Communication-Efficient Tracking of Distributed Cumulative Triggers

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

Abstract

There has been growing interest in large-scale distributed monitoring systems, such as Dynamic Denial of Service attack detectors and sensornet-based environmental monitors. Recent work has posited that these infrastructures lack a critical component, namely a *distributed-triggering mechanism* that fires when an aggregate of remote-site behavior exceeds some threshold. For several scenarios, the trigger conditions of interest are naturally *cumulative*, they continuously monitor the accumulation of threshold infractions (e.g., resource overuse) over time.

In this paper, we develop a novel framework and communication-efficient protocols to support distributed cumulative triggers. In sharp contrast to earlier work focusing on instantaneous violations, we introduce a general model of threshold conditions that enables us to track distributed cumulative violations over time windows of any size. In our system, a central coordinator efficiently tracks aggregate time-series data at remote sites by adaptively informing the sites how to locally filter their data and when to ship new information. Our proposed algorithmic framework allows us to: (1) provide guarantees on the coordinator's triggering accuracy; (2) flexibly tradeoff communication overhead versus accuracy; and, (3) develop an analytic solution for computing local filtering parameters. Our work is the first to solve the problem of communication-efficient monitoring for distributed cumulative trigger conditions using principled solutions with accuracy guarantees. We evaluate our system using time-series data generated from SNORT logs on PlanetLab nodes and demonstrate that our methods yield significant communication overhead reductions while simultaneously achieving high detection accuracy, even for highly variable data streams.

1 Introduction

Distributed monitoring systems aggregate and present information describing the status and performance of large distributed systems (e.g., server clusters and large Internet Service Provider (ISP) and enterprise networks). Remote monitor sites are typically deployed throughout the network (both at the network edge and inside the internal infrastructure) 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 a variety of purposes. Example applications abound. Consider, for instance, a network-wide anomaly detection system. In a typical enterprise network, many Intrusion Detection Systems (IDSs) are deployed across geographically-distributed vantage points to monitor network traffic. These "local" IDS views need to be continuously fused at a central Network Operations Center (NOC) to enable timely detection and warning for abnormal activities. As another example, ISP and enterprise NOCs employ distributed monitoring to continuously track the health of their infrastructure, identify element failures, and then track the performance of their failure recovery procedures; they also monitor load levels for hot spots as a part of capacity planning. Wireless sensornets for habitat, environmental, and health monitoring also continuously monitor and correlate sensor measurements for trend analysis, detecting moving objects, intrusions, or other adverse events.

We can abstract two key aspects of such large-scale monitoring systems. First, monitoring is *continuous*; that is, to ensure timely response to potentially serious problems, we need *real-time tracking* of measurements or events, not merely one-shot responses to sporadic queries. Second, monitoring is inherently *distributed*; that is, local data streams (e.g., IP traffic measurements) observed across several *remote monitor sites* need to be fused and/or correlated at a *coordinator site* to allow tracking of interesting phenomena over a *global* data stream. For instance, consider a collection of compromised hosts within an enterprise network launching a Distributed DoS (DDoS) attack to an outside destination address. 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 to evade detection). On the other hand, a monitoring system tracking the *aggregate* of the compromised host behaviors, can indeed reveal alarming levels of outgoing traffic to the destination. In a similar vein, Lakhina *et al*. [18] propose anomaly-detection methods that track the top eigenvalues of the global traffic matrix by monitoring all the link-load levels in large IP networks. In both scenarios, tracking the aggregate behavior over a physically-distributed monitoring infrastructure is much more revealing than tracking the local behavior of individual network elements or hosts.

Communication-efficient distributed monitoring. The distributed nature of monitor sites also typically implies important *communication constraints* owing to either network overhead restrictions (e.g., large volumes of distributed IP-monitoring traffic) or power limitations (e.g., sensor battery life), For instance, large enterprise networks typically do not

overprovision their interconnections to remote office sites, yielding severe communication restrictions for their enterprise IDS systems, as such systems typically generate enormous amounts of data that is pulled to a central NOC for further analysis by so-called "correlation engines" [1] that look for patterns across the logs of different machines. Such background management traffic coupled with regular inter-office traffic can easily saturate inter-site links. Furthermore, even though ISPs today typically overprovision their backbone networks, emerging continuous monitoring applications may require much finer time and/or data granularities, yielding significant measurement traffic volumes, even by ISP standards. For example, typical SNMP monitors today collect simple link statistics once every five minutes; however, for real-time anomaly detection, finer time scales are often necessary. As our implementation numbers show, simply collecting header information for each new TCP connection over 400 PlanetLab nodes produces a continuous continuous data stream of about 10Mbps at the collection site. And, of course, in any realistic large-scale monitoring setting, there could be tens or hundreds of distinct continuous queries running concurrently over the network infrastructure. The above scenarios clearly illustrate the need for intelligent, *communication-efficient distributed monitoring*, either to limit the burden on the underlying production network or to simply avoid overwhelming the centralized coordinator. Naive solutions that continuously "push" the local data streams directly to a collection site simply will not scale to large distributed systems.

Cumulative triggers. Several recent research proposals suggest architectures for efficient large-scale monitoring systems [3, 4, 14, 24]. Their vision articulates the need for distributed tools that monitor overall system activity. Other recent work [16, 17] argues convincingly that a critical component missing from such architectures is that of a flexible *distributed triggering* mechanism, that can efficiently detect when a global condition across a set of distributed machines exceeds acceptable levels. These early efforts have focused solely on *instantaneous* aggregate trigger conditions, where the goal is to fire the trigger as soon as the aggregate (typically, `SUM`) of the up-to-date local observations (e.g., site CPU utilizations or messages to a given destination) exceeds a pre-specified threshold. While such instantaneous triggers are undoubtedly a useful tool for several application scenarios, they also have some important limitations when it comes to monitoring distributed phenomena that are inherently *bursty*, such as network traffic. Fixing appropriate instantaneous threshold conditions (e.g., for anomaly detection) in such settings can be very difficult, and easily lead to numerous false positives/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 or malicious if they are allowed to *persist over time*. For instance, in our DDoS example, a clever attacker could try to "fly under the radar" by ensuring that the instantaneous aggregate volume of traffic to the victim is not large enough to raise any alarm signals; capturing the persistence of the aggregate traffic over time is key to detecting the attack. Another example where temporally-persistent violations can play an important role is that of *"burstable billing"* policies employed by ISPs for large enterprise network customers with multiple connections to the ISP's network. Typically, these customers are allowed to use up to a certain amount of bandwidth across all the links per month for a fixed fee, with additional charges if the allotted bandwidth is exceeded. Given the transient bursty nature of traffic, charging customers literally for each excess byte over their bandwidth allotment is too restrictive; instead, a much more flexible and intuitive billing policy is to assess extra charges only for bandwidth overuse that persists over time or exceeds the contracted allotment by a truly excessive amount.

The above scenarios clearly argue for a novel class of *cumulative* triggers, where 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 introduce a new class of distributed monitoring problems that cannot be captured using existing `SUM`-trigger mechanisms based on instantaneous sums of local values [16, 17]. In a nutshell, the accumulation of signal area can take a place over a time window of arbitrary size (not known a priori), whose boundary is defined based on the whole history of the aggregate signal (e.g., with periods of underutilization compensating for periods of overuse). This cumulative threshold condition cannot be expressed in terms of an instantaneous problem.

Our Contributions. In this paper, we introduce and formalize the concept of *distributed cumulative triggers*, and propose a novel algorithmic framework for the communication-efficient tracking of such global triggering conditions in a networked environment. Our proposed solution is built by combining in-network processing ideas [5, 8] with new insights based on *queueing theory*. Briefly, the monitors and coordinator are each assigned an amount of *slack* that carefully controls the discrepancy in the views of the data available at the coordinator and the remote monitors. One of our key insights is that this slack can be viewed as analogous to *queue sizes*. By sizing a set of distributed queues correctly, we can use them to determine when monitors should update the coordinator, and when the coordinator should fire a cumulative trigger. These queue sizes affect the resulting amount of communication overhead as well as the resulting false-alarm and missed-detection (i.e., false negative) rates. We develop an analytical solution for determining the queue sizes based on user-supplied target error rates for false alarms and missed detections. In this manner, the user can effectively control the tradeoff between communication overhead and alarm detection accuracy. To the best of our knowledge, our

work is the first to address the problem of communication-efficient tracking for distributed cumulative trigger conditions. We believe that, through the introduction of cumulative triggers, and the incorporation of new analytical and algorithmic insights from queueing theory for guaranteed-accuracy monitoring, our approach significantly broadens the scope of earlier distributed triggering and query-tracking proposals.

A thorough experimental evaluation over real-life distributed data streams collected from PlanetLab IDS monitors demonstrates 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%).

Prior Work. Database research on continuous distributed query processing has considered similar environments [2, 5, 8, 15, 19]; however, the focus is on the accurate estimation of the aggregate signal itself rather than catching a constraint violation. The database community has also explored centralized triggering mechanisms [11, 25]; however, the goal of minimizing communication overhead in widely distributed environments introduces new challenges. Jain *et al*. [16], 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 [9] 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*. [17], 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. Recent progress in distributed monitoring, profiling and intrusion detection [18, 20, 26, 27] aims to share information and foster collaboration between widely distributed monitoring boxes to offer improvements over isolated systems. These systems provide other examples of distributed monitoring systems for which our triggering tools would be useful.

Organization. The rest of the paper is organized as follows: we define the problem and evaluation metrics in Sec. 2; we discuss our approach in Sec. 3; we present solutions for varying-window triggers in Secs. 4; we evaluate the approach in Sec. 5; we discuss deployment issues and triggering extensions in Sec. 6; finally, we conclude in Sec. 7.

2 System Model and Problem Statement

As shown in Fig. 1, the distributed triggering system consists of a set of widely distributed monitoring nodes $m_1, m_2, \ldots, m_n$ and a coordinator node X. The concept of monitor in our setting is very general. It can be a monitoring sensor,

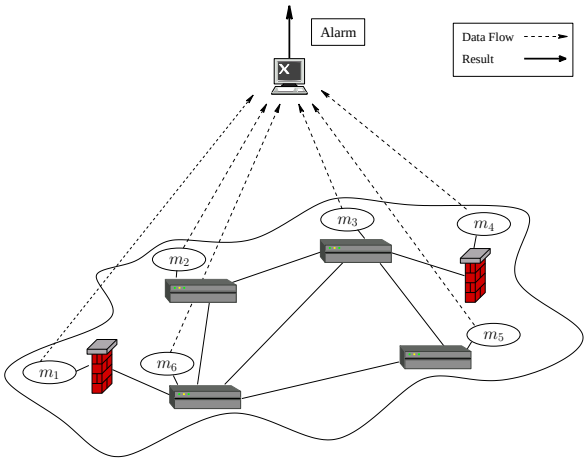


Figure 1: The system setup.

functional software on an end host or firewall, or a software module embedded in a router. Each monitor continuously produces time series signals $r_i(t)$ on the variable(s) or condition(s) selected for monitoring. A monitor's output can be very general, for example, it can be any subset, or any combination of: 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 which acts as an aggregation and detection point. The purpose of the coordinator is to track conditions across its monitors and to fire a trigger whenever some limitation on the aggregate behavior of a subset of nodes is violated. In general, such a coordinator can aggregate and correlate the incoming time series signals using any typical aggregation function (such as `SUM, AVG, MIN, MAX`, etc.) or more complex correlation functions (such as the top eigenvalues of the global measurements matrix [18]). We focus primarily on simple linear aggregators, using `SUM` as our main example. We should stress, however, that our system is general and can be extended to accommodate domain-specific knowledge; furthermore, simple `SUM` aggregates can actually enable more sophisticated distributed detection tools, as discussed in [13].

All communication happens only between monitoring nodes and the coordinator, and no communication happens among monitoring nodes. If all the monitors sent their time series signals continuously (and there were no delay and no loss in the network), then the coordinator would have perfect knowledge of the signals (*i.e.*, global state) and would fire the trigger accurately. By "accurately" we mean that the coordinator can make two kinds of mistakes when it has imperfect knowledge: either a violation among monitors occurs and the coordinator fails to catch it (we call this a *missed detection*), or no violation occurs yet the coordinator thinks that one has (called a *false alarm*). Clearly, continuously sending all the monitored signals is extremely costly in terms of communication overhead and can overwhelm the coordinator.

Our intent here is to enable the coordinator to fire its trig-

gers with high accuracy while using as little communication as possible. We make use of three avenues for reducing overhead: (1) when the time series itself does not change "much," no updates are sent to the coordinator since the most recent information sent is still valid; (2) we focus on the accuracy of firing the trigger and not on estimating the aggregate time series signal; and, (3) we leverage the coordinator's global view by letting it inform each monitor the level of accuracy that it must report. To simplify the exposition, our discussion assumes that communication with 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 [19].

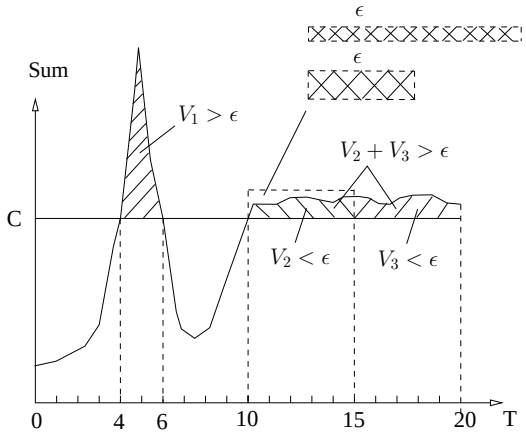


Figure 2: Cumulative violations.

2.1 Types of Threshold Conditions

Let C denote the distributed trigger threshold. Our goal is to track the trigger condition *approximately* to within a specified *error tolerance* ϵ, and our tracking algorithms exploit this error tolerance to minimize communication costs. Since we are dealing with continuous time series of measurements, the notion of exceeding a threshold is intimately related to the length of time over which a violation may occur. Our focus in this paper is on a new class of persistent, *cumulative* threshold violations.

Cumulative Threshold Condition. The basic idea is that, to capture temporally-persistent phenomena, a violation can be defined in terms of the accumulation of excess *area* of the underlying signal over windows of time. The coordinator can achieve this by computing a violation penalty that accrues over time, and fires the trigger condition when the penalty becomes excessive. During a window with (time-varying) 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 size 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 $\max_\tau\{V(t, \tau)\} > \epsilon$. One of our key insights in this work is that, by exploiting an analogy to queuing theory, our system can track varying-window trigger conditions effectively, without having to retain the entire signal history or check the condition against all possible τ.

Other Threshold Conditions. Earlier work on distributed triggers [9, 17] has focused solely on *instantaneous* threshold conditions, where the goal is to detect if $\sum_i^n r_i(t)$ exceeds a threshold C by more than a given error tolerance at any time instant t. An easy 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. (Since τ is fixed, such triggering conditions can be easily reduced to the instantaneous case, at least for simple aggregates like `SUM`.)

While undoubtedly useful in several settings, 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 over-react to natural, transient phenomena which are very common in practice. 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) may never 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, $[6, 7]$). However, in a time-window of size 2, the penalty V_1 does exceed ϵ. In several application scenarios, it is important to detect both types of violations regardless of the specific time window in which they occur. Fig. 2 also illustrates the key difference between fixed-window and cumulative violation. Consider, for instance, a fixed window size $\tau = 5$. When the violation (on average) is small, it will not trigger alarms in time periods $[10, 15]$ or $[15, 20]$ with total excess violations $V_1, V_2 < \epsilon$. On the other hand, since the violation persists over time (across the $[10, 20]$ window), assuming $V_1 + V_2 > \epsilon$, a cumulative trigger with the same threshold would readily detect the problem. Thus, by not fixing a window size beforehand, our

cumulative triggering mechanism can capture a wide variety of persistent violation scenarios while avoiding the pitfalls of fixed time granularities.

2.2 Problem Statement

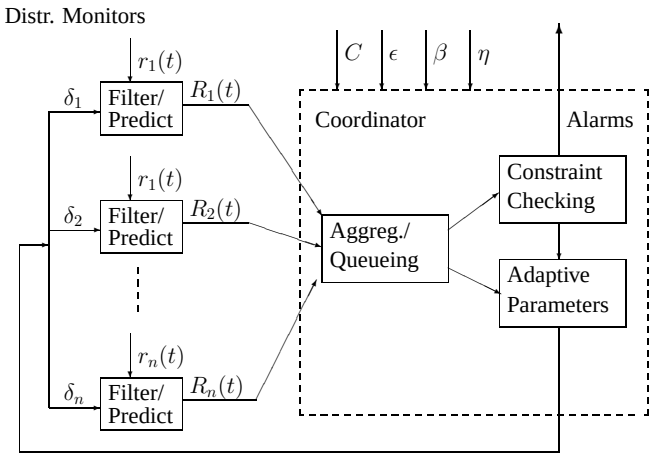
Based on our earlier definitions, we say a *missed detection* occurs if $\max_\tau\{V(t,\tau)\} > \epsilon$ and the system does *not* fire the corresponding trigger. Conversely, a *false alarm* occurs whenever $\max_\tau\{V(t,\tau)\} \leq \epsilon$ and the system fires a trigger. We define the *missed-detection 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. Both β and η are system inputs that can be tuned to achieve a target false-alarm and missed-detection rate. Allowing these parameters 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 problem we address herein is to design the protocols resident at the monitors and at the coordinator in order to guarantee that the distributed trigger check at the coordinator is accurately fired as the local monitor signals evolve over time. A user can specify the desired error tolerance ϵ, as well as the target missed-detection rate β and false-alarm rate η as inputs to our system — the triple (ϵ, β, η) essentially denotes the accuracy level that our tracking schemes target. Thus, the goal is to guarantee the trigger fires with (ϵ, β, η)-accuracy while simultaneously keeping communication overheads low. We measure the communication overhead for our techniques as a fraction of the original time series (*i.e.*, complete signal data) sent to the coordinator; thus, a 10% overhead indicates that the data transferred between monitors and coordinator is only $\frac{1}{10}$th of all the measurement data observed at the monitors. We can now define the cumulative triggering problem that we address in the remainder of this paper.

•**Cumulative Trigger Tracking:** Design monitor and coordinator protocols that trigger an alarm if the (global) condition $\max_\tau\{V(t,\tau)\} > \epsilon$ holds *at any time* t, with accuracy (ϵ, β, η), while imposing minimal communication overhead on the network.

3 Our Approach: An Overview

This section discusses several key elements of our novel distributed trigger tracking approach. Fig. 3 depicts the components of our system, where $r_i(t)$ denotes the actual time series observed at monitoring node i, and $R_i(t)$ denotes the approximate representation of $r_i(t)$ that is available at the coordinator. In general, $R_i(t)$ can be based on any type of *prediction model* for site m_i that tries to predict the site's behavior over time (*e.g.*, based on the recent past of $r_i(t)$). A



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 3: Our distributed trigger tracking framework.

simple model might set $R_i(t)$ to the latest $r_i(t)$ value communicated from the site, or an average of recent communication, but more sophisticated prediction models [5, 6] can be used. Our techniques remain applicable regardless of prediction model specifics.

The key idea is that, at any time t, $\sum_{i=1}^{n} R_i(t)$ captures the the coordinator's view of the global state, while each monitor node m_i uses its prediction to filter updates to the coordinator by continuously tracking the deviation of its "true" state $r_i(t)$ from the corresponding prediction $R_i(t)$. This filtering is based on *local monitor slack* parameters $\delta_i > 0$ that, intuitively, upper bound the amount of drift between the coordinator's view of site i's data stream and the actual $r_i(t)$ signal. As long as the prediction accurately captures the local stream behavior (*i.e.*, within δ_i bounds), no communication is needed. Meanwhile, the coordinator continuously monitors its up-to-date global prediction to ensure that its cumulative penalty across any possible time window stays below the required trigger threshold and triggers when that condition is violated.

A Queuing Perspective on Cumulative Distributed Triggers. Our cumulative trigger conditions pose novel algorithmic problems that have not been addressed in earlier work on data streaming. *Window-based* stream processing [7, 10] typically focuses only on the case of (time- or arrival-based) windows *of fixed size* over the stream; such techniques are clearly not useful in our case, since the window sizes of the (potential) trigger violation are not known a priori. Instead, our key observation is that we can accurately model the monitoring of a cumulative trigger condition (see Sec. 2) using a

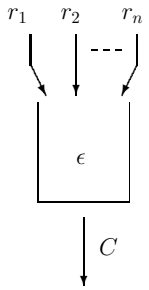


Figure 4: Cumulative violation and queue overflow.

simple *queuing model* (see Fig. 4), as stated by the following theorem.

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. 5 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. 5 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 queuing 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.

Our algorithms and analyzes for efficiently tracking cumulative distributed triggers depend crucially on the above equivalence. While the model in Fig. 4 illustrates the conceptual equivalence between a cumulative trigger violation and an overflowing queue, we point out that this is an idealized centralized model. It is idealized because it assumes the complete accurate signals for $r_i(t)$ can feed the queue, and that there is a single queue. In our distributed environment, adjustments are necessary. We extend the queueing idea to the

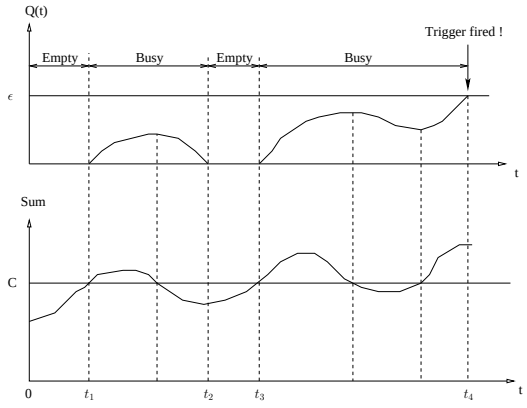


Figure 5: Queuing model for a cumulative trigger.

distributed environment by placing queues at all the monitors and at the coordinator. Our task is then to design algorithms to convert the centralized queue with size ϵ into a coordinator queue with size θ and a set of local monitor queues with size $\delta_1, \ldots, \delta_n$, while still guaranteeing the necessary false alarm and missed detection rate.

Adaptive Threshold-based Slack Allocation. Besides maintaining an up-to-date estimate of the global state $\sum_{i=1}^{n} R_i(t)$, the job of the coordinator entails two key steps: (1) simulating a queuing process to check constraint violations, and (2) adaptively determining slack θ (coordinator queue size) for itself and individual local slacks δ_i (monitor queue size) for each monitor. Local slacks can vary over time and are *adaptively* recomputed to maximize the effects of local filtering, and thus, to minimize overall communication. Our adaptive slack allocation schemes exploit the trigger condition to allow for much "looser" (and thus, more effective) filters at monitors when the signal stays well *below* or *above* the C threshold. This observation is one of the key motivations for building adaptivity into our distributed trigger monitoring system.

4 Distributed Cumulative Triggers

The simple queuing model discussed in Sec. 3 is ideal since it relies on observing the true aggregate $\sum r_i(t)$. However, in our distributed environment, the global coordinator only observes approximate predictions $R_i(t)$ of the local signals (sent in by the monitors). We extend our queueing anomaly to the distributed environment by placing queues at the coordinator (to catch the violations through an overflow) along with queues at each of the monitors (to perform filtering). This distributed queuing model is depicted in Fig. 7.

Extending the queuing analogy to individual monitors, our model captures the effects of local prediction-based filtering at m_i through a monitor queue of size δ_i (the local slack) with an arrival rate of $r_i(t)$ (the actual local signal) and a drain rate of $R_i(t)$ (the local prediction last sent to the co-

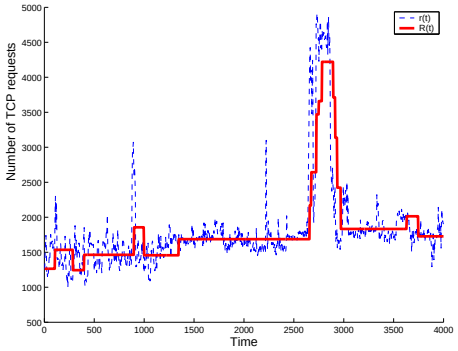


Figure 6: Local prediction-based filtering.

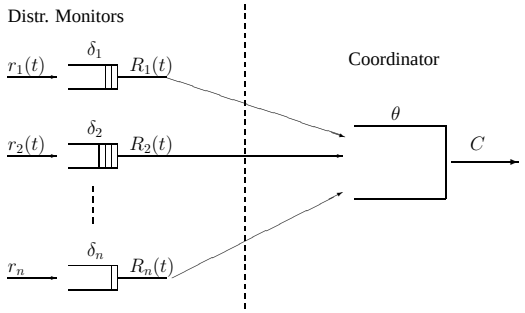


Figure 7: Distributed queuing model: cumulative triggers.

ordinator)[1]. Monitor sites simply track their local queue occupancy and notify the coordinator (also attaching a more recent prediction) when their local occupancy exceeds their δ_i bounds. As an example, Fig. 6 shows the (true) $r_i(t)$ and (smoothed) $R_i(t)$ curves for a real data set (number of TCP requests in 5-minute intervals over a two-week period on a PlanetLab node), using a static prediction model (*i.e.*, the prediction used was exactly the last value at the local monitor), and a queue size of 5,000. Periods where $R_i(t)$ remains constant imply that $r_i(t)$ stays consistently within bounds (*i.e.*, no communication).

On the other side, the coordinator simulates a queue of size θ with an arrival rate equal to the (up-to-date) aggregate prediction $\sum_{i=1}^{n} R_i(t)$ and a drain rate equal to the trigger threshold C; as in Sec. 3, the coordinator fires a trigger violation if its queue overflows. Intuitively, while the local slacks δ_i at the remote monitors aim to filter out local variations in individual $r_i(t)$ signals, the *coordinator slack* parameter θ is important for effectively canceling out variations *across monitors* (*e.g.*, think of distinct $r_i(t)$'s moving in opposite directions). Of course, one of the coordinator's key tasks is, given the desired trigger threshold ϵ, miss-detection rate β, and false-alarm rate η parameters, to determine the local monitor slacks δ_i ($i = 1, \ldots, n$) and coordinator slack θ that optimize the overall communication costs while ensuring the required trigger-detection guarantees. Our coordinator algorithms compute these slack parameters continuously (based on available updates from the monitors) to ensure that our system adapts to changing local monitor characteristics over time.

In the remainder of this section, we present the details of our cumulative trigger tracking protocols, as well as the queuing analysis that drives our parameter settings and the resulting approximate triggering guarantees; finally, we discuss how our protocols can naturally adapt to varying monitor characteristics.

[1]Note that all queues are primarily used as conceptual tools in our development to guide our understanding and analysis; our implementation just simulates these queues through simple counters.

4.1 Our Trigger-Tracking Protocols

The Local Monitor Protocol. Given a local slack parameter δ_i (determined by the coordinator), the trigger-tracking protocol run at each monitor site m_i is fairly straightforward. Let t_i^{prev} denote the time of the last update message from m_i to the coordinator, and let $R_i(t)$ be the most recent prediction model for $r_i(t)$ sent to the coordinator. At any time t, monitor m_i continuously tracks the cumulative deviation of $r_i(t)$ from its prediction $R_i(t)$ over the interval $[t_i^{prev}, t]$ as $d_i(t) = \int_{t_i^{prev}}^{t} (r_i(x) - R_i(x))dx$, and checks the condition $|d_i(t)| \leq \delta_i$ (*i.e.*, the monitor ensures that the absolute cumulative difference between its actual stream and the coordinator's corresponding prediction is upper bounded by its local slack δ_i. Whenever $|d_i(t)| > \delta_i$, the monitor sends an update message to the coordinator that includes $d_i(t)$ and an up-to-date prediction $R_i(t)$, and resets $d_i(t)$ to zero[2].

The Coordinator Protocol. Driven by the up-to-date predictions and deviations communicated from local monitors, the coordinator continuously simulates a queue of size equal to the coordinator slack θ and arrival/drain rates equal to the total prediction $\sum_{i=1}^{n} R_i(t)$ and the trigger threshold C, respectively (Fig. 7). 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 or departures from the queue (depending on whether $d_i(t)$ is positive or negative), where $d_i(t)$ is the observed cumulative deviation of $r_i(t)$ from its prediction at m_i. The coordinator continuously tracks this complex arrival process at its local queue and fires the trigger condition whenever the queue overflows. Finally, either periodically or on each monitor update, the coordinator can recompute the monitor and coordinator slack parameters in order to adaptively optimize communication costs for changing monitor characteristics. A high-level pseudo-code description of both the local-monitor and coordinator protocols is depicted in Fig. 8.

[2]Note that, unlike traditional queuing, local monitor "queue" occupancies are allowed to become negative, if predictions consistently underestimate the true local signals. Such conditions are important to detect and bring to the coordinator's attention since they can result in more up-to-date predictions and enable canceling out of cross-site variations.

Procedure Monitor(i, δ_i)
Input: Monitor index i, local slack parameter δ_i.
1. **while** (**true**) **do**
2. t := current time; t_i^{prev} := time of last update to coordinator
3. $d_i(t) := \int_{t_i^{prev}}^{t} (r_i(x) - R_i(x))dx$
4. **if** ($|d_i(t)| > \delta_i$) **then**
5. Send update message $(i, d_i(t), R_i(t))$ to coordinator
6. Set $d_i(t) := 0$
7. **if** (new slack δ_i^* is received from coordinator) **then**
8. Set $\delta_i := \delta_i^*$

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. 4.2)*
8. **if** (adaptive allocation) **then disseminate**($\{\delta_i\}$)

Figure 8: Procedures for (a) local monitor update processing, and (b) distributed trigger tracking at the coordinator.

4.2 Queuing Analysis for Slack Estimation

In this section, we present an analysis of a simplified variant of our distributed queuing model (Fig. 7), and discuss the application of our results to estimating effective settings for the monitor and coordinator slack parameters in our system. Compared with the idealized (centralized) queuing model of Fig. 4 and Sec. 3, our goal here is to break the idealized central queue of size ϵ into a coordinator queue (of size θ) and a set of local monitor queues (of sizes $\delta_1, \ldots, \delta_n$). 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 filters absorb enough local update traffic 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 queuing 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). We make two key assumptions to make the analysis tractable. First, we assume uniform local slack parameters, where $\delta_i = \delta$ for all i (in Sec. 4.3 we briefly discuss non-uniform parameters). Second, we assume an $M/M/1$ queuing model for the coordinator queue.[3]. 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 queuing activity). Since the overall "mass" of the true aggregate signal is preserved over time, the coordinator can also accurately estimate λ_r as $\lambda_r = \lambda_R + (\lambda_e - \lambda_d) \cdot \delta$. [4]

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. 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[5].

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).

[3] In the Appendix, we also provide analyses under other possible queuing models, such as $M/D/1$.

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

[5] Proofs of theorems can be found in the Appendix.

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* at 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 $\sigma^2 = \sum_{i=1}^{n} \sigma_i^2$. It is also important to note that the condition in Theorem (1) naturally adapts to the current true state of the signal and its distance from the trigger threshold C through its direct dependence on the $\rho = \frac{\lambda_r}{C}$ ratio.

Now, consider the false alarm rate η. Observe that, in our distributed queuing 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.

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 $(\epsilon,\ \beta,\ \eta)$, our coordinator algorithms use the derived system of two non-linear equations (Theorems 1 and 2) to solve for the optimal (under our assumptions) coordinator- and monitor-slack values θ and δ (Step 7 in Fig. 8(b)). The local slacks δ are then distributed to the monitors for tracking their local prediction deviations.

4.3 Adaptive Slack Allocation

Clearly, at any time instant t, Theorems (1) and (2) can be used at the coordinator (with the up-to-date estimates of queue arrival rates and variances) to provide optimal settings $\theta(t)$, $\delta(t)$ for our system slack parameters. Thus, our system can naturally adapt to changing monitor characteristics. It is important to note, however, that the aggregate statistics employed in our queuing analysis are likely to be quite stable (*i.e.*, vary slowly over time). This implies that, in practice, frequent re-calibration of the θ and δ parameters is not necessary, and, in fact, could cause system instability and excessive communication. In this section, we discuss two simple schemes for filtering δ updates at the coordinator; while such schemes may no longer offer the performance guarantees of Theorems 2 and 3, they also limit the sensitivity of the system to transient variations and the number of required δ-slack disseminations from the coordinator to the monitors. We also briefly discuss non-uniform allocation of local monitor slacks.

Min-based $\delta(t)$ Filtering. Consider a scenario where the coordinator estimates a new $\delta(t)$ value that is greater than the previously disseminated local slack. In this situation, the coordinator may choose *not* to disseminate the new slack value to save $O(n)$ messages, at the cost of more conservative filtering (and, thus, maybe more messages) at the local monitors. This choice is correct for a transient change in δ due to local stream variability; in addition, it can only *reduce* the actual miss detection and false alarm rates (albeit at the cost of extra communication). Our *min-based filtering scheme* is based on this intuition: it applies a low-pass filter on the current $\delta(t)$ value based on a window of h time instants, computing the local slack as $\bar{\delta}(t) = \min\{\delta(t-h), \ldots, \delta(t)\}$ and disseminating new slack values based solely on changes in $\bar{\delta}(t)$. Note that the `min` function is just one way of trying to capture the long-term trend for δ and other aggregates (*e.g.*, `AVG`) can also be applied. Unlike `min`, however, these are not "safe" and may cause more miss detections and/or false alarms.

Discretization-based $\delta(t)$ Filtering. We can apply the intuition that we really only need to update the local slack value δ when we see a significant change in its value. As such, we can quantize the range of slack values into intervals I_0, I_1, ..., and update the monitor slacks only when $\delta(t)$ moves across interval boundaries. Since the available slack is usually on the order of ϵ, we can use intervals of size $\frac{\epsilon}{b}$, where b is a quantization parameter (thus, $I_k = ((k-1)\frac{\epsilon}{b}, k\frac{\epsilon}{b}]$). Large b values yield tighter intervals and more accurate local δ settings, but also imply more sensitivity to transient changes and increased communication, thus giving rise to interesting accuracy and cost tradeoffs.

Non-Uniform Local-Slack Allocation. To achieve load balance and further reduce communication overhead, instead of using identical $\delta(t)$ for all monitors, the coordinator can compute and distribute non-uniformly the local monitor slacks $\delta_1(t), \ldots, \delta_n(t)$. For instance, the coordinator can distribute slacks in proportion to locally observed variances (*i.e.*, $\delta_i(t) = \frac{\sigma_i \cdot n\delta}{\sigma}$) providing more "cushion" to sites with higher variability.

5 Evaluation

In this section, we use our protocol implementation and protocol simulator with a real wide-area network activity dataset, to evaluate our methods for distributed cumulative triggers.

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

5.1 Implementation and Data

We implemented our triggering system using Java, and deployed the monitor protocol on 40 PlanetLab nodes along with the coordinator protocol on a single PlanetLab host. SNORT sensors were activated on each monitor node and have been continuously running for approximately one year. In the deployment, our Java module extracts information from these logs in periodic epochs, the size of which can ranges from 5 seconds to 10 minutes. These epochs determine the underlying time unit of the resulting time series data. For the examples presented herein, we use the time series of the number of TCP requests per 5 minutes time window. We have checked our results, especially the reduction on communication overhead, against results for other time granularities. Clearly, time series with different underlying time scales will exhibit different amounts of volatility, which in turn affect the communication overhead. We observe 85% to 96% of communication reduction when using time series with 5 minute time bins, while in time series with 5 second time windows, we observed 70% to 90% of communication reduction. We thus believe the data presented herein are representative of the general gains possible using our methods.

In addition to our implementation, that confirms proper functioning of our code, we have also developed a trace-driven simulator. The simulator emulates our protocols and is fed with the SNORT time series as input. This simulator serves many purposes. First, because our Java code was deployed only recently, the simulator allows us to evaluate our methods on the time series data produced from the SNORT sensors throughout this last year. Also our code is currently deployed on 40 machines whereas the SNORT sensors are deployed on 200 machines. Using the simulator with all 200 SNORT time series allows us to do some scalability assessment. Third, the simulator is also useful for rerunning experiments which is very important for evaluating our protocols under a wide variety of settings (*e.g.*, target accuracy levels).

In addition to this PlanetLab SNORT dataset, we also conducted some evaluations on two other datasets. One monitors per-connection packet rates from sources spread over a wide-area network [21], another is a dataset of temperatures from an environmental sensor network. Among these three datasets, the time series extracted from the SNORT logs (number of TCP requests/5 min) were the most volatile and thus the most difficult to handle. Due to lack of space, we have elected to present the results from the most challenging data set, namely SNORT data. The results for the other datasets illustrate the same properties as those presented here, but achieve even greater communication overhead reduction due to smoother time series data. In most of the plots below, we used a time series of length 2 weeks (corresponding to 4000 data points per time series per node).

5.2 Performance Metrics

We examine the performance of our protocol for cumulative violations. We start with experiments in which the monitors are given uniform slacks $\delta_i = \delta$, and evaluate the effect of non-uniform slack allocation (i.e., heterogeneous queue sizes) later on.

In our evaluations, the target performance level is specified by the usual triplet parameters (ϵ, β, η). We use these values with our models to compute the monitor and coordinator queue sizes (δ, θ) for the simulator, which we then drive with the PlanetLab TCP request time series data as input. The simulator's outputs are the false alarm and missed detection rates actually achieved by our system. The false alarm rate *achieved* by our system when run on real data is 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, denoted by η^*, is then given by 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 missed detection rate, denoted by β^*, achieved by our system is given by the ratio of the number of missed detections over the number of real constraint violations.

For each experiment, we compute the communication overhead 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). Then communication overhead is calculated as $num/(m \cdot n)$ which gives the per-node communication cost.

Thus, one single experiment consists of the following. We feed an input triple (ϵ, β, η) and the PlanetLab data into our model, and compute the monitor and coordinator queue sizes (δ, θ) using Theorems 1 and 3. Recall that the computation uses the data variability σ, along with the enqueue and dequeue rates λ_R, λ_e and λ_d. We ran our simulator using each pair of selected queue sizes, with the actual SNORT traces as input, for 40 nodes, each of which has 4,000 values in the time series. This produces a single result for our three performance metrics $(overhead, \beta^*, \eta^*)$. We used hundreds of triples of (ϵ, β, η) to generate all the points in the graphs below.

5.3 Model Validation

In Table 1, we give a few examples of the actual false alarm rate (η^*) and missed detection rate (β^*) that occurred in the system, along with the corresponding target η and β that was given as input. We can see that the achieved β^* and η^* are always lower than the target β and η. These results indicate

	Desired		Achieved	
ϵ	β	η	β^*	η^*
0.2	0.02	0.02	0.008	0.008
0.2	0.02	0.04	0.000	0.023
0.2	0.02	0.06	0.008	0.030
0.2	0.04	0.02	0.000	0.020
0.2	0.04	0.04	0.008	0.031
0.2	0.04	0.06	0.000	0.023
0.4	0.02	0.02	0.010	0.010
0.4	0.02	0.04	0.000	0.018
0.4	0.02	0.06	0.000	0.026
0.4	0.04	0.02	0.028	0.009
0.4	0.04	0.04	0.028	0.036
0.4	0.04	0.06	0.010	0.035

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

that our model finds upper bounds on the detection performance, and that it is always safe to use our model's derived queue size parameters δ and θ; although it also implies that there is future work to do in identifying further optimizations that reduce the communication cost.

In our experiments, we observed that the size of the time-window needed to catch each of the violations varied from 5 to 100 minutes. There is no good single value of a fixed window size that would have caught all of these events. This broad range illustrates that indeed the notion of a time-varying window for violation detection is needed, and this provides motivation for the idea of cumulative triggers.

5.4 Experiment Configuration

Clearly the reduction in communication overhead is a function of the time series themselves, and smoother data streams will yield larger overhead reductions. We now examine two properties of our data to be sure that the general observations we make are not artifacts of a particular time series. We also use these next two plots to help us select the experiments to run for the remainder of the evaluation.

Our target constraint C is data dependent. The value of C would typically lie near the extreme behavior of the data since the triggers are usually designed to detect unhealthy behavior. In the following experiment we select C to be the value of the 85^{th} percentile of the distribution of all 4,000 values (time instants) of $\sum r_i(t)$. Similarly, we try the 90^{th} and 98^{th} percentiles as different values for the threshold C. In Fig. 9 we plot the communication overhead as a function of the error tolerance for each of these four values of C. In all cases, the shapes of the monotonically decreasing curves are very similar to each another. For any particular value of ϵ, the communication reduction is substantial. A communication overhead in the range of 10-20% means that we only need 80-90% of the original time series data to fire the triggers with high accuracy (the exact amount depends upon the target accuracy level). We elect to use C corresponding to the data value at the 90^{th} percentile of the distribution for the remainder of our experiments.

The amount of communication bandwidth used between our monitors and the coordinator will depend upon the data, and it is intuitive — more volatile data will use more bandwidth. In order to see the range of gain we achieve on communication overhead reduction for different sets of time series, we did the following. Of the 200 PlanetLab SNORT logs, we selected 40 machines (time series) at a time. We did this by first computing the variance of each of the 200 time series and then sorting them. We selected three different sets of 40 machines each: the "high volatility" set are the nodes with the 40 largest variances, the "low" set used the 40 machines with the lowest variances, while the "middle" volatility set selected 40 nodes at random. The communication overhead reduction versus error tolerance for these three sets of machines is given in Fig. 10. For all experiments (one for each dot), we used $\beta = \eta = 0.06$. As expected, for a given value of ϵ, the communication overhead decreases as the volatility of the data decreases. The fact that this graph matches our expectations can be taken to indicate that our protocol and its implementation are doing what they are suppose to do. We see that even with the most volatile set, we still achieve efficient communication. In the remainder of our experiments, we use the middle volatility set.

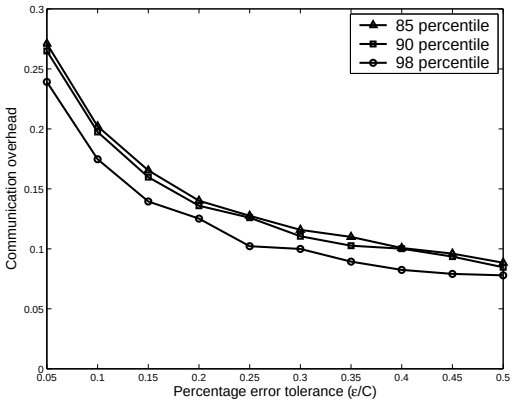


Figure 9: Impact of constraint violation threshold C.

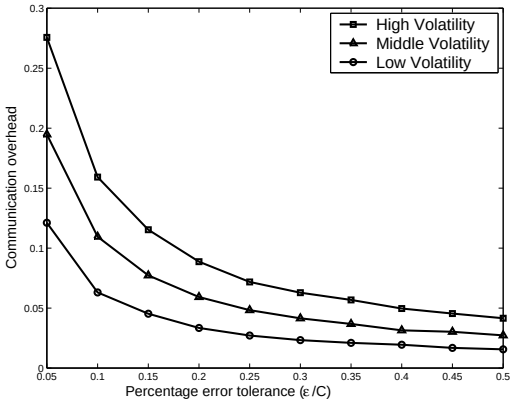
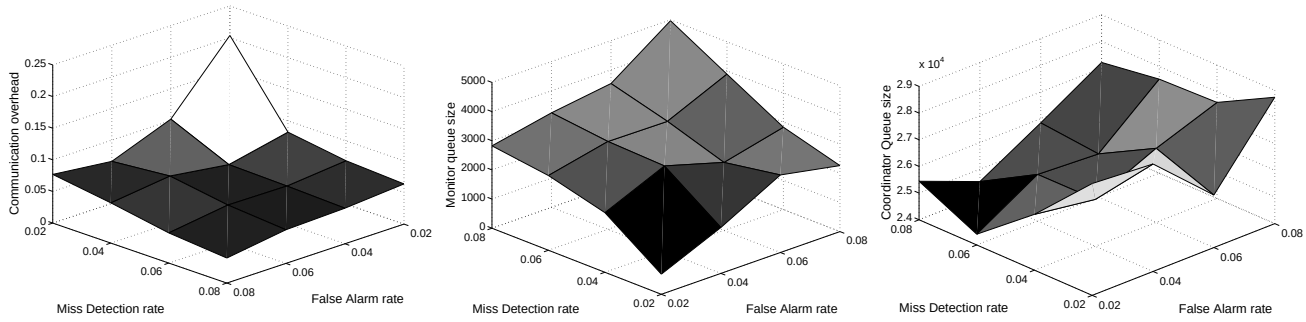


Figure 10: Impact of volatility on overhead.



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

Figure 11: Parameters design and tradeoff between false alarm, miss detection and communication overhead.

5.5 Detection Performance vs. Communication Overhead

We examined the tradeoffs between the false alarm and missed detection rates with the communication overhead and the two queue sizes. We use $\epsilon = 0.2C$ for these experiments. Fig. 11(a) shows the communication overhead tradeoff, while (b) and (c) show the monitor queue and coordinator queue sizes used for each achieved performance level (β^*, η^*). Note that to facilitate viewing of the 3-dimensional plots, the order of increasing β^* and η^* in Fig. 11(a) differs from that in (b) and (c).

In Fig. 11(a) we see that the 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. 11(a) is consistent with Figs. 9 and 10. 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. 9, 10, and 11(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 today's systems. Moreover today's prototypes do not provide any guarantees.

Fig. 11(b) shows that as the tolerable false alarm rate increases, the local queues increase in size because we can do more filtering at the monitors, which in turn brings down the overhead. This explains why the overhead decreases with increasing false alarm rate. A similar behavior occurs when the tolerable missed detection rate is raised.

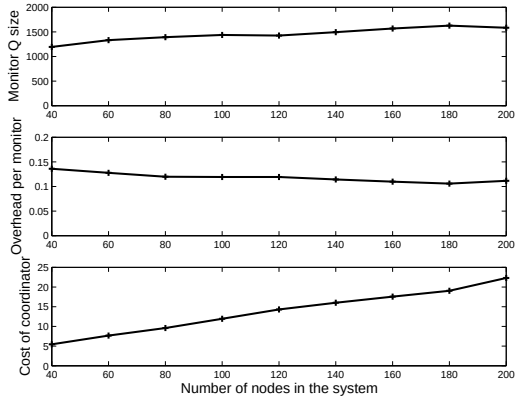


Figure 12: Communication overhead versus system size.

Looking at both (b) and (c) together, we see that a small change in (β, η) can lead to sizable change in the local queue, but relatively small amounts of change in the coordinator queue. Because the coordinator does not vary much, even when we change the accuracy requirements, we conclude that cancellation across the signals of different monitors is indeed occurring.

5.6 System Scalability

We now examine our system's scalability as the number of distributed monitors grows. Recall that one of the key reasons for controlling the communications cost is to avoid overwhelming the coordinator should it receive lots of data from many monitors. The communications overhead metric we have been using until now (namely $num/n \cdot m$) is an average value for the overhead *per monitor*. This therefore captures how much reduction can be done on each typical time series. However the communications bandwidth coming into the coordinator is the sum of all these filtered time series. We refer to this as the communications *cost*. This cost with respect to the coordinator can be computed from num/m. This captures the average number of messages the coordinator receives in one time slot.

We plot the communications cost as a function of the number of monitors in Fig. 12. We varied the number of monitors from 40 to 200, and used the target performance triplet

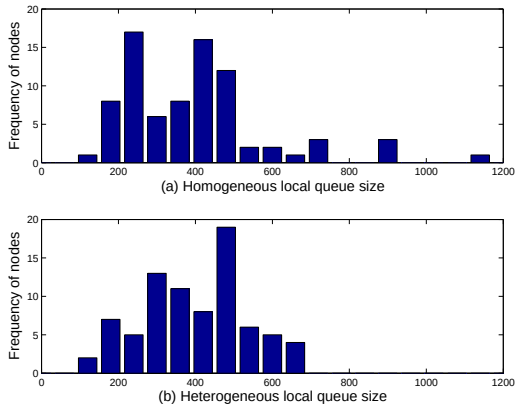


Figure 13: Number of messages per node.

$(\epsilon, \beta, \eta) = (0.2C, 0.06, 0.06)$. For each system size n, we run 5 rounds of experiments, each of which runs on n randomly picked monitors. In this Figure, as the system size increases: 1) the communication overhead of each monitor decreases slightly; and 2) the communication cost of coordinator increases slowly with the slope roughly being 0.1. This result indicates that the communication cost increases sublinearly as system size increases, and that our system thus scales gracefully. The intuition here is that as the number of monitor nodes increases, when one monitor queue overflows, it is more likely that there will be an underflowing queue elsewhere, and this leads to more signal cancellation at the coordinator. Our algorithm captures this trend and enables monitors to use larger queue sizes to filter out more updates, which in turn results in less communication overhead.

We note that the monitor and coordinator queues grow as the system scales. We point out that this is not related to scalability because when our solutions are implemented there is no need to implement actual buffers; instead the queues are implemented as counters, and the queue sizes correspond to maximum counter values.

5.7 Non-Uniform Slack Allocation

Finally we consider the case of non-uniform slack allocation. Recall that this means that the queue sizes the coordinator assigns to each monitor will be heterogeneous. We run this experiment using a network with 80 nodes over 4000 time slots, in which each node is randomly assign a data stream. We run this experiment twice, once with homogeneous queue sizes and once with heterogeneous queue sizes. The performance in terms of our three main performance metrics is nearly identical. The values for (overhead,β^*, η^*) are $(10.7\%, 0.030, 0.032)$ with homogeneous queues and $(10.6\%, 0.020, 0.035)$ for heterogeneous queues.

The difference in these two systems will be in terms of how much data each monitor sends in these two scenarios. We measured the volatility of our 80 data streams through their standard deviation. The distribution of these standard deviations was a bell shaped curve that ranged from 100 to 2000, indicating that we are indeed using a collection of time series with a broad range of volatility. One might hypothesize that more volatile time series need to send more data than less volatile ones (or at least that these two time series would differ; the amount sent clearly depends upon the queue size as well). Consider the plots in Fig. 13. In (a) we illustrate the number of messages sent by each monitor for the homogeneous queue case. On average around 440 messages are sent by each node over all time slots. We see that there exists some "hot-spot" heavy nodes, that send more than 1100 messages in the experiment, a great deal more than other monitors. In (b) we provide the same plot as (a) but for the heterogeneous case. We see that the distribution of per-node messages sent is concentrated in a smaller region (200-700), and there are no longer any unusually heavy nodes. Using non-uniform slack allocation can remove "hot-spot" nodes because it allocates more slack to nodes with more volatile data streams. This show another feature of our system, namely that we can achieve some kind of load balancing by using non-uniform slack allocation, without paying any penalty in terms of error and overhead performance.

6 Discussion

In this section, we discuss two practical issues for a real-world deployment and a potential extension.

Fault Tolerance. Our system has a single coordinator that is responsible for triggering. There are several approaches that can be used to tolerate this single point of failure, including having monitors multicast data to multiple coordinators, and using hierarchical aggregation structures [23] or peer-to-peer topology management [28]. Regardless of the choice of fault-tolerance mechanism, our distributed trigger scheme offers benefits and remains applicable.

Hierarchical Structure. Our approach can be extended to a multi-level tree structure with the following benefits: 1) reducing the coordinator's communication and processing workload because the roots of subtrees can perform partial aggregation and detection; 2) mapping monitors in different administrative or network domains into different subtrees (one for each domain) to exploit spatial locality.

Complex Triggers. In this paper, we solved a distributed triggering problem for anomalies defined as a `SUM` function exceeding a threshold. Our solution detects threshold violations with specified accuracy while minimizing communication overhead, as well as providing the flexibility for users to trade off communication overhead with detection accuracy. However, our queuing-based approach can support other general anomaly types. For example, in [13], we explore complex triggers that detect network-wide anomalies in a dynamic Origin-Destination network traffic flow matrix by: a) using a Principal Components Analysis (PCA) technique to decompose network traffic into normal and residual components; b) applying a threshold function to detect anomalies

on residual components [18]. We believe that our model of simple, efficient, and extensible triggers can support a variety of monitoring tasks and can be composed with existing query and detection techniques to enhance applications with sophisticated distributed detection capabilities.

7 Conclusion and Future Work

We have presented a novel solution to the problem of efficient cumulative triggering on an aggregate constraint condition in a distributed monitoring system. We believe our work is the first to address constraint detection over a time-varying window. Our solution relies on a key insight of focusing on accurate triggering, and not ϵ-accurate aggregate value reporting. This insight 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 queuing theory-based problem definition, which makes analytical solutions possible; and performing a detailed evaluation of our schemes (and representative alternatives) 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. One area is support for more general and complex correlation functions (other than our choice of `SUM`) by incorporating sketching techniques and advanced anomaly detection algorithms into our framework. Another area is the use of a multi-level tree hierarchy to further reduce the processing and communication workload at the coordinator.

References

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

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

[3] CHUN, B., HELLERSTEIN, J., HUEBSCH, R., MANIATIS, P., AND ROSCOE, T. Design considerations for information planes. In *WORLDS* (2004).

[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] CORMODE, G., GAROFALAKIS, M., MUTHUKRISHNAN, S., AND RASTOGI, R. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *ACM SIGMOD* (2005).

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

[8] DELIGIANNAKIS, A., KOTIDIS, Y., AND ROUSSOPOULOS, N. Hierarchical in-network data aggregation with quality guarantees. In *EDBT* (2004).

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

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

[11] HANSON, E. N., BODAGALA, S., AND CHADAGA., U. Trigger condition testing and view maintenance using optimized discrimination network. *IEEE TKDE*, 14(2) (2002).

[12] HUANG, L., GAROFALAKIS, M., JOSEPH, A., AND TAFT, N. Communication-efficient tracking of distributed triggers. UC Berkeley Tech. rep., May 2006.

[13] HUANG, L., GAROFALAKIS, M., HELLERSTEIN, J., JOSEPH, A., AND TAFT, N. Toward sophisticated detection with distributed triggers. Under submission, April 2006.

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

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

[16] 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).

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

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

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

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

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

[22] PITTS, J., AND SCHORMANS, J. *Introduction to IP and ATM design and performance with applications and analysis software*. John Wiley and Sons, 1996.

[23] RENESSE, R. V., BIRMAN, K., AND VOGELS, W. Astrolabe: a robust and scalable technology for distributed system monitoring, management and data mining. *ACM Trans. on Computer Systems*, 21(2) (2003).

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

[25] WIDOM, J., AND S.CERI. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, 1996.

[26] 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).

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

[28] ZHAO, B., HUANG, L., STRIBLING, J., JOSEPH, A., AND KUBIATOWICZ, J. Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays. In *IEEE ICNP* (2003).

8 Appendix

8.1 Instantaneous & Fixed-Window Triggers

Our goal for instantaneous triggers is to track an instantaneous violation *approximately* to within the specified error tolerance ϵ around threshold C. For example, if the aggregation function is a `SUM` function on n nodes, then the trigger fires for any time instant t where $\sum_{i=1}^{n} r_i(t) > C + \epsilon$.

Our work here and the work in [17] independently developed algorithms for tracking instantaneous triggers. The algorithms for tracking instantaneous trigger violations are significantly simpler than those for the cumulative case, and follow along the framework discussed in Sec. 3 (Fig. 3). The tracking protocol at the coordinator and local monitors is quite simple and, at a high level, very similar to the solutions proposed in earlier work for approximate aggregate tracking (*e.g.*, [15, 19]): The coordinator determines the amount of global monitor slack Δ and an allotment of Δ to individual local slacks δ_i such that $\Delta = \sum_i \delta_i$. Then, for each i, the δ_i value is communicated to monitor m_i which continuously

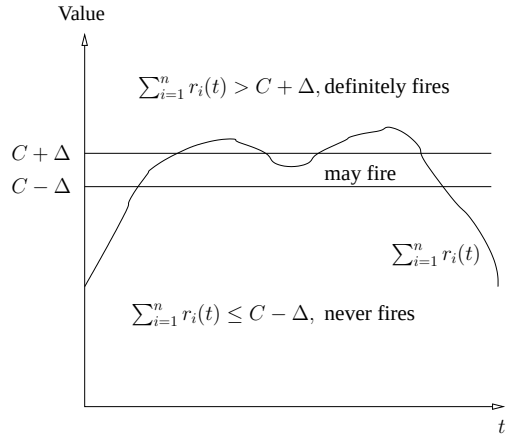


Figure 14: Instantaneous trigger tracking guarantees.

tracks the (instantaneous) difference $d_i(t) = r_i(t) - R_i(t)$ between the true local signal and its (most recent) prediction. Whenever $|d_i(t)| > \delta_i$, m_i sends an update message to the coordinator that includes the up-to-date value of $r_i(t)$, along with an up-to-date prediction $R_i(t)$ that satisfies the local filtering constraint at m_i. The coordinator continuously tracks the aggregate predictions across all monitor sites, and fires a trigger condition violation whenever $\sum_{i=1}^{n} R_i(t) > C$.

Based on the above protocol, it is not difficult to show (see, *e.g.*, [19]) that the above tracking algorithm always guarantees a $\pm\Delta$ additive bound for the predictions tracked at the coordinator; this, in turn, directly implies the following result.

Theorem 4 *The above instantaneous trigger-tracking scheme is guaranteed to: (1)* fire *whenever* $\sum_i r_i(t) > C + \Delta$*; and, (2)* never fire *whenever* $\sum_i r_i(t) < C - \Delta$.

In other words, Theorem 4 asserts a "band of uncertainty" (of size 2Δ) around the trigger threshold C, where our simple tracking algorithm may or may not fire a trigger violation; an illustration is shown in Fig. 14. A straightforward application of the above theorem with $\Delta = \epsilon$ (essentially, directly applying the techniques of [15, 19]) would ensure that our algorithm tracks the instantaneous distributed trigger to within ϵ additive error (as discussed in Sec. 2). Similarly, the convex optimization algorithms of Olston et al. [19] (based on the idea of marginal gains) can be used to determine the optimal allocation of the (fixed) total slack $\Delta = \epsilon$ to local monitor slacks δ_i. Clearly, however, such an approach is far too conservative: Our global slack should be able to adapt to changing local signals at the monitors based on the required threshold value (Sec. 3). We now discuss such an adaptive, threshold-based approach that provides the required ϵ-error guarantees — our experimental results in Sec. 5 clearly demonstrate the benefits of such adaptivity in practice.

Adaptive Instantaneous Trigger Tracking. The key idea of our adaptive scheme is quite simple. Unlike [15, 19], we are not interested in ϵ-error approximations to the true aggregate signal $\sum_i r_i(t)$, unless its value is close to the trigger threshold C. When $\sum_i r_i(t) < C$, the additional slack should be exploited to effectively minimize updates from monitors to the coordinator. Formally, for any time instant t, the coordinator estimates the total amount of available slack as:

$$\Delta(t) = C + \epsilon - \sum_{i=1}^{n} R_i(t)$$

and can distribute that slack to local monitor δ_i's (*e.g.*, using a marginal-gains strategy, as in [19]). (The key idea here is to allocate slack based on the expected reduction in the number of monitor messages per unit of slack, as estimated from the recent update history for each monitor at the coordinator.) Thus, aggregate signals that are (expected to be) far from the trigger threshold imply additional slack and, therefore, reduced communication for each local monitor.

Compared to the simple instantaneous tracking scheme described earlier in this section, the local monitor protocol remains unchanged while the coordinator protocol changes in order to effectively adapt to changing values of $\Delta(t)$. Specifically, when the coordinator receives a monitor update (at, say, time t), it recomputes the current global slack $\Delta(t)$, based on which it computes and disseminates new local slacks δ_i to individual monitors. The following theorem shows that our adaptive scheme indeed guarantees ϵ-approximate instantaneous trigger tracking.

Theorem 5 *Employing an adaptive global monitor slack equal to* $\Delta(t) = C + \epsilon - \sum_{i=1}^{n} R_i(t)$*, where* $R_i(t)$ *denotes the up-to-date prediction from monitor* m_i *(for all* i*) ensures that the coordinator check* $\sum_{i=1}^{n} R_i(t) > C$*: (1)* always fires *if* $\sum_i r_i(t) > C + \epsilon$*; and, (2)* never fires *if* $\sum_i r_i(t) < C - \epsilon$.

Of course, as with our adaptive scheme for cumulative triggers, the key here is to avoid an overly sensitive coordinator that disseminates new δ_i's for small, transient changes in $\Delta(t)$. Our coordinator algorithms achieve this through min-based and discretization-based filtering steps on $\Delta(t)$, similar to those described in Sec. 4.3. For instance, Fig. 15 depicts the the smoothing effect of the two filtering steps on a real-life aggregate signal.

Obviously, the adaptive global slack $\Delta(t)$ can be distributed across local monitor slack values δ_i in a non-uniform manner in order to minimize overall communication. As shown in [8, 19], in the case of a fixed total slack, the optimal allocation point is achieved by equalizing the individual marginal-gain ratios across all monitors. Similar results can be shown to hold in the case of adaptive total slack as well, and our implementation (discussed in Sec. 5) employs such a marginal-gains-based allocation scheme.

Extension to Fixed-Window Triggers. The tracking algorithms described in this section for the instantaneous trigger problem are also naturally applicable to the case of fixed-window triggers. Assuming a (fixed) window size τ, the idea is, for each monitor m_i to maintain its running local aggregate over the last τ time instants $s_i(t) = \int_{t-\tau}^{t} r_i(x)dx$

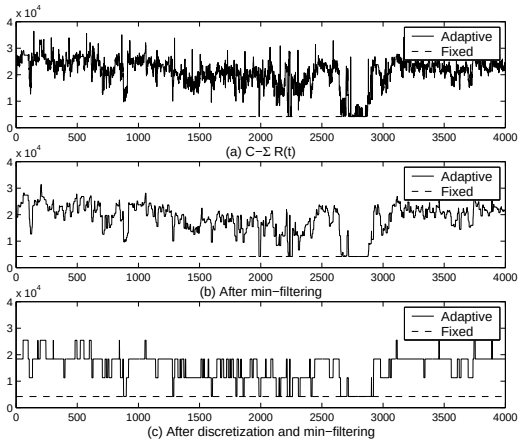


Figure 15: Effect of min- and discretization-based filtering.

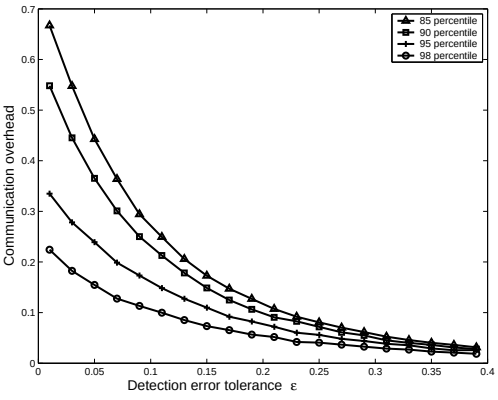


Figure 17: Impact of target C on communication overhead.

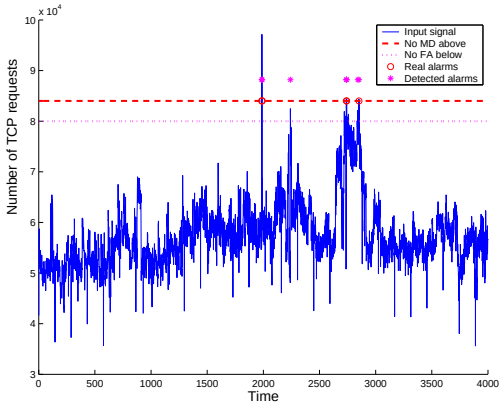


Figure 16: Triggering on instantaneous violation.

(which is trivial to do assuming $O(\tau)$ space). Then, the fixed-window trigger over the $r_i(t)$ signals is essentially transformed into an instantaneous trigger over the $s_i(t)$ window aggregates, and all the techniques discussed earlier in this section are naturally applicable.

8.2 Evaluation of Instantaneous Triggers

The same data and implementation setup for cumulative triggers are used to evaluate instantaneous triggers. Recall that our assurance for instantaneous triggers is slightly different from cumulative triggers. Given an error tolerance ϵ, our algorithms assert a "2ϵ-zone of uncertainty" around the trigger threshold C, and have zero false alarm and zero miss detection outside this band.

We first show in Fig. 16 a sample of aggregate time series used in experiments and the triggering performance guaranteed by Theorem 5. In the graph x-axis is time and y-axis is signal value. The solid curve denotes the time series signals. The dotted line denotes the value that no false alarm should be triggered when signals are below this line, and the dashed line denotes the value that no miss detection should happen when signals are above this line. The region between the two lines are the uncertainty zone. In the experiment, we set trigger threshold C to be the value of the 99^{th} percentile of the distribution of all values in the time series. Error tolerance $\epsilon = 0.05$, so the size of uncertainy zone is $2\epsilon \cdot C = 0.1 \cdot C$. Circles denote the real violations under this setup, and stars denote triggers fired by our protocol. We can clearly see that our protocol guarantees the desired performance: detecting all real violations and only has false alarms inside the uncertainty zone. When aggregate signals are above uncertainty zone, no trigger is missed, indicated by one star for each circle; if aggregate signals are below uncertainty zone, no trigger is fired, indicated by no star for signals under the dotted line. Inside the uncertainty zone, our protocol have one false alarm, indicated by the second star in the graph.

A set of following results show the tradeoff between the error tolerance and the communication overhead incurred. We start with how the value of C and the data volatility impact the communication overhead under different error tolerance, followed by the comparison of our work with existing approaches and the scalability of our system.

Fig. 17 shows the relationship between the communication overhead and the target threshold C when setting C to be the $85^{th}, 90^{th}, 95^{th}$ and 98^{th} percentile of the distribution of all aggregate signals. We first see from the graph that for all different C values, communication overhead decreases quickly as error tolerance ϵ increases. More important, for any given ϵ, the communication overhead decreases as target threshold C increases. This is expected, when C is large as the 98^{th} percentile, aggregate signals are always far away from C. So our protocol always keeps a loose eye on signals from individual monitors, thus paying low cost to achieve the desired detection accuracy; however, when C is small as 85^{th} percentile, aggregate signals are always close to C, and our protocol has to keep a close eye on individual signals to see whether they cause constraint violations, thus paying relative high cost to achieve the desired detection accuracy. However, when the error tolerance is big enough, the protocol pays low cost even when C is small. This demonstrate that our ideal to let total slack adapt according to aggregate signals is very

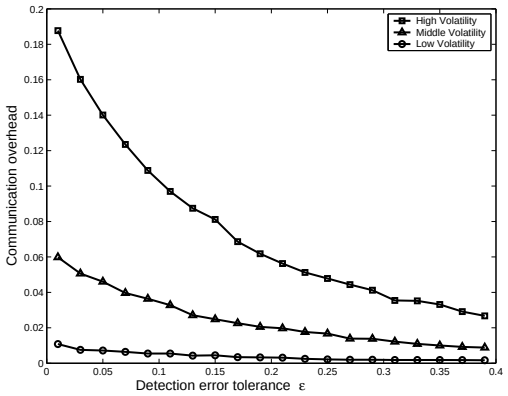


Figure 18: Impact of volatility on communication overhead.

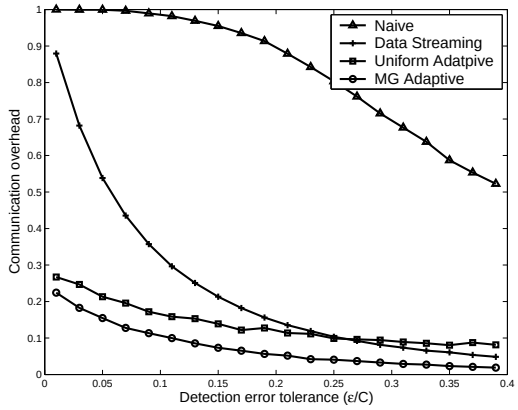


Figure 19: Comparing our approach to existing approaches.

effective in reducing communication overhead.

Fig. 18 shows the impact of data volatility on communication overhead. The experiment setup is similar as that in Fig. 10. As expected, for every given value of ϵ, the communication overhead decreases quickly as the volatility of the data decreases, indicating that our solution is adaptive to data properties.

We are now comparing our work with previous work. The only adaptive filtering approach similar to our problem is that proposed by Olsten *et al.* in [19] in the context of data streaming. We also compare our solution with a naive approach that might be attractive for its simplicity, as the δ_i are fixed (non-adaptive) and homogeneous. In any time epoch, each monitor i sends an update to the coordinator only if its time series $r_i(t) > \frac{(C+\epsilon)}{n}$. [7] We consider two versions of our protocol. In one, called *uniform adaptive detection*, the δ_i's are homogeneous but adapt due to the adapting total slack Δ. In the second version, called *MG (Marginal Gain) adaptive*, the local slacks are heterogeneous and are computed according to their Marginal Gain in reducing communication cost. The method for computing the MG is given in [19] where the authors prove that this is optimal for when there is a given total amount of slack to distribute to the monitors. Both our MG adaptive version and the data streaming scheme use the same approach when doing heterogeneous local slack. The essential difference is that in the data streaming scheme, the total slack to be distributed remains constant, whereas in the MG adaptive scheme the total slack is adaptive and varies according to how far we are from the triggering constraint. Recall that we can do this because our goal is to estimate the trigger accurately, while the data streaming scheme's goal is ϵ-accurate estimation of the aggregate signal.

The results of this comparison are shown in Fig. 19. For these tests, we set C to be the 98^{th} percentile of the aggregate signal. As one would intuitively expect, the communication overhead decreases as error tolerance increases. (Note that Theorem 5 essentially guarantees no false alarms/missed detections outside the "uncertainty" zone for our instantaneous triggering scheme.)

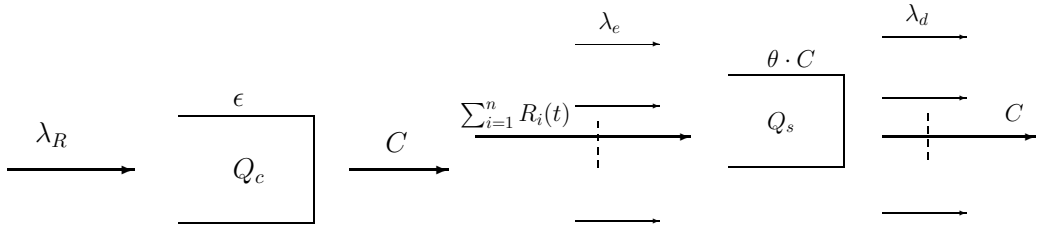
We see that our MG adaptive scheme substantially outperforms both the naive and data streaming methods, at any error tolerance level. Our uniform adaptive scheme also outperforms data streaming in the lower ranges of error tolerance (*e.g.*, $\epsilon < 0.27$ in this case). The strong performance of our schemes illustrate that adapting the global slack is a powerful idea for reducing communication overhead. *This implies that when designing systems for distributed triggering, rather than signal estimation, a very different level of communication efficiency is achieved.*

We also observe that our MG adaptive scheme outperforms our uniform adaptive scheme. This implies that allowing heterogeneous filters at each monitor can lead to communication reduction; however the gain resulting from adaptivity of the total slack is certainly more dramatic than the gain resulting from allowing heterogeneity of local slack across monitors.

The improvement of our schemes over previous methods is particularly notable in the low error tolerance region (*e.g.*, $\epsilon < 0.1$). For example, when $\epsilon = 0.1$, our approach can filter out more than 90 of the original time series signals while achieving the desired detection accuracy. with transmission of only 10 of the original signal. Our scheme results in 3.5 times less communication overhead than the data streaming technique which sends 35 of the original signal for the same target accuracy level. For even smaller values of ϵ, the gains of our schemes over existing schemes are even greater. Thus, we conclude that our scheme is well suited for distributed triggering systems with very low error tolerances.

Fig. 20 shows the scalability of our protocol for instantaneous triggers. We measure the communication overhead as a function of the number of distributed monitors, which vary from 20 to 200 in the system. The experiment setup is similar as that shown in Fig. 12. As seen from the result, the (per-node) communication overhead is relative stable and slightly decreasing when the system size increases, indicating that our protocol is scalable to large system.

[7] We do not compare with [16] since, unlike our schemes and [19], they offer no strict error guarantees.



(a) *The centralized idea model* (b) *The distributed solution model*

Figure 21: Queuing model for slack estimation.

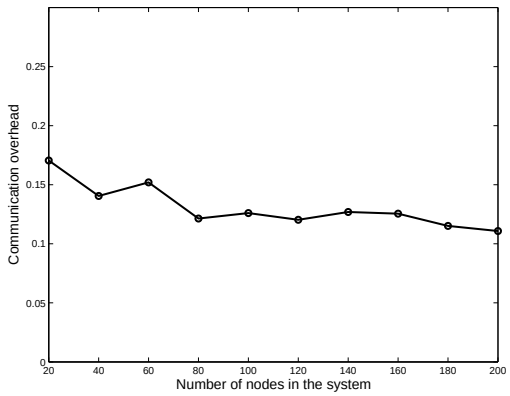


Figure 20: Communication overhead versus system size.

9 Appendix

We present here the proofs of theorems in the paper. For the varying-window trigger, both the centralized ideal model and the distributed solution model are shown in Figure 21. Let the coordinator queue be Q_c with size ϵ in the ideal model, and be Q_s with size θ in the solution model.

9.1 Proof of Theorem 2

Miss detections happen if both monitor queues and coordinator queue in the solution model are so large that they absorb enough update traffic to mask a real trigger violation. Let the occupancies (in unit of δ) of monitor queues over time be random variables $\alpha_1, ..., \alpha_n$, then we have $-\delta < \alpha_i < \delta$ in our setting. It is reasonable to assume that each α_i follows an independent Normal $N(0, \sigma_i)$ distribution. Then the aggregate occupancy of monitor queues, $S = \sum_{i=1}^{n} \alpha_i$, follows a Normal $N(0, \sigma^2)$ distribution, where $\sigma^2 = \sigma_1^2 + ... + \sigma_n^2$. Let $F()$ denotes the CDF of $N(0, \sigma^2)$, then the probability that monitor queues have aggregate occupancy more than x is $1 - F(x)$.

In the centralized model, arrivals of $\sum_i r_i(t)$ overflow queue Q_c with probability, which is relative small in real system because constraint violations are rare events. So $\sum_i r_i(t)$ should be less than C on average over time, however, $\sum_i r_i(t)$ is bigger than C in some periods and causes Q_c to overflow. When assuming an $M/M/1$ model for the coordinator queue at the granularity of δ, we have following approximation for Q_c: 1) length of Q_c is $\frac{\epsilon}{\delta}$; 2) enqueue of $\sum_i r_i(t)$ is approximated by a Poisson arrival with (average) rate $\frac{\lambda_r}{\delta}$; 3) dequeue is approximated by a Poisson arrival with rate $\frac{C}{\delta}$. With this setup, the overflow probability of Q_c in centralize model can be determined by [22]

$$Pr(l_r > \frac{\epsilon}{\delta}) = \left(\frac{\lambda}{\mu}\right)^{\frac{\epsilon}{\delta}+1} = \left(\frac{\lambda_r}{C}\right)^{\frac{\epsilon}{\delta}+1} = \rho^{\frac{\epsilon}{\delta}+1}$$

where $\rho = \frac{\lambda_r}{C}$ is the queue utilization. When Q_c is overflowed, it is possible that Q_c has occupancy (in unit of δ) $l_r = \frac{\epsilon}{\delta} + i$, for each $i = 1, ..., \infty$, each of which has probability:

$$\begin{aligned} Pr\left(l_r = \frac{\epsilon}{\delta} + i | l_r \geq \frac{\epsilon}{\delta} + 1\right) &= \frac{\rho^{(\frac{\epsilon}{\delta}+i)}(1-\rho)}{\rho^{\frac{\epsilon}{\delta}+1}} \\ &= \rho^{i-1}(1-\rho) \end{aligned}$$

A miss detection happens when Q_c has occupancy $\frac{\epsilon}{\delta} + i$ (which causes constraint violation), but Q_s has occupancy $l_s \leq \frac{\theta}{\delta}$ (otherwise, Q_s is overflowed and the trigger fires). This happens because monitor queues hold too much fluid and have aggregate occupancy more than $\frac{\epsilon}{\delta} + i - \frac{\theta}{\delta} = \frac{\epsilon-\theta}{\delta} + i$, which has probability $1 - F\left(\frac{\epsilon-\theta}{\delta} + i\right)$. So the miss detection rate (probability) β can be approximated as

$$\begin{aligned} \beta &= Pr(l_s \leq \frac{\theta}{\delta} | l_r \geq \frac{\epsilon}{\delta} + 1) \\ &= \sum_{i=0}^{\infty} \left\{ \left[1 - F\left(\frac{\epsilon-\theta}{\delta} + i + 1\right)\right] \cdot \rho^i(1-\rho) \right\} \end{aligned}$$

When using i as a continuous variable, we get the integral version of the equation.

If assuming an $M/D/1$ model for the coordinator queue, we can approximately compute its queue length distribution as [22]

$$Pr(l_r > x) = \exp\left[-2x\left(\frac{\mu-\lambda}{\lambda}\right)\right] = \pi^x$$

where $\pi = \exp\left[-2\left(\frac{\mu-\lambda}{\lambda}\right)\right]$. With this model, β can be computed as

$$\begin{aligned} \beta &= Pr(l_s \leq \frac{\theta}{\delta} | l_r \geq \frac{\epsilon}{\delta} + 1) \\ &= \sum_{i=0}^{\infty} \left\{ \left[1 - F\left(\frac{\epsilon-\theta}{\delta} + i + 1\right)\right] \cdot \pi^i(1-\pi) \right\} \end{aligned}$$

9.2 Proof of Theorem 3

False alarms happen when a combination of chunk bursts in the solution model causes Q_s to overflow even though the true aggregate signals have not caused Q_c to overflow in the centralized model.

On the granularity of δ, we have following approximation for Q_s in the solution model: 1) length of Q_s is $\frac{\theta}{\delta}$; 2) enqueue of $\sum_i R_i(t)$ is approximated by a Poisson arrival with (average) rate $\frac{\lambda_R}{\delta}$; 3) dequeue is approximated by a Poisson arrival with rate $\frac{C}{\delta}$. 4) chunk enqueue from all monitors are approximated by a Poisson arrival with rate λ_e, and chunk dequeue by a Poisson arrival with rate λ_d. Then, the overflow probability of Q_s is

$$Pr(l_s > \frac{\theta}{\delta}) = \left(\frac{\lambda}{\mu}\right)^{\frac{\theta}{\delta}+1} = \left(\frac{\lambda_R + \lambda_e \cdot \delta}{C + \lambda_d \cdot \delta}\right)^{\frac{\theta}{\delta}+1}$$

Apparently, the solution model is more bursty than the centralized model, and $\frac{\theta}{\delta}$ is less than $\frac{\epsilon}{\delta}$. So $Pr(l_s > \frac{\theta}{\delta})$, the overflow probability in the solution model, is bigger than $Pr(l_r > \frac{\epsilon}{\delta})$, the overflow probability in the centralized model. The false alarm rate (probability) η can be simply approximated by

$$\begin{aligned}\eta &= \frac{\left[Pr(l_s > \frac{\theta}{\delta}) - Pr(l_r > \frac{\epsilon}{\delta})\right]}{Pr(l_r > \frac{\theta}{\delta})} \\ &= 1 - \left(\frac{\lambda_r}{C}\right)^{\frac{\epsilon}{\delta}+1} \Big/ \left(\frac{\lambda_R + \lambda_e \cdot \delta}{C + \lambda_d \cdot \delta}\right)^{\frac{\theta}{\delta}+1}\end{aligned}$$

Using $M/D/1$ queuing model, the overflow probability and false alarm rate can be computed as

$$\begin{aligned}Pr(l_r > \frac{\epsilon}{\delta}) &= \exp\left[-\frac{2\epsilon}{\delta}\left(\frac{C-\lambda_r}{\lambda_r}\right)\right] \\ Pr(l_s > \frac{\theta}{\delta}) &= \exp\left[-\frac{2\theta}{\delta}\left(\frac{\mu-\lambda}{\lambda}\right)\right] \\ &= \exp\left[-\frac{2\theta}{\delta}\left(\frac{C-\lambda_r}{\lambda_r + \lambda_d \cdot \delta}\right)\right] \\ \eta &= \frac{\left[Pr(l_s > \frac{\theta}{\delta}) - Pr(l_r > \frac{\epsilon}{\delta})\right]}{Pr(l_r > \frac{\theta}{\delta})} = 1 - \frac{Pr(l_r > \frac{\epsilon}{\delta})}{Pr(l_s > \frac{\theta}{\delta})} \\ &= 1 - \exp\left[\frac{2\theta}{\delta}\left(\frac{C-\lambda_r}{\lambda_r + \lambda_d \cdot \delta}\right) - \frac{2\epsilon}{\delta}\left(\frac{C-\lambda_r}{\lambda_r}\right)\right]\end{aligned}$$

9.3 Proof of Theorem 4

The simple scheme for instantaneous trigger-tracking has the following guarantee for each monitor m_i:

$$|r_i(t) - R_i(t)| \leq \delta_i$$

Summing over i on $R_i(t) \geq r_i(t) - \delta_i$, we get $\sum_{i=1}^n R_i(t) \geq \sum_{i=1}^n r_i(t) - \Delta$. Whenever $\sum_{i=1}^n r_i(t) > (C + \Delta)$, the coordinator has

$$\sum_{i=1}^n R_i(t) > (C + \Delta) - \Delta = C$$

and immediately fires the trigger.

With the same reasoning, the scheme guarantees $\sum_{i=1}^n R_i(t) \leq \sum_{i=1}^n r_i(t) + \Delta$. When $\sum_{i=1}^n r_i(t) < (C - \Delta)$, the coordinator has

$$\sum_{i=1}^n R_i(t) < (C - \Delta) + \Delta = C$$

and never fires the trigger.

9.4 Proof of Theorem 5

The detection error of the adaptive scheme for instantaneous trigger-tracking is caused by the value discrepancy between monitors and the coordinator. With $\Delta(t) = C + \epsilon - \sum_{i=1}^n R_i(t)$, the value discrepancy of the scheme can be analyzed as follows.

- When $\sum_i R_i(t) \geq C$, we have $\Delta(t) \leq \epsilon$. The value discrepancy between monitors and coordinator is upbounded by ϵ. The coordinator always has an ϵ- approximation of aggregate signals produced by monitors, and has the desired detection guarantee as that in Theorem 4.

- When $\sum_i R_i(t) < C$ at time t, monitors have $\sum_i r_i(t)$ and coordinator believes monitors have $\sum_i R_i(t)$. This accrues value discrepancy $\sum_i r_i(t) - \sum_i R_i(t)$. Because the setting of $\Delta(t)$, total value discrepancy can be big up to $C+\epsilon-\sum_i R_i(t)$ at any subsequent time $t' > t$. So, without triggering any value update at monitors, the change from $\sum_i r_i(t)$ to $\sum_i r_i(t')$ can be at most

$$\begin{aligned}DR = (C - \sum_i R_i(t) + \epsilon) - (\sum_i r_i(t) - \sum_i R_i(t)) \\ = C - \sum_i r_i(t) + \epsilon\end{aligned}$$

  This "no-update"would not cause constraint violation, because the value of $\sum_i r_i(t')$ unknown to coordinator is at most

$$\sum_{i=1}^n r_i(t') \leq \sum_i r_i(t) + C - \sum_i r_i(t) + \epsilon = C + \epsilon$$