Monitoring Cloud Service Unreachability at Scale

Paper #1422 (1570667992), 9+1 pages

***Abstract*—We consider the problem of network unreachability in a global-scale cloud-hosted service that caters to hundreds of millions of users. Even when the service itself is up, the "last mile" between where users are, and the cloud is often the weak link that could render the service unreachable.**

We present `NetDetector`, a tool for detecting network-unreachability based on measurements from a client-based HTTP-ping service. `NetDetector` employs two models. The first, `GA` (Gaussian Alerts) models temporally averaged raw success rate of the HTTP-pings as a Gaussian distribution and flags significant dips below the mean as unreachability episodes. The second, more sophisticated approach (`BB`, or Beta-Binomial) models the health of network connectivity as the probability of an access request succeeding, estimates health from noisy samples, and alerts based on dips in health below a client-network-specific SLO (service-level objective) derived from data. These algorithms are enhanced by a drill-down technique that identifies a more precise scope of the unreachability event. We present promising results from `GA`, which has been in deployment, and the experimental BB detector over a 4-month period. For instance, `GA` flags 49 country-level unreachability incidents, of which 42 were labelled true positives based on investigation by on-call engineers (OCEs).

I. Introduction

The rise of cloud computing has meant the migration of hitherto on-premise applications such as email and productivity tools to the cloud. While the cloud infrastructure and the services hosted there are engineered to be highly available, the effective availability as seen by users is limited by the "last mile" that connects clients to the cloud. [1] The reason is that unlike with its own infrastructure, which a cloud provider can monitor directly (as indeed solutions such as Amazon CloudWatch [1], Google Cloud Platform's Stackdriver [2] and Azure Monitor [3] do), the provider has little visibility into the broader Internet infrastructure. Indeed, the nature of network unreachability is such that the service might be unaware that users are trying to reach it but are unable to. Often an unreachbility incident comes to light only after users complain, often through third-party services such as DownDetector [4] or social media channels such as Twitter.

Our goal with `NetDetector` is to enable effective monitoring of end-to-end network reachability. While it is attractive to attempt such monitoring based on observations of the organic traffic itself on the server side, e.g., by looking for dips in traffic as signs of unreachability, we argue that such an approach is stymied by practical hurdles, specifically variation in server-side traffic volume due to temporal shifts in user behaviour (for example, un-anticipated holidays) and retrial logic incorporated into the client application that may mask and distort observations of network unreachability. We argue that having a client-side signal is highly desirable.

In this paper, we focus on the signal from a client-side "HTTP ping" service akin to Odin [5], which in a nutshell provides us two pieces of information: the number of out of band probing pings (HTTP GETs) attempted by the client and the number that succeeded. `NetDetector` has to, nevertheless, contend with a number of challenges, outlined shortly. The design of a scalable, fault tolerant HTTP ping infrastructure is not a contribution of our paper, for that we follow Odin [5] closely. Our contribution lies in algorithms to process the noisy and variable data obtained therefrom.

First, the number of HTTP probing pings attempted by a set of clients in a certain scope (for example, country) varies significantly with time of day, day of week and holidays. Therefore, computing a raw "success rate" is not always meaningful; for instance, 1 failure out of 2 attempts is very different from 500 failures out of 1000 attempts. Second, unlike with a cloud-hosted service itself, for which we can set absolute availability requirements (e.g., five 9's [6]–[8]) and then engineer for it, in our setting, the last-mile, which is outside the control of the service, can be quite variable in quality. For instance, it is likely that clients in South Africa would have worse connectivity to a service hosted in Europe than clients in Europe itself would, so it would not be appropriate to set the same service-level objective (SLO) for availability for both sets of clients. Third, the scope of impact could vary depending on the underlying failure. For instance, the failure could impact just a subnet or an AS (Autonomous System) in a particular metro or the AS network as a whole or an entire country. Therefore, we need to be able to discover the scope of impact based on data.

In `NetDetector`, we start with a simple approach to detect unreachability incidents, which addresses the above challenges by restricting ourselves to those regions where a minimum threshold on the number of attempts (HTTP pings) per unit intervals of time (5 minutes) is met. This is to ensure that raw success rate so computed is accurate. This approach, dubbed `GA` (short for Gaussian Alerts), models the temporally averaged success rate for a country as a Gaussian distribution, estimating the mean and the standard deviation. It then looks for improbable dips in the success rate and flags these as unreachability incidents subject to certain alerting criteria. Flagged incidents are investigated by an on-call engineer, who also labels the alert as a true or false positive.

While this is effective for large scopes, for example, countries where the raw number of attempts is stable and high, it faces difficulties in narrower scopes, such as metros and ISPs and countries where the number of pings may be low, for

[1] Despite the name, the "last mile" could, in fact, be a wide-area path, depending on where the client is located relative to the cloud infrastructure.

example Nigeria. Because the natural scope of unreachability events are typically much smaller than entire countries, this difficulty is real and needs to be addressed.

While GA has been in production use with Mega, a large cloud-based productivity service serving hundreds of millions of users, for more than an year, we also wanted to explore a more systematic approach to addressing the challenges noted above. To this end, we also developed a second, experimental detector, dubbed BB (short for Beta Binomial), which models the attempts and successes in the Bayesian framework of a hidden semi-Markov binomial process, with a time varying success probability, p, that is specific to each {client-aggregate, endpoint} pair. As prior for p, it uses a Beta distribution, with its parameters derived from data. This framing makes it convenient for the posterior of p to be updated as observations of attempts and successes are made.

To a rough approximation, when the median of the posterior of p drops below a client location specific SLO, BB treats it as a candidate for raising an unreachability event alert. The details are described in Section III. The SLO is set in a data-driven manner, with the goal of separating "normal" operation (which, depending on the client location, might still mean less-than-perfect reachability) from the abnormal periods.

By default, NetDetector monitors availability at the level of countries, a choice motivated by the current practice in the operations team of the large cloud-hosted service, Mega, that we study. However, once an issue has been flagged at the country level, NetDetector employs a novel MAXSAT-based drill down technique to identify the more fine-grained scope that is affected. The rationale for this two-step procedure is that it allows us to focus attention on significant incidents that "move the needle" at the level of a country, while still identifying the problematic scope precisely.

In this paper we present the findings from a 4-month period: of the 49 events flagged by GA during this period, 42 were found to be true positives. We also ran BB alongside GA and find that it flags many more incidents, 122 in the same 4-month period, including correctly flagging 38 out of the 42 true positives labeled in the case of GA (i.e., a low false negative rate of 4/42). While many of these result in a marginal dip in health at the country level, our drill-down technique reveals a much sharper dip at a finer scope such as ISP or ISP-metro. While the on-call engineers(OCE) today do not investigate issues at such a finer scope, we present analysis of how both GA and BB perform relative to an "ideal" approach that flags unreachability incidents strictly in decreasing order of their impact on user accesses.

II. Problem Context

Our work is set in the context of a global-scale, online email and productivity service (anonymized as Mega), with hundreds of millions of users. While such services are engineered to be "always on", issues in the wide-area path between the cloud servers and clients (e.g. cable cut, router misconfigurations [9], etc.) may render the service unreachable, impacting users. Thus, it is important for the service

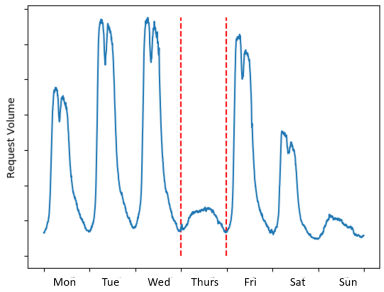

Fig. 1. Request volume dip during holiday in India on Thursday, 15 Aug

providers to be aware whenever their users are unable to reach their desired endpoint for some sustained length of time. This might appear paradoxical since a service provider such as Mega may not have any direct control on preventing or mitigating such problems on the Internet. However, based on our conversations with the operations team at Mega, there is significant business value in detecting faults, including unreachability issues, before the customer notices and complains. Knowledge about such ongoing incidents plays a crucial role in routing traffic around problem spots, extending help and information to partner ISPs, guiding future network peering decisions, and of course, proactively communicating the nature and extent of the problem to its customers.

Users of Mega connect to the service using a rich client (i.e., a native application on the client platform) or just a web browser. Unreachability is monitored with respect to the HTTP front end of Mega, which is what clients communicate with. The front end receives the client request and directs it to the appropriate back end, possibly routing it via a front end in a different region depending on the relative location of the user and their data.

A seemingly obvious way to infer unreachability episodes would be to monitor the volume of clients requests received per unit time at the front end of Mega and look for anomalous dips. This request volume based analysis, however, is beset with many practical problems. First, the request volume is a noisy signal that could, in general, be impacted by factors other than network unreachability, specifically shifts in user-behaviour. For instance, the volume could drop because of a holiday or a sports event that keeps users off their computers. While one could use out-of-band signals (e.g., calendar information) to factor in such events, this is challenging because of the global footprint of the service (e.g., the schedule of holidays would be different in different countries and in some cases even across regions in a country) and moreover there could be unscheduled events that are hard to anticipate (e.g.,a breaking news event that glues users to their TVs and keeps them away from their computers). In Figure 1, we show an example of a seemingly "anomalous" dip in request volume that is due to shift in user behaviour rather than network unreachability. In addition to these *natural* variations, the dips in the volume can be easily masked by retries, both human initiated and application initiated, trying to compensate for poor network quality. While we could, in principle, augment

the client to convey this information, effecting such a change is not easy in the context of a large, live service.

As an alternative, we employ an approach patterned after the Odin systems [5], which injects a Javascript-based measurement agent into the client to have it perform out of band HTTP "pings", i.e., HTTP GETs of a small object from the server. The agent reports back the statistics of attempts and successes through a redundant telemetry channel (e.g., using staging servers on a CDN), which ensures that the telemetry reaches `Mega` unless, of course, the client is fully disconnected. Thus, unreachability detection in `NetDetector` focuses on the attempt and success counts obtained from this Odin-like system. Note that our methodology leverages and builds on a scalable, fault tolerant HTTP ping infrastructures [5]. Our contribution lies not in the design of such a service but in data analysis algorithms that process the noisy and variable data produced by such a service.

III. Design of Unreachability Detection

`NetDetector` has been designed keeping in mind the challenges it has to contend to.

First, the set of users adversely affected by an unavailability incident could sometimes be a small fraction of the global user-base, for example, users of a specific ISP-metro combination, while at other times the fraction may be sizeable, such as all users behind a severed trans-oceanic link. Service providers need to monitor the endpoints from different parts of the world and from different modes of access (e.g., rich clients and web browsers,) so that they maintain necessary awareness about such incidents. The sheer combinatorial variety of possible subsets of the user-base that could be adversely affected at any given time, makes detecting such problems very challenging.

Another problem is that the number of HTTP probing pings attempted by the client-side measurement agent varies as a function of the number of clients online. As a result, Fewer HTTP pings will emanate from scopes with fewer users or clients. One way to mitigate this problem is by simply dialing up the rate of HTTP pings in such scopes , but this would burden clients with additional measurement overhead. Instead, we incorporate in our framework the uncertainty arising from the low volume of HTTP pings.

A. Overview of `NetDetector`

Our system to detect unreachability, called `NetDetector`, consists of three loosely coupled logical parts, one is set up for monitoring, another for alerting and a third for refining the unreachability incident to the most narrow scope possible. The entire system builds upon a client side, HTTP-based active probing infrastructure similar to Odin [5] described briefly in Section II. We use two systems for monitoring, `GA` to detect egregious incidents affecting large scopes, and `BB` to detect both smaller dips in unavailability and to incidents affecting regions where the number of probing requests per unit time has to be kept low. The alerting business logic, the probing infrastructure and the system to narrow the scope of affected regions are common to both the monitoring systems.

For practical reasons, we aggregate client data at multiple resolutions – at a country level to begin with, but then refine on-demand to finer resolutions to clearly define the boundaries, or scope of the affected user population. Such on-demand refinements play an important role in debugging incidents, discussed more in Section IV.

1) What is `health` *:* Our goal here is to focus on unavailability due to application agnostic factors that impact clients on the Internet, e.g., DNS resolution failures, routing issues, heavy congestion, etc.; `Mega` already employs cloud-hosted canaries to monitor the application itself. Therefore, we focus on a fixed request type — HTTP GETs of a small, static object — purpose-built for the monitoring we care about.

In our paper we ascribe a restricted and quantifiable meaning to '`health` '. Here it stands for the probability that a service-endpoint succeeds in responding correctly to a client's request. To make monitoring practical, clients are aggregated into client-sets, making `health` a property of (client-set,service-endpoint) pairs. Since these probabilities would be different for different request types, a fixed request type purpose built for monitoring is used as described in [5]. The requests are designed to elicit identical, uncached responses – an image that is small enough to fit in the payload of a single (typical) IP packet. The probability of failure of such out-of-band-HTTP-probes gives `health` an operational and measurable definition.

B. Gaussian Alert (`GA`) Model

The design of `GA` is based on traditional control charts introduced by Shewart as a means of statistical process control (SPC) [10]. This is applied to the stream of raw success rates computed as a ratio of the number of successful probes in a measurement interval (five minutes) over the number of probing requests made in the measurement interval.

An underlying assumption in SPC is that measurements (in our case, the outcome of a probe) taken from a process in control are random samples from an unchanging distribution with a desired finite mean μ and standard deviation σ. If `health` of our service is p, then the outcome of the probes correspond to a Bernoulli random variable with mean p and standard deviation $\sqrt{p(1-p)}$. In a traditional control chart, these raw measurements are sequentially averaged over some k observations at a time. As a consequence of the central limit theorem (CLT) the distribution of the averages approaches a Gaussian distribution, regardless of the distribution of the measurements (as long as the rather weak requirements of CLT are met). This Gaussian distribution has the same mean μ and a standard deviation of $\sigma/\sqrt{k}$.

CLT is an asymptotic phenomenon, the degree to which the distribution of the averages resembles a Gaussian distribution depends on k, as well as, how far the original distribution is from a Gaussian distribution. Our invocation of CLT to justify our Gaussian assumption is motivated by simplicity. Strictly speaking this is unnecessary as we can model the measurements exactly without the need for such approximations (as we shall do shortly when we present `BB`). However, the invocation

does simplify the detection algorithm – GA. In our case we obtain the mean $\mu = p$ and standard deviation $\sigma = \sqrt{\frac{p(1-p)}{k}}$ of the Gaussian distribution.

Typically, 3 sigma limits (or 99.7%) are used to set the upper and lower control limits in statistical process control charts i.e. the observed process is said to be "in-control" within the 3 sigma band. However; the width of band can be readjusted as an appropriate multiple t of standard deviation corresponding to any non-zero budget.

Restrictions of our data collection backend forced us to deviate a little from the standard algorithm. Rather than averaging every k measurements of raw data, online measurements are averaged over fixed interval of time (five minutes) yielding an effective k that varies dynamically with the number of measurements made in that interval. Even if one were to dynamically coalesce adjacent intervals, one need not obtain an equal number of attempts coalesced bin. The consequences of this limitation is, however, mild — rather than defining a fixed width band around the running mean, the width varies with the number of attempts made, because the standard deviation varies along with it.

C. Limitations of GA

Now, we discuss the various issues that motivate the design of our more sophisticated detection algorithm, BB.

1) Inadequacy of monitoring raw failure rates: While, it might be tempting to define health in terms of the raw failure rate, there are many reasons why this would be problematic: (i) the number of probing requests attempted over a fixed time interval is a function of number of clients online, consequently it varies considerably, yielding estimates with fluctuating confidences, (ii) the raw proportion of failures – computed as a ratio of number failures to the number of attempts – fluctuates, at times wildly, with a non-constant magnitude (scaling inversely as the square-root of the number of attempts), and finally, (iii) even if the number of requests were hypothetically fixed, the number of failed requests would be stochastic, not deterministic, even when the `health` is constant, much like the number of 'tail' outcomes of a coin tossed a fixed number of times.

2) Smart temporal averaging: One way to address the problem of fluctuations is to apply a simple smoothening filter on the raw failure rates, for example, exponentially weighted moving averages [11]. Such methods, however, fail to account for many complexities. First, variation in the number of probing requests continuously alters the uncertainty associated with the raw failure-proportion rate, as shown in Figure 2 (note that decrease in the number of attempts coincides with the increase in fluctuations in apparent failure rate). Simple smootheners do not track this dependency of fluctuations on the confidence band of the smoothened estimates, neither does it adjust the degree of smoothening on the number of attempts.

Another challenge is the sudden shifts in the `health` of the system, say due to a change in the underlying network conditions (e.g., congestion, routing instability). Such shifts mean that the measurements across adjacent intervals may

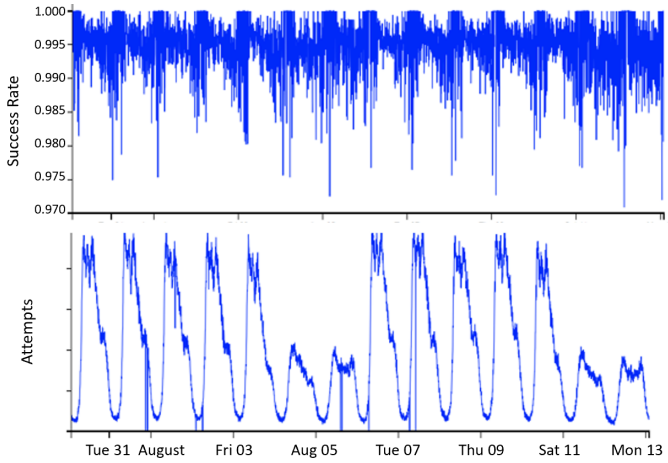

Fig. 2. Effect of fluctuations in the number of attempts (bottom) on the success rate (top).

not be identically distributed, i.e., the behaviour is non-IID. Kalman filtering is a classical approach for addressing non-IID behaviors [12]; however, it is only applicable for processes with a Gaussian distribution. In our setting, the health of the system typically remains close to 1.0. At such levels, the distribution of the successes (or failures) deviates strongly from a Gaussian distribution making Kalman filters unsuitable. Our approach with BB (described in III-D), deviates from the traditional Kalman filtering approach, in that, we use a Beta-Binomial distribution more fitting for our use-case.

3) Inadequacy of static thresholds for alerting: Once `health` has been estimated, we need to decide when to flag an unreachability alert. Even if `health` were to remain constant, the variation in the number of attempts would induce strong fluctuations on the observed failure rates. This precludes the use of a static alerting thresholds – when the number of probing attempts is low, the confidence in the observations is low as well, hence the observed failure rate needs to be egregiously low to warrant an alert.

While the time scale and the magnitude of fluctuations in health varies over networks, geographies and time periods, one also needs to account for the variations over time. For example, 5 shows periodic drops in health within a day, which we believe are caused by the rise and fall in the level of congestion depending on the time of the day. Although it is important that the `health` monitor reports low values during such drops, it is equally important that the alerting machinery ignores such consistent and predictable drops as long as they are within their expected limits.

D. Modeling and Monitoring Health in BB

Observed over short periods of time, the non-deterministic aspects of number of failed requests are captured well by a Binomial model. For the Binomial model to fit well, the probability of failure must remain constant over the observation period and the failures be independent. The former can be controlled by the length of the observation window and the latter by choosing a stateless request type whose

failures do not affect subsequent such requests. According to the Binomial model, the probability of k successes in N attempts is $\binom{N}{k}p^k(1-p)^{N-k}$ where p is the probability of success of a single request, typically close to 1.0. This is of significance because approximation of a Binomial distribution by a Gaussian breaks down at such extremes. The deviation from Gaussian behavior also makes it difficult to justify the use of classic Kalman filtering.

The problem of estimating health, as we have posed it in our paper, is akin to estimating the probability of a tossed coin landing as 'heads' from an online stream of tosses, with the important caveat that, unknown to us, the coin is secretly but non-adversarialy changed from time to time. In this streaming setting, our goal is to estimate the probability of the coin in use at any point in time.

1) Hidden Semi-Markov Binomial Process : If the underlying failure probability were to remain unchanged, the failure proportion computed over an infinite time window would be a good estimator of `health` . In the real world, however, `health` is non-stationary and we are restricted to finitely many measurements. To incorporate these, we extend the Binomial model by incorporating two key non-stationarities – (i) changes in the health over time and (ii) changes in the rate of attempts over time. Our Bayesian generative model[2] considers the attempt sequence as a fixed input and only tries to model the failure sequence conditioned on it.

Let $\mathbf{x}(t) = (s(t), a(t))$ denote a time indexed tuple of number of successes $s(t)$ and number of attempts $a(t)$, $t \in \mathbb{N}$. The tuple sequence $\mathbf{x}(t)$ is modeled as a discrete time stochastic process, with t incremented at every time step (for example, every 5 minutes) up to a finite horizon. Recall, we model $s(t)$ conditioned on $a(t)$, the latter being assumed given. We assume that `health` denoted by $\theta(t)$ is piece-wise constant in time, This allows incorporating temporal changes in `health` without any loss in generality as such *simple* functions can represent any *measurable signal* to arbitrary accuracy.

We define a parameterized formal specification for generating artificial data and tune its (hyper)parameters in the empirical Bayesian framework [13]. Since the (hyper)parameters are designed to be interpretable, their tuned values give us insight into the state of health of the system (for the client-set, service-endpoint pair) and how it fluctuates.

Our proposed generating model consist of three processes (i) a stochastic process for allocating temporal lengths to time segments that have constant health — we will call this the *segment generation* process (ii) a process to assign a health value to each segment defined by *segment generation* process – we will call this health assignment process and (iii) given a number of attempts made in each equal sized time bin lying in a segment, assign a number of successes according the Binomial distribution.

Segment generation: For the segment generation process one can chose any distribution over the integers, for example, Geometric, Negative Binomial, Poisson etc [14]. A choice other than the Geometric gives the process memory and hence a non-Markovian, in particular a Semi-Markovian character. This expands the capability of the model beyond that of Markovian models. In our evaluations Negative Binomial distribution performed better than the Geometric (indicating that the data is indeed non-Markovian) but at some cost of increased complexity of the implementation. As a practical trade off, We persisted with the Geometric distribution

$$\texttt{Geom}(t+1;\gamma) = \gamma(1-\gamma)^t,$$

where γ is the sole parameter of the distribution and $t+1$ represents the length we are modeling. The expected length is given as $\texttt{E[t]} = 1/\gamma$.

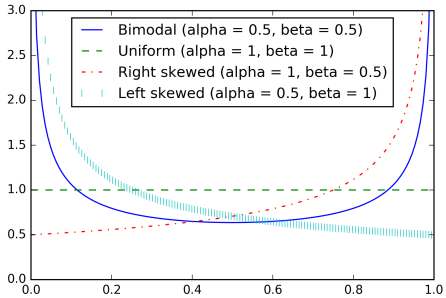

Fig. 3. Beta probability density functions (PDF's).

Health assignment: For the health assignment process, any model that assigns probability over $[0, 1]$ would suffice as its purpose is to assign a `health` value, a probability, to each segment. We opt for the `Beta`distribution, specified as

$$\texttt{Beta}(\theta;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1},$$

where $\Gamma(z)$ is the Gamma function defined as $\int_0^\infty x^{z-1}\exp^{-x}$ and α, β are its parameters. `Beta`distribution is particularly suitable for a few reasons – (i) it is a flexible class of distributions that can incorporate any of the following — uniform distribution, unimodal distribution, bimodal distribution, left skewed distribution, right skewed distribution etc as shown in Figure 3, (ii) it is a conjugate prior to the binomial distribution [15], our choice for the stochastic process assigning number of failures. This conjugacy property greatly simplifies the implementation as integrals necessary to compute in the intermediate steps have a closed form. Furthermore, the updated distribution $P(\theta|s)$ found after incorporating the data s turns out to be a `Beta`distribution again, but with different but easy to update parameters, as shown below.

$$\begin{aligned} P(\theta|s) &\propto \texttt{Binom}(s;\theta,a)\texttt{Beta}(\theta;\alpha,\beta) \\ &= \binom{a}{s}\theta^a(1-\theta)^{a-s}\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \texttt{Beta}(\theta;\alpha+s,\beta+a-s) \end{aligned}$$

These three subprocesses are assembled together as shown schematically in Figure 4 to give our final model. Further details of the model are explained in the appendix.

[2]A generative model is a formal specification that can generate artificial data, usually with the goal of matching characteristics of observed data.

Fig. 4. Probabilistic graphical model used by BB.

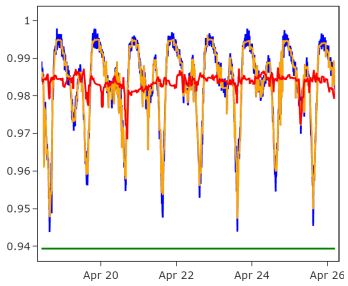

Fig. 5. Periodicity in health in China during 19 Apr - 26 Apr. Raw success rate in blue, estimated health in yellow, de-seasonalized health in red. Green indicates SLO.

E. Alerting on Unreachability Incidents

After estimating health , we turn to the question of alerting based on it. If a service-level objective could be set independently, say based on business considerations, our task would simply be to flag an unreachability incident when health dips below the chosen SLO. However, there are several complexities, as we discuss here.

1) Periodic fluctuation in health *:* We see that there is often a fluctuation in health , with a period of a day. For example, Figure 5 shows the health estimate for China for a week (Sunday, 19 Apr 2020 through Saturday, 25 Apr 2020). While puzzling at first, we believe these fluctuations are due to three primary reasons.

First, many enterprises block outbound probing traffic, for example, to prevent employees from accessing certain sites during office-hours. These attempts are still counted thereby pulling down the apparent health of the country. We verified this by discarding all probing attempts done by the enterprise networks. Second, the level of congestion could vary depending on time of day. We would expect the level of congestion, and hence the failure rate, to be higher during the daytime hours. However, the shifting of users (from residential networks to enterprise and the opposite) could also result in a less busy day and hence, low congestion. Third, the mix of client networks changes. For example, during day-time more users connect through enterprise networks whereas in the night they connect via a less-well-connected home networks.

In BB, we seek to neutralize the impact of such periodic fluctuations in health regardless of the cause. Our approach is based on Seasonality and Trend decomposition using Loess (STL) [16]. In a nutshell, STL helps to remove the the seasonality components, or periodicity, in the health time series. Compared to techniques such as Fast Fourier Transform that have a strict definition of periodicty, STL offers the advantage of being tolerant of jitter in the time period of the seasonality component. STL is used to de-seasonalize the median of the posterior of the health distribution. Figure 5 shows the the raw success rate (in blue), median of estimated health (in yellow), and de-seasonalized median (in red) , over 7 days for China. We see strong periodic fluctuations in raw success rate and estimated health. The magnitude of fluctuations are substantially diminished after de-seasonalization.

2) Data-driven SLO setting: The spread in Internet quality across locations results in variations in health . Consider, for instance, that USA had more instances of higher-than-target latency because of a tighter latency target compared to other countries [17]. Therefore, we set the SLO on a per-country basis.

The SLO assigned to countries characterizes their typical minimum quality of Internet connectivity. Therefore, it is natural to set SLO at a value that is breached only infrequently. We aggregate the CDF of the estimated health over a month on a rolling basis and set an upper bound to the SLO at 1st percentile of the monthly CDF. In addition to rarity of SLO breach, it is also desirable that alerting frequency be insensitive to small fluctuations in the SLO. This dictates that SLO passes through a low-density region of the health distribution. To avoid selecting a narrow crevice of density, the aggregated CDF is smoothened and then the rightmost local minimum of the CDF, smaller than the chosen upper bound, is deemed the SLO of the country.

IV. DrillDown

While monitoring and unreachability detection in NetDetector happens at the country level by default, it is important to resolve problems to a more fine-grained scope, where possible, to aid problem investigation and resolution. While fine-grained scopes can be defined in many ways, for the purposes of our discussion, we focus on the granularity of autonomous systems (AS), metros, and (AS, Metro) pairs. Here, metro denotes a metropolitan area, which could correspond to a city or even a larger region such as a state.

We devise the DrillDown technique for ascribing blame to one or more of the individual entities (ASes, metros, and/or (AS, Metro) combinations) contained within the country. We do this in two steps: first, we make a determination of whether any (AS, Metro) pair is affected; then we ascribe blame to possible units of failure in a manner that accounts for the ones that are (un)affected while remaining parsimonious (i.e., ascribing blame to as few entities as possible).

When NetDetector detects an unreachability incident at a country level, it tabulates measurement at the finer resolution of a (AS, Metro) pair. We compute health for such pairs, both during the period of the country-level incident (termed the "incident period") and also before and after it. We label a pair as "affected" if the magnitude of the dip in its health during the incident period is greater than that of the dip in health at the country level. If the dip in health of the pair is smaller than a quarter of the dip at the country level, we label it as "unaffected". Otherwise, we label it as "inconclusive".

Then, we proceed to ascribe blame to a suitable set of entities by using a MAXSAT solver, as explained in detail below. An *entity* could be anything whose failure can explain

the unreachability event, e.g., a link that is severed or an upstream AS with a routing loop. For the purposes of this paper, we restrict ourselves to the metros, ASes, and (AS, Metro) combination entities, in addition to the default country entity. If an entity is blamed, it also means all of its children would be "covered". For example, ascribing blame to a particular metro entity would cover all (AS, Metro) entities for that metro, while blaming the country entity would cover all AS, metro, and (AS, Metro) entities contained within that country. Conversely, if an entity is blamed but one of its children has been labelled as "unaffected", we term this as a mistake. A mistake is also incurred when an entity marked "affected" is left "uncovered". Our goal is to minimize such mistakes while being parsimonious in attributing blame.

`DrillDown` works by assigning a Boolean variable to each entity that could have failed. These variables are unknowns that need to be solved for, much like variables in a a system of simultaneous equations. A solution is an assignment of a Boolean value to all them. Assignment of 1 indicates that the corresponding entity has failed, whereas 0 indicates otherwise. The solver tries to find a suitable assignment that satisfies the set of clauses derived from health observations made at the resolution of (AS, Metro) pair.

Following Occam's Razor, not only do we want an assignment that satisfies the *hard* clauses, we also want to find the most frugal assignment. A `MAXSAT` solver tries to maximize the number of *soft* clauses it can satisfy in addition to all the *hard* clauses. As a result, we can encourage parsimony by adding each entity as a negated soft clause. This ensures that assigning a 1 to any Boolean variable, thereby potentially blaming more entities than necessary, is discouraged. For instance, a clause such as $\neg\text{Affected}(NYC)$ ensures that it gets uncovered (dissatisfied) whenever the metro NYC gets blamed. The consequence of posing the problem this way is that frugality is automatically incorporated.

Clause Generation: An observation that the $(\text{AS}_\text{i}, \text{Metro}_\text{j})$ pair is affected generates the *hard* disjunctive clause:

$$\texttt{Clause}_{\texttt{ij}} = \text{Affected}(\text{AS}_\text{i}) \vee \text{Affected}(\text{Metro}_\text{j}) \vee \text{Affected}(\text{AS}_\text{i} \wedge \text{Metro}_\text{j}).$$

This clause encodes that either the AS or the metro or the combination has incurred a failure and so must be blamed.

Similarly, the observation that the $(\text{AS}_\text{k}, \text{Metro}_\text{l})$ pair is unaffected generates the *hard* conjunctive clause

$$\texttt{Clause}_{\texttt{kl}} = \neg\text{Affected}(\text{AS}_\text{k}) \wedge \neg\text{Affected}(\text{Metro}_\text{l}).$$

Finally, we add a set of *soft* negated clauses, one each for every entity, for example

$$\texttt{Clause}_{\texttt{m}} = \neg\text{Affected}(\text{Metro}_\text{m})$$
$$\texttt{Clause}_{\texttt{n}} = \neg\text{Affected}(\text{AS}_\text{n})$$
$$\texttt{Clause}_{\texttt{o}} = \neg\text{Affected}(\text{AS}_\text{n} \wedge \text{Metro}_\text{m})$$

The variables that are assigned a value of 1 in the solution are interpreted as entities having blame ascribed to them.

While the combinations of possible user subsets (AS, Metro, (AS, metro)) could be large and this is further exacerbated by the fact that `MAXSAT` problems are NP-hard, there exist SAT-solvers made available by PySAT library [18] making `DrillDown` feasible.

V. Evaluation

We now report the overall statistics of the alerts thrown by each of the `GA` and `BB` detectors in `NetDetector` over a 4-month period, from 1st Jan through 7 May 2020. As noted earlier, `GA` has been in production use, so the alerts it throws are investigated and labeled, while `BB` is an experimental detector and so its alerts are not investigated. We adapt over evaluation methodology accordingly.

Table I reports the statistics of the alerts thrown by `GA`. A total of 49 unreachability alerts were thrown by GA during the period of our study, each of which was investigated by the on-call engineers (OCEs), by correlating with data from sources such as Down Detector [4], Thousand Eyes [19], and other manual processes. Based on these investigations, 42 out of the 49 alerts were deemed to be true positives and the remaining 7 false positives.

This methodology does not lend itself to a direct analysis of false negatives, since we do not know the ground truth, i.e., the full and complete list of actual unreachability incidents. However, we have qualitative evidence that `GA` has a low false negative rate; before `GA` was rolled out as the first detector in `NetDetector`, the operation team would hear customer complaints about problems that they were unaware of. But once `GA` was rolled out, such complaints became few are far between, suggesting that at least major incidents were likely not being missed. However, this does not rule out the possibility of there being less severe incidents that were both missed by `GA` and did not elicit customer complaints (or perhaps were transient and went away before users could start complaining). This point is pertinent when we discuss the `DrillDown` results in Section V-A.

Detector	# flagged	# TPs	# FPs
`GA`	49	42	7

TABLE I
UNREACHABILITY INCIDENTS STATISTICS FLAGGED BY `GA`.

Table II reports the statistics of the alerts thrown by `BB`. `BB` throws many more alerts than does `GA`— a total of 122 compared to 49. Since `BB` is experimental and its alerts are not investigated by the OCEs, we confine our analysis here to the universe defined by the 49 alerts thrown by `GA`, each of which was investigated, yielding 42 true positives and 7 false positives. Of the 42 true positives, `BB` flags 38, implying 4 false negatives. Of the 7 false positives, `BB` flags only 1. While we cannot, in the absence of full investigation, generalize to the full set of 122 alerts thrown by `BB`, these results do point to the promise of `BB`, specifically its ability to keep false negatives low (4 out of 42) and also false positives low (1 out of 7).

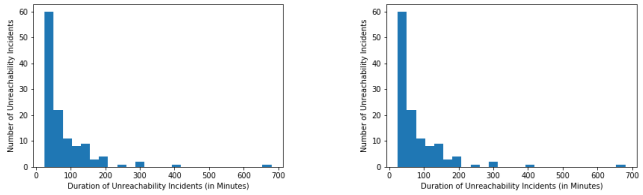

Fig. 6. Histogram of the duration of all unreachability incidents flagged by GA (left) and BB (right)

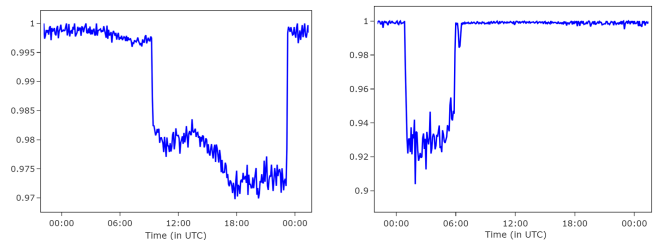

Fig. 7. Examples of long-lasting incidents for France on Apr 7 (left) and Denmark on Feb 18 (right).

In Section V-A, we will show that while many of the incidents flagged by BB were not flagged by GA and hence were not investigated, these do point to a significant dip in health at finer scopes, suggesting that there was indeed a real issue, though confined to a small subset of users.

Detector	# flagged	Confined to universe of 49 incidents flagged by GA		
		# TPs	# FNs	# FPs
BB	122	38	4	1

TABLE II
UNREACHABILITY INCIDENTS STATISTICS FLAGGED BY BB.

Next, we present the statistics of the alerts thrown by GA and BB, sliced along various dimensions.

1) Incidents by Duration: Figure 6 shows the histogram of the duration of unreachability incidents detected by GA and BB. The width of each bucket in the histogram is set by the Freedman–Diaconis rule [20], and is 22.79 minutes for GA and 26.2 minutes for BB. While the majority of incidents last under 10s of minutes, we note that a few span several hours. Figure 7 shows two such long-lasting incidents in France (825 minutes) and Denmark (305 minutes).

2) SLOs for Different Countries: In Table III, we report the SLO derived for a few countries. We see SLOs ranging from a high of 0.996 (i.e., "normal" corresponding a success probability of 99.6% for the HTTP ping measurements) down to 0.940. In general, these data-derived SLOs accord with what we would expect given the combination of location relative to Mega's cloud infrastructure, the economic level (as a proxy for the quality of Internet infrastructure), and the prevalence of middleboxes such as firewalls that could impede accesses.

A. DrillDown Analysis

Table IV shows the distribution of the finer-grained scopes blamed by DrillDown, corresponding to the incidents flagged by GA and BB.

Country	SLO	Country	SLO
UK	0.996	USA	0.995
Japan	0.995	Russia	0.991
India	0.988	China	0.940

TABLE III
THE DATA-DRIVEN SLOS FOR VARIOUS COUNTRIES

Detector	#Incidents	AS	Metro	AS,Metro
GA	49	79	0	130
BB	122	116	4	358

TABLE IV
DISTRIBUTION OF DrillDown BLAME ASSIGNMENT

The blame for a single incident may be ascribed to more than one type of entity, e.g., blame for an unreachability incident in Indonesia on 15 Feb 2020, blame was ascribed to the Central Java metro (i.e., all ASes there were affected) and also to specific (AS,metro) pairs elsewhere — AS 4787 (PT Cyberindo Aditama) in Jakarta and also AS 7713 (PT Telekomunikasi Indonesia) in West Java. These overlapping blame assignments cause the sum of columns 3-5 in Table IV to exceed the number of incidents flagged (49 and 122, respectively, for GA and BB). We observe that blame is most often ascribed to specific (AS,Metro) pairs and sometimes to entire ASes. Blame is rarely ascribed to an entire metro, which is as we would expect since the network infrastructure of the different ISPs in a metro is generally independent.

Next, we drill down into a few incidents. Specifically, we consider one incident each where there was a more significant dip in health at the level of an AS and an (AS,metro) pair compared to the dip at the country level. Both these are drawn from the alerts thrown by BB for which we do not have labels, since these alerts were not thrown by the GA detector in production and hence were not investigated by the OCEs.

Figure 8 shows two incidents (Bulgaria on 05 Feb 2020 and Ecuador on 13 Mar 2020) flagged by BB on the country-level. As depicted, the dip in health for AS 41313 (shown in (b)) is much sharper than that for another AS (AS 42794, Ultracom Ltd.). Similarly, in Ecuador we see the DrillDown ascribes blame to the pair of AS 27668 (ETAPA EP) and the metro Provincia del Guayas, for which health shows a much sharper dip than for the same AS in a different metro (Provincia de Pichincha) and also for the country of Ecuador. The time to execute DrillDown for Bulgaria is 9 seconds whereas for Ecuador is 2.7 seconds.

These results show that even when an incident is not investigated by the OCEs, DrillDown is able to find a smaller scope that is affected. In other words, these are likely to be true positives though limited in impact because of the specific scope that is affected.

VI. CONCLUSION

We have presented NetDetector, a system to detect unreachability in the context of a large online service, Mega, using HTTP pings. We have presented a simple approach, GA, to flagging unreachability incidents, which is in production

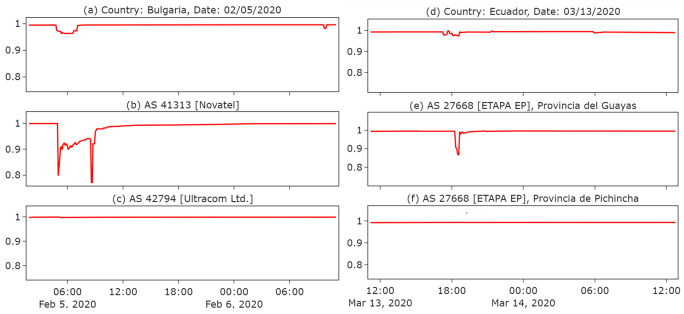

Fig. 8. `DrillDown` in action. (a) and (d) show incidents flagged by `BB` at the country-level, (b) and (e) are examples of scopes were the blame was ascribed which see a significantly larger dip, (c) and (f) are scopes that remain unaffected. The x-axis show the time of day (UTC). Time to execute `DrillDown`- 9 seconds and 2.7 seconds respectively.

use, and have also devised a principled, data-driven, experimental approach, `BB`, based on estimating the `health` of the clients' connectivity to the service, and changes therein, over time, and also what constitute "normal" health for a particular client set. We also present a novel formulation to drill-down into unreachability incidents to ascribe blame to more find-grained entities. A 4-month evaluation of `NetDetector` in collaboration with the operations team at `Mega` has shown promising results: a high rate of true positives for `GA` (42/49) and a low rate of false negatives for `BB` (4/42).

VII. Appendix

Let us introduce some notation to specify the details of the model used for segment generation (i.e., partitioning time into possibly unequal intervals of constant `health`) and health assignment (i.e., assigning a `health` to each interval). Our modeling horizon is finite and indicated by n, i.e., we model the observation for n discrete steps. Binary Vector $\tau \in \{0,1\}^n$ is n dimensional and defines the segment boundaries and let T be the set of all such vectors. A value of 1 at index i indicates that a segment ends at time step i. The right and left indices of a segment j is indicated by l_j and r_j, respectively. We assume that the first segment begins at index 0 and that the last segment may be open, i.e. $r_k \geq n$ where kth segment is the last. By definition, $0 = l_0 < r_0 = l_1 \cdots l_k \leq n-1$. Health vector $\theta \in [0,1]^n$ indicates the value of `health` at each index of time.

The set of segments induced by τ is $\pi(\tau) = \{(l_i, r_i) | \ \forall j \in [l_i, r_i) \tau_j = 0, \tau_{r_i} = 1, \text{if } l_i > 0 \ \tau_{l_i - 1} = 1\}$, i.e., the segments are contiguous and non-overlapping intervals in $[0, n-1]$, where the end of each segment is marked with a 1 in the τ vector. Similarly the set of segments induced by θ is denoted by $\pi(\theta) = \{(l_i, r_i) \mid \forall j,k \in [l_i, r_i) \ \theta_j = \theta_k, \ \theta_{r_i} \neq \theta_{r_i+1}\}$, i.e., the segments are defined by the piece-wise constant health. Mismatched segmentations, where the segments induced by τ and θ do not match, are assigned a probability of 0.

Let $\kappa = \sum_i \tau_i$ denote the total number of segments in the n time steps modeled. According to our model, the joint distribution $P(\tau, \theta, \mathbf{x})$ over the variables $\tau, \theta, \mathbf{x}$ factorizes as

$$P(\tau, \theta, \mathbf{x}) = P(\tau)P(\theta|\tau)P(\mathbf{x}|\theta). \tag{1}$$

In other words, we first segment time into intervals of constant health. Then given this segmentation, we assign the health corresponding to each segment. Finally, given the estimated health, we generate the success/failure counts for each interval. Note we have overloaded the notation $P(\cdot)$ to denote different probabilities — these are given as:

$$P(\tau) = \begin{cases} \prod\limits_{j \in [0 \cdots \kappa)} \gamma(1-\gamma)^{r_j - l_j}, \text{if } \tau_{n-1} = 1 \\ (1-\gamma)^{r_\kappa - l_\kappa} \prod\limits_{j \in [0 \cdots \kappa - 1)} \gamma(1-\gamma)^{r_j - l_j} \text{otherwise} \end{cases}$$

$$P(\theta|\tau) = \begin{cases} 0 & \text{if } \pi(\tau)! = \pi(\theta) \\ \prod\limits_{j \in [0 \ldots \kappa)} \texttt{Beta}(\theta_{l_j}, \alpha, \beta) & \text{otherwise} \end{cases}$$

$$P(\mathbf{x}|\theta) = \prod \lim_{j \in [0 \ldots n)} \texttt{Binom}(\mathbf{s}_j, \mathbf{a}_j, \theta_j)$$

The posterior $P(\mathbf{x}|\theta)$ over the latest `health` is obtained as

$$P(\theta_n|\mathbf{x}) = \sum_{\theta_1 \ldots \theta_{n-1}} P(\theta|\mathbf{x}) \text{ where} \tag{2}$$

$$P(\theta|\mathbf{x}) = \sum_{\tau \in T} P(\tau, \theta|\mathbf{x}) = \sum_{\tau \in T} \frac{P(\tau, \theta, \mathbf{x})}{\sum_{\tau \in T} \int_\theta P(\tau, \theta, \mathbf{x})}. \tag{3}$$

Let us denote the denominator $\sum_{\tau \in T} \int_\theta P(\tau, \theta, \mathbf{x})$ by $Z(0, n)$. Using the factorization of the joint distribution (1) we obtain

$$Z(0,n) = \sum_{\tau \in T} P(\tau) \int\limits_{\theta | \pi(\tau) = \pi(\theta)} P(\theta|\tau)P(\mathbf{x}|\theta) \text{ where}$$

$$\int\limits_{\theta | \pi(\tau) = \pi(\theta)} P(\theta|\tau)P(\mathbf{x}|\theta)$$

$$= \prod_{j \in [0 \ldots n)} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(a_j + 1)}{\Gamma(s_j + 1)\Gamma(a_j - s_j + 1)}$$

$$\times \frac{\Gamma(\alpha + s_j)\Gamma(\beta + a_j - s_j)}{\Gamma(\alpha + \beta + a_j)}.$$

The expressions above may be substituted in equation (2) to obtain the posterior probability of the health of the system. Computing the sum over all partitions, $Z(0, n)$, is exponential in n, so we resort to dynamic programming method and form the following recursive relation

$$Z(0,n) = \sum_i Z(0,i)\texttt{Geom}(n-i, \gamma) \times \int_\theta \texttt{Beta}(\theta; \alpha, \beta) \prod_{i \cdots n} \texttt{Binom}(s_i; a_i, \theta).$$

A. Hyperparameter Estimation

The set of hyperparameters that need to be tuned are (i) γ corresponding to the segment length distributions and (ii) α, β that characterize the `Beta`priors. These parameters are tuned per (country, end-point) pair. In order to tune these values we compute the marginalized likelihood of the observed data as a function of these parameters using equation (1). Following the empirical Bayes principle [13] We choose that value of parameter that maximizes this marginalized likelihood.

References

[1] "Amazon CloudWatch," https://aws.amazon.com/cloudwatch/.

[2] "Stackdriver," https://cloud.google.com/monitoring/.

[3] "Azure Monitor overview," https://docs.microsoft.com/en-us/azure/azure-monitor/overview.

[4] "DownDetector," https://downdetector.com/.

[5] M. Calder, R. Gao, M. Schröder, R. Stewart, J. Padhye, R. Mahajan, G. Ananthanarayanan, and E. Katz-Bassett, "Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System," in *NSDI*, 2018.

[6] "Amazon Compute Service Level Agreement," https://aws.amazon.com/compute/sla/.

[7] "Google Compute Engine Service Level Agreement (SLA)," https://cloud.google.com/compute/sla.

[8] "SLA summary for Azure services," https://azure.microsoft.com/en-in/support/legal/sla/summary/.

[9] "Cloudflare outage on July 17, 2020," https://blog.cloudflare.com/cloudflare-outage-on-july-17-2020/.

[10] W. Shewhart and W. Deming, *Statistical Method from the Viewpoint of Quality Control*. Courier Corporation, 1986.

[11] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting with Exponential Smoothing: The State Space Approach*. Springer-Verlag, 2008.

[12] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*. Prentice Hall, 2000.

[13] H. Robbins, "An empirical bayes approach to statistics," *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, 1956.

[14] W. Feller, *An Introduction to Probability Theory and Its Applications*. Wiley, 1968, vol. 1.

[15] J. M. Bernardo and A. F. M. Smith, *Bayesian Theory*. New York: John Wiley & Sons, 1994.

[16] W. S. Cleveland and S. J. Devlin, "Locally weighted regression: An approach to regression analysis by local fitting," *Journal of the American Statistical Association*, vol. 83, pp. 596–610, 1988.

[17] Y. Jin, S. Renganathan, G. Ananthanarayanan, J. Jiang, V. N. Padmanabhan, M. Schroder, M. Calder, and A. Krishnamurthy, "Zooming in on Wide-area Latencies to a Global Cloud Provider," in *ACM SIGCOMM*, 2019.

[18] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437.

[19] "ThousandEyes," https://www.thousandeyes.com/.

[20] "Freedman–Diaconis Rule," https://en.wikipedia.org/wiki/Freedman-Diaconis_rule.

[21] P. Gu, J. Purdom, J. Franco, and B. Wah, "Satisfiability problem: Theory and applications, chapter algorithms for the satisfiability sat problem: A survey," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 19–152, 01 2002.

[22] "RIPE Atlas," https://atlas.ripe.net/.

[23] "SamKnows," https://samknows.com/.

[24] "Internet Outage Detection and Analysis (IODA)," https://ioda.caida.org/.

[25] "Dynatrace," https://www.dynatrace.com/.

[26] "AS Rank of AS 9121 (TTNet)," https://asrank.caida.org/asns?asn=9121&type=search.

[27] W. Willinger, V. Paxson, R. H. Riedi, and M. S. Taqqu, "Long-range dependence and data network traffic," 2001.

[28] L. Quan, J. Heidemann, and Y. Pradkin, "Trinocular: Understanding Internet Reliability Through Adaptive Probing," in *ACM SIGCOMM*, 2013.

[29] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An Information Plane for Distributed Services," in *OSDI*, 2006.

[30] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. E. Anderson, "Studying Black Holes in the Internet with Hubble," in *NSDI*, 2008.

[31] E. Katz-Bassett, C. Scott, D. R. Choffnes, Í. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy, "LIFEGUARD: Practical Repair of Persistent Route Failures," in *ACM SIGCOMM*, 2012.

[32] A. Schulman and N. Spring, "Pingin' in the Rain," in *ACM IMC*, 2011.

[33] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data," in *ACM CoNEXT*, 2007.

[34] Í. Cunha, R. Teixeira, N. Feamster, and C. Diot, "Measurement Methods for Fast and Accurate Blackhole Identification with Binary Tomography," in *ACM IMC*, 2009.

[35] A. Dainotti, R. Amman, E. Aben, and K. C. Claffy, "Extracting Benefit from Harm: Using Malware Pollution to Analyze the Impact of Political and Geophysical Events on the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 31–39, 2012.

[36] A. Dainotti, C. Squarcella, E. Aben, K. C. Claffy, M. Chiesa, M. Russo, and A. Pescapé, "Analysis of Country-wide Internet Outages Caused by Censorship," in *ACM IMC*, 2011.

[37] P. Richter, R. Padmanabhan, N. Spring, A. Berger, and D. Clark, "Advancing the Art of Internet Edge Outage Detection," in *ACM IMC*, 2018.

[38] R. Padmanabhan, A. Schulman, D. Levin, and N. Spring, "Residential Links Under the Weather," in *ACM SIGCOMM*, 2019.

[39] D. Shipmon, J. Gurevitch, P. M. Piselli, and S. Edwards, "Time series anomaly detection: Detection of anomalous drops with limited features and sparse examples in noisy periodic data," Google Inc., Tech. Rep., 2017. [Online]. Available: https://arxiv.org/abs/1708.03665

[40] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato, "BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks," in *USENIX ATC*, 2014.

[41] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-Wide Traffic Anomalies," in *ACM SIGCOMM*, 2004.

[42] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP Misconfiguration," in *ACM SIGCOMM*, 2002.

[43] R. Teixeira and J. Rexford, "A Measurement Framework for Pin-Pointing Routing Changes," in *ACM SIGCOMM workshop on Network troubleshooting*, 2004.

[44] M. Zhang, C. Zhang, V. S. Pai, L. L. Peterson, and R. Y. Wang, "PlanetSeer: Internet Path Failure Monitoring and Characterization in Wide-Area Services," in *OSDI*, 2004.

[45] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: Pushing Experiments to the Internet's Edge," in *NSDI*, 2013.

[46] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California Fault Lines: Understanding the Causes and Impact of Network Failures," *ACM SIGCOMM*, 2011.

[47] D. Koller and N. Friedman, *Probabilistic Graphical Models*. MIT Press, 2009.