Communication-Efficient Tracking of Distributed Cumulative Triggers

Ling Huang* Minos Garofalakis† Anthony D. Joseph* Nina Taft‡
*UC Berkeley †Yahoo! Research ‡Intel Research Berkeley
{hling, adj}@cs.berkeley.edu minos@yahoo-inc.com nina.taft@intel.com

Abstract

In recent work, we proposed D-Trigger, a framework for tracking a global condition over a large network that allows us to detect anomalies while only collecting a very limited amount of data from distributed monitors. In this paper, we expand our previous work by designing a new class of queries (conditions) that can be tracked for anomaly violations. We show how security violations can be detected over a time window of *any* size. This is important because security operators do not know in advance the window of time in which measurements should be made to detect anomalies. We also present an algorithm that determines how each machine should filter its time series measurements before back-hauling them to a central operations center. Our filters are computed analytically such that upper bounds on false positive and missed detection rates are guaranteed. In our evaluation, we show that botnet detection can be carried out successfully over a distributed set of machines, while simultaneously filtering out 80 to 90% of the measurement data.

Keywords: Distributed Triggering, Network Monitoring, Anomaly Detection, Data Aggregation, Queueing Theory.

1 Introduction

Network-wide anomaly detection systems rely on large-scale distributed monitoring systems to collect, aggregate and present information describing the status and performance of the network under observation. Many types of networks, including server clusters, enterprise networks, ISPs, and sensor networks, employ such systems to track the health of their networks. Remote monitor sites are typically deployed throughout the network and, thus, their data streams present information from multiple vantage points. The ensemble of these monitors leads to the creation of numerous, large, and widely-distributed time-series data streams that are continuously monitored and analyzed for anomalous conditions, either benign or malicious. In typical monitoring systems, all the measurement data collected across the remote monitors is shipped to a data fusion center (i.e. a network operations center) for processing. The fusion center can raise alarms should it detect anything abnormal.

Today's distributed monitoring systems suffer from two important limitations. The first is scalability - both in terms of how fast detections can be made (i.e. it is desirable to shrink the time scale from hours and minutes to seconds and milliseconds), and in terms of how many distributed monitors can be tracked simultaneously. The second limitation (as in recently proposed large-scale monitoring systems [4, 12, 20]) is their lack of flexibility in detecting when a sophisticated *global condition* across a set of distributed machines exceeds acceptable levels.

To address the scalability issues, we designed D-Trigger, a general framework in prior work [9, 10] for global constraint tracking. The key to achieving scalability is to reduce the amount of data needed for anomaly detection. In [10], we provided an example of a particular anomaly detector that when re-designed to fit in our framework, could continue to achieve high detection accuracy while only using 10-20% of the original data. In this paper, we focus on the problem of lack of sophistication in the types of conditions that can be tracked for violations.

The new queries and their supporting algorithms that we develop here, were designed in accordance with our framework. This framework is described in Sec. 2, but we briefly mention a few points here. To reduce the amount of data transmitted through the network, we advocate engaging the monitors in filtering their own data, and only sending the operations center new data "as needed". What data is needed by which machine depends upon the global constraint being tracked. Because the operations center is doing the detection, and it now operates under a limited view of the global data, it can make mistakes. To bound these mistakes, the filtering at the monitors needs to be done in coordination with the operations center. Our framework proposes that the filtering be done in such a way that detection errors are bounded.

In order to enable broader and more flexible conditions for triggering, we introduce a new class of triggers, called *cumulative triggers*. These triggers allow one to detect cumulative violations that are persistent over time and are spread across a distributed set of machines. One of our main contributions is to enable such potential violations to be measured over time without specifying a fixed window size *a priori*. Our key enabling insight is that the filter parameters needed to support these triggers can be viewed as analogous to queue sizes.

The motivation for such triggers comes from the following. Current designs for large-scale monitoring systems focus solely on *instantaneous* trigger conditions, where the goal is to fire the trigger as soon as the aggregate (i.e., `SUM`) of observations (e.g., CPU utilization or number of messages sent)

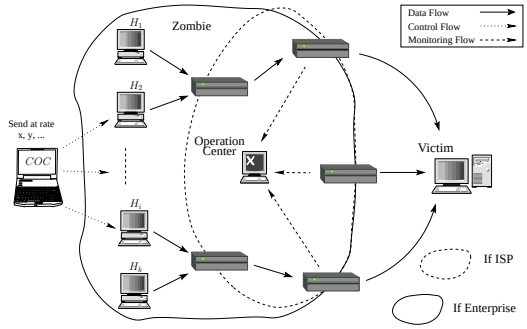


Figure 1: Example: Distributed detection of botnet attacks.

across distributed machines exceeds a pre-specified threshold. While such triggers are undoubtedly useful, they are limited when it comes to monitoring distributed phenomena that are inherently *bursty*, such as network traffic and server load. Fixing appropriate instantaneous threshold conditions for anomaly detection can easily lead to numerous false positives and negatives. Exceeding a threshold for a short period of time could very well be allowed as natural bursty behavior; on the other hand, even violations that are small in magnitude could be harmful if they are allowed to *persist over time*. Persistent violations are better observed by measuring activity over a window of time. However, the task of selecting a particular window size over which something is measured is a headache that has long plagued operators because a single window size cannot accommodate all of their needs. In addition to ISPs, managers of distributed server systems have found that measuring average server load using fixed sized windows is unsatisfactory in that it is insufficient to identify good or bad system behavior [23].

One could implement a cumulative trigger by naively pushing all of the monitoring data to the central operations site. However, following the philosophy advocated in our framework, we design filtering schemes so as to accurately detect cumulative trigger violations with limited data. Our second main contribution is thus an algorithm for computing the filtering parameters at each local monitor. Based on our analogy to queues, we use queueing theory as an analytical tool. Our analytical solution provides guarantees that user supplied target false alarm and missed detection rates will be met. These user-supplied target error rates enable the tradeoff between reducing communication overhead and alarm detection accuracy.

We apply our algorithms in the context of botnet detection as depicted depicted in Fig. 1. In botnet attacks, multiple zombies (recruited machines) try to open a large number of TCP connections to a single server (the victim). Assume a set of hosts have been recruited by a botnet, some subset of which reside in our given network. An external commander gives them an order to launch a large number of connections to a victim, that in this case resides outside our network. In many cases, tracking the traffic level at each individual host may not raise any serious alarms (e.g., intelligent botnets prevent compromised machines from transmitting at their maximum level so as to evade detection). However, by tracking the `SUM` of the number of simultaneous TCP connections from these hosts to a given destination, the excess or overload would be more easily revealed[1]. Upon detection ISPs could block this unwanted traffic via filters on the gateways. In an enterprise network the hosts could simply be disconnected.

In our experimental evaluation, we use real-life distributed data streams collected from PlanetLab IDS monitors. We demonstrate that our schemes can easily guarantee target accuracy levels of around 98% while typically sending less than 20% of the original time-series data (i.e., a communication reduction of over 80%).

2 Background

A basic distributed triggering system consists of a set of widely distributed monitoring nodes $m_1, m_2, \ldots, m_n$ and a coordinator node X. Each monitor continuously produces time series signals $r_i(t)$ on the variable(s) or condition(s) selected for monitoring. Examples of a monitor's output include: number of SYN requests per second, number of DNS transactions per hour, volume of traffic per minute at port 80, and so on. These time series signals are sent to coordinator X. The coordinator acts as an aggregation and detection point, whose purpose is to track conditions across its monitors and to fire a trigger whenever some constraint on the aggregate behavior of a subset of nodes is violated. The coordinator can aggregate the incoming time series signals using any typical aggregation function - such as `SUM, AVG, MIN, MAX`, etc. We focus herein on the linear SUM aggregator because our goal is to enable the SUM aggregation over time-varying windows, and because this aggregator is useful for botnets. In [9] we illustrated that our framework can support more complex correlation functions such as the top eigenvalues of the global measurements matrix.

Our framework advocates a philosophy and provides design guidelines for supporting anomaly detectors over such large-scale distributed monitoring systems. Our belief is that many anomaly detection applications can be successful without backhauling *all* of the monitored data. Because our applications are focused on anomaly detection, most of the time the traffic will be "normal" and there is no need to send such data to the fusion center. The goal is thus to send only the monitoring data that is "needed" for the anomaly detector to work properly. To achieve this, we propose engaging the monitors in filtering their data locally by installing triggers on their machines. The monitors *only* send updates to the data fusion center when the local trigger fires.

The approach of filtering the measurement data at the local monitors has critical implications for detection accu-

[1] A caveat is that the overload could be due to a flash crowd.

racy, because the operations center (called 'coordinator' hereafter) carries out the detection. The coordinator's view of the global state will be approximate because some elements of the global data can become stale (until the next update). Because of its approximate view of the global state, the coordinator could make errors in detection. The coordinator can make two kinds of mistakes: either a violation among monitors occurs and the coordinator fails to catch it (called a *missed detection*), or no violation occurs yet the coordinator thinks that one has (called a *false alarm*).

Our framework advocates that if this approximate view of the global state at the operations center is carefully managed, then accurate detection can nevertheless still be achieved. The idea is to bound how 'approximate' or perturbed it is allowed to become. The filtering scheme needs to be selected so as to ensure these bounds are never exceeded. Since, different security applications require tracking of different conditions, each new anomaly detector that is supported requires the development of a new filtering scheme along with an analytical method for guaranteeing the detection accuracy bounds are met. In designing the filtering, we leverage the coordinator's global view by having it choose the level of accuracy that each monitor must report. This last point indicates some of the complexities of our problem setting, since local filtering needs to be done based upon the global constraint that is being tracked.

A nice property of our framework is that it achieves *continuous* tracking, where by 'continuous' we mean the following. The remote monitors can collect data as fast as they are designed to operate (i.e., this could be 5 minutes if they collect SNMP data, or on the order of a few milliseconds, if they collect flow statistics - such as netflow). Today monitors either summarize their data by aggregating at coarse time scales (e.g., computing averages over long windows of multiple minutes) before sending it to an operations center, or they send every measurement they make. Neither of these options is desirable. In our framework, the local monitors do continuous monitoring and the operations center has the *perception* that it receives all the measurements. The filtering strategies are chosen so that the operation center finds out anything important (relative to the anomalies it tracks) as soon as something happens, thereby essentially achieving continuous monitoring.

Prior Work. Database research on continuous distributed query processing has considered similar environments [3, 5, 13, 17]; however, the focus is on the accurate estimation of the aggregate signal itself rather than catching a constraint violation. Jain *et al*. [14], propose using uniform thresholds across all monitors, and eventually detect instantaneous threshold violations without giving any guarantees on the size of the violation; in contrast, we place strict bounds on the size of the violation that our schemes seek to enforce within specified error rates. Dilman and Raz [7] propose algorithms for detecting whether the sum of a set of numeric values from distributed sources exceeds a user-supplied threshold value. More recently, Keralapura *et al*. [15], formalized the instantaneous thresholded counting problem and gave static and adaptive algorithms, as well as a detailed optimality analysis. Our approach goes further by providing both a firm detection guarantee for *cumulative* trigger conditions, as well as the flexibility for users to trade off communication overhead with detection accuracy.

Our prior work in [10] provides a solution for the particular application of volume anomalies using instantaneous triggers combined with PCA techniques. The methods proposed in this paper are fundamentally different: Instead of PCA and matrix perturbation theory [9, 10], our results here are based on queueing theory. The need for different methods result from (i) the different types of triggers (instantaneous versus cumulative) that are supported; and, (ii) the different accuracy guarantees provided. In [10], we offer guarantees only on false alarm rates, whereas in this work we provide guarantees on both false alarm rates and missed detection rates.

3 Cumulative Triggers

We now explain a new kind of threshold violation that can be detected through the use of *cumulative triggers*. The goal is to detect violations that persist in time, without having to specify a fixed aggregation time window beforehand. Let C denote the network-wide trigger threshold that is resident at the coordinator. In cumulative triggers, the threshold condition is defined in terms of the accumulated excess *area* of the aggregate signal over time: (bytes $\times$ time) or (number of connections $\times$ time). Abstractly, a cumulative trigger condition should fire when the excess area of the observed aggregate signal over a time window *of any size*, exceeds the pre-specified cumulative threshold. Such cumulative triggering conditions cannot be captured using existing `SUM`-trigger mechanisms based on instantaneous sums of local values [14, 15].

To capture temporally-persistent phenomena, the coordinator computes a violation penalty that accrues over time, and fires the trigger condition when the penalty becomes excessive. During a time window of size $\tau = \tau(t)$, the penalty at time t accrued over the interval $[t - \tau, t]$ is defined to be

$$V(t, \tau) = \max\{0, \int_{t-\tau}^{t} \sum_{i=1}^{n} r_i(w)dw - C \cdot \tau\}.$$

(We maximize this term with zero to keep the penalty non-negative.) Our cumulative triggering mechanism does not depend on any fixed window τ; instead, a cumulative trigger fires at time t if penalty $V(t, \tau) > \epsilon$ *for any* window size $\tau \in [1, t]$. Thus, intuitively, we fire the trigger if there is *some time window* that causes the cumulative penalty to exceed the ϵ constraint; or, more formally, if $\max_\tau\{V(t, \tau)\} > \epsilon$, where $\max$ is computed on all possible τ over the entire

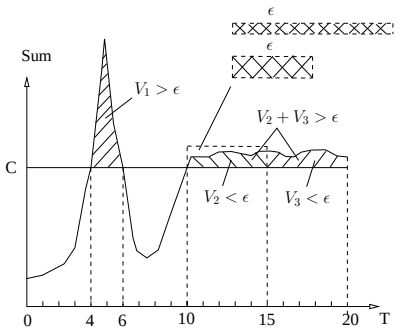


Figure 2: Cumulative violations.

signal history. One of our key insights in this work is that, by exploiting an analogy to queuing theory, our system can track cumulative trigger conditions effectively, without having to retain the entire signal history or check the condition against all possible τ.

We allow the user or network operator to specify an error tolerance ϵ which indicates that it is sufficient to track the global state (i.e., the aggregated time series) approximately with an error bounded by ϵ. We will exploit this error tolerance to gain additional savings in communication overhead.

Earlier work on distributed triggers [7, 15] has focused solely on *instantaneous* threshold conditions, where the goal is to detect if $\sum_i^n r_i(t)$ exceeds a threshold C at any time t. An generalization of the instantaneous case are *fixed-window triggers*, where the goal is to detect the condition $V(t,\tau) > \epsilon$ at any time t, *for a given, fixed time window* τ.

Instantaneous and fixed-window triggers are inherently limited when it comes to signals where transient bursty behavior is the norm, such as IP network traffic. Depending on the threshold value, an instantaneous trigger may easily overreact to natural, transient phenomena. With fixed-window triggers, choosing the right window size τ can be problematic for several reasons. If we use a small τ (short window), and the violation lasts for a long time but is small in magnitude, the system is likely to miss it altogether. For example, in Fig. 2, the persistent (but small) violation occurring in time slots $[10, 20]$ could go undetected with a window size of $\tau = 5$ because the penalty (over any 5 time slots), V_2 or V_3, does not grow to exceed ϵ. If, on the other hand, the violation were short in duration but large in magnitude, the system would miss it if a large τ (long window) is used. In our example figure, a short but large violation occurs during the time period $[4, 6]$. With a window of size 5 time units, this violation is likely to get averaged out because the positive penalty in period $[4, 6]$ is canceled out by the negative contribution in period $[3, 4]$ (or, $[6, 7]$). This illustrates the difference between fixed sized windows and cumulative violations with varying window sizes. With a fixed window of size 5, both these violations would have been missed. However, with cumulative conditions, a window of size 10 would have caught the violation in $[10, 20]$ in our first example since the penalty $V_2 + V_3$ exceeds ϵ; a window size of 2 would detect the violation in $[4, 6]$ in our second example since the penalty V_1 does exceed ϵ. We can thus see the flexibility in not having to specify a priori the time window over which a potential violation is measured.

4 Problem Statement

Because our approach of using limited data introduces errors, we allow the network operator to input their tolerable error levels. We allow three such inputs. The first input, ϵ specifies the error tolerance on the size of the violation. We also allow network operators to input their tolerance on false alarms and missed detections.

In our system, a *missed detection* occurs if $\max_\tau\{V(t,\tau)\} > \epsilon$ and the system does *not* fire the corresponding trigger. Conversely, a *false alarm* occurs whenever $\max_\tau\{V(t,\tau)\} \leq \epsilon$ and the system fires a trigger. We define the *missed-detection rate* β as the fraction of missed detections over the total number of real violations, and the *false-alarm rate* η as the fraction of false alarms over the total number of triggers fired. Allowing β and η to be inputs, creates a flexible system in which different deployments can be tailored to their own needs. For example, some systems may consider minimizing false alarms more important than minimizing missed detections; other systems may take the opposite view.

The user input thus constitutes a triple (ϵ, β, η) that essentially denotes the accuracy level that our tracking schemes target. The problem we address herein is to design the protocols resident at the monitors and at the coordinator in order to *guarantee* that the coordinator's trigger fires, whenever $\max_\tau\{V(t,\tau)\} > \epsilon$, *at any time* t, with (ϵ, β, η)-accuracy while simultaneously keeping communication overhead low.

To simplify the exposition, our discussion assumes that communication between the monitors and the coordinator are instantaneous. In the case of non-trivial delays in the underlying network, techniques based on time-stamping and message serialization can be employed to ensure correctness, as in [17].

5 Distributed Cumulative Triggers

5.1 The Queueing Model

Our approach to **supporting cumulative violations** without having to specify windows of time a priori is to use insights from queueing theory. Earlier work on data streaming uses *window-based* stream processing [6, 8] and focuses only on the case of (time- or arrival-based) windows *of fixed size* over the stream. Such techniques are not useful in our case, since the window sizes of the (potential) trigger violation are not known ahead of time. Instead, our key observation is that

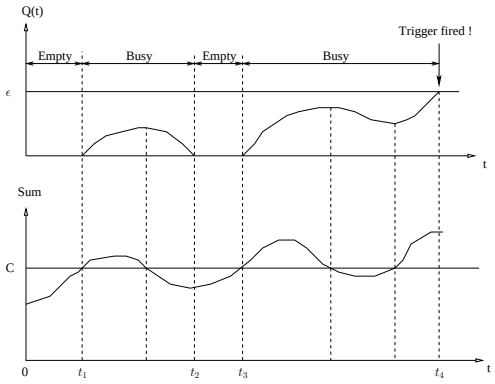


Figure 3: Queueing model for a cumulative trigger.

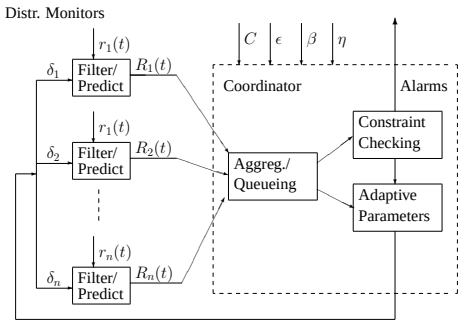
we can accurately model the monitoring of a cumulative trigger condition (see Sec. 3) using a simple *queueing model* as stated by the following theorem[2].

Theorem 1 *Consider a queue of size ϵ with an arrival rate equal to the actual aggregate signal $\sum_{i=1}^{n} r_i(t)$ and a drain (i.e., service) rate equal to the trigger threshold C. A cumulative trigger should fire (i.e., $\exists\tau$ s.t. $V(t,\tau) > \epsilon$) if and only if the above queue overflows.*

Essentially, cumulative triggering aims to guarantee that $\sum_i r_i(t)$ does not exceed C in the long-term, however, it allows $\sum_i r_i(t)$ to be bursty (i.e., $\sum_i r_i(t)$ can be *any* amount larger than C in *any* time window, but the volume of the burstiness should not exceed ϵ). Thus, cumulative triggering does not care about instantaneous sums or averages over a fixed size window; it cares only whether (across *any possible time scale*) the accumulated violation (penalty) exceeds ϵ and causes queue overflow.

As an example, the bottom half of Fig. 3 depicts a sample aggregate time-series signal $\sum_{i=1}^{n} r_i(t)$, while the top half shows the occupancy of the above-described queue, $Q(t)$, over time. Clearly, if the queue overflows at some time t, then there must be some time $t^s < t$ denoting the start of a *busy period* $[t^s, t]$ (*i.e.*, a period during which the queue is persistently non-empty; that is, $t^s = \max\{x | x \leq t$ and $Q(x) = 0\}$) ending at t with a queue occupancy $Q(t) \geq \epsilon$. Fig. 3 shows two busy periods, $[t_1, t_2]$ and $[t_3, t_4]$, the second of which results in sufficient queue buildup to fire the trigger. It is not difficult to see that, by our queueing model, $Q(t) = V(t, t - t^s)$, so that $Q(t) > \epsilon$ (*i.e.*, a queue overflow) indeed implies that our trigger should fire. Similarly, for any time window $\tau \leq t$, $V(t, t - t^s) \geq V(t, \tau)$ (i.e., windows smaller or larger than the latest busy period can only reduce the cumulative size of the violation). In other words, $Q(t) = V(t, t - t^s) = \max_\tau\{V(t, \tau)\}$, implying the cumulative trigger should fire if and only if the queue overflows. The model in Theorem 1 captures the equivalence between an overflowing queue and a violation of the cumulative trigger constraint.

[2]Due to lack of space, the proofs for all of our theorems are omitted, but can be found in [11].



Symbol	Meaning
X	Coordinator, coordination and detection center
m_i	Monitor sites ($i = 1, \ldots, n$)
$r_i(t)$	True local time-series signal at m_i
$R_i(t)$	Most recent prediction model for $r_i(t)$
C	Trigger threshold
ϵ	Error tolerance for threshold violation
δ_i	Local monitor slack parameters
θ	Coordinator slack parameter
β	Miss detection (*i.e.*, false negative) rate
η	False alarm (*i.e.*, false positive) rate

Figure 4: Our distributed trigger tracking framework.

5.2 Our Trigger-Tracking Protocols

The architecture of our system is depicted in Fig. 4. The role of the monitor is to track it's own time series data and to decide when to send the coordinator an update based on a filtering scheme. Let $r_i(t)$ denote the actual time series observed at monitoring node i. If a monitor decides at time t^{prev} to send the coordinator a sample of its data, it sends $R_i(t^{prev})$, which is an approximate representation of $r_i(t^{prev})$. If in a subsequent time $t > t^{prev}$, the monitor sends nothing, then the coordinator assumes that $R_i(t^{prev})$ is a good approximation for $r_i(t)$. In general, $R_i(t)$ can be based on any type of *prediction model* for monitor m_i. For example, a simple model is to set $R_i(t^{prev}) = r_i(t^{prev})$ at the update time. Time series prediction models and other sophisticated prediction models [5, 13] could also be used.

The role of the coordinator is twofold. First, it makes global anomaly detection decisions based a queueing model with parameter θ, using the received updates $R_i(t)$ from the monitors. Second, it computes the slack parameters δ_i for all the monitors based on its view of the global state and the condition for triggering an anomaly. The slack parameters δ_i are sent to the monitors whenever they change. The monitors use slack parameters when tracking the drift between the actual time series signal and the prediction function; whenever this drift exceeds the allowed slack, the monitor sends the coordinator an updated prediction $R_i(t)$. Intuitively, these slacks are used to bound the difference between the coordinator's view of the data and the actual data.

The simple queueing model as in Theorem 1 is ideal since it assumes the true aggregate $\sum r_i(t)$ feeds a single coordi-

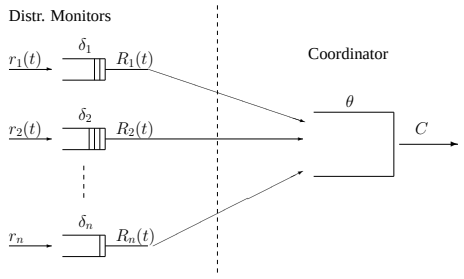


Figure 5: Distributed queueing model: cumulative triggers.

nator queue. We extend the ideal queueing model to the distributed environment by placing queues at each of the monitors in addition to the one queue at the coordinator. This distributed queueing model is depicted in Fig. 5. Our task is then to design algorithms to convert the centralized queue model of size ϵ into a coordinator queue of size θ and a set of local monitor queues of size $\delta_1, \ldots, \delta_n$, while still guaranteeing the necessary false alarm and missed detection rates.

The Local Monitor Protocol. In our distributed model, each local queue has an arrival rate or $r_i(t)$, a drain rate of $R_i(t)$ and a size of δ_i. Let t_i^{prev} denote the time of the last update message from m_i to the coordinator. At any time t, the size of the monitor's queue captures the cumulative deviation of $r_i(t)$ from its most recent prediction $R_i(t^{prev})$ over the interval $[t_i^{prev}, t]$, namely $d_i(t) = \int_{t_i^{prev}}^{t} (r_i(x) - R_i(x))dx$. Should the local queue overflow, i.e., when $|d_i(t)| > \delta_i$, this means the drift has exceeded the allowed slack. At this time the monitor sends the coordinator an update on its time series. It sends the current value $r_i(t)$, a prediction $R_i(t)$ for near-term future values, and the current value $d_i(t)$. The amount $d_i(t)$ corresponds to the cumulative deviation of $r_i(t)$ from its most recent prediction. At the time of the update, the local queue also resets $d_i(t)$ to zero. Note that, unlike traditional queueing, local monitor queue occupancies are allowed to become negative, if predictions consistently overestimate the true local signals. Such conditions are important to detect and bring to the coordinator's attention since they also capture excessive drift and thus lead to more updates. Sending underflow information to the coordinator can also enable cross-site variations to cancel out (thus avoiding false alarms).

We point out that the queues we are using here are models, not actual physical queues. In an implementation a queue that stores data is not needed. Instead only a counter is needed that is incremented and decremented according to the queueing models herein.

The Coordinator Protocol. In our distributed queueing model, the coordinator's queue has an arrival rate of $\sum_{i=1}^{n} R_i(t)$, a drain rate equal to the trigger threshold C, and is of size θ, as in (Fig. 5). In addition to the continuous "arrivals" at rate $\sum_{i=1}^{n} R_i(t)$ to the coordinator queue, each update from monitor m_i also introduces a *chunk* of $d_i(t)$ arrivals into the queue. Note that if the queue underflows (drops below zero), then $d_i(t)$ is negative. The coordinator continuously tracks this complex arrival process at its queue and fires a trigger violation if its queue overflows. A high-level pseudo-code description of the coordinator protocol is depicted in Fig. 6.

Procedure Coordinator(ϵ, β, η)
Input: Trigger error threshold ϵ; miss-detection/false-alarm rates (β, η).
1. **while** (**true**) **do**
2. Continuously simulate a virtual queue Q of size θ with arrival rate $\sum_i R_i(t)$ and drain rate C
3. **for each** (monitor update $(i, d_i^*(t), R_i^*(t))$ received) **do**
4. Set local prediction $R_i(t) := R_i^*(t)$
5. Enqueue the $d_i^*(t)$ chunk in the virtual coordinator queue Q
6. **if** (Q overflows) **then**
fire("trigger violation"); **break**
7. Compute new optimal settings for local slacks $\{\delta_i\}$ and coordinator slack θ based on (ϵ, β, η) and maintained statistics *(Sec. 5.3)*
8. **if** (adaptive allocation) **then disseminate**($\{\delta_i\}$)

Figure 6: Procedures for distributed trigger tracking at the coordinator.

Intuitively, the local slacks δ_i at the remote monitors aim to filter out local variations in individual $r_i(t)$ signals, while the *coordinator slack* θ is useful for canceling out variations *across monitors* (*e.g.*, when distinct $r_i(t)$'s moving in opposite directions). In addition to tracking the global constraint, one of the coordinator's key tasks is to compute values for δ_i ($i = 1, \ldots, n$) and θ that lower communications costs yet guarantee that none of the three errors (ϵ, β, η) exceed their tolerance levels. In order to be adaptive, the coordinator can recompute and redistributed these slacks either periodically or upon each monitor update. In the next section, we give our algorithm for computing these slack values.

5.3 Queueing Analysis for Slack Estimation

We now present an analysis of a simplified variant of our distributed queueing model (Fig. 5), and discuss the application of our results to estimating effective settings for the monitor and coordinator slack parameters in our system. The existence of the local δ_i filters obviously reduces communication costs by allowing monitors to "absorb" updates with no communication to the coordinator. At the same time, however, this local filtering also makes the arrival process at the coordinator queue more *bursty* by introducing bursts of queue arrivals and departures when the filter constraints at local monitors are violated. Thus, abstractly, the role of the coordinator queue (of size θ) is to allow for such bursts to be effectively absorbed (or, cancel each other out) as long as the cumulative trigger bound is not exceeded.

The system slack parameters (δ_i's and θ) interact with each other as well as the input error threshold ϵ, miss-detection rate β, and false-alarm rate η parameters in complex ways. Intuitively, given an error threshold ϵ for our trigger monitor, we

would like to *maximize* the size of the local-monitor filters δ_i, as that would obviously minimize the number of monitor updates to the coordinator. However, larger monitor filters also imply larger (more bursty) chunks of arrivals/departures at the coordinator queue (due to monitor updates) which may, in turn, cause: (1) *false alarms* when a combination of bursts causes the queue to overflow even though the true aggregate signal has not violated the trigger condition; and, (2) *miss detections* when the local monitor filters absorb enough traffic variability to mask a real trigger violation. To minimize the false alarm problem, we would like to have a large coordinator queue size θ to absorb the monitor bursts — however, the size of the coordinator slack θ and monitor slacks $\delta_1, \ldots, \delta_n$ are also clearly constrained by the overall error threshold ϵ that our triggering schemes must try to guarantee.

In what follows, we employ queueing theory to analytically explore the aforementioned tradeoffs (under some simplifying assumptions), and obtain results that provide effective settings for our system slack parameters for a given input triple (ϵ, β, η). Our approach is to develop two non-linear equations relating δ and θ to the parameters (ϵ, β, η) as well as the model parameters. These two equations can then be solved simultaneously to derive δ and θ.

We make two key assumptions to make the analysis tractable. First, we assume uniform local slack parameters, where $\delta_i = \delta$ for all i[3]. Second, we assume an $M/M/1$ queueing model for the coordinator queue[4]. Under the $M/M/1$ assumption, let λ_r and λ_R denote the mean "arrival rates" for the true signal and predicted signal, respectively (*i.e.*, the estimated averages of $\sum_i r_i(t)$ and $\sum_i R_i(t)$ over time). Similarly, let λ_e and λ_d be the mean arrival rates for enqueue and dequeue chunks (respectively) at the coordinator. Note that, the λ_R, λ_e, and λ_d rates are directly observable at the coordinator, and can be computed empirically (*e.g.*, through averaging over a time window of recent queueing activity). Since the overall "mass" of the true aggregate signal is preserved over time, the coordinator can also accurately estimate λ_r as $\lambda_r = \lambda_R + (\lambda_e - \lambda_d) \cdot \delta$. [5]

Now, consider the effect of θ and δ on the miss detection rate β. It is not difficult to see that having $\epsilon \geq \theta + n \cdot \delta$ always guarantees a miss detection rate $\beta = 0$. However, this condition is simply too conservative and may result in excessive communication, especially if (a) some $\beta > 0$ is acceptable, or (b) the true value of the cumulative violation $\max_\tau\{V(T,\tau)\}$ is well below the ϵ threshold. Essentially, fixing a total slack of ϵ is an overly conservative, non-adaptive solution. As proved in [11], the following theorem presents a more versatile, less conservative analytical result relating the miss-detection rate to ϵ, θ, and δ, under the assumption of normally-distributed local "queue" sizes.

Theorem 2 *Assume an $M/M/1$ model for the coordinator queue, and that the aggregate occupancy of all local monitor "queues" follows a Normal $N(0, \sigma^2)$ distribution. Then, setting*

$$\int_{x=0}^{\infty} \left[1 - F\left(\frac{\epsilon - \theta}{\delta} + x + 1\right)\right] \rho^x (1-\rho) dx = \beta \qquad (1)$$

guarantees a miss detection rate $\leq \beta$, where $F()$ denotes the CDF of $N(0, \sigma^2)$, and $\rho = \frac{\lambda_r}{C}$ denotes the average coordinator queue utilization (over time).

The assumption of a zero mean for the aggregate occupancy of all local monitor queues is motivated by the fact that, over a large enough window of time, the true and predicted signal rates are approximately equal (*i.e.*, $\lambda_R \approx \lambda_r$). Similarly, the normality assumption can be justified under the assumption of *independent updates* across local monitors and the law of large numbers (for large enough n)[6]. To estimate the aggregate variance σ^2 in our system, each local monitor m_i continuously tracks the up-to-date variance σ_i^2 of its local occupancy and ships that information to the coordinator in its update messages if there is a significant change with respect to the most recent measurement; the coordinator then estimates the aggregate variance as $\sigma^2 = \sum_{i=1}^{n} \sigma_i^2$. Note that Theorem (2) has the ability to support adaptivity through its dependence on $\rho = \frac{\lambda_r}{C}$. As the rate λ_r evolves, so will ρ, and the resulting value computed for δ.

Now, consider the false alarm rate η. Observe that, in our distributed queueing model, the arrival and drain rates at the coordinator queue can be naturally approximated as $\lambda_R + \lambda_e \cdot \delta$ and $C + \lambda_d \cdot \delta$ (respectively), whereas the corresponding rates for the idealized (centralized) case are simply λ_r and C. Based on this observation and our $M/M/1$ assumption, we can prove the following result (see [11] for details).

Theorem 3 *Assume an $M/M/1$ model for the coordinator queue. Then, setting:*

$$1 - \left(\frac{\lambda_r}{C}\right)^{\frac{\epsilon}{\delta}+1} \Big/ \left(\frac{\lambda_R + \lambda_e \cdot \delta}{C + \lambda_d \cdot \delta}\right)^{\frac{\theta}{\delta}+1} = \eta \qquad (2)$$

guarantees a false alarm rate $\leq \eta$.

Given a triple of trigger-tracking requirements (ϵ, β, η), our coordinator algorithms use the derived system of two non-linear equations (Theorems 2 and 3) to solve for the optimal (under our assumptions) coordinator- and monitor-slack values θ and δ (Step 7 in Fig. 6(b)). The local slacks δ are then distributed to the monitors. This theorem also is a function of the queue input rates, and thus these two equations can be solved again as often as desired; as the time series

[3] In a technical report [11], we evaluate how using non-uniform parameters can provide greater communiction reduction.

[4] In [11], we also provide analyses under other possible queueing models, such as $M/D/1$.

[5] Note that (unlike λ_r and λ_R) λ_e and λ_d here are in units of chunks (of size δ).

[6] Experience with several real data sets shows that a Normal model of aggregate local occupancy is accurate under reasonable time windows.

ϵ	Target β	Achieved β^*	Target η	Achieved η^*
0.2	0.02	0.008	0.02	0.008
0.2	0.02	0.008	0.06	0.030
0.2	0.04	0.000	0.02	0.020
0.2	0.04	0.008	0.04	0.031
0.4	0.02	0.010	0.02	0.010
0.4	0.02	0.000	0.06	0.026
0.4	0.04	0.028	0.02	0.009
0.4	0.04	0.028	0.04	0.036

Table 1: *Target vs. achieved detection performance.*

change, the queue input and drain rates will evolve and thus $\theta(t)$, $\delta(t)$ can be updated over time. Thus supporting changes in the data's underlying statistics is straightforward (see [11] for more details).

6 Evaluation

6.1 Implementation and Data

We have implemented D-Trigger using Java, and deployed the monitor protocol on 40 PlanetLab nodes along with the coordinator protocol on a single PlanetLab node. SNORT [2] sensors have been continuously running on each monitor node for approximately one year. Our Java module extracts information about the number of TCP requests per fixed time window from these logs, and D-Trigger uses this information to detect network overload conditions resulting from bursts of short TCP connections or periods of many, long TCP connections. While the sizes of time windows (and underlying time unit of the time series data) can range from from 5 seconds to 10 minutes, we have elected to use a 5 minute window. We explored the effects of other time windows and in time series with 5 minute windows, we observed 85% to 96% of communication reduction, while with 5 second time windows, we observed 70% to 90% of communication reduction. Thus, we believe the data presented herein are representative of the general gains possible using our methods. Furthermore, this time series demonstrates the need for cumulative triggers because we observed that the size of the time window needed to detect violations varied from 5 to 100 minutes (*i.e.*, no single fixed window size would have caught all events).

6.2 Performance Evaluation Model

Using our implementation, we developed a trace-driven simulator that takes in a time series and can be used for running large-scale experiments under controlled conditions and evaluating D-Trigger's performance. Given a target performance level specified by the triplet parameters (ϵ, β, η), our model uses Theorems 2 and 3, the data variability σ, and the enqueue and dequeue rates λ_R, λ_e and λ_d, to compute the (uniform) monitor and coordinator queue sizes (δ, θ) which are used by the simulator to process the SNORT time series data. The simulator's outputs are the actual observed (*achieved*)

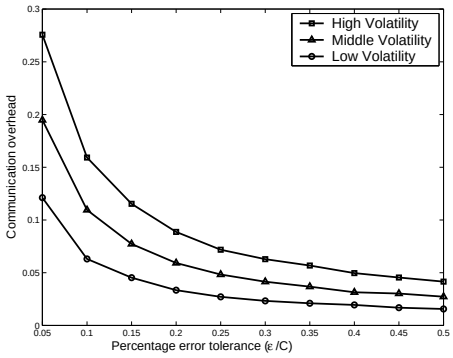


Figure 7: Impact of volatility on overhead.

false alarm and missed detection rates, which are computed as follows. If a trigger is fired, but no corresponding real violation occurred within 3 time intervals (1 interval before, during, and after) of the detected one, then we count it as a false alarm. The achieved false alarm rate (η^*) is the ratio of the number of false alarms over the total number of triggers fired. For each real violation, if no trigger is fired within the 3 time intervals around the real violation, we count this as a missed detection. The achieved missed detection rate (β^*) is the ratio of the number of missed detections over the number of real constraint violations.

We compute the communication overhead (per-node communication cost) as follows. Let num be the number of messages exchanged between monitors and the coordinator, including both the signal updates from monitors to coordinator as well as the filter updates from the coordinator to the monitors. Let n be the number of monitors and m the number of values in each monitor's time series. Thus $m \cdot n$ indicates the worst-case communication overhead (giving the coordinator perfect knowledge), and the communication overhead is $num/(m \cdot n)$.

Table 1 provides several examples of achieved false alarm (η^*) and missed detection (β^*) rates, along with the corresponding target (input) η and β. The table shows that the achieved β^* and η^* are always lower than the target β and η, indicating that our model finds upper bounds on the detection performance, and its derived queue size parameters δ and θ are always safe to use. The results also imply that there are additional optimizations that could reduce the communication cost further.

Clearly the reduction in communication overhead depends on the time series data themselves. We now examine our data's properties to ensure that the our general observations are not artifacts of a particular time series, and use these results to help select the time series and parameters used in our experiments.

The communication bandwidth used between monitors and the coordinator depends upon the data (intuitively, more volatile data uses more bandwidth). To explore the range of communication overhead reductions for different sets of time

series, we selected 40 machines (time series) at a time from the 200 SNORT time series, by first computing the variance of each of the 200 time series and then sorting them.

We selected three different sets of 40 machines: a "high volatility" set of nodes with the 40 largest variances, a "low" set of the 40 nodes with the lowest variances, and a "middle" volatility set of 40 nodes selected at random. The communication overhead reduction versus error tolerance using $\beta = \eta = 0.06$ for these three sets of machines is given in Fig. 7. In all cases, the shapes of the monotonically decreasing curves are similar to each another, and the communication reduction is substantial. A communication overhead of 0.1-0.2 means that only 10-20% of the original time series data is needed to fire triggers with high accuracy. The exact amount depends upon the volatility of the input data, and as expected, the communication overhead decreases as the data's volatility decreases. The fact that the graphs match our expectations indicates that, even with the most volatile set we considered, our protocol and its implementation still achieve efficient communication. For the experiments in this section, we use the middle volatility set.

The target constraint C is data dependent and since triggers are usually designed to detect anomalies, it typically lies near the extreme behavior of the data. We set C to the values of the 85^{th}, 90^{th} and 98^{th} percentile of the distribution of all 4,000 values (time instants) of $\sum r_i(t)$, and observed that the communication overhead as a function of the error tolerance is similar for these three C values. Thus, for the experiments in this section, we set C to the data value at the 90^{th} percentile of the distribution.

6.3 Performance versus Overhead

We examined the tradeoffs between false alarm and missed detection rates, communication overhead, and the queue sizes. Using $\epsilon = 0.2C$, Fig. 8(a) shows communication overhead tradeoffs, (b) and (c) show monitor queue and coordinator queue sizes for each achieved (β^*, η^*) pair. Note that to facilitate viewing of the 3-D plots, the order of increasing β^* and η^* in (a) differs from that in (b) and (c).

Fig. 8(a) shows that communication overhead decreases quickly as β and η increase. The basic phenomenon here is that for any error type (ϵ, β, and η are different error types), the communication overhead can be reduced if we can tolerate higher errors. In this sense, Fig. 8(a) is consistent with Fig. 7. What is surprising is that the range of communication overhead is very limited (4-20%), implying that even when very low false alarm and missed detection rates are desired, we can still achieve efficient communication. For example, when $\beta = \eta = 0.04$, we can filter out 92% of the original signal. We point out that looking across Figs. 7 and 8(a), we see that the communication overhead is typically in the range of 5-20%, even when looking at it from different perspectives (in terms of volatility, percentage error tolerance, constraint definition, and target performance levels). While these numbers are particular to our dataset, we nonetheless therefore believe that our methods can regularly achieve significant data reduction even for low target error rates. Comparing our system to distributed monitors today that do not support distributed cumulative triggers, we see that we achieve difficult monitoring tasks with less than 80% of the monitored data compared to centralized solutions.

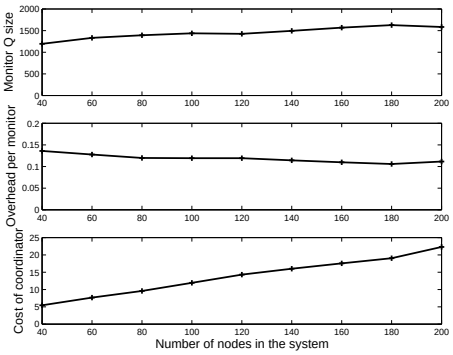


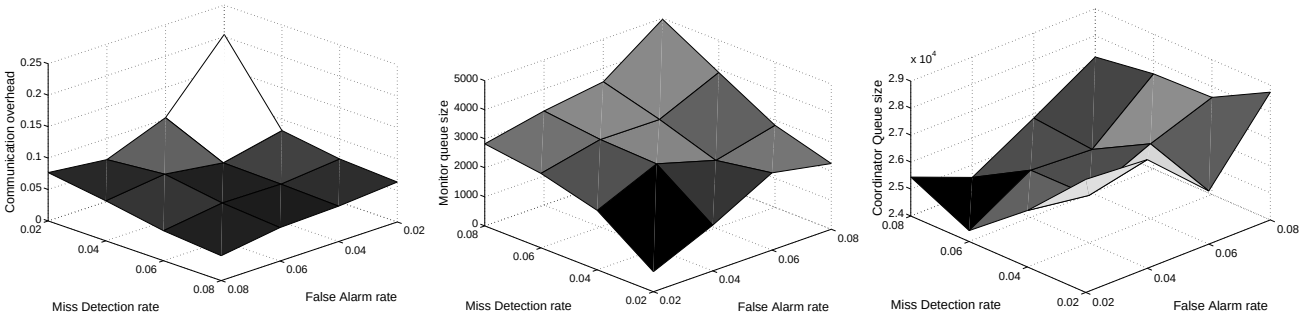
Figure 9: Communication overhead versus system size.

Fig. 8(b) shows that as the tolerable false alarm rate increases, local queues increase in size because more filtering can be done at monitors, which in turn brings down the overhead. This result explains why overhead decreases with increasing false alarm rate and a similar behavior occurs when the tolerable missed detection rate is raised. Looking at both (b) and (c) together, we see that a small change in (β, η) leads to sizable change in local queues, but relatively small amounts of change in the coordinator queue. Because the coordinator does not vary much, even with changes in accuracy requirements, we conclude that cancellation across the signals of different monitors is indeed occurring.

6.4 System Scalability

One key reasons for controlling communications costs is to avoid overwhelming the coordinator, so we examine scalability as the number of distributed monitors grows. Instead of measuring communications overhead (*i.e.*, $num/n \cdot m$), the average overhead *per monitor*, we measure the communications *cost* (*i.e.*, using num/m), the total communications bandwidth into the coordinator, to capture the average number of messages the coordinator receives in a time slot.

In Fig. 9, we plot communications cost as a function of the number of monitors (ranging from 40 to 200) and set $(\epsilon, \beta, \eta) = (0.2C, 0.06, 0.06)$. We ran 5 rounds of experiments for each system size n, each of which ran on n randomly picked monitors. The system scales gracefully, since as the system size increases: 1) the communication overhead of each monitor decreases slightly; and 2) the coordinator's communication cost increases slowly with a slope of roughly 0.1 (indicating that communication cost increases



(a) *Communication overhead* (b) *Monitor queue size* (c) *Coordinator queue size*

Figure 8: Parameters design and tradeoff between false alarm, miss detection and communication overhead.

sub-linearly as system size increases). Intuitively, our algorithm captures the trend that as the number of monitor nodes increases, when one monitor queue overflows, it is more likely there will be an underflowing queue elsewhere, leading to more signal cancellation at the coordinator and less communication cost and overhead.

Note that monitor and coordinator queues grow as the system scales, however, this does not affect scalability because an implementation does not actually use buffers; instead queues are implemented as counters, and queue sizes correspond to maximum counter values.

7 Conclusion and Future Work

We have presented a novel solution to the problem of efficient aggregate constraint detection over a time-varying window (*cumulative triggering*) in a distributed monitoring system. Our solution relies on a key insight of focusing on accurate triggering instead of ϵ-error aggregate value reporting, which can yield a greater than 80% reduction communication overhead, while preserving high detection accuracy.

Our contributions include: providing a mathematical definition of cumulative distributed triggering; using a queueing theory-based problem definition, which makes analytical solutions possible; and performing a detailed evaluation of our schemes using real world and trace-based streaming data. Overall, the combination of our contributions offers users the power to tradeoff desired detection accuracy and performance with communication overhead.

We envision several areas for future exploration, including additing fault-tolerance to the single coordinator, applying cumulative triggers over more sophisticated correlation functions (other than our choice of `SUM`), and using multi-level tree hiearchies to further reduce the processing and communication workload at the coordinator.

References

[1] ArcSight. `http://www.arcsight.com/`.

[2] Snort. `http://www.snort.org/`.

[3] CHERNIACK, M., BALAKRISHNAN, H., BALAZINSKA, M., CARNEY, D., ETINTEMEL, U., XING, Y., AND ZDONIK, S. Scalable distributed stream processing. In *CIDR* (2003).

[4] CLARK, D., PARTRIDGE, C., RAMMING, J. C., AND WROCLAWSKI, J. T. A knowledge plane for the internet. In *ACM SIGCOMM* (2003).

[5] CORMODE, G., AND GAROFALAKIS, M. Sketching streams through the net: Distributed approximate query tracking. In *VLDB* (2005).

[6] DATAR, M., GIONIS, A., INDYK, P., AND MOTWANI, R. Maintaining stream statistics over sliding windows. In *ACM-SIAM SODA* (2002).

[7] DILMAN, M., AND RAZ, D. Efficient reactive monitoring. In *IEEE INFOCOM* (2001).

[8] GAROFALAKIS, M., GEHRKE, J., AND RASTOGI, R. Querying and mining data streams. Tutorial in VLDB (2002).

[9] HUANG, L., GAROFALAKIS, M., HELLERSTEIN, J., JOSEPH, A., AND TAFT, N. Toward sophisticated detection with distributed triggers. In *MineNet* (2006).

[10] HUANG, L., NGUYEN, X. L., GAROFALAKIS, M., HELLERSTEIN, J. M., JORDAN, M., JOSEPH, A.D., AND TAFT, N. Communication-efficient online detection of network-wide anomalies. To appear in *INFOCOM* (2007).

[11] HUANG, L., GAROFALAKIS, M., JOSEPH, A., AND TAFT, N. Communication-efficient tracking of distributed cumulative triggers. UC Berkeley Tech. rep., EECS-2006-139 (2006).

[12] HUEBSCH, R., AND ET AL. Querying the internet with pier. In *VLDB* (2003).

[13] JAIN, A., CHANG, E. Y., AND WANG, Y.-F. Adaptive stream resource management using kalman filters. In *ACM SIGMOD* (2004).

[14] JAIN, A., HELLERSTEIN, J. M., RATNASAMY, S., AND WETHERALL, D. A wakeup call for internet monitoring systems: The case for distributed triggers. In *HotNets* (2004).

[15] KERALAPURA, R., CORMODE, G., AND RAMAMIRTHAM, J. Communication-efficient distributed monitoring of thresholded counts. In *ACM SIGMOD* (2006).

[16] LAKHINA, A., CROVELLA, M., AND DIOT, C. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM* (2004).

[17] OLSTON, C., JIANG, J., AND WIDOM, J. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD* (2003).

[18] PADMANABHAN, V. N., RAMABHADRAN, S., AND PADHYE, J. Netprofiler: Profiling wide-area networks using peer cooperation. In *IPTPS* (2005).

[19] PAXSON, V., AND FLOYD, S. Wide-area traffic: the failure of poisson modeling. *IEEE/ACM Trans. on Networking*, 3(3) (1995).

[20] SPRING, N., WETHERALL, D., AND ANDERSON, T. Scriptroute: A facility for distributed internet measurement. In *USITS* (2003).

[21] XIE, Y., KIM, H.-A., O'HALLARON, D. R., REITER, M. K., AND ZHANG, H. Seurat: A pointillist approach to anomaly detection. In *RAID* (2004).

[22] YEGNESWARAN, V., BARFORD, P., AND JHA, S. Global intrusion detection in the domino overlay system. In *NDSS* (2004).

[23] ZHANG, S., HUANG, L. Personal Communication. November, 2003.

[24] ZHANG, Y., GE, Z.-H., GREENBERG, A., AND ROUGHAN, M. Network anomography. In *IMC* (2005).