Measuring and Troubleshooting Large Operational Multipath Networks with Gray Box Testing

Hongyi Zeng[†*], Ratul Mahajan[§], Nick McKeown[†]
George Varghese[§], Lihua Yuan[§], Ming Zhang[§]
[†]Stanford University [§]Microsoft
{hyzeng,nickm}@stanford.edu, {ratul,varghese,lyuan,mzh}@microsoft.com

ABSTRACT

Troubleshooting large operational networks is extremely difficult due to the extensive usage of multipath routing. We present NetSonar, a system that localizes performance problems in such networks. It uses *planned tomography*, whose input comes from a novel test technique that maximizes component coverage while minimizing probing overhead. Earlier techniques are either white box (assuming complete knowledge of network's forwarding state) or black box (assuming no knowledge). We argue that the former is infeasible in large networks and the latter is inefficient. We use *gray box* technique that needs only coarse forwarding information (e.g., multipath configuration without knowledge of router-internal hash functions). NetSonar deals with nondeterminism in multipath by computing *probabilistic path covers*, and localizes faults accurately with minimal test overhead via *diagnosable link covers*. We describe our experience deploying NetSonar in a global inter-datacenter network. In a one-month period, NetSonar triggered 66 alerts, of which 56 were independently verified.

1. INTRODUCTION

In large networks, performance faults such as high latency are not only commonplace but also difficult to localize. Ideally, these faults would be repaired *before* users complain. In the global inter-datacenter (DC) network that we study, operators tried to achieve this by monitoring SNMP counters (e.g., link load and drops) at routers and alerting when the values cross a threshold. Unfortunately, these counters are noisy and lead to many false alerts (Section 7.3). Consequently, operators routinely ignore them and instead rely on user complaints to trigger fault isolation. Without a systematic method, and bedeviled by a large number of possible paths between two hosts, we have observed many faults that took several hours to detect and localize.

Given the shortcomings of SNMP counters, operators [24] and researchers [8, 10, 21, 29, 30] have proposed active testing. In such testing, test agents at the network boundary send end-to-end probes (e.g., pings) along network paths to detect faults. The results of the probes are then fed to an localization algorithm using what is known as network tomography [5]. Classic network tomography [2, 11, 16] assumes that measurements are *fixed* and seeks to maximize the accuracy of inferences for a given input set of measurements.

However, in many networks, there is an opportunity to plan the placement of test agents and decide which probes to send. We can generate a test plan for optimal localization while minimizing probing overhead. The results from executing the test plan can be fed to a localization algorithm. We call this approach *planned tomography*. Our primary contribution is a new test plan generator for planned tomography in multipath networks, and deployment results that show its benefits over using router counters.

A straightforward testing technique is to choose all nodes as test agents and to send test messages between *all pairs* of test agents. This is a *black box* method because it requires no knowledge of the network. It shares the shortcomings of any black box tester—lack of scalability and inability to reason about coverage. To effectively test large networks, many test agents are needed, which leads to an intractably large number of probes per test interval [24]. Even if one could afford all-pairs testing, it is difficult to reason about which nodes and links are covered, especially when load balancing over multiple paths is used in the network.

To overcome these shortcomings, researchers have proposed the equivalent of *white box* testing [4, 17, 22, 28]. These techniques need detailed information about the network such as traceroute logs or FIB (forwarding information base) from each router. Using this information, they compute optimal locations for test agents and a subset of all possible probes that can cover all the routers and links in the network.

White box testing is impractical for large operational networks for three reasons. First, existing white box testers do not handle multipath networks which is precisely the complication (exponential number of paths between sources and destinations) that exacerbates manual fault isolation. With ECMP, traceroute logs and

[*]Hongyi Zeng was an intern at Microsoft Research while doing this work. He is now with Facebook.

the FIB do not fully describe how a given packet will be forwarded; the outgoing link of a packet is decided by hashing packet headers. However, the hash function is often not known to the network operator. Second, white box testing is complicated by the amount of information it needs to gather. Today, router CPUs tend to be underpowered, and it is not possible to frequently read the millions of FIB prefixes at each of hundreds of routers. Finally, white box techniques assume that traceroute logs and FIB state represent a consistent snapshot across all routers, which is hard to obtain in a large network with high churn.

We thus argue that testing in large networks be *gray box*. Gray box network testing relies on partial, easily-obtainable information about the network structure. It uses this information to efficiently cover all links and routers, thus offering simultaneously the practicality of black box testing and efficiency of white box testing. Based on the information available for a given network, different gray box testing techniques can be developed.

We develop one such gray box test technique. It uses as input the set of available paths between a source and a destination (but not the exact path taken by a given packet). To handle multipath routing, it computes the test plan using *probabilistic path covers* – probes between a source S and destination D use a computed number of packet headers such that all paths between S and D are exercised with high probability. To aid fault localization, plan computation also uses *diagnosable link covers* – each link is covered by a configurable number of probes to allow rapid isolation and to ameliorate the effects of routing churn.

We develop a network tester, called NetSonar, which uses our test technique. We deploy it in a global inter-DC network of a large online service provider, hereafter referred to as IDN, with test agents located in the DCs. Our results are encouraging. In a one-month period, NetSonar detected 66 faults involving high latency and localized them to a router or link in the core of the network. Of these, 56 faults (84.8%) were cross-verified with other monitoring data sources. Several of the faults that we detected were deemed high priority, and upon investigation, the operators confirmed that NetSonar's localization was correct. We also found that SNMP-based alerting would have generated 87 times as many alerts. The vast majority of these appear to be false alarms as they are not accompanied by any end-to-end performance degradation.

2. MOTIVATION

Consider the simple topology in Figure 1 with four routers and four hosts that can act as test agents. Assume that our goal is to test all 3 inter-router links for gray faults (i.e., those that go undetected by low-level monitoring such as keep alive messages) that cause

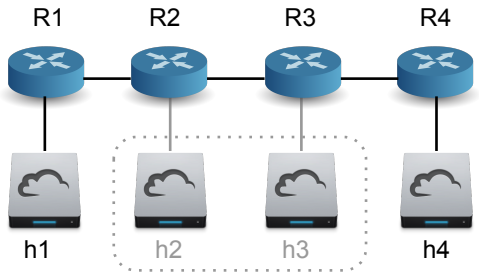

Figure 1: Probing based on knowledge of the network can reduce the number probes by strategically selecting probes. All-pairs testing will generate 12 probes, while only 2 probes ($h1 \rightarrow h4$, $h4 \rightarrow h1$) can cover all links.

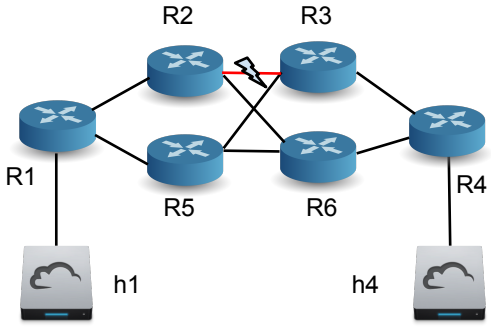

Figure 2: A link with performance problems is difficult to detect in the presence of multipath routing.

packet loss or delay. A black box approach such as all-pairs can achieve this goal, but with 12 (4×3) probes. Such quadratic behavior cannot scale to large networks.

A white box approach can compute that the same test coverage can be achieved using only 2 probes: $h1 \rightarrow h4$ and $h4 \rightarrow h1$. Both are needed because we must test the links in both directions. Such minimal probe sets can be computed by the Min-Set-Cover algorithm, as in [3,4,28]. While a white box technique can compute a small set of probes, as mentioned before, it needs highly detailed information (e.g., the entire FIB state) that is difficult to acquire in large networks.

Further, current white box approaches do not consider multipath routing, which is prevalent in large networks. Figure 2 illustrates the challenge posed by multipath routing. The network employs equal-cost multipath (ECMP): four shortest paths between $R1$ and $R4$ are used in parallel ($R1-R2-R3-R4$, $R1-R2-R6-R4$, etc.). Assume that the link $(R2, R3)$ is faulty. There is no guarantee that a probe sent between $h1$ and $h4$ will traverse the faulty link. Whether the faulty link is covered depends on packet headers in probes and the

(unknown) hash function used by $R1$ and $R2$.

We develop NetSonar based on the observations above. We call it gray box because it relies on some knowledge of network routing but not highly precise information on how individual routers forward traffic. In particular, we assume that the *set* of available paths (four paths in Figure 2) between each pair of ingress-egress routers is known, without knowing the precise path a given packet header traverses.

3. RELATED WORK

NetSonar builds on the long, rich line of work in network fault localization. While it is almost impossible to cover all prior techniques, we classify prior work into five categories and describe how we relate to each.

Unplanned tomography: NetSonar's goal is to identify links with faults such as congestion, which falls squarely under Boolean Network Tomography, pioneered by Duffield et al. [11], followed by many others (e.g., [8, 10, 20, 21, 29, 30]). Besides, more general inference algorithms include SCORE [16] and Sherlock [2]. Unplanned tomography assumes that the set of input measurements is fixed (and uncontrollable). It thus cannot provide guarantees on failure detection and coverage (Section 8.2). NetSonar leverages the inference algorithms developed in this line of work, but provides them as input the results of its planned testing.

Planned testing: Many researchers have used optimization techniques to plan tests. Bejerano and Rastogi [4] consider the problem of minimizing test agents and probes to cover every link. Nguyen and Thiran [22] extend these results to diagnose multiple failures. Kumar and Kaur [17] further explore test agent placement in the face of routing dynamics. Barford et al. [3] propose a time-variant weighted based cover algorithm. ATPG [28] extends traditional link covers to more general forwarding and access control rule covers. Huang et al. [13] evaluate the Nguyen and Thiran's approach in a controlled testbed; they point out several practical issues such as scalability and errors caused by a lack of consistent snapshots.

NetSonar improves existing planned testing methods in two ways. First, all these works cannot handle multipath networks. As we show in Section 4.3, handling multipath networks is non-trivial, as it requires techniques such as probabilistic path covers. Second, minimizing the number of test agents or probes may lead to *identifiability* problem, which means bad links are not uniquely identifiable. Existing approaches usually require iterative active probing [3] after problem detection, or direct probing of router interfaces [4, 17]. Our diagnosable link covers provide a simple, practical way to balance probing efficiency and identifiability.

Scalable end-to-end monitoring: Scalable end-to-end monitoring techniques infer end-to-end measures (e.g., latency) between *all* test agents using measurements taken between *some* test agents. Techniques include rank-based [6], SVD [7], and Bayesian experiment design [27]. In principle, the results of such methods could be fed into a network tomography solution to isolate high delay links. However, this is an indirect and unnecessarily complex way to do fault diagnosis. Further, the evaluations of these techniques only demonstrate their effectiveness for inferring end-to-end measures, not for fault localization.

Multipath measurement: Recent studies [1, 9, 23] point out that traditional host-based diagnosis tools such as ping and traceroute are inadequate for multipath networks. While Paris Traceroute [1] also varies packet headers like NetSonar in order to cover paths, it outputs a topology, not the tuples required for coverage. It also uses a *dynamic* scheme based on hypothesis testing to discover a multipath topology. By contrast, NetSonar uses *static* analysis of the topology to compute the number of random tuples needed.

Using other data sources: NetDiagnoser [10] takes control plane messages into account in Binary Network Tomography. Others [14, 15, 18, 19] focus on temporal correlation between different network events using statistical mining. These tools complement NetSonar by exploring several different dimensions.

In summary, the first contribution of NetSonar is test plan computation for large multipath networks; this is different from other work that computes test plans but cannot handle multipathing [4, 17, 21, 28] and require complete network knowledge. The second contribution of NetSonar is deployment experience in IDN, together with comparison with SNMP counter methods.

4. NETSONAR

We designed NetSonar to cover all links and routers in a global backbone network, mainly targeting performance problems such as packet loss and latency spikes. Today, global backbone networks typically use MPLS to establish many parallel label switched paths (LSPs) between two sites. NetSonar examines the network topology and LSP configuration to generate test plans with high link coverage. The plans are then executed by corresponding test agents—currently, ping and traceroute agents. Once the test data is collected, post-processing cleans the data, triangulates the problem and provides reports to human operators.

4.1 Overview

NetSonar works in three phases: computing covers, executing test plan, and localizing faults.

In the first offline bootstrapping phase, NetSonar computes a cover by reading the set of LSPs in the network.[1]

[1]In an IP (non-MPLS) network, NetSonar can use shortest paths or link weights to bootstrap.

It then uses its knowledge of routes and available test agents to compute a set of test probes that forms two covers: First, for each pair of test agents connected by N LSPs, it generates a *probabilistic path cover* by choosing k random TCP port-pairs to send test packets between them. If we assume that the (unknown) hash function in the edge router maps every 5-tuple (IP source/destination addresses, TCP source/destination ports, IP protocol type) to a route with equal probability, the relationship of N and k follows the analysis of the "coupon collector's problem" (Problem 1). Second, across the entire network, we generate a *diagnosable link cover* to cover all links – in other words, if a single link exhibits a performance fault, there is sufficient information for the tester to localize that link without further probing. This is done by selecting pairs of test agents so that each link is covered multiple times.

In the second online testing phase, for all these 5-tuples, NetSonar sends low-frequency traceroute and high-frequency ping *simultaneously*. Why is traceroute needed when we already know the paths through LSPs? First, LSP information may be outdated when the actual probes are sent out. More fundamentally, the LSP information is insufficient. Assume there are two paths $p1$ and $p2$ between S and D. Recall that a probabilistic path cover only guarantees a certain probability of covering both $p1$ and $p2$. However, because of the non-determinism, *for a specific packet*, the test agent does not know whether it is mapped to $p1$ or $p2$. Hence, its ping results cannot be mapped to the correct links. Traceroute maps each chosen 5-tuple (and ultimately associated pings) to a specific path. Note that traceroute triggers ICMP responses from routers; thus it cannot run very often, currently once every five minutes for each 5-tuple. At the same time, NetSonar sends ping probes for each 5-tuple chosen in the diagnosable link cover much more frequently, currently every 3 seconds between any pair of agents. We call this combined ping-traceroute approach *traceable probes*. Thanks to the traceroute done earlier, NetSonar knows the path taken by each ping packet.

Finally, in the offline analysis phase, NetSonar collects ping and traceroute results, and uses a fault localization algorithm to pinpoint the faulty spot.

The entire NetSonar workflow is an *open loop*. This is to ensure that even when forwarding information changes *between* phases, each phase can still capture relatively accurate information. For example, in the offline phase, the cover calculation does not depend on the traceroute results in the online phase. Similarly, in the final analysis phase, NetSonar does not need to issue more probes (back to the online phase) since each link has already been covered multiple times.

4.2 Components

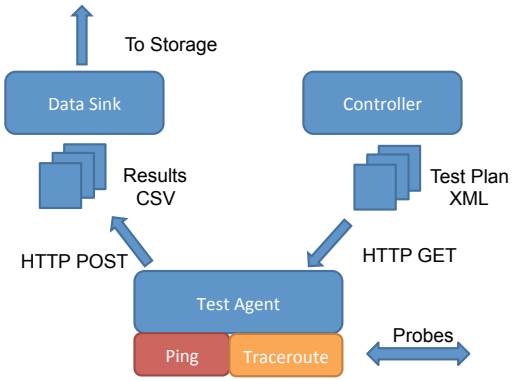

Figure 3: NetSonar Components

NetSonar contains three loosely coupled main components: test agents, the controller, and the data sink, as depicted in Figure 3.

Test agents: Test agents running on DC servers execute the test plan. Each test agent contains two test clients: a TCP traceroute client and a TCP ping client. Traceroute client collects only path information; ping client collects latency information.

We choose TCP ping and traceroute because TCP is the dominant traffic type, hence the test results closely reflect application-perceived performance. Each ping probe is a TCP SYN packet sent to an open remote TCP port. We measure the time gap between this SYN packet and the returned ACK packet as the latency. A TCP traceroute probe is a set of SYN packets with small TTLs, so that routers along the path can return ICMP Time-Exceeded messages.

Each test agent downloads its own test plan – a list of commands – periodically. The commands are executed by both traceroute and ping clients. Traceroutes and pings in the a single test use *the same* 5-tuples, so that we know the exact path for each ping result in a multi-path network. Notice that these traceable probes also provide certain level of robustness against path changes - even the LSP data is slightly outdated, we are still able to map ping results to correct paths at the time of testing. Both ping and traceroute intervals are configurable. The frequency of traceroute is chosen to avoid overwhelming routers.

Controller and data sink: The logically centralized controller periodically reads the network topology, LSP data, as well as the health of test agents, and generates test plans for individual test agents that cover all links and routers. The controller ensures that traceroute-triggered ICMP responses from any router and the number of commands assigned to any individual test agent are below certain safe thresholds. Moreover, the loss of individual test agents will not result in a large drop in coverage. The data sink collects the results in comma-separated values (CSV) from test agents, and uploads

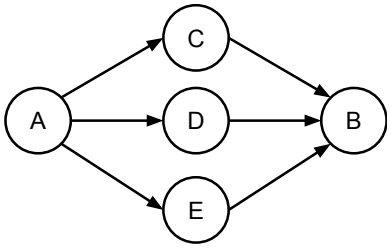

Figure 4: Source multipath. Source router A can select one of three paths ($N = 3$).

the aggregated data to data storage for analysis.

4.3 Probabilistic Path Covers

Following Paris traceroute [1], we disambiguate different load balancing paths by varying the 5-tuple headers of ping and traceroute probes between the same pair of test agents. A fundamental question is: without knowledge of the hash functions used by routers, how many different 5-tuples are needed to cover all links and interfaces between a pair of test agents?

4.3.1 Source Multipath

We start by examining the *source multipath* network, where load balancing only happens at the source router. A example source multipath network is depicted in Figure 4: router A is the source router that chooses path among $A-C-B$, $A-D-B$, and $A-E-B$. This load balancing approach is popular in LSP-based wide-area networks such as IDN, where multiple paths are set up between two edge routers, and only the edge router can decide the exact path of a particular class of packets.

Consider a router that has N next-hops for a class of outgoing packets. In the worst case, we need all possible 5-tuples assuming an unknown, adversarial hash function. Instead, we assume: (1) Each 5-tuple is hashed to a next-hop with a uniform probability of $1/N$; (2) Each tuple is treated by the hash function independently. The first assumption is reasonable because of the prevalent use of ECMP to spread the traffic uniformly; the second is reasonable because most router hash functions treat tuples statelessly. With these assumptions, the number of tuples needed can be derived from the Coupon Collector's problem [12]:

PROBLEM 1 (COUPON COLLECTOR'S PROBLEM). *Suppose that there are N different coupons, equally likely, from which coupons are being collected with replacement. What is the probability of collecting all N coupons in less than k trials?*

The analogy is easily seen: each next-hop is a different coupon; each sending of a probe with a specific 5-tuple is equivalent to drawing a coupon from the pool. Let T be the number of trials needed, we can calculate

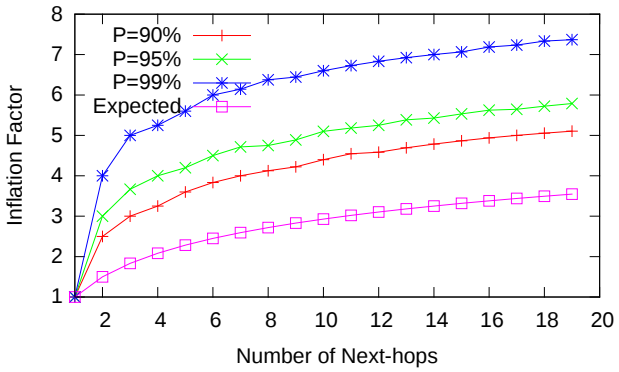

Figure 5: The number of 5-tuples needed to exercise N next-hops, with different confidence.

the expectation as follows:

$$\mathbb{E}(T) = N \sum_{i=1}^{N} \frac{1}{i} \sim O(N \log(N))$$

The expected value of T does not guarantee 100% coverage. One can calculate the exact probability:

$$\Pr(T \leq k) = \sum_{i=1}^{N} (-1)^{(i+1)} \binom{N}{i} \left(1 - \frac{i}{N}\right)^k, \text{ where } k \geq N$$

Figure 5 shows the number of 5-tuples needed to exercise different next-hops. We define the "inflation factor" as T divided by N. For instance, 75 randomly selected 5-tuples with inflation factor 4.7 can exercise 16 next-hops with 90% confidence. To achieve 99% confidence, we need approximately 110 5-tuples with inflation factor 6.9. Note that the percentage here represents the confidence to cover *all* next-hops, not the percentage of next-hops covered. The analysis above guides NetSonar in creating a test plan to cover all links and routers in a multipath network at a given confidence level. We call each set of 5-tuples between a pair of test agents a *probabilistic cover*.

We will show in Section 8.1 that when test agents are deployed at all sites, we can cover all links and routers with a smaller inflation factor (e.g., 2 in IDN). This is because even if some paths are missed between a pair of test agents, the links on these paths can still be covered by other test agent pairs. By contrast, a sparse deployment requires larger inflation factor.

4.3.2 Multilevel Multipath

Today's data center networks (such as a fat-tree network) often employ a more general load balancing scheme than source multipath, which we refer to as *multilevel multipath*. In multilevel multipath, routers are categorized into levels. A set of "symmetric" links connecting two levels share the traffic load *equally*. Figure 4 shows

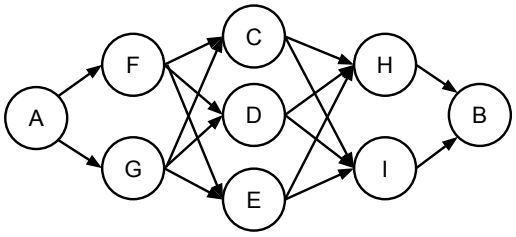

Figure 6: Multilevel multipath with 3 levels of edge, aggregation and core. N is determined by the maximum number of links between any two levels ($N = 6$).

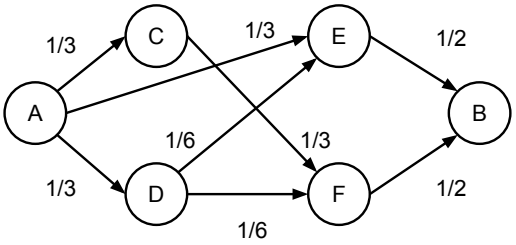

Figure 7: General multipath. N can be determined by the minimum traversal probability ($N = 6$).

a full-mesh, 3-level network: A and B are two edge routers; F, G and H, I are two aggregation levels; and C, D, E form the core level. All three levels can independently make load balancing decisions. For example, router F can choose C, D, or E as its next hop.

Observe that the number of unique paths grows exponentially – in this example, there are $2 \times 3 \times 2 = 12$ paths between A and B. However, this does *not* imply that we should plug $N = 12$ into Problem 1. This is because a packet sent from A to B has a probability of $1/2$ to traverse any particular link between edge and aggregation levels such as (A, F), and a probability of $1/6$ to traverse any link between aggregation and core such as (F, C). Hence, we need approximately $2 \log(2)$ and $6 \log(6)$ packets respectively to cover these links. Since the same packet will traverse all levels, we only need the maximum (not the sum or product) of these two numbers, which is equivalent to $N = 6$.

More generally, in a multilevel multipath network, the results in Problem 1 can apply where N equal to the *maximum* number of links between any two levels.

4.3.3 General Multipath

A natural way to generalize multilevel multipath is to remove the constraints of levels and symmetry. Such general multipath routing is common in large ISP networks. In general, the set of paths between a source and destination forms a single-source, single-sink directed acyclic graph, where each node represents a router, and each edge marks a (possible) forwarding direction. Figure 7 shows such an example. For simplicity, assume that each node sends packet to all its next hops with equal probability as in ECMP.

We first calculate the probability of a random packet sent from A to B traversing a particular link, called *traversal probability*:

1) Sort the nodes topologically so that any prior hop of a node X occurs before X in the sorted order.

2) Initialize the traversal probability of all links and nodes to 0 except for the source node which is assigned probability 1.

3) Start from the source node in topological order: for each node with probability of p and m outgoing links, assign p/k to each outgoing link and add p/m to each of its next-hop node's probability. Note that we can generalize to unequal load balancing (as in weighted cost multipath) by having each node add a weighted portion of its probability to its next-hop. We can add these probabilities because the events are disjoint: a particular packet must arrive from exactly one prior hop.

In Figure 7, each link is annotated with its traversal probability. For example, a random packet sent from A to B has probability of $1/3$ traversing link (C, F).

Next, we find the link l with minimum traversal probability p. In our example, l is either (C, E) and (C, F), both with $p = 1/6$. We can now use the coupon collector formula of Problem 1 with $N = \lceil 1/p \rceil$. This is because with $O(N \log N)$ packets we will (almost for sure) traverse l. This also implies that we will traverse *all other links* that have higher traversal probability. Note that this is a coarse upper bound; finding the exact number is known as the non-uniform coupon collector's problem [25].

4.4 Diagnosable Link Covers

A probabilistic path cover only covers all paths between *a pair* of test agents. NetSonar's goal is to generate a list of test targets for each test agent so that the overall set of tests forms what we call a *diagnosable link cover*. More formally, assume the network topology is a directed graph $G = (V, E)$, where V are nodes and E are links. Besides the topology, we also know a set of *paths*. Each path is a set of links that connects two nodes. We can treat each path as a subset of E, and choose a *minimum set of paths* to cover all links in E. This is known as a NP-Hard "Min-Set-Cover" problem. A well-known $O(N^2)$ approximation (hereafter referred to as MSC) can solve this problem, where N is the number of paths. MSC first initializes the uncovered edge set U to E. In each iteration, MSC greedily chooses the path p that covers the maximum number of uncovered links (maximize $|p \cap U|$), and removes links in p from U. The algorithm terminates when there are no links left in U.

```
U ← E
# Initialize counters
for link ∈ U do
    visited[link]← 0
repeat
    path, score ← FIND_MAX(U, paths)
    paths ← paths − path
    if score > 0 then
        for link ∈ path do
            visited[link]++
            if visited[link] == α then
                U ← U − link
until U == ∅ or score == 0 or paths == ∅
```

Figure 8: MSC-α algorithm. A link is only removed when it is covered at least α times.

NetSonar modifies the basic MSC algorithm to handle multipath and to enable open-loop diagnosis. To handle multipath, we group the paths by source/destination pair. In each step, *either* all links in the probabilistic cover P for a source-destination pair are chosen, *or* none of them are chosen. This is because we cannot control the coverage of an individual path in a probabilistic cover. Furthermore, recall that the number of probes in the probabilistic cover between a pair of test agents depends on the number of paths between them. Hence, when choosing a probabilistic cover P in each iteration, instead of using $|P \cap U|$ as the scoring function, we normalize $|P \cap U|$ by the number of probes to favor a P that covers new edges most economically.

We need one more twist for open-loop diagnosability. The original MSC aims to minimize the overall number of probes. However, this can introduce ambiguity in fault localization. For example, if two links l_1 and l_2 are *only* exercised by path p but no other paths, in the original MSC, both links are considered "covered" and no new path will try to exercise these links. If p encounters latency problems, it is not clear whether l_1 or l_2 is the culprit. Traditionally, such ambiguity is removed by sending additional probes after performance problems are discovered [27, 28]. However, in a large network like IDN, such close-loop, multi-iteration diagnosis can be difficult to implement due to the additional latency in the measurement data processing pipeline.

NetSonar takes a different approach. We introduce a parameter α. NetSonar uses MSC-α (Figure 8) which requires each link to be exercised by *at least* α times, except when less than α paths can exercise a link, in which case the link is exercised maximum possible times ($\leq \alpha$). This modification can be easily done by adding a counter to each link initialized to 0. As shown in Figure 8, we increment the counter every time a path is chosen that covers the link, and only remove the link when the counter reaches α. Note that MSC-1 is equivalent to the original MSC, and MSC-∞ is "all-pairs" where all paths are exercised.

The intuition behind MSC-α is that by requiring each link to be covered α times, it is more likely to be covered in different combinations of links as part of different paths. Thus, we can distinguish a culprit link from other healthy links (during fault localization) by observing the health of different paths. Although MSC-α is very simple, it is a practical way to improve what is referred to as *identifiability* in the tomography literature [5], leveraging the fact that NetSonar can control the generation of test probes. As we show in Section 8.3, MSC-α also improves resilience to path changes.

5. DATA ANALYSIS

After computing covers and gathering path and latency data, NetSonar analyzes the data as follows. First, NetSonar cleans the ping and traceroute data to reduce noise caused by missing or inaccurate test probes. Next, NetSonar integrates information from other sources, such as topology, LSPs and device configurations, and stores the resulting data in an analysis database. Finally, it detects and localizes network performance faults.

Noise reduction: One common source of noise in our traceroute data is "unknown" hop, due to missing ICMP response messages. ICMP messages may be missing for various reasons: e.g., a misconfigured router or packet drops caused by an overloaded router or server.

We found all routers in IDN use the incoming interface IP address as the source address in the ICMP message. [2] This allows us to reconstruct most missing traceroute hops using our knowledge of the topology: if a router r is missing, but the incoming interface I of the next hop router responds, we can recover r by identifying the interface that corresponds to the remote end of I in the network topology.

Another common source of noise is inflated latency measurements caused by server load. Latency values up to 10-times the average appear frequently in our data. To deal with this, we discard the top 1% of all latency measurements, i.e., we only take the 99th percentile value as a data point during a ping aggregation window. If the latency in the current aggregation window exceeds a predefined threshold compared to the average of the last 3 data points, we consider it a *latency spike*.

We identify two common types of faults in IDN as follows. For each of them we develop a detector to process the data NetSonar collects.

Latency inflation detector: Latency spikes may arise as a result of packets taking a longer path. By mapping each ping to a specific path, we can identify such latency inflation by comparing the paths before,

[2]This behavior is actually non-standard: RFC1812 requires the ICMP message's source address to be one of the *outgoing* interfaces.

during and after each latency spike.

Troubled link detector: Latency spikes that cannot be explained by route changes may indicate a "troubled" link along the path. We can identify the culprit by triangulating the latency results collected on different paths. While other inference algorithms do exist [10,11,21,27], we chose Sherlock [2] for its relative simplicity and robustness to noise (recall that our test agents run on busy production servers).

We feed latency spikes, along with their associated path information, into Sherlock to locate faulty links. Sherlock considers *all* possible failure scenarios (assuming at most m simultaneous faulty links), and assigns a score to each scenario to quantify how well it matches the behaviors we observe. For the single failure case ($m = 1$), Sherlock works as follows starting with a list of all spikes and paths.

1) In each iteration, assume only one link is "bad". Any path that traverses the "bad" link is also labeled as "bad". All other paths are labeled as "good".

2) After all paths are labeled, calculate the following: (a) e_f: explained failures, the number of paths that encounter latency spikes and are labeled as "bad"; (b) u_f: unexplained failure, the number of paths that encounter latency spikes but are labeled as "good"; (c) u_s: unexplained success, the number of paths that have no latency spike but are labeled as "bad".

3) Given a link labeled as "bad", calculate a score $S = e_f/(e_f + u_f + u_s)$. When this iteration finishes, return to Step 1 and try other candidate links until all links have been tried.

4) Output all the links ordered by their scores from high to low.

The process above can be easily modified to handle multiple faulty links; details are in [2].

6. IMPLEMENTATION

Each NetSonar component is kept simple for reliability and to minimize load on the servers. Every 30 minutes, the controller generates test plans in XML files and serve them via a web server. Every test agent periodically reloads its own XML file from the controller and runs the set of tests. Results are uploaded to the data sink, which in turn uploads them to a distributed file system on which the analysis code is run directly.

Controller: The NetSonar controller, implemented in C#, contains two parts: a file generator and a web server. The file generator periodically reads network topology and LSP information. It then uses the test plan generation algorithms described before to select paths that can cover all links. Finally, these paths are clustered by the source test agents; each test agent has a dedicated XML file containing a list of test targets.

Figure 9 shows an example XML file that is run by Host A's test agent. In this example, Host A should send traceroute every 300 seconds to Host B with remote port 1100 and local port 50000. The "physicalIP" field is used to avoid unnecessary DNS queries from a large number of agents. The "level" field identifies a test level. For example, a test agent can participate in both "global" and "north_america" tests. Data reported to the data sink is tagged with the "level" field. The web server hosts all the XML files so they can be downloaded by test agents; it does not participate in test plan generation.

Test agent: Each test agent, implemented in C++, contains a management module and several test clients. Periodically, the management module downloads its XML file from the controller, and dispatches the tests to the two test clients used by NetSonar: TCP ping and TCP traceroute. The module is also responsible for uploading test results to the data sink.

Data sink: The NetSonar data sink is also a web server. The test agents `POST` 2KByte test results each time in the form of comma-separated values, including timestamp, source/destination IP and port, latency, path, and other metadata. The results are uploaded to a distributed file system for post-analysis.

Practical considerations: Being part of a global DC infrastructure, NetSonar is implemented for scalability and reliability. First, both the controller and the data sink are implemented as simple, distributed web services. For example, each logical controller is actually a cluster of machines that hosts the same set of XML files. The HTTP `GET` requests from test agents are automatically load balanced across different machines, to avoid the failure of a single machine bringing down the entire system. A watchdog program monitors the health of machines and removes bad ones from the system.

Second, the test agent is fault tolerant. To avoid a malfunctioning controller disrupting the entire network, test agents have a local hard limit on the number of tests they can generate per minute. If an agent loses its connection to the controller or the data sink, it uses the cached XML file to continue testing and buffers the results locally, but will stop testing after a certain period of time. Following conventional fault-tolerant design practice, test agents, controllers, and data sinks are distributed across different availability zones, and do not share top-of-rack switches or power supplies.

Finally, the HTTP interface between different components uses plain-text and is standard-compliant, facilitating debugging and profiling. It also allows components to evolve independently. For example, we have developed new test plan generation algorithms in the controller without touching the data sink or test agents.

7. DEPLOYMENT AND EVALUATION

We deployed NetSonar as part of IDN's DC management system. As a result, test agents are automat-

```
<Testlist server="A" majorVersion="1" minorVersion="0">
  <Peer uri="test://B:1100:50000" physicalIP="10.0.0.1" interval="300000" level="global"/>
  <Peer uri="test://C:1100:50000" physicalIP="10.0.0.2" interval="300000" level="global"/>
</Testlist>
```

Figure 9: Example XML file for a test agent.

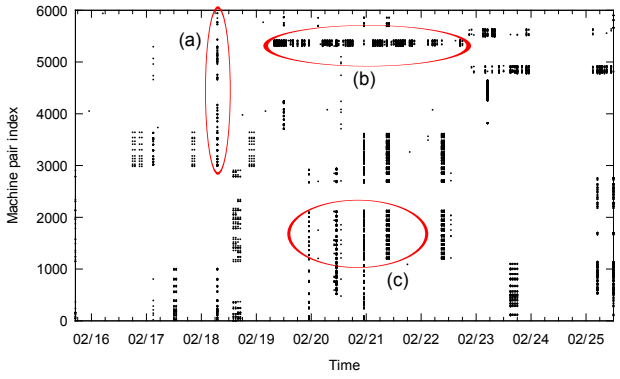

Figure 10: Spikes detected by NetSonar, showing (a) vertical strips, (b) horizontal strips, and (c) periodic spikes. Neighboring devices are given consecutive indexes.

ically deployed to *all* hosts in a DC and co-exist with other production services. This approach is unlike some other monitoring systems, which only deploy a small number of dedicated vantage points. It helps to cover more routers and links. However, these machines may be also running other services such as web indexing, which means that their test results can be inaccurate when the servers are busy.

Four machines, acting as both the controller and the data sink, manage the entire NetSonar system. Every 30 minutes, the controller extracts the network topology and LSP information from various data sources and re-generates XML files for test agents. Our current deployment is limited to a modest number (8) of DCs.

We report on incidents captured by NetSonar during a one-month period spanning Feb and March, 2013.

7.1 Failure Characteristics

Part of the incident heatmap generated by NetSonar is shown in Figure 10. Each black dot indicates the time and the machine pair that experiences a latency spike. The y-axis is machine-pair index number ordered first by the source host name and then by the destination host name. Due to this naming scheme, hosts in the same DC will be adjacent to each other. In the graph, vertical strips indicate an incident affecting many machine pairs at the same time, and the horizontal strips indicate an incident affecting a small number of machine pairs but for a long time.

We observed the following using NetSonar's output:

Frequent spikes: To reduce false positives, we use a conservative spike detection threshold of 50 ms, which means that each dot represents at least a 50 ms latency jump from the average of past 3 data points. Each data point is the 99th percentile latency during a one-hour ping aggregation window. Even with such a conservative threshold, over the entire one-month period, there are over 40,000 machine-pair spikes. Each spike is not a unique incident, as an incident can cause many spikes (across time or machine-pairs).

Bad agents: To increase coverage, NetSonar agents co-locate with production machines. Even if we used dedicated machines as test agents, performance impacting incidents such as software updates, hardware failures, and power cycling are inevitable. Hence, some test agents experience sustained periods of high latency. The horizontal strip (b) in Figure 10 indicates one such "bad test agent" during Feb 19 - 22. The width of the strip indicates that this test agent experienced high latency when communicating with all other test agents.

Periodicity: There are some periodic, vertical strips (c) in the lower half of Figure 10. The interval between the spikes are roughly 12 hours, at 12AM and 12PM respectively. Moreover, the spike incident was DC wide. This is likely to be caused by periodic application level activities, such as website indexing. Some of these activities affect test agents directly, while others generate cross-traffic affecting test agents.

7.2 Validation

To validate the root cause of spike incidents inferred by NetSonar, we manually cross-verify the top 5 suspects of each incident using other data sources such as SNMP counters. NetSonar captured 140 incidents during Feb-March, 2013. Of these incidents, 84 were localized to edge devices. That means either the top suspect was the machine itself or there were local issues *inside* the DCs. We could not validate these incidents due to the lack of reliable, independent information for cross-verification. The remaining 66 incidents were localized to core devices. The most common problem was troubled link, e.g., due to large, bursty inter-DC transfers. 56 out of 66 incidents can be validated using SNMP counters and other data sources; that is, these other data sources confirmed that the router interface inferred as culprit by NetSonar was indeed overloaded or drop-

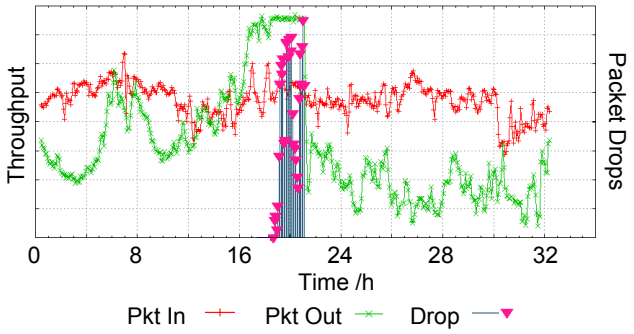

Figure 11: A congestion incident captured by NetSonar. SNMP counters show packet drops. This is due to a sudden increase of outgoing packets. Only relative scales are shown.

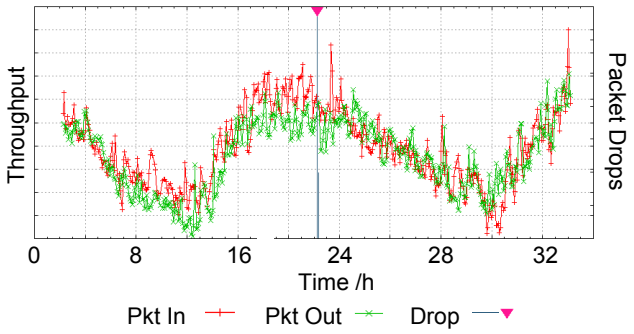

Figure 12: A sudden packet drops within just a few minutes. However, the root cause is unclear since the incoming and outgoing traffic are stable according to SNMP counters.

ping packets (without being overloaded) at the time of the incident.

We cannot validate the remaining 10 core incidents with the data sources we have. This does not necessarily mean that these problems were incorrectly localized. Due to the lack of the ground truth, we can only validate NetSonar alerts that correlate well with other available data sources.

To provide insight into the nature of faults, we report on 3 typical incidents in detail.

Case 1: Congestion Figure 11 illustrates the router counters in a typical troubled link incident reported by NetSonar. The outgoing traffic increases by 2x in just a few minutes and packets started being dropped. Noticing the increased latency due to the congestion, NetSonar successfully triangulated to this interface. This is likely due to an unplanned inter-DC activity initiated by the application. The traffic went back to normal after about 3 hours.

Case 2: Packet drops without obvious reasons Figure 12 shows a packet drop incident without obvious reasons. NetSonar found that the latency between

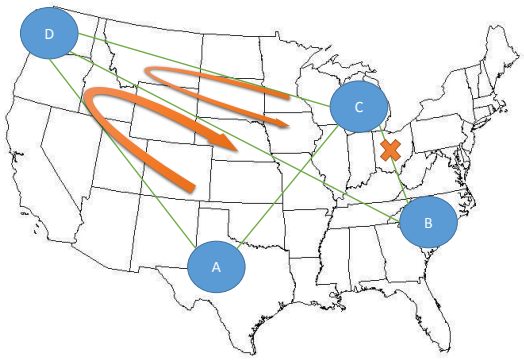

Figure 13: A fiber cut between two DCs C and B causes the traffic to reroute to west coast. Figure does not represent the actual DC locations

various DCs had increased. It narrowed down the suspect to the link between two DCs. Validation showed a short-lived packet loss of hundreds of packets per second for around 5-10min, which triggered the 99th percentile latency spikes. However, the root cause was not clear from SNMP counters alone, because the traffic was relatively stable during this period. In NetSonar we do not directly measure packet loss. Instead, packet loss is captured as higher latency due to TCP retries.

Case 3: Fiber cut Due to a maintenance-related activity by a circuit provider, the link connecting two sites B and C (Figure 13) lost 20 Gbps of bandwidth. Because there are multiple links between B and C, and only a portion of the links were cut, one might not correctly localize the problem with just ping and traceroute. In fact, NetSonar captured the latency increase between these two DCs using the troubled link detector and pinpointed the culprit as devices in D since they appeared in several spikes. However, at the same time, NetSonar's latency inflation detector captured the route changes from B-C to B-D-C, and A-C-B to A-D-B. By combining both results, we were able to correctly understand what was happening.

7.3 Comparison with SNMP counters

Today, many large network operators still rely on SNMP counters as a primary tool for network monitoring, which can be surprising given the significant progress in network tomography and other monitoring techniques. In our evaluation, SNMP counters are also an important data source to cross-validate the findings of NetSonar. However, we find that *solely* using SNMP counters for monitoring, without additional testing, is inefficient and often misleading.

To illustrate the shortcomings of SNMP counters, we

compared alerts from April, 2013 reported by SNMP counters and NetSonar. SNMP counters are polled every 5 minutes, while NetSonar's ping aggregation window is 60 minutes. To enable fair comparison, we aggregated all SNMP alerts in a 60-minute window as *one* alert. The SNMP alerts were generated based on two thresholds: 1) when link utilization exceeds 90% of capacity; 2) when the error or drop rate is higher than 1,000 packets per minute. These criteria are extracted from the network operator's alerting system.

We found 36 NetSonar alerts during this period, where 30 of them could be verified by SNMP counters; we could not identify the root cause of the remaining 6 latency incidents. On the other hand, we saw 1,052 high utilization alerts, and 2,091 error/drop alerts from SNMP counters, most of which do not have any impact on end-to-end latency as measured by NetSonar. The total number of SNMP alerts is almost 87x the number of NetSonar alerts. It equates to 104.8 alerts per day, which is well beyond what operators can handle.

The key reason for such a high false positive rate is that SNMP alerts are generated based on 5-minute aggregate data and hence does not always reflect packet-level end-to-end performance. For example, a 90%+ link utilization may look high but if the traffic is smooth, there could be very little queuing or congestion. Similarly, 1,000 error/dropped packets per minute may look high. But consider a 10 Gbps link rate and 1,500 bytes packet size; the average error/drop rate over the 5-minute interval is merely 0.002% which is again barely noticeable from the perspective of an individual flow. Raising the two thresholds would reduce false positives, but it would also dramatically increase the false negatives and cause us to miss most of the NetSonar alerts. In fact, there simply does not exist an ideal "threshold" that can achieve both low false positive rate and low false negative rate. This problem was also echoed in our conversations with IDN operators.

8. SIMULATION

In this section, we evaluate NetSonar's ability to locate troubled links using simulations. We first evaluate the effectiveness of probabilistic path covers and diagnosable link covers. We then study the impact of latency spike threshold, measurement noise, ping aggregation window, and traceroute frequency.

The topology and MPLS configuration used in our simulations are based on IDN. In all tables below, the "$\leq N$" columns mean the fraction of trials in which NetSonar captures the culprit in the top N suspects. All results are based on 1,000 trials. The "Total" and "Max" columns denote the total number of probes in the network and the maximum number of probes traversing a single router *in one "round"* (during which all links are tested at least once). Unless stated otherwise, we pick an inflation factor of 2 for probabilistic covers, use MSC-2 in diagnosable covers, and simulate a single faulty link.

Inflat. factor	Accuracy (%)			Probes (#/round)	
	≤ 1	≤ 2	≤ 3	Total	Max
0.5	76.9	82.4	84.7	2,401	280
1	89.8	94.1	96.1	4,813	582
2	92.1	96.0	98.8	9,626	1,100
3	95.7	97.6	99.8	14,439	1,653

Table 1: Accuracy and overhead of probabilistic covers. "Inflation factor" indicates the number of 5-tuples chosen per site pair, normalized by the number of LSPs between two sites.

Algorithm	Accuracy (%)			Probes (#/round)	
	≤ 1	≤ 2	≤ 3	Total	Max
MSC-1	74.6	87.1	94.9	8,073	800
MSC-2	92.1	96.0	98.8	9,626	1,100
MSC-3	96.0	98.8	99.9	18,957	2,206
All-pairs	98.5	99.3	99.9	51,621	7,054

Table 2: Accuracy and overhead of diagnosable covers with different α values.

8.1 Accuracy and Overhead

Probabilistic path covers: A large inflation factor can increase the path coverage, but it also adds probing overhead. Table 1 shows how inflation factor in probabilistic cover affects performance. The total and maximum number of probes grows almost linearly with the inflation factor. NetSonar can achieve high accuracy when inflation factor > 1. This is because test agents are present in all sites. Even when some paths are not covered by one test agent pair (because a relatively small inflation factor is chosen), MSC-2 ensures that there are some other test agent pairs which will exercise the links on those uncovered paths.

Diagnosable link covers: α determines how redundant a diagnosable link cover is. Table 2 compares the diagnosable link covers under different α values.

Overall, MSC-2 attains a good balance between overhead and accuracy. Compared to MSC-1, MSC-2 increases the accuracy to over 90% with only 1.4x probing overhead. Further increase in α has little effect on accuracy and only results in larger probing overhead.

Multiple faulty links: Multiple simultaneous faulty links, while less common, can occur in a large network. We evaluate how NetSonar performs with multiple faulty links by selecting top N or top X% suspects inferred by Sherlock. Table 3 shows that, with 2 or 3 faulty links, NetSonar may not be able to accurately localize the culprits to the top 2 or 3 suspects. However, in over 90% of the cases, NetSonar can still capture the

N	Accuracy (%)			
	$\leq N$	0.5%	1%	2%
1	92.1	100	100	100
2	46.4	85.1	94.5	97.6
3	19.1	56.2	79.6	92.9

Table 3: Accuracy with multiple failures. N denotes the number of simultaneous failures. The percentage in "Accuracy" column denotes the failures fall in top X% suspects.

Algorithm	Accuracy (%)			Coverage (%)
	≤ 1	≤ 2	≤ 3	
NetSonar	92.1	96.0	98.8	100
All-pairs +Single Path	60.9	68.4	73.8	78.2
Random pairs +Prob. Cover	71.9	79.7	82.8	84.5

Table 4: NetSonar provides higher accuracy and coverage compared to either ignoring multipathing or using unplanned tomography.

culprits within the top 2% candidates. In other words, NetSonar can still successfully help operators eliminate a vast majority of the links from the suspect set.

8.2 Multipathing and Planned Tomography

We compare NetSonar's coverage techniques to two simple alternatives suggested by the existing literature. First, most existing planned tomography schemes (e.g., [4, 28]) assume a single forwarding path between sites, and do not consider multipathing. To simulate this, we pick one random path from the set of paths between two sites for measurement. Table 4 shows that, even with these all-pairs measurements, ignoring multipath causes diagnosis accuracy to drop by more than 30% compared to NetSonar. The lack of probabilistic path covers and the single path assumption results in only 78.2% of links being covered.

Second, we assume a probabilistic cover for multipathing and consider using unplanned tomography, where the measurements are not designed using a test plan (Section 3). We simulate unplanned tomography by randomly picking the same number of pairs of test agents as NetSonar's diagnosable link cover. The random choice simulates the lack of planning. The last row in Table 4 shows that accuracy is 20% lower than NetSonar with the same testing overhead, again due to reduced link coverage. These experiments suggest that we need to incorporate *both* multipathing and planning to achieve better coverage.

8.3 Robustness and Parameters

False negatives (%)	Accuracy (%)		
	≤ 1	≤ 2	≤ 3
0	92.1	96.0	98.8
5	91.5	95.6	97.9
10	91.0	95.2	97.6
20	90.6	94.5	96.6

Table 5: Accuracy as false negative rate varies due to threshold selection.

Bad agents (%)	Accuracy (%)		
	≤ 1	≤ 2	≤ 3
0	92.1	96.0	98.8
0.25	90.0	94.6	95.4
0.5	79.3	88.0	92.7
1	44.7	59.2	72.9

Table 6: Accuracy as agent noise varies.

False Negatives in Measurement: To detect a latency spike, we use a threshold to compare the 99th percentile latency in the current aggregation window to the average of the past three data points. To prevent raising too many *false positives*, we pick a fairly conservative threshold (50 ms) to filter out most small latency variations. However, such a conservative threshold may generate *false negatives*, i.e., "bad" probes that traverse a faulty link are mistakenly labeled as "good". In the following simulations, we randomly and deliberately relabel a certain percentage of "bad" probes as "good".

Table 5 shows how NetSonar performs under different percentage of false negatives. Surprisingly, even with 20% of false negatives, the ≤ 3 accuracy is still above 95%. This is because with MSC-2, one faulty link will be covered by multiple probes between different test agent pairs. Thus, a small percentage of false negatives will have little impact on fault localization accuracy, as long as the faulty link is captured by a sufficient number of other probes. This result also indicates that a conservative latency spike threshold works well in practice.

Bad agents: False positives can arise from bad agents, which would nudge Sherlock away from the actual culprit. In this simulation, we assume a certain fraction of bad agents which will distort latency measurements, with a probability varying from 10% to 90%. Table 6 shows that 0.5% of bad agents will cause noticeable drop in localization accuracy. Because of the use of 99th percentile latency in spike detection, even 10% of distorted latency measurements will trigger false positives.

We identify bad agents by examining all latency measurements from the same agent. If an agent reports latency spikes from all of its probes (to different targets), we will discard all of its latency measurements. By applying this simple trick, we can restore the localization

Path changes	≤ 2 Accuracy (%)		
	MSC-1	MSC-2	MSC-3
0	87.1	96.0	98.8
1	74.6	88.6	95.3
2	72.6	84.0	93.7
3	65.2	81.3	84.8

Table 7: Accuracy as the number of path changes per aggregation window varies for various values of α in diagnosable link cover. Only ≤ 2 accuracy is shown.

TR Period (min)	Accuracy (%)			Overhead (pps)	
	≤ 1	≤ 2	≤ 3	Total	Max
0	92.1	96.0	98.8	∞	∞
5	89.3	95.8	98.1	32.1	3.7
10	87.5	94.1	96.3	16.0	1.8
20	82.4	90.3	94.0	8.0	0.9

Table 8: Accuracy as traceroute frequency varies

accuracy to the level when there is no false negative (first row in Table 6).

Path changes within aggregation windows: Path changes may occur during an aggregation window. In other words, the latency measurements may actually correspond to multiple paths. In the following simulations, we introduce path changes by randomly picking a new LSP within the same pair of test agents. We then vary the number of path changes in an aggregation window. When a latency spike is detected, NetSonar will blame *all* paths appearing in the same window.

Table 7 shows how path changes affect accuracy and overhead. The first row represents the ideal case where we can exactly map each latency measurement to the correct path. When the number of changes grows, accuracy drops. We see that the diagnosable link cover provides extra protection against path changes. For example, in the last row, MSC-2 has 16.1% higher accuracy compared with MSC-1 when there are 3 path changes in the same aggregation window. In a diagnosable link cover, the same link is tested independently by multiple probes. Even if probes between certain pairs of test agents fail to map to a correct link, other probes traversing the same link can still provide accurate identification of a troubled link.

Path changes between traceroutes: If a path's lifetime (from the time when a new path is set up to the time when the path is torn down) is shorter than the traceroute probing period, such a path will not be captured by traceroute and some latency measurements may be mapped to incorrect paths. In the following simulations, we again introduce path changes as in the last experiment but with a median life time of 15 minutes. In Table 8, the first row (0 min TR period) denotes the ideal case where we can associate each ping with the correct path at the cost of traceroutes sent at an infinite frequency. We can see a traceroute period between 5 and 10 minutes strikes a good balance between overhead and accuracy.

9. LIMITATIONS

Matching measurements: NetSonar combines ping and traceroute but ping measures round-trip *latency*, while traceroute only reports forward *path* information. This mismatch can lower the diagnosis accuracy - the long ping latency caused by bad reverse paths may be incorrectly attributed to the forward path reported by traceroute. This problem can be solved by applying One-Way-Ping [26], or probing reverse path information with a reversed 5-tuple (swapping source/destination IP and TCP port) by the receiving test agent.

Single administrative domain: NetSonar assumes the entire network belongs to the same administrative domain and that the network operator knows the topology as well as basic forwarding information (such as LSPs) in advance. Otherwise, the host selection algorithm cannot perform path and link covers, which may reduce the coverage and diagnosis accuracy. In this case, the system has to fallback to all-pairs probing, where no attempt is made to reduce probing overhead.

IP level probing: NetSonar relies on IP level probing (traceroute) to map pings to physical paths. For some devices, such probing may not be as useful. For example, the Ethernet level link aggregation groups (LAGs) prevent traceroute from discovering the physical links used, since the entire group is a single IP link.

Uneven hashing: In Section 4.3, we assume that hash functions always distribute traffic to all next-hops uniformly, or at least with a known distribution as in weighted cost multipath (WCMP). This may not be true in some cases. For example, a broken hash function can create unknown, non-uniform traffic distribution, which may lower NetSonar's path/link coverage.

Intra-DC diagnosis: NetSonar performs well in troubleshooting inter-DC performance problems, and in theory it can also be used in intra-DC fault localization. However, our trial deployment experience reveals that while the cover algorithms work, there are still several practical difficulties. First, intra-DC latency is extremely small (usually much less than 1ms), hence the measurements are much more sensitive to machine "hiccups". By comparison, inter-DC latency is usually more than 20 ms. Second, intra-DC distributed applications, such as web indexing, can simultaneously overload many devices. Such behavior is likely to confuse our fault localization algorithm.

10. CONCLUSION

NetSonar is a large-scale network tester: it treats the whole network as a device-under-test, and automatically generates high-coverage test plans. NetSonar handles the complexities of real networks including changing paths and multipath routing. NetSonar is also a *gray box* tester: it utilizes partial forwarding information and deals with the remaining uncertainty by computing efficient probabilistic and diagnosable covers that allow diagnosis in a single pass. Both probabilistic and diagnosable covers are general ideas that apply to networks that belong to a single administrative domain. Beyond these ideas, to the best of our knowledge, NetSonar is the first tester to report deployment experience in a large operational inter-DC network beyond simulations [3, 17] and testbeds [13].

Many aspects of NetSonar's design are driven by imperfect knowledge of the data plane today. We do not know the hash functions used by multipath routing; so we need to use traceroute to map pings to paths. Router CPUs are underpowered and responsible for other critical tasks such as route computation; so we can only infrequently harvest snapshots of the data plane to bootstrap testing. One might argue that these problems are temporary, and will disappear with newer router hardware and the stronger consistency between data and control planes that SDN provides.

However, even in a future network with perfect technology, we believe some uncertainty is inevitable, at least for non-technical reasons such as policies and organizational boundaries. Thus, gray box testing will still be needed to achieve trustworthy results. Moreover, our experience with NetSonar shows that, contrary to what one may think at first, accounting for uncertainty is practical. It can be done with minimum overhead while achieving high accuracy. We hope that NetSonar is a first step towards building tools that balance what can be known about networks with their unknowns.

11. REFERENCES

[1] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the internet. IMC '07, pages 149–160. ACM, 2007.

[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. SIGCOMM '07, pages 13–24. ACM, 2007.

[3] P. Barford, N. Duffield, A. Ron, and J. Sommers. Network performance anomaly detection and localization. INFOCOM'09, pages 1377–1385, 2009.

[4] Y. Bejerano and R. Rastogi. Robust monitoring of link delays and faults in IP networks. *Networking, IEEE/ACM Transactions on*, 14(5):1092–1103, 2006.

[5] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: recent developments. *Statistical Science*, 19:499–517, 2004.

[6] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. SIGCOMM '04, pages 55–66. ACM, 2004.

[7] D. Chua, E. D. Kolaczyk, and M. Crovella. A statistical framework for efficient monitoring of end-to-end network properties. SIGMETRICS '05, pages 390–391. ACM, 2005.

[8] I. Cunha, R. Teixeira, N. Feamster, and C. Diot. Measurement methods for fast and accurate blackhole identification with binary tomography. IMC '09, pages 254–266. ACM, 2009.

[9] I. Cunha, R. Teixeira, D. Veitch, and C. Diot. Predicting and tracking internet path changes. SIGCOMM '11, pages 122–133. ACM, 2011.

[10] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. Netdiagnoser: troubleshooting network unreachabilities using end-to-end probes and routing data. CoNEXT'07, pages 18:1–18:12. ACM, 2007.

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

[12] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, Inc., 1968.

[13] Y. Huang, N. Feamster, and R. Teixeira. Practical issues with using network tomography for fault diagnosis. *SIGCOMM Comput. Commun. Rev.*, 38(5):53–58, Sept. 2008.

[14] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. SIGCOMM '09, pages 243–254. ACM, 2009.

[15] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson. Studying black holes in the internet with hubble. NSDI'08, pages 247–262. USENIX Association, 2008.

[16] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. IP fault localization via risk modeling. NSDI'05, pages 57–70. USENIX Association, 2005.

[17] R. Kumar and J. Kaur. Practical beacon placement for link monitoring using network tomography. *Selected Areas in Communications, IEEE Journal on*, 24(12):2196–2209, 2006.

[18] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B. Huntley, and M. Stockert. Rapid detection of maintenance induced changes in service performance. CoNEXT'11, pages 13:1–13:12. ACM, 2011.

[19] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J. Wang, Z. Ge, and C. T. Ee. Troubleshooting chronic conditions in large ip networks. CoNEXT'08, pages 2:1–2:12. ACM, 2008.

[20] H. Nguyen, R. Teixeira, P. Thiran, and C. Diot. Minimizing probing cost for detecting interface failures: Algorithms and scalability analysis. INFOCOM'09, pages 1386–1394, 2009.

[21] H. Nguyen and P. Thiran. The boolean solution to the congested ip link location problem: Theory and practice. INFOCOM'07, pages 2117–2125, 2007.

[22] H. X. Nguyen and P. Thiran. Active measurement for multiple link failures diagnosis in ip networks. PAM'04, pages 185–194, 2004.

[23] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush. From paris to tokyo: On the suitability of ping to measure latency. IMC '13, pages 427–432. ACM, 2013.

[24] All-pairs ping service for PlanetLab ceased. `http://lists.planet-lab.org/pipermail/users/2005-July/001518.html`.

[25] H. V. Schelling. Coupon collecting for uneqal probabilities. *The American Mathematical Monthly*, 61(5):pp. 306–311, 1954.

[26] S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, and M. Zekauskas. A One-way Active Measurement Protocol (OWAMP). RFC 4656 (Proposed Standard), Sept. 2006.

[27] H. H. Song, L. Qiu, and Y. Zhang. Netquest: a flexible framework for large-scale network measurement. *IEEE/ACM Trans. Netw.*, 17(1):106–119, Feb. 2009.

[28] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown. Automatic test packet generation. CoNEXT '12, pages 241–252. ACM, 2012.

[29] M. Zhang, C. Zhang, V. Pai, L. Peterson, and R. Wang. Planetseer: internet path failure monitoring and characterization in wide-area services. OSDI'04, pages 12–12. USENIX Association, 2004.

[30] Y. Zhao, Y. Chen, and D. Bindel. Towards unbiased end-to-end network diagnosis. *IEEE/ACM Trans. Netw.*, 17(6):1724–1737, Dec. 2009.