StrobeLight: Lightweight Availability Mapping and Anomaly Detection

James W. Mickens, John R. Douceur, William J. Bolosky
Microsoft Research
mickens,johndo,bolosky@microsoft.com

Brian D. Noble
University of Michigan
bnoble@umich.edu

Abstract

Large-scale distributed systems span thousands of independent hosts which come online and go offline at their users' whim. Such availability flux is ostensibly a key concern when systems are designed, but this flux is rarely measured in a rich way post-deployment, either by the distributed system itself or by a standalone piece of infrastructure. In this paper we introduce StrobeLight, a tool for monitoring per-host availability trends in enterprise settings. Every 30 seconds, StrobeLight probes Microsoft's entire corporate network, archiving the ping results for use by other networked services. We describe two such services, one offline and the other online. The first service uses longitudinal data collected by our StrobeLight deployment to analyze large-scale trends in our wired and wireless networks. The second service draws live StrobeLight measurements to detect network anomalies like IP hijacking in real time. StrobeLight is easy to deploy, requiring neither modification to end hosts nor changes to the core routing infrastructure. Furthermore, it requires minimal network and CPU resources to probe our network of over 200,000 hosts.

1 Introduction

As distributed systems are built at increasingly larger scales, it becomes more difficult to understand the relationship between host availability and distributed system performance. Loosely coordinated, independently administered hosts display a wide variety of availability patterns [8, 10, 25]. Providing robust services atop this churning substrate requires substantial effort during the design and implementation of the distributed system. Thus, all distributed systems are guided by at least a crude characterization of host availability in the deployment environment.

Unfortunately, once these systems are deployed, they rarely include a component for collecting and analyzing system-wide, fine-grained availability data. Historical availability traces exist (e.g., [8, 10]), but they were collected by one-shot tools that were not intended to be permanent, stable pieces of the distributed infrastructure. The permanent monitoring tools in existence often focus on monitoring path characteristics, not individual host availability, so they issue measurements to and from a small set of vantage points. For example, RON [4] and iPlane [24] track latency and loss rates between a set of topologically diverse end points, but these machines are assumed to be highly available and small in number; no mechanism is provided for testing individual host availability inside a stub network. CoMon [29] provides uptime monitoring for individual PlanetLab hosts, but it does not scale to hundreds of thousands of machines. Furthermore, it requires modifications to end hosts, which may be difficult in non-academic settings where people are leery of installing new software.

In overlays like Pastry [32] and storage systems like TotalRecall [9], hosts probe the availability of select peers, but this data is not archived in a public directory, preventing global analysis. Schemes to distribute such data exist [20, 26], but large-scale data mining is difficult due to the number of wide-area data fetches required, as well as the need to perform cryptographic calculations to verify measurements submitted by untrusted peers.

The lack of a persistent infrastructure for availability monitoring is unfortunate because it could benefit a wide variety of systems. For example, distributed job allocators [5] could use historical uptime data in concert with availability prediction [25] to assign high priority tasks to machines that are likely to be online for the expected duration of the job. Distributed storage systems could also use a live feed of availability measurements to guide object placement and increase data availability [1].

To address such needs, we introduce StrobeLight, a tool for measuring availability in an enterprise setting containing hundreds of thousands of hosts. StrobeLight issues active probing sweeps at 30 second intervals, archiving ping results for the benefit of other distributed services that might find them useful. We describe two examples of such services. The first is an offline data-miner for longitudinal availability traces; such an application might be useful for distributed storage systems trying to make decisions about replica allocation. The second StrobeLight service monitors network-wide availability in real time, raising alarms for anomalies such as network partitions and IP hijacks. StrobeLight detects such problems using a new abstraction called an *availability fingerprint*. Under normal conditions, a subnet's fingerprint changes very slowly. Thus, StrobeLight raises an alert when the similarity between consecutive fingerprints falls below a threshold. Using Planetlab experiments, simulations, and a real enterprise deployment, we show that our detection system is accurate and fast.

By using standard ICMP probes to test availability, StrobeLight avoids the need to install new software on end hosts or deploy new infrastructure within the routing core. By collecting data from a few centrally controlled vantage points, StrobeLight avoids the trust and complexity issues involved with distributed solutions while making it easy for other systems to access availability data.

This paper provides three primary contributions. From the technical perspective, it demonstrates that frequent, active probing of a large host set is cheap and practical. From an analytical perspective, it introduces new techniques for analyzing availability traces that contain temporal gaps (see Section 3.3). Finally, the paper introduces a new, fine-grained trace of wired and wireless availability in a large corporate environment. Using this trace, we can validate results from previous studies that used coarser-grained data [10, 25]. We also discover an interesting property about the stability of subnet availability. From the qualitative perspective, subnet uptime is consistent across weeks—for example, the relative proportion of diurnal hosts is unlikely to change. However, from the quantitative perspective, subnet availability may fluctuate by more than 25% across a month (see Section 3.4.2).

2 Design and Implementation

The design of our availability measurement system was guided by three principles. First, keep the system simple. Second, make the system unobtrusive. Third, collect fine-grained data.

Keep it simple. Our primary design principle was to keep everything simple, a philosophy reflected in many different ways. We wanted to avoid solutions which required new software to be installed on end hosts, an arduous task that is difficult to justify on a corporate-wide basis. Similarly, we hoped to avoid major modifications to our internal routing infrastructure. Large-scale decentralization of the probing infrastructure was not a primary concern. Although coordinated distributed monitoring has certain benefits, previous experience had taught us that the road to a bug-free distributed protocol is fraught with peril [11]. Thus, we thought hard about the costs and benefits of a coordinated peer-to-peer design, and ultimately rejected it. One motivating factor was our development of analysis techniques which tolerate temporal gaps in availability data (see Section 3.3). These techniques shifted the payoff curve between the better coverage and robustness of a distributed, coordinated solution and the reduced complexity of a centralized one.

Don't annoy the natives. We wanted a system that was unobtrusive—we did not want our measurement activity to disrupt normal network traffic or add significant load. We also required a straightforward mechanism to turn off measurement activity in specific parts of the network. The latter was important because previous experience had taught us that at some point, our new network infrastructure would break someone else's experiment or interact with other components in unexpected ways. When such scenarios arose, we wanted the capability to quickly remove the friction point.

Collect high-resolution data. We wanted our tool to collect per-host availability statistics at a fine temporal granularity. This would allow us to validate previous empirical studies which used coarser data sets [10, 25]. It would also make the service more useful for anomaly detection, since disruptions like IP hijacking may only last for a few minutes [31].

These design considerations led to several "non-goals" for our system.

Infinite scalability is overkill. Our solution only needed to scale to the size of an enterprise network containing hundreds of thousands of hosts. Building a measurement system to cover an arbitrary number of hosts in an arbitrary number of administrative domains would have been extremely challenging. For example, active availability probing from foreign domains might trigger intrusion detection systems. Organizations might also be reluctant to provide outside access to DNS servers and other infrastructure useful for identifying "live" end hosts.

Complete address disambiguation is difficult. Another barrier to performing arbitrary-scale, cross-domain host monitoring is the widespread use of NATs, firewalls, and DHCP. These technologies can create arbitrary bindings between hosts and IP addresses, and prevent some machines from being seen by external parties. Devising a comprehensive monitoring system that can pierce this heterogeneous cloud of addressing policies is an important research topic. However, this goal was beyond the scope of our project. By focusing on enterprise-level solutions, we hoped to avoid many of the issues mentioned above; NATs were relatively rare in our corporate environment, and we could configure our firewalls to trust packets generated by our new monitoring system.

2.1 The Winning Design: StrobeLight

As shown in Figure 1, we eventually chose a centralized architecture in which a single server measured availability throughout our entire network. To determine which IP addresses to test, the server would download hostname/IP

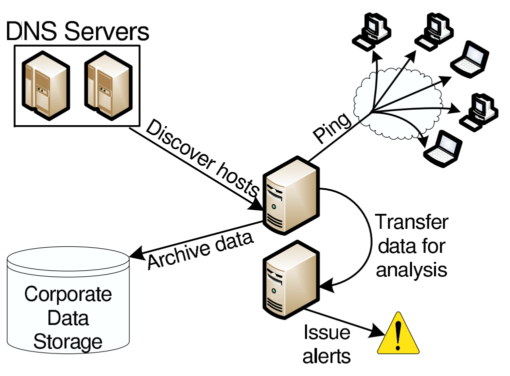


Figure 1: StrobeLight Architecture

mappings from corporate DNS servers. It would then test host availability using standard ping probes issued at intervals of 30 seconds. Recent probe results would be transferred to an analysis server for real-time anomaly detection, and longitudinal data would be archived in the corporation's standard distributed data store.

This design, which we named StrobeLight, was very attractive from the implementation and deployment perspectives. No new code would have to be pushed to end hosts or internal routers, and the only additional hardware required would be the probing server and the analysis engine. We also expected the probing process to have a light footprint. The total volume of request/response traffic would be trivial compared to the overall traffic level in the corporate network. Furthermore, we would not have to deal with control or synchronization issues that might arise in a more decentralized design. Our main concerns involved performance and fault tolerance. We feared that a single server might be overloaded by sending probes for hundreds of thousands of machines every 30 seconds. A centralized probing design also had obvious ramifications for fault robustness. Despite these weaknesses, we committed to the single-server design due to its relative ease of implementation, and we pledged to revisit the design if we encountered undue difficulties after deployment.

2.2 Implementation and Deployment

The core probing infrastructure was deployed first. The pinging daemon, consisting of 2,200 lines of C++ code, runs on a standard desktop PC with a 3.2 GHz CPU, 2 GB of RAM, and a gigabit Ethernet card; this machine resides within a corporate subnet in Redmond, WA. At boot time, the daemon reads an exclusion file which specifies the set of IP prefixes that should never be pinged. This file allows us to selectively exclude parts of the network from our probing sweeps. To determine which IP addresses to ping, the daemon downloads zone files from Microsoft's DNS servers at 2:10 AM each day. At any given moment, these zone files contain entries for over 150,000 IP addresses scattered throughout the world. This set of addresses evolves over time due to the introduction of new hosts and the decommissioning of old ones.

Due to these factors, an address may not appear in every DNS snapshot. Since StrobeLight only probes the addresses mentioned in the zone data, an IP may have gaps in its availability history. To deal with these gaps, StrobeLight describes the availability of an address as online, offline, or unknown. The first two categories result from the outcome of a ping probe, whereas the third is assigned to an IP which was not probed at a particular time.

Once the probing daemon had produced a sizable archive of availability data, we were able to test the offline analysis engine. This engine, totaling about 5,000 lines of C++ code, provides a set of low-level classes to represent per-host availability. It also defines a high-level query interface for use by data mining programs. We used this interface to generate the results in Section 3. Importantly, the interface defines a subnet of size N as a set of N consecutive and *allocated* IP addresses; the queryer chooses the starting address, N, and the time period over which "allocated" is defined. An address is considered allocated during a given time period if it appeared in a zone file at least once during that period. In practice, we often set N to a small number like 256 and investigate the subnets contained within a Class A or B prefix.

2.3 Operational Experiences

The probing server has run with few interruptions for almost three years, and it has not struggled with the network load generated by the ping sweeps. We currently spread each sweep across 25 seconds to avoid load spikes on our shared network infrastructure, but brief "full throttle" experiments show that our current prober can scan 270,000 hosts in 7.9 seconds (roughly 35,000 hosts a second).

In general, our ping traffic has not bothered the other members of our network. We occasionally receive emails from the network support staff when they unveil a new intrusion detection system and they conclude that our probing machine is infected with an IP-scanning virus; these incidents became rarer after we explained that StrobeLight was a piece of permanent infrastructure. We also received a complaint from another research group who claimed that our pings were causing problems for their wireless devices. After generating the appropriate exclusion file and restarting the daemon, we received no more complaints.

3 Application 1: Offline Analytics

In this section, we describe one application of StrobeLight, using it to gather long-term availability data for offline analysis. Such data could be used in several

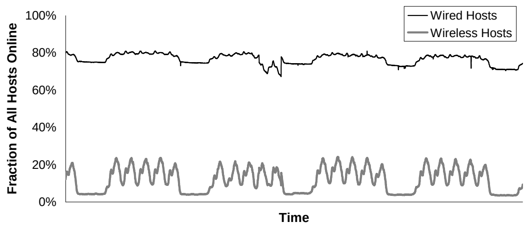


Figure 2: Global availability (10/21/2005 to 11/21/2005)

ways, e.g., to guide replication policy in a distributed data store [1, 9, 25]. In this section, we use the data in a more exploratory fashion, looking for interesting patterns in our wired and wireless networks. We restrict our analysis to IP addresses which appeared in at least 95% of the daily DNS snapshots. During the time period examined below, this included 138,801 wired IPs and 11,670 wireless IPs. In our corporate environment, the DHCP lease time is 20 days for wired machines and 3 hours for wireless ones. Thus, a wireless address is likely to be bound to multiple machines over the course of the day. Although we often refer to "hosts" and "IP addresses" interchangeably, the true unit of uniqueness is an address, not a host.

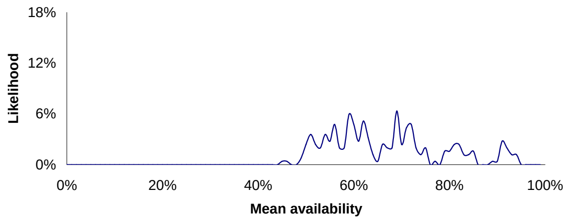
3.1 Global Trends

Figure 2 depicts aggregate availability fluctuations from October 21 to November 21 of 2005. The bulk of Microsoft's machines reside in the American west coast, so both the wired and wireless networks show large-scale diurnal trends aligned with the work day in this time zone. However, during these large-scale surges and declines in availability, there are regular, smaller-scale peaks and valleys. These additional periodic cycles are driven by phase-shifted diurnal behavior amongst Microsoft hosts in Europe and the Middle East.

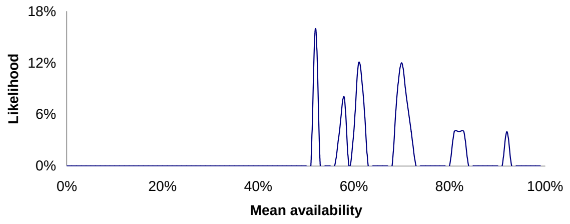
Comparing the two curves in Figure 2, we see that wireless IP addresses are much less likely to be associated with online hosts. However, the wireless network demonstrates stronger diurnal trends than the wired network. We investigate this issue further in Section 3.3.

3.2 Subnet-level Trends

We define the mean availability of a subnet as its average fraction of online hosts. Figure 3 shows the distribution of mean subnet availability in the wired network for subnets of size 256 and 2048. In both cases, mean subnet availability is always higher than 40%. Increasing the subnet size causes probability mass to coalesce around several regions of mean availability. This is a discretization artifact, since increasing the subnet size without increasing the total number of hosts results in fewer subnets to examine and less smoothness in the resultant distribution.



(a) 256 hosts per subnet



(b) 2048 hosts per subnet

Figure 3: PDF for mean subnet availability (wired)

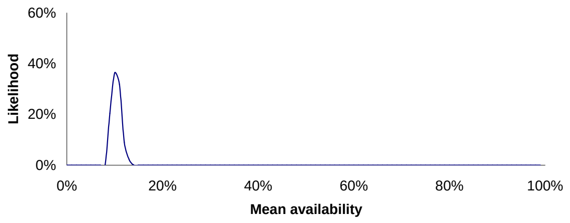
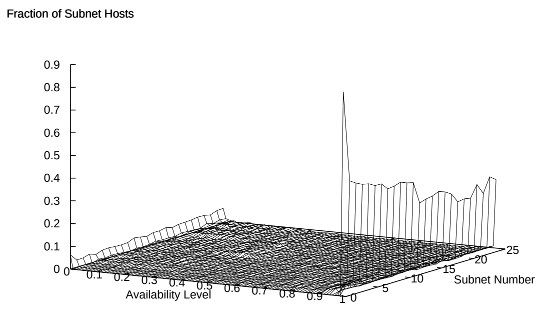


Figure 4: PDF for mean subnet availability (wireless)

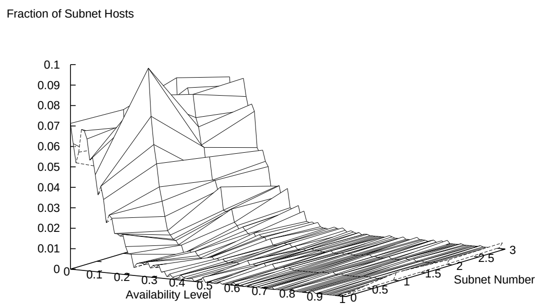
As expected, Figure 4 shows that wireless subnets have much lower mean availability. Figure 4 shows results for a subnet size of 256, but increasing the subnet size to 2048 results in an almost identical availability distribution. The relative lack of discretization artifacts is due to the greater homogeneity of wireless host availability. Figure 5 shows the distribution of per-host uptime fractions within each subnet. Each wired subnet has a skewed bimodal distribution, with a plurality of hosts having very high uptime and a smaller fraction having very low uptime. However, in every wired subnet, roughly 50% of the probability mass is spread across the "plateau" between the two modes. In contrast, the wireless subnets look more unimodal, with the majority of hosts having very low availability and much less probability mass sheared away from the mode.

3.3 The Availability of Individual Hosts

To understand the lower-level dynamics driving aggregate availability, we modified our previous taxonomy for classifying the uptime behavior of individual hosts [25]. In the unmodified scheme, a host is declared *always-on* if its uptime is greater than 90% and *always-off* if its uptime is less than 10%. If a host fails these tests, its availability signal is converted into the frequency domain using



(a) Wired subnets (2048 hosts per subnet)



(b) Wireless subnets (2048 hosts per subnet)

Each curve on the "availability level" axis is a pdf for per-host uptime fractions in a particular subnet. In each figure, the pdfs are sorted by standard deviation, with higher subnet numbers indicating larger standard deviations. The trends depicted in each graph are insensitive to subnet size.

Figure 5: Per-host availability within a subnet

a Fourier transform. If the resultant profile demonstrates harmonic peaks in the daily and weekly spectra, the host is labeled *diurnal*. If the spectral curve resembles the curve 1/f, i.e., it contains large amounts of low frequency energy, the host is labeled as *long stretch*, meaning that it has long, uninterrupted periods of uptime and downtime. Nodes failing all four tests are labeled as *unstable*. Such a designation usually implies that the host's availability is difficult to predict.

The standard algorithms for Fourier decomposition assume that signals are sampled at a uniform rate and that no samples are missing. In our data set, the assumption of a uniform sampling rate was almost always true, since the vast majority of probe sweeps were separated by 30 second intervals. However, missing samples were fairly common for two reasons. First, our network used DHCP to assign IP addresses to physical machines. When an address was dormant (i.e., unassigned), it did not show up in our zone files, meaning that we did not collect availability data for it during the dormant period. Second, the DNS servers occasionally failed, or misbehaved and returned extremely small zone files. Both of these phenomena introduce brief probing gaps for many hosts.

To deal with missing samples, we replaced the Fourier analyses with two entropy-based techniques. To determine whether an availability signal contained diurnal patterns, we adapted Cincotta's method for period detection in irregularly sampled time series [14]. Let $a_t \in \{0, 1\}$ be the value of an availability signal at time t. Given a hypothetical period τ, we calculate the phase of each a_t as $\phi_t = \frac{t}{\tau} - nearestInteger(\frac{t}{\tau})$; note that $\phi_t \in [-0.5, 0.5]$. We can interpret each (ϕ_t, a_t) pair as a coordinate in $\phi \times a$ space. If the hypothesized period τ is close to the signal's actual period (or a harmonic of it), the (ϕ_t, a_t) points will cluster in the coordinate space. This means that if we divide the coordinate space into bins, the resultant bin distribution will have low entropy. If the hypothesized period is not the signal's true period, points will be scattered throughout the $\phi_t \times a_t$ space and the bin distribution will have high entropy.

To determine whether an availability signal contains diurnal patterns, we check whether the entropy for a τ of 24 hours is less than the entropy for a τ of 23 hours. Availability signals with complex diurnal patterns may have entropy dips in other places, but finding one for a τ of 24 is sufficient for our purposes.

To determine whether an availability signal contains long-stretch behavior, we use an approximate entropy test [30]. Suppose that we have an arbitrary window of k consecutive samples from the signal. We define `ApEn(k)` as the additional information conveyed by the last sample in the window, given that we already know that previous $k - 1$ samples. Low values of `ApEn(k)` indicate regularity in the underlying signal. In particular, if we know that a host is not always-on, always-off, or diurnal, but it still has a low `ApEn(k)`, it is likely that the uptime regularity is driven by long-stretch behavior.

The choice of window size k is driven by the time scale over which "long stretch" is defined; k should be small enough that a stretch contains several windows, but not so small that `ApEn(k)` measures the incidence of small k-grams that are actually pieces of larger, more complex availability patterns. In the results presented below, we used a k of 8 and sampled our availability trace in steps of 15 minutes. This meant that we looked for long-stretch behavior at a time scale of roughly two hours. We defined hosts as long-stretch if their availability signal had an `ApEn(8)` of less than 0.16. This cutoff was determined by hand, but our results were not very sensitive to the exact value.

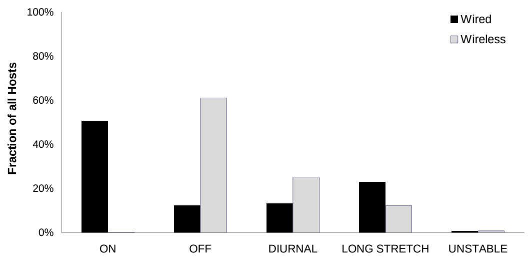


Figure 6: Availability taxonomy

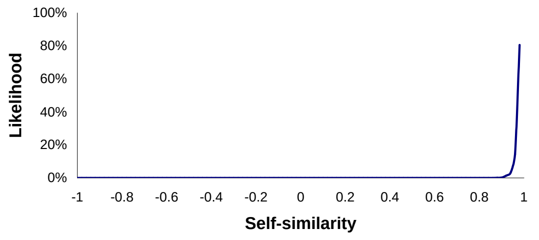
Figure 6 depicts the availability taxonomy for the wired and wireless networks. We found that roughly half of the wired hosts were always online. This result is congruent with smaller-scale observations of the Microsoft network which used an hourly sampling period [10, 25]. Indeed, the fact that the always-on fraction is the same at a finer sampling granularity implies that the natural time scale for availability fluctuation in wired corporate environments is hours, not minutes. This claim is further validated by the fact that almost none of the wired hosts had unstable availability. In other words, if a host was not always-on, always-off, or diurnal, then it at least had availability that was stable across one or two hours.

The wireless network was dominated by always-off machines, which comprised 61% of all hosts. The wireless network had almost twice as many diurnal machines as the wired network (25% versus 13% respectively) but almost half as many long-stretch hosts (12% versus 23%). These trends were unsurprising. In contrast to desktop machines that were always "plugged in," wireless devices with limited battery lives were more likely to have shorter sessions. Also, users often took these devices home at the end of the day, removing them from the physical proximity of a corporate access point. Thus, wireless connectivity exhibited stronger diurnal patterns and less long-stretch behavior than wired uptime.

3.4 Availability Fingerprints

Up to this point, we have investigated aggregate availability trends over a five week window. However, many network anomalies occur over a much smaller time scale. For example, an IP hijacking attack might only last for several minutes [31], and BGP misconfigurations can be just as transient [13].

Both types of anomaly change the mapping between IP addresses and physical hosts. In a hijacking attack, an entire range of IPs is bound to a different set of physical machines; similarly, a misconfigured router can cause arbitrary desynchronizations. Active availability probing can detect such problems if three conditions are true. First, the probing interval must be less than the duration of the desynchronization episode, lest the anomaly escape undetected between probing sweeps. Second, in the absence



Subnet self-similarity between successive probing sweeps is very high. The graph depicts results for wired subnets of size 256, but the outcome is insensitive to subnet size. The results are extremely similar for wireless subnets.

Figure 7: PDF for self-similarity of delta fingerprints (15 minute probe interval)

of anomalies, a subnet's availability "fingerprint" must be stable across multiple consecutive probing periods. This gives us confidence that when the fingerprint changes, an actual problem has arisen. Third, at any given moment, the availability fingerprint for each subnet should be globally unique. This allows us to detect routing problems in which two subnets have their IP bindings swapped.

With these desired characteristics in mind, we can provide a formal definition of a fingerprinting system. Given a specific subnet and a time window of interest, a fingerprinting algorithm examines per-host availability trends during that window and produces a bit-string that is a function of those trends. A fingerprinting system also defines a distance metric which determines the similarity of two fingerprints. To detect an anomaly in a subnet, we maintain a time series of its fingerprints and raise an alarm if the most recent fingerprint is too dissimilar from the previous one.

In the remainder of this section, we provide a concrete description of a fingerprinting system and evaluate its performance on trace data collected by StrobeLight. We focus on basic issues such as how a subnet's fingerprint evolves over time, and the accuracy with which we can distinguish two subnets based solely on their fingerprints. We present more applied results in Section 4, where we show how fingerprints can be used to detect anomalies within the enterprise and across the wide area.

3.4.1 Delta Fingerprints

During a single probe sweep, we test the availability of each known host in our network. Given a subnet of size s, we represent its probe results as an s-bit vector where a particular bit is 1 if the corresponding host was online and 0 if the host was offline or unknown (remember that a host is not probed if it is not mentioned in the current DNS mapping). We call such a vector an instantaneous or delta fingerprint because it represents a snapshot of subnet availability at a specific time.

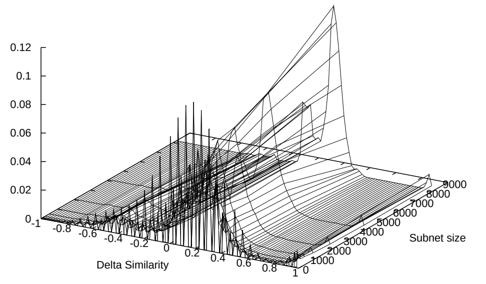


Figure 8: PDF for instantaneous cross-subnet similarity (15 minute probe interval)

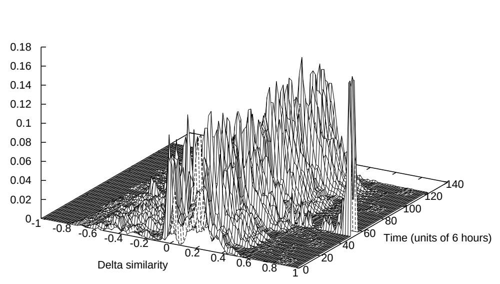


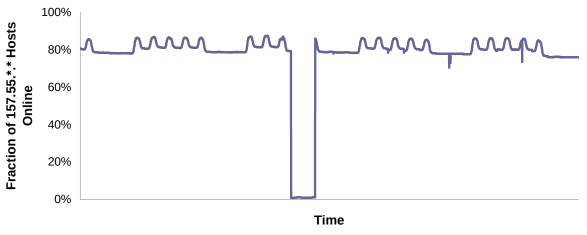
Figure 9: Temporal evolution of cross-subnet delta similarity (15 minute probe interval)

A natural distance metric for two delta fingerprints is the number of bit positions with equivalent values. Thus, we define the similarity of two fingerprints as the number of equivalent bit positions divided by s and normalized to the range $[-1, 1]$. For example, if two fingerprints match in half of their bit positions, they will have a similarity of 0. If they match in all positions or no positions, they will have a similarity of 1 or -1 respectively.

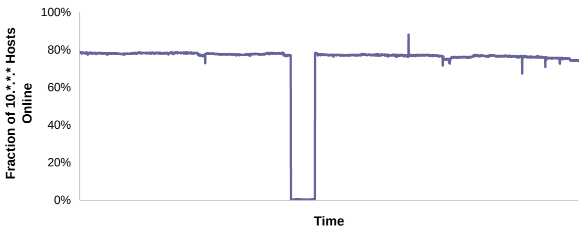
Given the availability probing period ρ, we define a subnet's self-similarity as the expected similarity of its fingerprints at time t and time $t + \rho$. Figure 7 depicts the pdf for self-similarity in the wired network with a ρ of 15 minutes. As shown in Section 3.3, the natural time scale of availability fluctuation in the wired network is hours, not minutes. Thus, with a 15 minute sampling granularity, delta fingerprints are very stable across two consecutive snapshots, with 95% of all fingerprint pairs exhibiting similarities of 0.96 or greater. Decreasing ρ results in even greater stability, which is possible since StrobeLight has a 30 second probing granularity.

The delta similarity of two *different* subnets at time t is simply the similarity of their fingerprints at t. Figure 8 depicts the pdf for cross-subnet similarity as a function of subnet size. As the subnet size grows, probability mass shifts towards the center of the similarity spectrum. However, even for subnets as small as 32 hosts, less than 2% of all subnet pairs have similarities greater than 0.8. The reason is that the various availability patterns described in Section 3.3 are randomly scattered throughout each subnet. For example, even though most subnets have a large set of always-on hosts, these hosts are randomly positioned throughout each subnet's fingerprint vector. Thus, two vectors are unlikely to have high correlations in all bit positions, and each fingerprint is likely to be globally unique.

The tiny peaks along the right side of Figure 8 indicate a small probability that at any given moment, two subnets have completely equivalent fingerprints. To understand the origin of these peaks, we plotted cross-subnet similarity as a function of time. Figure 9 indicates a large spike in fingerprint similarity during the middle of the trace period. As Figure 10 shows, this spike was synchronous with a dramatic availability drop in several IP blocks during November 3 and 4 of 2005. When these blocks went offline, their fingerprint vectors transitioned to an "all-zeros" state, leading to an immediate increase in cross-subnet similarity.



(a) Host availability in 157.55.*.*

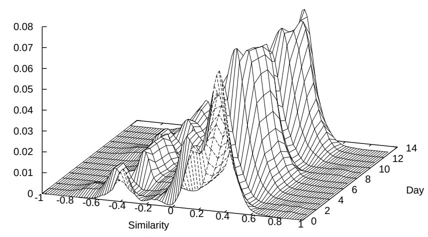


(b) Host availability in 10.*.*.*

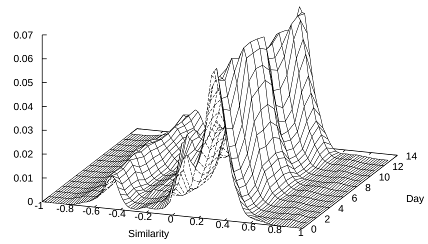
Network anomalies during November 3 and 4 of 2005 caused the spike in fingerprint similarity seen in Figure 9.

Figure 10: Punctuated availability disruptions

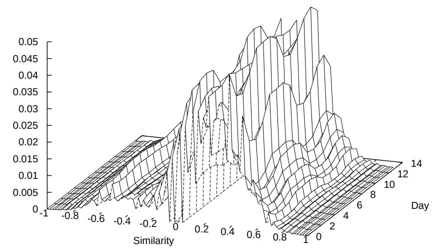
Once the anomaly terminated, the similarity distribution returned to a steady state in which all fingerprints were distinguishable. Thus, during the whole trace period, the global uniqueness property was only violated during the severe network disturbance. We return to the issue of anomaly detection in Section 4.



(a) 256 hosts per subnet, 24 hour window, 32-bit float per host



(b) 256 hosts per subnet, 24 hour window, 1-bit float per host



(c) 32 hosts per subnet, 24 hour window, 1-bit float per host

Figure 11: Cross-subnet similarity for wired and wireless subnets (24 hour window)

3.4.2 Fingerprinting Over Larger Windows

As currently described, a fingerprint is a bit vector representing the instantaneous availability of a set of hosts. In this section, we briefly describe how to extend our fingerprints to cover longer observation periods.

Cross-subnet Similarity: To create a fingerprint which covers a longer time window, we can associate each host with a floating point number instead of a single bit. Each float represents the mean availability of a host during the time period of interest. To compute the similarity between two floating point fingerprints, we examine each pair of corresponding floats and calculate the absolute magnitude of their difference. We sum these absolute magnitudes, divide by the subnet size, and then normalize the result to the range $[-1, 1]$.

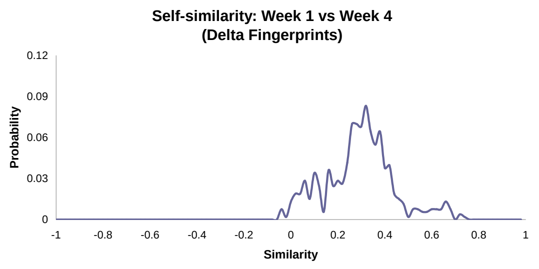
Figure 11(a) shows the temporal evolution of cross-subnet similarity using a day-long window; the subnet size was 256 hosts and each host was associated with a 32-bit floating point number. Comparing Figure 11(a) to Figure 8, we see that lengthening the fingerprint window does not change the fundamental distribution of subnet similarity. Most subnets are weakly similar or weakly dissimilar, but almost none are very similar or very dissimilar.

Figure 11(b) depicts cross-subnet similarity using a 24 hour window and "1-bit floats." In this scenario, a fingerprint contained a single bit for each host; the bit was 1 if the host was majority-online during the window and 0 if it was majority-offline. Comparing Figure 11(a) to 11(b), we see that using these truncated floats has little impact on the similarity distribution. Even if we decrease the subnet size to 32 hosts, Figure 11(c) shows that 1-bit floats provide enough resolution to keep the likelihood of perfect cross-subnet similarity well below 1%.

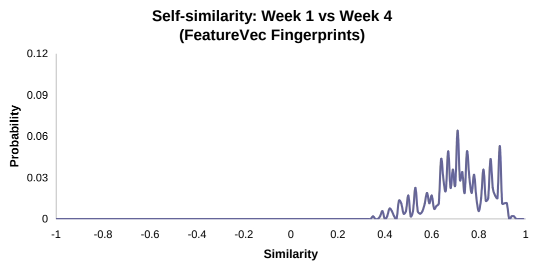
Using 1-bit floats, very little storage space is needed to maintain longitudinal fingerprint databases. For example, suppose that one needs to store fingerprints for a network containing 250,000 hosts. Using 1-bit floats, an individual snapshot would consume 250,000 bits (roughly 30 KB). Assuming a 24 hour window, a full year of data will only require 11 MB of storage space.

Self-similarity: Most subnets exhibit diurnal uptime. However, the true period of their availability is a week, not a day, since availability during the weekend lacks diurnal fluctuation and is depressed relative to that of the work week. Thus, if we examine subnet self-similarity using a day-long window, there are discontinuities during the transitions into and out of the weekend. However, one might expect self-similarity to be high using a week-long window, since this window size would precisely capture a full cycle of the seven day availability pattern.

Figure 12(a) shows the distribution of wired subnet self-similarity between the first and fourth weeks of our observation period. Although self-similarity was almost always positive, the correlation was unexpectedly weak, with the bulk of the probability mass residing between 0.0 and 0.5. This surprised us, since we had predicted that a host's availability fraction would not change much across weeks. Confronted with these results, we generated a new hypothesis, predicting that a host's availability *class* would vary less than its availability *fraction*. For example, the uptime fraction of a long-stretch host might vary between weeks, but its availability would be unlikely to transition from long-stretch behavior to (say) diurnal behavior.



(a) If we represent host uptime as a 32-bit floating point availability fraction, subnet self-correlation across weeks is mildly positive. However, raw subnet availability often varied by more than 25%.



(b) Self-correlation is higher if we represent host availability using a 2-bit enumeration type {ALWAYS-ON, ALWAYS-OFF, DIURNAL, OTHER} and check for behavioral equivalence amongst corresponding fingerprint entries.

Figure 12: Wired subnet self-similarity using week-long windows

To test this hypothesis, we devised a new type of fingerprint called a feature vector fingerprint. Instead of associating each host with a floating point availability fraction, we gave each host a 2-bit identifier representing whether it was always-on, always-off, diurnal, or "other" (either long-stretch or unstable). We defined the similarity between two feature vectors as the number of corresponding positions with equivalent feature identifiers. As before, we divided this number by the vector size and normalized it to the range $[-1, 1]$.

Figure 12(b) confirmed our hypothesis that, at the granularity of individual hosts, availability classes are more stable than availability fractions. However, subnet self-similarity was still lower than expected given the observed stability of weekly availability cycles at the subnet level. This topic remains an important area for future research.

4 Application 2: Detecting IP Hijacking

The Internet is composed of individual administrative domains called autonomous systems (ASes). The Border Gateway Protocol (BGP) stitches these independent domains together to form a global routing system [16]. Packets follow intra-domain routing rules until they hit an inter-AS border, at which point BGP data determines the next AS that will be traversed.

As currently described, StrobeLight detects intra-AS anomalies. For example, in Section 3.4.1, we showed how StrobeLight discovered the unreachability of several large subnets from within our corporate network. In this section, we describe how to detect BGP anomalies which affect subnet visibility from the perspective of external ASes. To detect such anomalies, we must deploy StrobeLight servers *outside* of the local domain. We describe the architecture for such a system and evaluate it using Planetlab experiments and simulations driven by our corporate availability trace.

4.1 Overview of IP Hijacking

An AS declares ownership of an IP prefix through a BGP announcement. This announcement is recursively propagated to neighboring ASes, allowing each domain to determine the AS chain which must be traversed to reach a particular Internet address. BGP updates are also generated when parts of a route fail or are restored. Since BGP does not authenticate routing updates, an adversary can fraudulently declare ownership of someone else's IP prefix and convince routers to deliver that prefix's packets to attacker-controlled machines. An attacker can also hijack a prefix by claiming to have an attractively short route to that prefix.

Zheng *et al* describe three basic types of hijacking attack [38]. In a *blackhole attack*, the hijacker simply drops every packet that he illegitimately receives. In an *imposture attack*, the hijacker responds to incoming traffic, trying to imitate the behavior of the real subnet. In an *interception attack*, the hijacker forwards packets to their real destination, but he may inspect or manipulate the packets before forwarding them.

Due to vagaries in the BGP update process, the attacker's fraudulent advertisement may not be visible to the entire Internet. This means that during the hijack, some ASes may route traffic to the legitimate prefix although others will not [6]. If the hijack causes divergence in external views of the prefix's availability, we can detect the attack by deploying multiple StrobeLight servers at topologically diverse locations.

For all but the least available subnets, a blackhole attack will create a dramatic instantaneous change in externally measured fingerprints. Fingerprint deviations may be less dramatic during an imposture attack; however, as we show in Section 4.3, two arbitrary subnets are still dissimilar enough to make imposture detection easy. Interception attacks cannot be detected through fingerprint deviations since the attacker will forward StrobeLight's probes to the real hosts in the target prefix. However, we describe a preliminary scheme in Section 4.4 that uses carefully chosen probe TTLs to detect such interceptions.

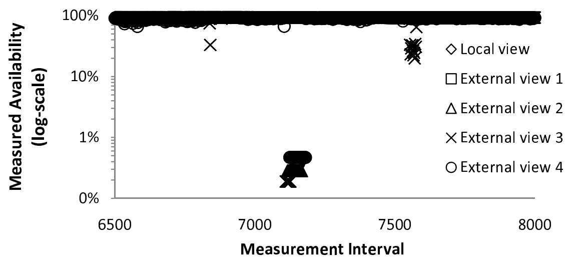


Figure 13: Availability of live IPs from different views

In our distributed StrobeLight system, the individual StrobeLight servers do not need to reside within the core routing infrastructure—they merely need to be deployed outside of the AS that they monitor. Furthermore, since anomalies are defined with respect to local measurements, there is little need for communication between the individual servers. Thus, a distributed StrobeLight system should be easy to deploy and maintain.

4.2 Does the Wide-Area Distort Probing?

As shown in Figure 7, a subnet's fingerprint changes very slowly under normal network conditions. However, that conclusion was derived from the perspectives of vantage points *inside* the enterprise. To detect BGP anomalies, StrobeLight servers must be deployed at *external* locations. This exposes probes to the vagaries of wide-area traversal, possibly increasing delay or loss in a way that destroys fingerprint stability during non-anomalous regimes.

To determine whether fingerprints could provide diagnostic power across the wide area, we deployed StrobeLight servers on 10 topologically diverse Planetlab hosts. From April to July of 2008, these servers probed 45 Class C subnets belonging to the computer science department at the University of Michigan. We also deployed a StrobeLight server inside the local campus domain. Each server launched a probe sweep every 30 seconds, similar to our deployment inside the Microsoft corporate network. The campus network contained roughly 11,000 live IP addresses. Figure 13 shows the measured availability of these addresses from the local perspective and those of four representative Planetlab hosts. Availability was always greater than 90% from the local vantage point. This was also true for the first two external views. The third and fourth views were measured from servers that were heavily loaded with other Planetlab experiments. Processor and network utilization were consistently high on these hosts; particularly severe spikes caused the StrobeLight servers to miss incoming probe responses and underestimate true domain availability by up to 80%. However, these incidents were rare, and would not arise in a real StrobeLight deployment that used dedicated probing machines.

Near time step 7100, external views 2, 3, and 4 were almost completely partitioned from the campus domain. This partition was caused by a switchgear failure at a Detroit Edison power plant that caused punctuated router failures throughout southeast Michigan. Interestingly, this event simulated a selective blackhole attack—although views 2, 3, and 4 were cut off from the local domain, view 1 enjoyed continuous connectivity. Thus, the Planetlab deployment showed two things. First, wide-area network effects do not destroy the diagnostic utility of availability fingerprints. Second, StrobeLight can detect blackhole attacks if probe servers are deployed at topologically diverse locations.

4.3 Imposture Attacks

Blackhole attacks are not subtle. In such an attack, the adversary drops all traffic destined for the target network, creating dramatic decreases in subnet availability and thus dramatic changes in subnet fingerprints. Imposture attacks are potentially more difficult to detect, since the adversary seeks to *mimic* the behavior of hosts in the target domain. In particular, we are interested in detecting spectrum agility attacks, first described by Ramachandran and Feamster [31]. The goal of a spectrum attack is to elude IP-based blacklists using short-lived manipulations of BGP state. Spammers hijack a large network, e.g., a /8 prefix, send a few pieces of spam from random IP addresses within the prefix, and then withdraw the fraudulent BGP advertisement a few minutes later. By using short-lived routing advertisements, spammers increase the likelihood that their hosts will be unreachable by the time that white hat forensics begin. By sending a small amount of traffic from each host, and by randomly scattering the traffic throughout a large address space, spammers avoid filtering by DNS-based blacklists [21].

To determine whether StrobeLight can detect spectrum attacks, we used simulations driven by availability data from the Microsoft network. We used this trace data instead of the Michigan data because it contained many more IP addresses, and spectrum attacks require large address spaces for maximum effectiveness. Our simulations used a trace gathered between July 29, 2006 and September 1, 2006. To include the largest possible host set in our evaluation, we did not filter hosts based on their unknown fraction. During this observation period, we saw 238,951 unique IP addresses. Our simulations examined the largest subnets demarcated by standard Class A/B/C rules. We also examined a "mega" subnet consisting of all IP addresses in the trace.

During each simulation run, we iterated through our availability data in strides of 15 minutes; during each iteration, we compared each subnet's fingerprint to that of a similarly sized attacker subnet in which a random fraction of hosts responded to StrobeLight's pings. StrobeLight

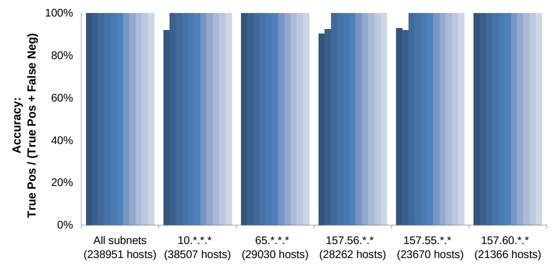


Figure 14: Detecting spectrum agility attacks

detected the spectrum attack if the similarity of the two fingerprints was beneath a threshold c. More specifically, let $f_{real,t}$ represent the fingerprint of a real subnet at time t and $f_{fake,t}$ be the fingerprint of the attacker subnet. Let $sim()$ compute the similarity of two fingerprints. Given a similarity cutoff c, we define StrobeLight's detection accuracy at time t as follows:

- `True positive:` $sim(f_{real,t-1}, f_{fake,t}) < c$. The attacker's fake subnet at time t is too dissimilar to the real subnet's fingerprint from the previous timestep. StrobeLight raises an alarm in this case.
- `False negative:` $sim(f_{real,t-1}, f_{fake,t}) \geq c$. The fake subnet is sufficiently similar to the real subnet that StrobeLight does not raise an alarm.

Every simulated comparison should raise an alarm, so there are no true negatives or false positives.

Figure 14 shows StrobeLight's detection accuracy in the five largest subnets. We also show results for an attack against the "mega-subnet" containing all hosts, since this is the best that we can approximate a large `/8` prefix. Each cluster of bars represents detection accuracy for a specific subnet. Within a cluster, the i-th bar is our detection accuracy when a random $i * 10\%$ of hosts in the attacker subnet respond to probes. For all of the results, we used a similarity cutoff c of 0.78; this value minimized the false negative rate.

In the mega-subnet containing 238,951 hosts, StrobeLight had perfect detection accuracy across all time steps. StrobeLight also had perfect accuracy for two of the five classful subnets. In the other three, detection accuracy for low response fractions dipped as low as 90%. These subnets were affected by a DNS failure which caused their hosts to spend part of the observation period in an unknown state. StrobeLight assumes that unknown hosts are offline, so an attacker could hijack these subnets during the DNS failure and evade detection by rarely responding to StrobeLight pings. However, StrobeLight would raise alarms at the beginning of the DNS anomaly, since a large number of hosts would appear to go offline suddenly. Thus, human operators would be more vigilant for additional problems during this time period. In Section 4.6, we return to the issue of StrobeLight's reliance on DNS infrastructure.

If an attacker could measure availability trends in our subnets, he could mimic the legitimate distribution of probe responses during the spectrum attack and avoid detection by StrobeLight. However, many organizations already perform ingress filtering of ping probes destined for internal hosts, eliminating the most obvious way for an adversary to collect availability data.

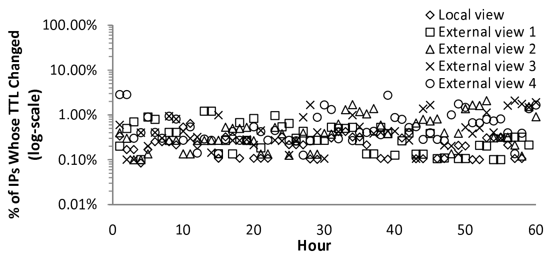
The attacker could try to spoof the IP address of a real StrobeLight server, and use the spoofed address to launch surveillance probes. There are several ways to deal with such an attack. One simple solution is to have the legitimate StrobeLight servers periodically audit each other using a shared-secret challenge/response protocol. If an attacker spoofs server S_0's address, and the spoof is visible by another server S_1, the fake S_0 will fail S_1's challenge, and S_1 can raise an alarm.

4.4 Interception Attacks

In an interception attack, the adversary convinces routers to send other people's traffic through attacker-controlled machines. These machines may inspect or tamper with the packets before forwarding them to their real destination. The current version of StrobeLight cannot detect such interceptions, since the interceptor does not drop legitimate probe packets or generate false probe responses. We have preliminary thoughts about how to modify StrobeLight to detect interceptions, and we briefly sketch some ideas below. However, a full exploration is left to future work.

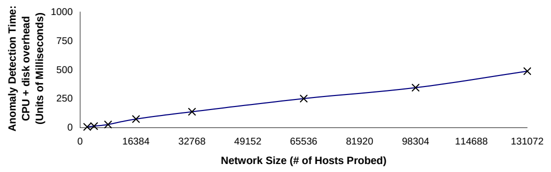
Since two arbitrary prefixes are likely to be topologically distant [38], an interception attack that affects a StrobeLight probing path should *lengthen* the route between the StrobeLight server and the monitored prefix. In theory, this will increase the latency from the server to the monitored prefix. So, the server can raise an alarm if it detects a correlated spike in response latencies across all prefix hosts. Unfortunately, latency may display significant jitter during non-anomalous conditions, so a naive implementation of this scheme will generate excessive false alarms.

Instead of looking for latency changes, StrobeLight could look for hop count changes. Previous research has shown that the hop count between two arbitrary prefixes is stable in the short to medium term [36, 38]. We verified this result with our StrobeLight deployment at the University of Michigan. Figure 15 shows the stability of hop counts from the internal Michigan server and from several external vantage points. Both internal and external servers recalculated their hop count to Michigan hosts once an hour; these recalculations were staggered across each hour. Recalculations typically resulted in TTL changes for less than 1% of all nodes, and we believe that most changes were due to lost tracing packets instead of actual host movement within the target domain.



Stub networks rarely change their location with respect to the network core. Thus, the hop counts between hosts in that stub and an external vantage point are stable.

Figure 15: Hop count stability



Each data point represents the average of 100 trials. Standard deviations were very small.

Figure 16: Scalability of Analysis Engine

Since interception attacks are likely to lengthen the route between a StrobeLight server and its target prefix, they are detectable by monitoring the hop count between the target prefix and the distributed measurement sites. This idea was first proposed by Zheng *et al* [38], and a variant could be integrated into StrobeLight. Each server would carefully set the TTLs of its probes to the expected hop count to the target prefix. A sudden increase in this path length will cause the probes to be dropped before they reach their destination; the StrobeLight server will perceive this as a sudden decrease in prefix availability and raise an alarm. This solution is more attractive than the latency-based scheme since hop counts are much more stable than latency. However, the hop count technique assumes that the attacker has limited topological knowledge. In particular, if the interceptor knows the routes connecting the target prefix, the StrobeLight servers, and the interceptor's routers, he can rewrite TTLs in a straightforward way to elude detection.

4.5 Performance

Anomaly detection consists of three steps: issuing the ping sweep from the probe machine, transferring the probe results to the analysis machine, and performing fingerprint calculations on the analysis machine. The first step is the slowest one, since we spread the probing sweep over several seconds to avoid noticeable network spikes. The second step should be fast even if the probing machine is different than the analysis machine, since probe results are just small bit vectors. As shown in Figure 16, the final calculation step is also fast. Figure 16 shows that once the analyzer has pulled the ping results onto local storage, the time needed to calculate new fingerprints and perform threshold calculations is less than half a second, even for networks with 130,000 hosts.

4.6 Discussion

StrobeLight queries DNS servers to determine which IP addresses to probe. Depending on one's perspective, this is a vice or a virtue. StrobeLight's sensitivity to DNS state means that it can detect some anomalies in DNS operation. However, this opens StrobeLight to DNS-mediated attacks in which adversaries try to disrupt StrobeLight's DNS fetches before tampering with BGP state. The IP prefixes owned by an enterprise are fairly stable, so we could manually configure StrobeLight with these prefixes and probe every address without regard to whether it was assigned internally (in fact, this is what we did for the StrobeLight deployment at the University of Michigan, since we lacked access to the DNS zone files). The penalty would be an increase in the prober's network load; also, if there are many unassigned addresses, cross-subnet similarity will naturally be higher, leading to more false alarms.

5 Related Work

Several commercial products provide enterprise-scale network monitoring without requiring end-host modification. For example, in the SiteScope system [17], a centralized server remotely logs into client systems and reads local performance counters. Tools like this collect a wider variety of data than StrobeLight, which only measures availability. However, StrobeLight can scan more machines per second, since it uses simple ping probes instead of comparatively heavyweight remote logins. StrobeLight is also easier to deploy in heterogenous end-host environments, since ICMP probes work "out-of-the-box" across all commodity operating systems, but remote login procedures can differ substantially across OSes.

Passive introspection of preexisting traffic can be used to infer path characteristics or host availability. For example, Padmanabhan *et al* record the end-to-end loss rate inside a client-server flow and use Bayesian statistics to extrapolate loss rates for interior IP links [28]. Passive detection of host availability is attractive for two reasons. First, it does not generate new traffic. Second, explicit probing may trigger intrusion detection systems on leaf networks, a problem occasionally encountered with active probing systems deployed on PlanetLab [34]. Despite these advantages, passive probing was ill-suited for our goal of tracking per-host availability in a large network. The time that a host is online is a superset of the time that it is generating network traffic, so passive observations of per-host packet flows may underestimate true availability. Also, a key design goal was to minimize the new infrastructure that had to be pushed to end hosts or the corporate routing infrastructure. Installing custom network introspection code on every end host was infeasible. Placing such code inside the core network infrastructure was also infeasible due to the complex web of proxies, firewalls, and routers that would have to be instrumented to get a full view of each host's network activity.

Most prior work on IP hijack detection has required modification to core Internet routers. Some systems require routers to perform cryptographic operations to validate BGP updates [2, 12, 19], whereas others require changes to router software to make BGP updates more robust to tampering [35, 37]. We eschewed such designs due to the associated deployment problems.

Several systems use passive monitoring of BGP dynamics to detect inconsistencies in global state [22, 23, 33]. These systems typically search for anomalies in one or more publicly accessible databases such as RouteViews [27], which archives BGP state from multiple vantage points, or the Internet Routing Registry [3], which contains routing policies and peering information for each autonomous system. Passive monitoring eases deployability concerns. However, data freshness becomes a concern when dealing with "eventually updated" repositories such as the IRR, and even RouteViews data is only updated once every two hours. Legitimate changes to routing policy may also be indistinguishable from hijacking attacks in terms of BGP semantics, making disambiguation difficult in some cases. In contrast, if our availability fingerprints indicate that a large chunk of hosts have suddenly gone offline or changed their availability profile, it is extremely unlikely that this is a natural phenomenon.

Hu and Mao were the first to use data plane fingerprints in the context of hijack detection [18]. In their system, a live BGP feed is monitored for suspicious updates. If an IP prefix is involved in a questionable update, its hosts are scanned from multiple vantage points using nmap OS fingerprinting [15], IP ID probing [7], and ICMP timestamp probing [18]. The results are presented to a human operator who determines if they are inconsistent. Our system differs in three ways. First, we do not require privileged access to a live BGP feed, easing deployability. Second, we continually calculate subnet fingerprints, whereas Hu's system only calculates fingerprints upon detecting suspicious BGP behavior, behavior which may take several minutes to propagate to a particular vantage point. Third, we can finish a probing sweep in less than 30 seconds, whereas several of Hu's scans may take several minutes to complete. Given the short-lived nature of spectrum agility attacks [31], we believe that quick, frequent scanning is preferable, if only to serve as a tripwire to trigger slower, "deeper" scans.

Zheng *et al* detect hijacking attacks by measuring the hop count from monitor hosts to the IP prefixes of interest [38]. For each prefix, the monitor selects a reference point that is topologically close to the prefix and lies along the path from the monitor to the prefix. In normal situations, the hop count along the monitor-reference point path should be close to that of the monitor-prefix path. When the prefix is hijacked, the hop count along the two paths should diverge. Zheng's system avoids the deployability problems mentioned above, since hop counts can be determined by any host that can run traceroute. However, the system assumes that a reference point can be found which is immediately connected to the target prefix and responds to ICMP messages; if the reference point is further out, the hijacker can hide within the extra hops. Our system only requires that end hosts respond to pings. Furthermore, our system tracks the availability of individual hosts, whereas Zheng's system only tracks the availability of a few representative hosts in each target prefix.

6 Conclusion

Many distributed systems would benefit from an infrastructure that collected high resolution availability measurements for individual hosts. Unfortunately, existing frameworks either do not scale, do not track every host in the network, or store data in such a way that makes global analysis difficult. In this paper we describe StrobeLight, an enterprise-level tool for collecting fine-grained availability data. Our current prototype has measured the uptime of hundreds of thousands of hosts in our corporate network for almost two years. Using the longitudinal data generated by this tool, we performed extensive analyses of availability in our wired and wireless networks. Using external Planetlab deployments and simulations, we also demonstrated how StrobeLight's real-time analysis engine can detect wide-area network anomalies. Our operational experiences indicate that StrobeLight's anomaly detection is fast and accurate.

References

[1] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *Proceedings of OSDI*, pages 1–14, Boston, MA, December 2002.

[2] W. Aiello, J. Ioannidis, and P. McDaniel. Origin Authentication in Interdomain Routing. In *Proceedings of CCS*, pages 165–178, Washington, DC, October 2003.

[3] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing Policy Specification Language (RPSL). RFC 2622, June 1999.

[4] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of SOSP*, pages 131–145, Banff, Canada, October 2001.

[5] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, November 2004.

[6] H. Ballani, P. Francis, and X. Zhang. A Study of Prefix Hijacking and Interception in the Internet. In *Proceedings of SIGCOMM*, pages 265–276, Kyoto, Japan, August 2007.

[7] S. Bellovin. A Technique for Counting NATted Hosts. In *Proceedings of the SIGCOMM Internet Measurement Workshop*, pages 267–272, Marseille, France, November 2002.

[8] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd IPTPS*, Berkeley, CA, February 2003.

[9] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. Total Recall: system support for automated availability management. In *Proceedings of NSDI*, pages 337–350, March 2004.

[10] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of ACM SIGMETRICS*, pages 34–43, Santa Clara, CA, June 2000.

[11] W. Bolosky, J. Douceur, and J. Howell. The Farsite Project: A Retrospective. *ACM SIGOPS Operating Systems Review*, 41(2):17–26, April 2007.

[12] K. Butler, P. McDaniel, and W. Aiello. Optimizing BGP Security by Eploiting Path Stability. In *Proceedings of CCS*, pages 298–310, Alexandria, VA, November 2006.

[13] D.-F. Chang, R. Govindan, and J. Heidemann. Locating BGP Missing Routes Using Multiple Perspectives. In *Proceedings of the SIGCOMM Workshop on Network Troubleshooting*, pages 301–306, Portland, OR, September 2004.

[14] P. Cincotta, M. Mendez, and J. Nunez. Astronomical Time Series Analysis I: A Search for Periodicity Using Information Entropy. *The Astrophysical Journal*, 449:231–235, August 1995.

[15] Fyodor. nmap security scanner. http://insecure.org/nmap/.

[16] IETF IDR Working Group. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[17] Hewlett-Packard Development Company. HP SiteScope software: Data sheet. White paper, August 2008.

[18] X. Hu and Z. Morley Mao. Accurate Real-time Identification of IP Prefix Hijacking. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–17, Oakland, California, May 2007.

[19] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure Path Vector Routing for Securing BGP. In *Proceedings of SIGCOMM*, pages 179–192, Portland, OR, September 2004.

[20] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of VLDB*, pages 321–332, Berlin, Germany, September 2003.

[21] J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Proceedings of IMC*, pages 370–375, Taormina, Sicily, Italy, October 2004.

[22] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. Topology-based Detection of anomalous BGP messages. In *Proceedings of RAID*, pages 17–35, Pittsburgh, PA, September 2003.

[23] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. Phas: A Prefix Hijack Alert System. In *Proceedings of USENIX Security*, pages 153–166, Vancouver, Canada, August 2006.

[24] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An Information Plane for Distributed Services. In *Proceedings of OSDI*, pages 367–380, Seattle, WA, November 2006.

[25] J. Mickens and B. Noble. Exploiting Availability Prediction in Distributed Systems. In *Proceedings of NSDI*, pages 73–86, San Jose, CA, May 2006.

[26] J. Mickens and B. Noble. Concilium: Collaborative Diagnosis of Broken Overlay Routes. In *Proceedings of DSN*, pages 225–234, Edinburgh, UK, June 2007.

[27] University of Oregon. *Route Views Project*. http://www.routeviews.org.

[28] V. Padmanabhan, L. Qiu, and H. Wang. Passive Network Tomography Using Bayesian Inference. In *Proceedings of SIGCOMM Internet Measurement Workshop*, pages 93–94, Marseille, France, November 2002.

[29] K. Park and V. Pai. CoMon: A mostly-scalable monitoring system for PlanetLab. *Operating Systems Review*, 40(1):65–74, January 2006.

[30] S. Pincus. Approximate entropy as a measure of system complexity. In *Proceedings of the National Academy of Science*, pages 2297–2301, USA, March 1991.

[31] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. In *Proceedings of SIGCOMM*, pages 291–302, Pisa, Italy, September 2006.

[32] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[33] G. Siganos and M. Faloutsos. Neighborhood Watch for Internet Routing: Can We Improve the Robustness of Internet Routing Today? In *Proceedings of INFOCOM*, pages 1271–1279, Anchorage, AK, May 2007.

[34] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using Planetlab for Network Research: Myths, Realities, and Best Practices. In *Proceedings of WORLDS*, pages 67–72, San Francisco, CA, December 2005.

[35] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. Listen and Whisper: Security Mechanisms for BGP. In *Proceedings of NSDI*, pages 127–140, San Francisco, CA, March 2004.

[36] R. Teixeira, S. Agarwal, and J. Rexford. BGP Routing Changes: Merging Views from Two ISPs. In *SIGCOMM Computer Communications Review*, pages 79–82, October 2005.

[37] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Detection of Invalid Routing Announcement in the Internet. In *Proceedings of DSN*, pages 59–68, Bethesda, MD, June 2002.

[38] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis. A Light-Weight Distributed Scheme for Detecting IP Prefix Hijacks in Real-Time. In *Proceedings of SIGCOMM*, pages 277–288, Kyoto, Japan, August 2007.