings every 5 minutes.

Solving for $F_{i,j}/N_{i,j}$ rather than $F_{i,j}$ in the original model (Equation 1) results in a second formulation and solution:

$$Y_e = \sum_{(\forall v_i, v_j \in V)} \left(\frac{F_{i,j}}{N_{i,j}} - P_{i,j}(e) \cdot X_e\right)^2 \tag{4}$$

$$X_e = \frac{\sum_{(\forall v_i, v_j \in V)} \left(\frac{F_{i,j}}{N_{i,j}} \cdot P_{i,j}(e)\right)}{\sum_{(\forall v_i, v_j \in V)} \left(P_{i,j}^2(e)\right)} \tag{5}$$

The solution in Equation 5 has the desired property of high contrast in the distribution of failure scores, as can be seen in Figure 4. Only 4 edges get a failure score over 0.5. However, this solution completely ignores the presence of successful pings between two vertices when there are no failed pings, i.e., $N_{i,j}$ is ignored when $F_{i,j} = 0$.

The above observation has lead to our final formulation and solution:

$$Y_e = \sum_{(\forall v_i, v_j \in V)} \left(\frac{F_{i,j}}{\sqrt{N_{i,j}}} - \sqrt{N_{i,j}} \cdot P_{i,j}(e) \cdot X_e\right)^2 \tag{6}$$

$$X_e = \frac{\sum_{(\forall v_i, v_j \in V)} \left(F_{i,j} \cdot P_{i,j}(e)\right)}{\sum_{(\forall v_i, v_j \in V)} \left(N_{i,j} \cdot P_{i,j}^2(e)\right)} \tag{7}$$

Figure 4 shows that the solution from Equation 7 offers a similar contrast in failure scores as Equation 5, while the presence of $N_{i,j}$ in the denominator ensures the use of successful pings in the calculations. We use Equation 7 to compute the failure scores for all edges and vertices that are part of a possible route of any failed ping, and produce a ranked list of edges and vertices based on that score.

4.3 Deviation Detection

The presence of *background noise* in large-scale network monitoring systems is common [6]. In our case, there exists background noise in both the input ping data as well as the generated failure scores.

Noise in the ping data typically appears in the form of false ping failures or delays, even though the underlying network is not experiencing any issues. In some cases, inflated ping latency measurements are caused by increased load on the servers running the ping service. In other cases, pings might get dropped by a router—even though regular applications do not exhibit any packet loss—due to the combination of two factors: (a) traffic saturation at a link and (b) pings having the lowest priority in network traffic.

Noise in the generated failure scores is primarily attributed to the unknown nature of network routes. Since the routes for failed pings are unknown and computed using probabilities, any edge or vertex across alternative routes can potentially have a non-zero failure score.

Background noise typically manifests as low failure scores for healthy edges and vertices. A simple solution would be to introduce a threshold for filtering out the low scores. Apart from the obvious drawback of introducing an ad-hoc threshold value, we would risk filtering out low failure scores that resulted from true partial failures. By definition, partial failures cause low packet drop rates and naturally lead to low failure scores. By simply looking at the value of a failure score, we cannot distinguish these two cases.

Statistical hypothesis testing: The key idea behind our solution is to look at failure scores over time in order

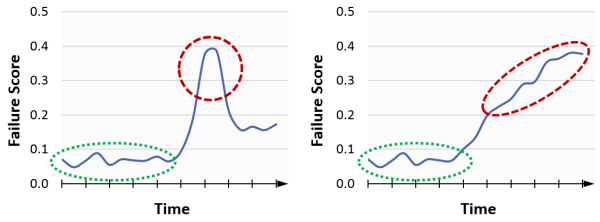

Figure 5: Detection of a spike and a gradual increase in a time sequence of failure scores.

to establish a score baseline for each edge and vertex, and then use *statistical hypothesis testing* to determine whether a current failure score is significantly different than its historical values. This approach allows us to filter out the false positives due to noise and only output edges and vertices with statistically significant failure scores.

Given a time sequence of failure scores $\mathbb{S} = \{s_0, s_1, ..., s_n\}$ for a particular edge (or vertex), we use a series of *Student's t-tests* to determine whether the addition of the latest score s_0 introduces a statistically significant change in the sequence of scores. In particular, we are interested in detecting two different score patterns over time, namely spikes and gradual increases, seen in Figure 5. Spikes are detected using *one-sample* Student's t-tests while gradual increases using *two-sample* Student's t-tests.

The one-sample Student's t-test is used to determine whether the latest score s_0 is likely to belong in the distribution of historical scores $\mathbb{S}_h = \{s_1, ..., s_n\}$, which forms the *null* hypothesis. Let n_h, μ_h, and σ_h^2 be the size, mean, and variance of $\mathbb{S}_h$, respectively. According to the t-test, the *null* hypothesis is rejected (i.e., the score is deemed significantly different) when

$$s_0 > \theta \cdot \mu_h + t_{\alpha, n_h - 1} \cdot \sqrt{\frac{\sigma_h^2}{n_h}} \tag{8}$$

where θ is a scaling parameter and $t_{a, n_h - 1}$ is the t value obtained from the t-distribution table using significance level $\alpha = 99\%$ and degree of freedom equal to $(n_h - 1)$. Currently, we use a dynamic scaling parameter $\theta = 1/\sqrt{\mu_h}$, which is set based on empirical evidence.

In the two-sample Student's t-test, we divide the set of scores $\mathbb{S}$ into two independent sets, the current set $\mathbb{S}_c = \{s_0, ..., s_c\}$ and the historical set $\mathbb{S}_h = \{s_h, ..., s_n\}$. Locations c and h in the sequence are configurable and control the size and gap between the two sets.

Let n_c, μ_c, and σ_c^2 be the size, mean, and variance of $\mathbb{S}_c$, respectively. Similarly, n_h, μ_h, and σ_h^2 are the size, mean, and variance of $\mathbb{S}_h$. The *null* hypothesis is that $\mu_c \leq \theta \cdot \mu_h$, for some scaling parameter θ. We use $\theta = 1/\sqrt{\mu_h}$ as discussed above. We account for the presence of θ in the *null* hypothesis by revising $\mu_c = \mu_c/\theta$ and $\sigma_c^2 = \sigma_c^2/\theta^2$. According to the t-test, we compute the t-statistic T as:

$$T = \frac{\mu_c - \mu_h}{\sqrt{\frac{\sigma_c^2}{n_c} + \frac{\sigma_h^2}{n_h}}} \tag{9}$$

The degree of freedom df is computed using:

$$df = \frac{\left(\frac{\sigma_c^2}{n_c} + \frac{\sigma_h^2}{n_h}\right)^2}{\frac{\left(\frac{\sigma_c^2}{n_c}\right)^2}{n_c - 1} + \frac{\left(\frac{\sigma_h^2}{n_h}\right)^2}{n_h - 1}} \tag{10}$$

Finally, the *null* hypothesis is rejected (and the latest score is deemed significantly different) when

$$T > t_{\alpha, df} \tag{11}$$

where $t_{\alpha, df}$ is the t value obtained from the t-distribution table using significance level $\alpha = 99\%$.

Both the one-sample and two-sample Student's t-tests are invoked each time a new failure score is computed. If any of the t-tests determines the score to be significant, the corresponding edge (or vertex) is added in the final ranked list.

4.4 Correlation Analysis

The algorithms discussed so far generate a ranked list of links and devices suspected to have caused ping failures. However, it is possible for a small set of links or devices to share the same failure score due to network redundancy and multipath routing. As an example, consider the case were three physical links are connecting two network devices for redundancy purposes. Multipath routing (and our probabilistic routing) postulates that all traffic between the two devices will be equally divided among the three links. Therefore, the probability that a failed ping will go through any of the three links is the same, which in turn implies that the three links will have the same failure score.

We address this issue by taking advantage of the availability and performance alerts produced by passive monitoring (recall Section 2.3). In particular, we perform a *left outer join* between our ranked list $\mathbb{L}$ and the set of device- and link-level network alerts $\mathbb{A}$ that are triggered during the time period of the analysis. This join retains all devices and links in $\mathbb{L}$ while appending the alerting information from $\mathbb{A}$ to the matching records. We do not remove the records in $\mathbb{L}$ that do not match $\mathbb{A}$ in order to capture the cases were a device or link is dropping packets without issuing any alerts. (We discuss such cases in the experimental evaluation in Section 6.) Instead, devices and links with alerting information are pushed to the top of the list since there is more evidence that they may be responsible for the observed ping failures.

5. DEPLOYMENT

We have implemented the fault localization approach discussed in Section 4 and incorporated it into the Windows Azure network monitoring infrastructure. Figure 6 shows the relevant components of the monitoring system:

- The **Data Store** holds the network topology information, the ping data, as well as local telemetry data collected from devices and other components of the network.
- The **Ping Service** is a distributed service that runs on all compute clusters of the data centers and is responsible for collecting and sending the ping data to the Data Store.
- Multiple **Device Data Collectors** are continuously collecting, aggregating, and sending local telemetry data to the Data Store.
- The **Analysis Worker** streams the topology, ping, and telemetry data, performs the fault localization analysis, and streams the results back to the Data Store.

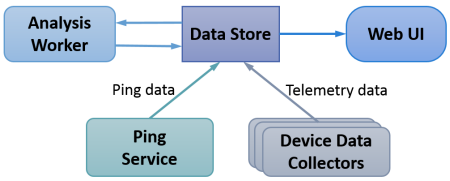

Figure 6: System architecture.

- The **Web UI** provides a web-based interface used by network operators to consume the results of the fault localization analysis in a user-friendly report.

The Ping Service and Device Data Collectors are continuously collecting large amounts of data that need to be analyzed. In particular, there are about 400 thousand pings and several hundreds to thousands of network events being generated every 15 minutes by the existing framework. In order to produce results in near real time, we process the data incrementally. Every five minutes, the Analysis Worker will (a) get the latest topology information from the Data Store and build the network graph, (b) stream in the ping and telemetry data from the last 15 minutes, (c) run the fault localization analysis presented in Section 4, and (d) stream out the ranked list of suspect links and devices, whose failures best explain the observed data.

6. EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness and efficiency of our approach to correctly localize issues in the entire Windows Azure network. The experimental evaluation is based on 73 real network incidents that were investigated by the Windows Azure network operations team during a period of three months. After sifting through large amounts of data and reports compiled primarily by network operators, we identified the (most likely) root cause for each incident. Root causes are divided into the following 10 categories:

- **Device Issue:** A network device experienced an issue such as hardware failure or misconfiguration.
- **Fiber Cut:** The fiber cable connecting two or more devices was damaged or cut.
- **Maintenance:** Devices and links were going through planned maintenance, which may have affected their performance or availability.
- **Transient Failure:** A device or link experienced a short failure and then recovered.
- **DNS Issue:** The Domain Name System (DNS) that translates domain names to IP addresses experienced an issue like incorrect name resolution.
- **BGP Issue:** The Border Gateway Protocol (BGP) that makes routing decisions based on paths experienced an issue like BGP sessions going down.
- **Software:** A software component like a Software Load Balancer had failed or operated incorrectly.
- **False Alarm:** There was no clear evidence that a real network issue had occurred and the network appeared to be healthy during the investigation.

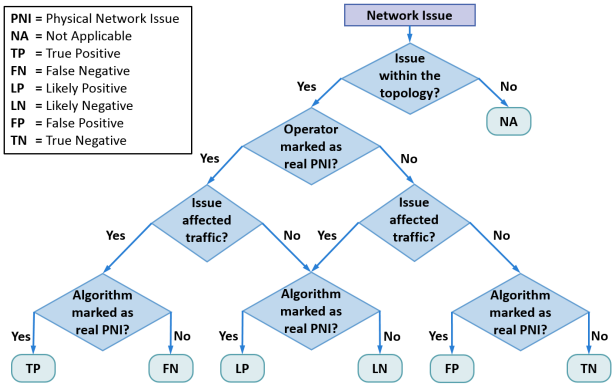

Figure 7: Derivation of localization accuracy classifications.

- **DDoS Attack:** A Distributed Denial-of-Service attack (DDoS attach) caused performance or availability issues to the network.
- **Peering Issue:** The issue was localized to outside the Windows Azure network.

Table 3 lists the total count of incidents for each root cause.

For the purpose of our evaluation, we define a real *physical network issue (PNI)* to be any issue experienced by a network device or link that resulted in dropped traffic and, therefore, has affected higher-level users. For example, a fiber cut or a device failure is considered a real physical network issue; these are the type of issues our algorithm is trying to localize. On the other hand, issues with DNS or software issues are not considered physical issues, and thus our algorithm should exclude them from localization.

6.1 Localization Accuracy

In this section, we evaluate the ability of our approach to accurately localize real physical network issues (PNIs) and to correctly exclude non-physical network issues. Unfortunately, in many cases, we do not have the ground truth of whether a network incident was caused by a real PNI or not. One option would be to consider the final ruling of the network operator as the ground truth. However, issues such as transient failures can, by definition, cause loss of traffic at the time of the incident but for the network to appear healthy by the time the operator investigates the incident. Hence, to evaluate the localization accuracy of our approach, we combine two sources of truth: (a) the expertise of the network operator in marking an issue as a real PNI or not, and (b) traffic data showing whether there was a significant loss of traffic near the location and around the time of the investigated incident.

When the two sources of truth are in agreement, localization accuracy is easily established. First, suppose the operator marked the incident as a real PNI and there was significant loss of traffic at the time of the incident. Using conventional terminology from binary classification, we declare the result of our localization approach to be *true positive (TP)* when our approach correctly identifies and localizes the issue and *false negative (FN)* when it failed to identify the issue. On the other hand, suppose the operator marked the incident as not a PNI and there was no observable loss of traffic. In this case, the localization result is considered *true negative (TN)* if the algorithm did

Table 3: Localization accuracy per root cause.

Root Cause	TP	TN	FP	FN	LP	LN	NA	Total
Device Issue	4	–	–	–	–	4	1	9
Fiber Cut	3	–	–	–	–	–	1	4
Maintenance	5	5	–	–	–	–	–	10
Transient Failure	3	–	–	–	11	–	–	14
DNS Issue	–	10	–	–	–	–	–	10
BGP Issue	–	5	–	–	–	–	–	5
Software	–	7	–	–	–	–	–	7
False Alarm	–	10	–	–	–	–	–	10
DDoS Attack	–	–	2	–	–	–	–	2
Peering Issue	–	–	–	–	–	–	2	2
Total	15	37	2	0	11	4	4	73

not produce any devices or links in the topology near the investigated incident; otherwise, it is a *false positive (FP)*.

When the two sources of truth are in disagreement (i.e., the operator believes there is a real PNI but there is no significant loss of traffic, or the operator believes there is no PNI but there is loss of traffic), we define two new localization accuracy classifications. The localization result is considered a *likely positive (LP)* when the algorithm identifies an issue and either the operator has marked it as a real PNI or there was significant loss of traffic. Conversely, the result is considered a *likely negative (LN)* when the algorithm does not identify an issue and either the operator has marked it as a non-issue or there was no substantial loss of traffic. Finally, if the issue is not located within the network graph, then we classify it as *not applicable (NA)*. For example, peering links that connect Windows Azure to the outside world are not part of our topology. Figure 7 summarizes the derivation of our localization accuracy classifications.

Table 3 lists the localization accuracy of our approach for each root cause for the 73 incidents. Overall, our approach was able to correctly identify and localize 15 real physical network issues (TPs) while correctly excluding 37 non-issues (TNs). The root causes for the TPs were device issues, fiber cuts, issues during planned maintenance, and transient failures. All DNS, BGP, and software issues as well as all false alarms and some maintenance issues were correctly classified as TNs. There were only 2 cases of FPs where our algorithm incorrectly output devices and links as suspects of causing failures; both incidents were likely the result of DDoS attacks. There were no cases of FNs.

There were also 15 incidents where the network operator and the traffic data were in disagreement. In particular, there were 11 cases of transient failures where the operator did not find a physical network issue even though there was significant loss of traffic observed for a short period of time (typically 5-10 minutes). In fact, in all cases there was additional evidence in the form of device- and link-level alerts suggesting there was indeed a physical network issue. We classify these cases as likely positives (LPs) even though we believe they are probably TPs. The remaining 4 cases are classified as likely negatives (LNs) and are attributed to device misconfigurations that did not affect traffic (and by extension, did not cause any user-perceived failures). Finally, there were 4 incidents that occurred outside the Windows Azure network and are thus classified as NA.

In the absence of our algorithm, the network operators must rely on their expertise as well as device- and link-level

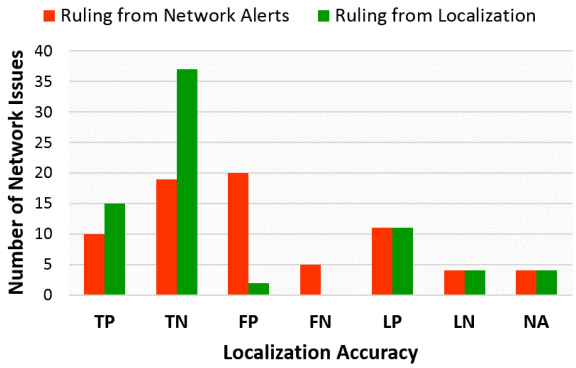

Figure 8: Localization accuracy of our approach compared to the conventional approach based on network alerts.

alerts from passive monitoring in order to localize network issues. Figure 8 compares the localization accuracy of the approach based on network alerts (we call this the *conventional* approach) and our approach. We observe that the conventional approach yields a large number of false positives; 20 compared to just 2 from our approach. It is fairly common for BGP, DNS, or software issues to cause device- and link-level alerts even though the actual devices and links are healthy and the traffic is not affected. In comparison, our approach is able to correctly determine that the issue is not related to physical devices or links.

Furthermore, the conventional approach suffered from 5 false negatives. There were 5 network incidents caused by a device (or link) malfunction but that device (or link) had no alerts associated with it (i.e., they were silent failures). We have seen this happen during device hardware failures, device misconfiguration, or planned maintenance. However, since the problematic device was causing traffic loss (and by extension ping failures or delays), our approach was able to detect and localize it correctly.

6.2 Localization Precision

While Section 6.1 demonstrated the ability of our approach to accurately localize network failures, it does not give any indication of how precise the localization is. In this section, we evaluate the precision of our approach using two metrics: one based on the size of the ranked list $\mathbb{L}$ of devices and links produced by the algorithm, and another based on the placement (i.e., rank) of the problematic device or link that caused the network issue in $\mathbb{L}$.

The first metric is termed *localization precision* and is inspired by a similar metric defined in [9]. The localization precision for a given network incident is defined as the ratio of the number of suspect devices and links after localization (i.e., the size of list $\mathbb{L}$) to that before localization. In other words, it is the fraction of devices and links that are likely to explain a particular network issue using our algorithm out of all the devices and links that can potentially cause that issue. We call the latter set the *potential set* $\mathbb{P}$.

The size of $\mathbb{P}$ depends on the scope of the investigation, which typically starts with some higher-level performance or availability alert for some part of the network. For example, suppose there is an availability alert for a particular cluster. Any ToR switch or CSP and their links could be potentially causing the issue. Table 4 lists the number of potential devices and links that might need to be checked for the 15 TP

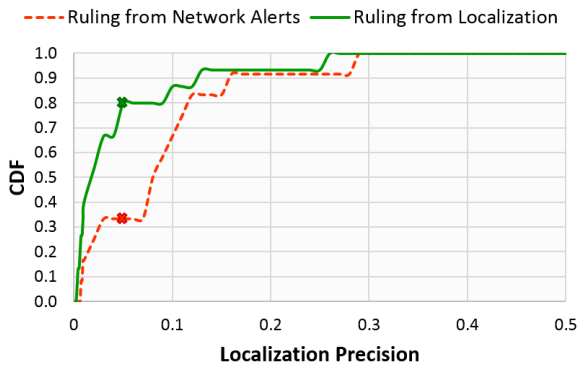

Figure 9: Localization precision of our approach compared to the conventional approach based on network alerts.

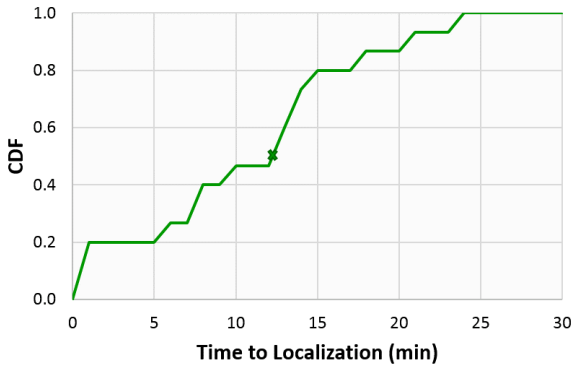

Figure 10: Time to localize real network incidents.

incidents. Of course, using domain knowledge and expertise, an operator will only check a subset of $\mathbb{P}$; but the estimate underscores the challenge faced by operators today.

Figure 9 shows the cumulative distribution functions (CDFs) of the localization precision of our approach and the conventional approach[2]. We observe that our approach localizes network issues to less than 2% for more than 50% of the failures and to less than 5% for more than 80% of the failures. On the contrary, only 33% of the failures are localized to less than 5% using the conventional approach. Table 4 further supports the poor localization precision of the conventional method as the number of distinct devices or links associated with network alerts is often much larger (up to an order of magnitude larger) than the total size of our ranked list $\mathbb{L}$. We conclude that our approach is able to localize the true issue very precisely from a large set of possible root causes for a given failure.

The second metric for evaluating the algorithm's precision is the *diagnostic rank*, which is the number of network devices or links deemed more likely culprits than the device or link that actually failed [11]. Assuming operators investigate failures in the order listed by our algorithm, the diagnostic rank reflects the overhead of identifying and resolving the issues. Table 4 lists the average diagnostic rank computed during the duration of each incident. On average, the operator will have to check less than 5 devices or links (and almost always less than 10) before identifying the real culprit, compared to 6–305 for conventional diagnosis. Hence, our approach can significantly reduce diagnosis time.

6.3 Efficiency

In this section, we evaluate the efficiency of our approach in terms of how quickly a particular network issue is localized. As discussed in Section 5, our algorithm is running continuously in real time and generates a ranked list of devices and links every 5 minutes (assuming it has identified a network issue). We compare the time the ranked list contained the problematic device or link for the first time with the starting time of the incident. Note that in most cases the exact starting time of an incident is not known, so we are using the starting time recorded in the incident report. Figure 10 shows the CDF of the time to localize real network incidents in Windows Azure. We observe that 50% of the incidents are localized within 12 minutes while the longest time to localize is 23 minutes. Typically, the time to localize depends on the severity of the incident; the higher the ping failure rate, the shorter the time to localize since the statistical hypothesis testing will quickly detect the deviation in the failure score. For example, the incident that took 23 minutes to localize had a ping failure rate between 1-2%.

It is interesting to note that in 3 incidents the device or link causing the issue was localized by our algorithm well before the starting time recorded by the operator—and presumably, before the time the operator started the investigation. Therefore, there is potential for our approach to be used in a more *proactive* way for localizing user-perceived failures. We plan to investigate this approach in the near future. Overall, our localization approach can identify and localize real physical network issues in a very short time period, significantly improving the ability of the operations team to localize and resolve network incidents.

[2]The localization precision for the conventional approach is based on the number of distinct devices and links with network alerts within the scope of the investigation.

7. RELATED WORK

In network tomography, link-level properties like link loss and packet latency are inferred using statistical approaches from end-to-end measurements [5]. In particular, BADABING [12] and Tulip [10] measure per-path characteristics to identify problems that impact network performance. These methods, along with some commercial products as well, use active testing (e.g., traceroute) to pinpoint faulty links. However, they have only been applied to smaller networks since applying an active testing technique that will sufficiently cover a worldwide network is prohibitively expensive. Our approach, on the other hand, does not require any controlled testing infrastructure and relies only on ping data, which is typically available as a connectivity signal in data center networks.

There exists a large body of work on detecting and localizing performance problems across network and services. Sherlock [2] captures the dependencies between various components of the IT infrastructure (e.g., DNS service, load balancers, network components) and tries to localize failures to a small number of components. Similarly, Shrink [8] and SCORE [9] have their own dependency model, which they use to find the most likely root causes of faults in wide-area networks. In contrast, our approach localizes failures on the underlying physical network, both within and across data centers, and as such, can complement the aforementioned tools. Furthermore, we have shown that our approach is

Table 4: Statistics for a sample of real network issues in Windows Azure. The device abbreviations are given in Figure 2.

Investigation Scope	Root Cause (Device/Link)	# Potentially Failed Devs/Links	# Devs/Links with Network Alerts	# Devs/Links in our Ranked List	Average Diagnostic Rank
Core Network	Fiber Cut (CR–CR Link)	79	6	10	9.3
Core Network	Fiber Cut (CR–CR Link)	262	–	12	10.5
Core Network	Fiber Cut (CR–CR Link)	868	85	5	1.0
Core Network	Transient (CR–CR Link)	452	72	5	3.0
Datacenter	Device Issue (DSP)	1178	132	24	3.0
Datacenter	Device Issue (CR)	951	6	9	3.0
Datacenter	Device Issue (CR)	453	–	3	2.0
Datacenter	Maintenance (CR–CR Link)	845	62	39	9.7
Datacenter	Maintenance (AR)	584	6	147	3.0
Datacenter	Maintenance (CR)	891	8	8	3.4
Datacenter	Transient (BL–DSP Link)	951	98	95	5.2
Multiple Clusters	Maintenance (CR–BL Link)	9173	228	29	3.0
Cluster	Device Issue (ToR)	1052	305	20	2.8
Cluster	Maintenance (CSP–ToR Link)	3364	287	8	6.7
Cluster	Transient (CSP)	1306	–	28	1.5

scalable, can produce results in near real time, and can handle densely connected topologies in addition to loosely connected ones such as wide-area networks.

8. CONCLUSIONS

In this paper, we described a novel and successful application of statistical data mining techniques for localizing user-impacting availability and performance issues in a worldwide data center network in near real time. A unique aspect of our work is that it is agnostic to the routing intricacies and forwarding aspects of the network and is, therefore, a scalable solution. We have deployed our solution on the Windows Azure network monitoring infrastructure with a special focus on producing a high confidence, real-time signal that can be used by operations teams to detect and localize physical network issues. Our experimental evaluation on real network incidents has validated both the efficiency and effectiveness of our approach.

9. ACKNOWLEDGMENTS

We would like to thank the Windows Azure Network teams that collaborated with us, especially Albert Greenberg for introducing us to this problem as well as Monika Machado and Tina Stewart for providing operational support.

10. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proc. of the ACM SIGCOMM Conf. on Data Communication*, pages 63–74. ACM, 2008.

[2] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *Proc. of the ACM SIGCOMM Conf. on Data Communication*, pages 13–24. ACM, 2007.

[3] L. A. Barroso and U. Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-scale Machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.

[4] Y. Bejerano and R. Rastogi. Robust Monitoring of Link Delays and Faults in IP Networks. *IEEE/ACM Transactions on Networking*, 14(5):1092–1103, 2006.

[5] N. Duffield. Network Tomography of Binary Network Performance Characteristics. *IEEE Transactions on Information Theory*, 52(12):5373–5388, 2006.

[6] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proc. of the ACM SIGCOMM Conf. on Data Communication*, pages 350–361. ACM, 2011.

[7] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proc. of the ACM SIGCOMM Conf. on Data Communication*, pages 51–62. ACM, 2009.

[8] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. In *Proc. of the ACM SIGCOMM Workshop on Mining Network Data*, pages 173–178. ACM, 2005.

[9] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP Fault Localization Via Risk Modeling. In *Proc. of the 2nd Symp. on Networked Systems Design & Implementation (NSDI)*, pages 57–70. USENIX, 2005.

[10] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet Path Diagnosis. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP)*, pages 106–119. ACM, 2003.

[11] R. Niranjan Mysore. *Automated Scalable Management of Data Center Networks*. PhD thesis, University of California, San Diego, 2013. `http://escholarship.org/uc/item/7mb1w3rv`.

[12] J. Sommers, P. Barford, N. Duffield, and A. Ron. Improving Accuracy in End-to-End Packet Loss Measurement. In *Proc. of the ACM SIGCOMM Conf. on Data Communication*, pages 157–168. ACM, 2005.

[13] H. H. Song, L. Qiu, and Y. Zhang. NetQuest: A Flexible Framework for Large-scale Network Measurement. *ACM SIGMETRICS Performance Evaluation Review*, 34(1):121–132, 2006.

[14] Windows Azure Services, 2014. `http://www.windowsazure.com/en-us/services/`.

[15] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic Test Packet Generation. In *Proc. of the 8th Intl Conf. on Emerging Networking Experiments and Technologies (CoNext)*, pages 241–252. ACM, 2012.

[16] Y. Zhao, Y. Chen, and D. Bindel. Towards Unbiased End-to-end Network Diagnosis. In *Proc. of the ACM SIGCOMM Conf. on Data Communication*, pages 219–230. ACM, 2006.