D-Trigger: A General Framework for Ef cient Online Detection

Ling Huang

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-119
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-119.html

September 24, 2007

D-Trigger: A General Framework for Efficient Online Detection

by

Ling Huang

B.S. (Beijing University of Aeronautics and Astronautics) 1994
M.S. (Beijing University of Aeronautics and Astronautics) 1997

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Anthony D. Joseph, Chair
Professor Joseph M. Hellerstein
Professor Bin Yu

Fall 2007

The dissertation of Ling Huang is approved:

Professor Anthony D. Joseph, Chair Date

Professor Joseph M. Hellerstein Date

Professor Bin Yu Date

University of California, Berkeley

Fall 2007

D-Trigger: A General Framework for Efficient Online Detection

Ling Huang

Abstract

D-Trigger: A General Framework for Efficient Online Detection

by

Ling Huang

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Anthony D. Joseph, Chair

As the Internet evolves into a valuable and critical service platform for our business and daily life, there has been growing interest in large-scale distributed monitoring for network infrastructures. The monitoring systems collect and aggregate information describing status and performance of networked systems. Remote monitor sensors are typically deployed throughout the network generating numerous large and widely-distributed, continuous or discrete timeseries data streams, representing large-scale information flows from multiple vantage points. Existing research in system monitoring and data management involves periodically pushing all monitored data to a central Network Operation Center (NOC) for sophisticated analysis and anomaly detection. However, such a "periodic push" approach suffers from timescale and size scalability limitations. Many anomalies occur on much smaller time scales than typical polling periods. Detecting events on a second sub-second time scale requires that the volume of measurement data transmitted through the network increase dramatically because the monitoring data must be collected on a second (or sub-second)

time scale. Similarly, an order of magnitude (or more) increase in the number of monitors also causes a massive growth in the volume of collected data, and could overload the central processing site's network capacity, especially for networks such as sensor networks, wireless networks, and enterprise networks.

In this dissertation, we design and develop D-Trigger as a general framework for efficient online anomaly detection. D-Trigger addresses the lack of efficiency and flexibility in today's distributed monitoring and anomaly detection systems, and proposes a general framework which gracefully integrates a variety of decision-making and optimization algorithms for online detection. The key goals and accomplishments in this dissertation are to: 1) enable real-time detection where the system's state is tracked continuously, so even the smallest anomalies will be exposed; 2) significantly reduce the data collected for anomaly detection, thus reducing the communication burden placed on the network; 3) guarantee desired detection accuracy even with the reduced amount of collected data. To achieve the three goals, D-Trigger combines in-network processing at distributed local sites, and decision making at the NOC. The combination of distributed local processing strategies, sophisticated detection algorithms, and theoretical analysis tools enables D-Trigger to perform in-network tracking with very high detection accuracy and low communication overhead. In addition, D-Trigger is able to accommodate a broad set of statistical learning algorithms for the detection of various unusual events, including botnet attacks, volume anomalies in an ISP network, electric power grid anomalies, etc.

Our work on D-Trigger has resulted in an efficient detection system which is capable of detecting a wide-range of anomaly types in distributed systems in near real-time with bounded detection error. The system can be applied to a wide variety of monitoring problems and domains,

ranging from simple monitor functions (e.g., SUM, AVG, MIN, and MAX) to complex mathematical functions, and spanning areas such as sensor networks, enterprise networks, and even power distribution networks.

Professor Anthony D. Joseph
Dissertation Committee Chair

Acknowledgments

There are a lot of people whom I want to thank for their direct and indirect contributions in making this dissertation possible.

First of all, I would like to thank my advisor, Anthony Joseph, for the guidance, support, and honest criticisms he has provided throughout the course of this work. I am grateful for the numerous meetings that he had with me to discuss our research problems and to guide me through them with his profound knowledge in many diverse areas. His mastery of the research world has saved me from the fire on more than one occasion. Without his invaluable support and constant guidance, I could not have completed this dissertation.

I would also like to extend my thanks to Nina Taft and Minos Garofalakis. During my extended visits to Intel Research Laboratories, Nina and Minos graciously took me "under their wing", and provide me a challenging and, at the same time, friendly environment for me to work in. Nina had numerous meetings with me to discuss our research problems and to guide me through them with her profound knowledge. Minos always provided a inspiring, encouraging and insightful way to guide me to a deeper understanding of knowledge work. He is personally responsible for pointing Anthony and I to the distributed trigger work that led to this dissertation.

A number of other faculty members have had a hand in guiding me through the maze that is graduate school. I've often sought and received insightful advice and suggestion from Joe Hellerstein, Mike Jordan, John Kubiatowicz, Randy Katz, Ion Stoica, Bin Yu and Ben Zhao. Both Joe and Mike played important roles in helping me establish the framework for in-network anomaly detection. Joe and Bin were on my qualifying exam committee and gave me many insightful comments during my dissertation work. Ben has been there several times offering insightful advice and

perspective.

I would like to thank my co-author XuanLong Nguyen for his inspiring conversation and hard work. He was key in helping prove the soundness and properties of several approximation algorithms in our system. I would like to express my sincere gratitude to Boon Thau Loo, Yitao Duan, Fang Yu, Matthew Caeser, Byung-Gon Chun and Blaine Nelson for reading my papers and giving me insightful suggestions on various topics. My appreciation also goes to other members in various research groups at UC Berkeley. I acknowledge Hao Zhang, Chen Shen, Li Zhang, Feng Zhou, Peng Li, Weidong Cui, Li Yin, Jingtao Wang, Wei Xu for their valuable insight, patience for numerous questions, and entertainment after work.

Finally, my deepest heartfelt thanks to my family and close friends, who have helped me get through some tidal-waves in the last few years. There were times when I was at the edge of physical and emotional exhaustion, and they pulled me up and gave me courage to continue. I thank my best friends Dan Huang, Yitao Duan, Nan Ding, Wei Liu, Maoyi Huang, Zhangshuan Hou, Rui Huang, Xiaohu Liu. I am deeply indebted to my wife Jialing Lang for her love and understanding. Her support and encouragement was in the end what made this dissertation possible. My family in China has been very supportive of me all these years. My parents, sister, and brother have been constant sources of encouragement. They cheer for me in good times, and comfort me in bad times. It is my family that gives me the strength to overcome the innumerable obstacles that I encountered during my PhD study. I dedicate this dissertation to them.

To my family

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 A New Paradigm for Network Monitoring

Today's Internet has evolved into a pervasive and critical infrastructure for daily life and business activities. Large-scale *distributed monitoring and anomaly detection systems* have been largely deployed for health monitoring and detection of unusual events on the Internet. They 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 de-

ployed 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 of 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, to determine when and where capacity upgrades are needed. Wireless sensornets for habitat, environmental, and health monitoring also continuously monitor and correlate sensor measurements for trend analysis, and detection of moving objects, intrusions, or other adverse events.

Consider the example application of botnet detection in detail. In botnet attacks, multiple zombies try to open a large number of TCP connections to a single server (the victim). Such a scenario is depicted in Fig. 1.1. This figure can represent either an enterprise or ISP network. If the hosts in the figure reside inside the network, then this figure captures an enterprise network. If the hosts reside outside the network infrastructure, then the ensemble of routers can represent an ISP network. Assume a set of hosts have been recruited by a botnet, some subset of which reside in our given network. An external commander gives them an order to launch a large number of connections to a victim, that in this case resides outside our network. By tracking the number of simultaneous TCP connections at an individual machine or monitor, one might not detect an unusually high number of connections all headed towards the same destination. However, by tracking the `SUM` of the number of simultaneous TCP connections from these hosts, all with the same destination, one could more easily detect the excess or overload. Both enterprise and ISP networks employ central operation centers that could track such sums. If an ISP deployed our proposed functionality,

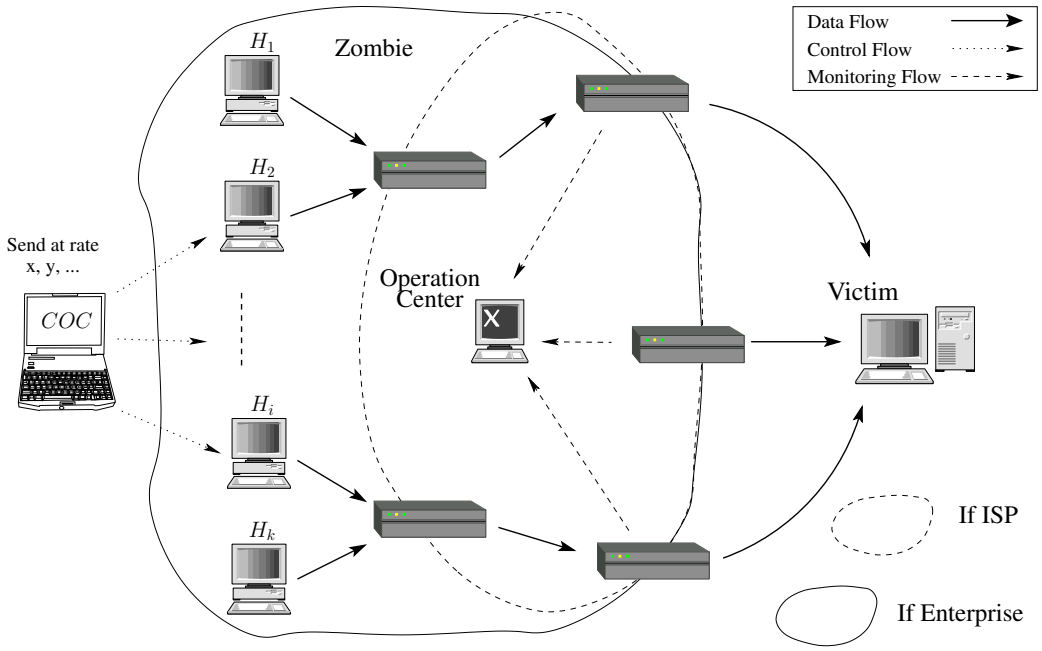


Figure 1.1: Example: Distributed detection of botnet attacks.

by using monitors on gateways and pushing data to the operations center, then it could block this unwanted incoming traffic via filters on the gateways. If an enterprise network could detect such traffic, the hosts could simply be disconnected. We point out that the goal here is not to completely squelch the entire Internet-wide botnet attack, nor to completely protect the victim (that, in this case, lies outside our network domain). Instead, the goal is to reduce an enterprise or ISP's liability by blocking the attack traffic emanating from its network.

This is a typical example of detecting whether there are a collection of compromised hosts within a 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 com-

promised host behaviors, can indeed reveal alarming levels of outgoing traffic to the destination. In some cases, simple linear aggregate functions like `SUM, MIN, MAX`, are enough to perform the detection; however, in other cases, more sophisticated anomaly detection functions have to be used to detect usual events in a large system. For example, Lakhina *et al.* [41] 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.

Over the past few years, the defining characteristics of online anomaly detection have been identified as posing new challenges that are not addressed by either traditional data management systems or network monitoring systems. The three fundamental key aspects of such large-scale monitoring and detection systems can be abstracted as follows:

1. 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. Queries in these monitoring situations are typically long running correlation analysis and evaluation functions over the data, which continuously run and return answers as they are found.

2. Monitoring is inherently *distributed*; that is, the data streams required to answer the monitoring queries are distributed throughout the network. 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.

3. Monitoring needs to track global detection functions defined over distributed datasets. Beside

continually recording system state, one primary goal in monitoring a system is to ensure that all is well. This is typically achieved by maintaining a well-defined set of logical predicates or detection functions over the entire system. In many situations, tracking the detection functions over system status collected from physically-distributed vantage points is much more revealing than tracking the local behavior of individual network hosts. At the same time, a monitoring system needs to accommodate a wide-range of correlation and decision functions, so that it can satisfy the requirements of different applications under a variety of situations.

1.2 Challenges for Efficient Online Monitoring and Detection

In distributed monitoring environments, useful monitoring data are produced continuously, and are spread across multiple distributed monitor sites. For many monitoring and detection functions, however, users and applications have to aggregate and access the data at the central NOC in a continuous way. This is because centralized data access is easy for network management (e.g., enforcing policies at a single point), and further more, many detection and correlation functions are simple to evaluate with global data at a single point. While monitoring and detection procedures tend to become simpler when reduced to centralized data access tasks, a significant challenge remains: that of collecting, shipping and using data across the system efficiently and effectively.

1.2.1 Communication Constraints

The distributed nature of monitor sites 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 typi-

cally do not overprovision their interconnections to remote office sites (e.g., crossing city, state, or country boundaries), yielding severe communication restrictions for their enterprise Intrusion Detection systems (IDS), 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.

In a bandwidth scarce environment, network communication resources can only be used at some premium. This premium for network usage may stem from the fact that increased congestion may cause service quality degradation for all applications that use the network. Alternatively, the premium may be manifest as a monetary cost, either in terms of direct payment to a service provider or as a loss of revenue due to an inability to sell consumed resources to others. As a result of communication resources being a valuable commodity, we have seen that the system monitoring and detection applications described above have to operate in a communication heavily constrained

environment.

While the sensor network community worries a lot about communication and computation efficiency [29, 26, 21, 13], few existing anomaly detection approaches in the network community consider communication efficiency. They always assume that complete data can be collected, and can be either continuously or periodically shipped to the central NOC for anomaly detection. For example, Lakhina *et al* propose the PCA-based method for network-wide traffic anomaly detection in [41]. They proposed that distributed monitors continuously measure the total volume of traffic (in bytes) on each network link, and periodically push all recent measurements to the NOC. The NOC then performs Principal Component Analysis (PCA) on the assembled data matrix to reveal traffic anomalies that were not detectable in any single link-level measurements. While effective, such a "periodic push" approach suffers from two scalability limitations:

1. The first limitation has to do with the time scale of operation and how fast anomalies can be detected. The work by Lakhina, *et al* was initially shown to work at 5 and 10 minute time scales [41]. However many anomalies occur on much smaller time scales. If the method were employed on a second or sub-second time scale, then the volume of measurement data transmitted through the network would increase dramatically because the monitoring data would need to be collected on a second (or sub-second) time scale.

2. The second scalability limitation has to do with the effect of increasing the number of monitoring sites. An approach in which *all* monitors upload *all* of their data to a central processing site regularly, creates two problems. It may overload the central processing site. Also, sending such large quantities of data through the network is a problem for certain kinds of networks such as sensor networks, many wireless networks, and enterprise networks (that do not over-

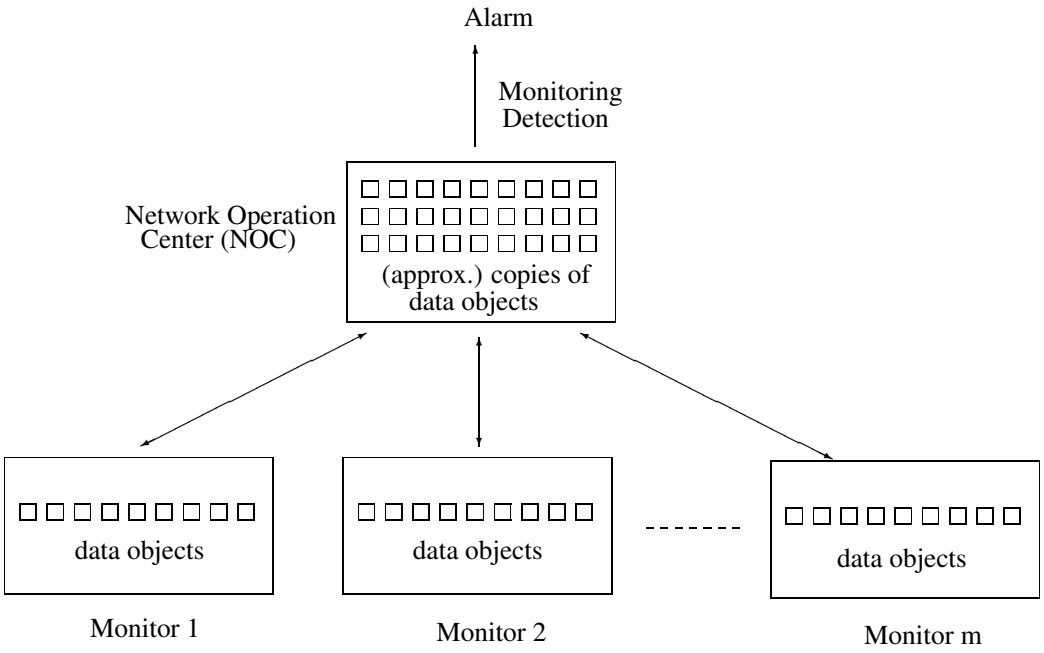


Figure 1.2: The abstract data management model.

provision inter-site connectivity). Although such measurement overhead may be supportable in today's ISPs, it may not in the future as we move towards petascale monitoring infrastructures that will monitor hundreds or thousands of network data features. The combined effect of increasing the number of monitors while simultaneously reducing the time scale of operation could lead to an explosion in the volume of data collected for this application.

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. Naïve solutions that continuously "push" the local data streams directly to a collection site simply will not scale to large distributed systems.

1.2.2 Efficient Data Management

The database community has done extensive research on approximate data replication protocols for efficient management of distributed and continuous data streams [4, 11, 47, 35]. Based on a one-level tree structure, they study the problem of how to efficiently and effectively facilitate centralized access to distributed data objects (i.e., a stream is a data object with values changing over time continuously). One typical way is to maintain copies of data objects of interest at the central NOC using an operation called replication, as illustrated abstractly in Figure 1.2. In a typical network monitoring system, the central NOC maintains replicas of data objects whose master copies are distributed across multiple remote and distributed monitors. Various information update and data synchronization protocols have been proposed to keep replicas synchronized to some degree with remote master copies using communication links between the NOC and each source. Keeping replicas exactly consistent requires propagating all master copy updates from monitors to the replicas at the NOC, which is infeasible or prohibitively expensive in many cases: data collections may be large or frequently updated, and network communication and computation resources may be limited due to the high communication costs incurred by this approach.

In many applications of network monitoring and detection systems, exact data is often not a requirement. It is a common practice to use approximate replication techniques, such as periodic refreshing or change/event-driven information updates, to reduce communication costs. In this case, the key issue is to study the fundamental tradeoff between the communication cost incurred for data synchronization and the degree of synchronization achieved. This characteristic property is referred to the cost-accuracy tradeoff, as shown in Figure 1.3. A particularly interesting tradeoff is to let users specify a minimum allowable accuracy level (i.e., fix an x-axis position in Figure 1.3), and

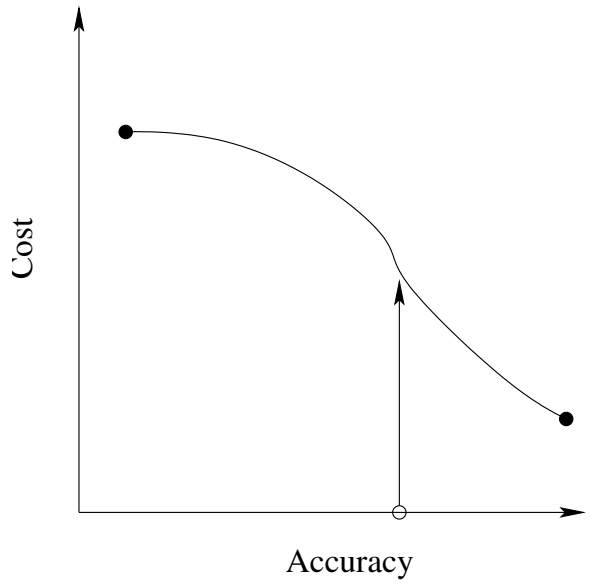


Figure 1.3: Tradeoff between cost and accuracy.

the data management system attempts to minimizing the communication cost while meeting the specified accuracy requirement.

Consider the *stream processing* problem of evaluating multiple *continuous queries* over distributed streamed data, which usually incurs significant communication overhead in the presence of rapid update streams. Olston *et al*. [47] proposed an adaptive filtering approach for reducing communication cost, by taking advantage of the fact that many applications do not require exact accuracy for their continuous queries. They allow users to submit a minimum accuracy level along with continuous queries to the stream processor, and the stream processor installs filters at the remote monitoring sites. The filters adapt to changing conditions to process data locally, so that they minimize communication cost while guaranteeing that all continuous queries still receive the updates necessary to provide answers of adequate accuracy at all times. In this way, users are offered fine-grained control over the tradeoff between query answer precision and communication cost.

1.2.3 Detection Instead of Approximation

One of the primary goals for a monitoring system is to ensure the target system performs well. For this goal, we do not need to record and aggregate global system state continuously at all times; instead, we want the monitoring system to detect and react in the event of occasional violations of system pre-defined constraints. Ideally, this detection and response should be achieved in a manner that remains both timely and efficient at scale. Stream processing protocols, however, are ill suited to this task. Existing stream processing approaches mainly focus on data approximation using simple linear aggregate functions. They always aim at collect, store and aggregate network status at the central NOC within an $\epsilon-$error bound. They are well suited to answering approximate queries and continuously recording system status for trend analysis. However, for detection purpose, they suffer from excessive query overhead in face of bursty data, partly because they always aim at ϵ-error approximation of system status regardless of actual system conditions. Always providing ϵ-error approximation of system status wastes resource for detection applications, which only care about 0-1 information (i.e., "normal" vs. "abnormal"). As shown later on in Chapter 3, this is unnecessary and incurs large communication cost for anomaly detection applications.

In addition, it is unclear how ϵ-error approximation can achieve a fine-grained tradeoff between accuracy and communication cost for detection applications, especially when sophisticated detection functions are used instead of simple linear function like `SUM`. This is one of the main research goals and achievements in this dissertation.

1.3 Overview of D-Trigger

Several recent research proposals suggest architectures for efficient large-scale monitoring systems [14, 15, 33, 58]. Their vision articulates the need for distributed tools that monitor overall system activity. Other recent work [36, 38] 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. Building upon this vision, we have designed D-Trigger as a general framework for efficient online detection by adaptively tracking system global conditions.

1.3.1 The System Setup

D-Trigger is designed as a distributed triggering system where continuous data aggregation is needed to gather timely information for continuous 0-1 queries that involve detection of an anomaly or constraint violation (*e.g.*, aggregate traffic, memory usage, or other parameter exceeding a threshold). Constraint violation may be defined in terms of different metrics, such as: instantaneously exceeding a threshold (*e.g.*, peak CPU load), exceeding a threshold over a time-varying window to detect an on-going heavy load or continuous burden on the system, or exceeding an average rate limit. These types of definitions can be used to support an IDS, provisioning decisions, performance/availability monitoring, etc.

As shown in Figure 1.4, our D-Trigger system consists of n distributed *monitors* and a central *coordinator* node. Monitors are distributed inside the network, each of which continuously produces time series signals on the monitored data. Users or applications register triggers at the coordinator via arbitrary threshold or anomaly detection functions. The coordinator is responsible

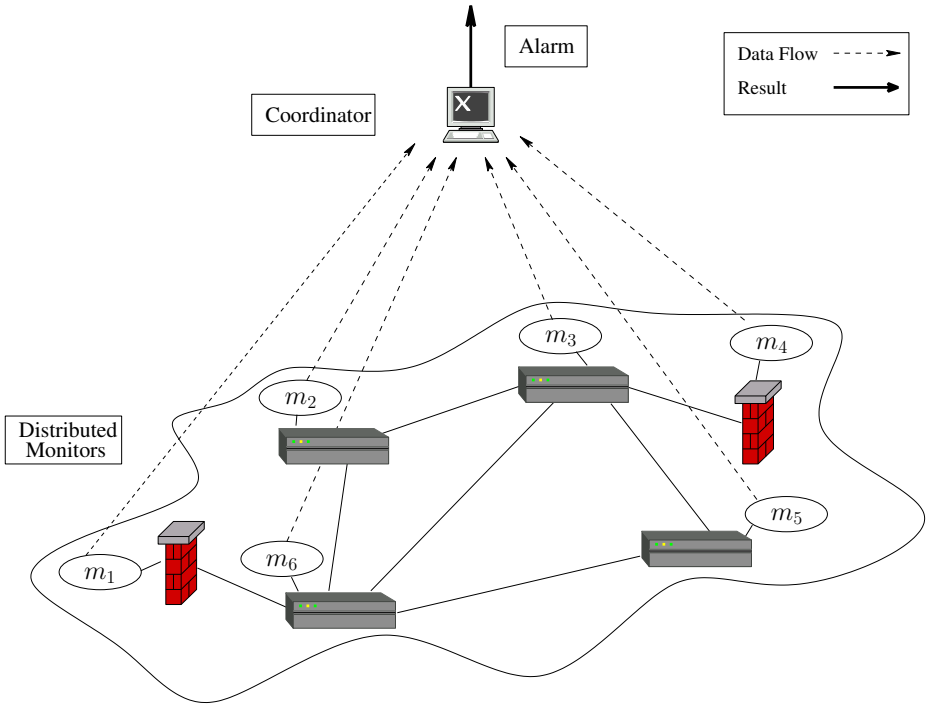


Figure 1.4: The system setup.

for continuous evaluation of the functions and firing a trigger if the aggregate (function-transformed) signal of the monitors exceeds a pre-specified threshold (See Fig. 1.4). Although the detection functions are specified at the coordinator, they are defined on the distributed information collected and produced by monitors. So the coordinator needs timely information updates from monitors to continuously track the detection functions. It also give monitors feedback and guidelines on how and when to do information updates.

1.3.2 The Problem and Key Challenges

Our work on D-Trigger focuses on collaborative anomaly detection, involving large sets of distributed monitors across large network systems. Anomaly detectors are perfect candidates for achieving a communication-efficient solution if done properly. Anomalies are rare events in a well-

performing system, so most of the time the monitoring data should be normal and the coordinator does not need a detailed view of what is happening in the system. It is not necessary for monitors to send updates of every bit of information to the coordinator because the anomaly detectors do not need most of it anyway. Only when things start to become questionable, does the coordinator need a more precise view of system status.

However, in a dynamic environment where system status change rapidly, it is not a easy task for the detection center to figure out when and how much information is needed to fire the trigger at the user required accuracy level. Naïvely designed solutions easily incur large communication cost in order to achieve the desired detection goal. The key challenges that D-Trigger faces can be summarized into the following two aspects:

1. Because monitoring applications are inherently continuous and distributed, algorithms and protocols in D-Trigger should be designed to minimize the communication overhead that it introduces, while still guaranteeing user-specified detection accuracy. To achieve the guaranteed accuracy, the D-Trigger protocols need to find out which information is important, and when and how to update information efficiently; the D-Trigger algorithms need to configure protocol parameters based on concrete theoretic analysis, so that the coordinator can gets "just enough" information to guarantee the specified detection accuracy.

2. As a general framework for efficient online monitoring and detections, D-Trigger should go beyond simple triggers (see more descriptions in Section 1.4) to accommodate a broad range of anomaly detection algorithms. They are crucial for effective detection of various unusual events, including botnet attacks, volume anomalies on an ISP network, electric power grid anomalies, etc. The challenge is, with incomplete information, guaranteeing accuracy re-

quirements for linear detection functions is not easy, and to do so for sophisticated anomaly detection algorithms (e.g., PCA-method) is even harder!

To reduce the communication cost, D-Trigger engages distributed monitors in local information processing at the edge of the network. Monitors only send the processed data to the coordinator, which can be approximated or filtered versions of the original data. Because the coordinator has imperfect knowledge of the monitored data (it receives filtered versions), it can make mistakes — leading to possible *false alarms* and/or *missed detections*. The problem here is to design protocols for the monitors and coordinator such that the coordinator fires the trigger with high accuracy while simultaneously using as little communication between the monitors and coordinator as possible, so that fine-grained tradeoffs between detection accuracy and communication cost can be achieved.

The D-Trigger system gracefully integrates a variety of approximation and optimization algorithms to address the inefficiency and inflexibility in today's distributed monitoring and anomaly detection systems. D-Trigger is designed with a focus on data collection for anomaly detection, and brings together the best techniques from continuous data streaming, online machine learning, and distributed signal processing. D-Trigger combines in-network processing at distributed local sites, and decision making at the NOC. The combination of distributed local processing strategies, sophisticated detection algorithms, and theoretical analysis tools enables D-Trigger to perform in-network tracking with very high detection accuracy and low communication overhead.

1.4 Going Beyond Simple Triggers

These early efforts on distributed triggers focused solely on *instantaneous* aggregate trigger conditions [36, 38, 56], 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 they are a useful tools for several application scenarios, they are insufficient for sophisticated detection in many networking applications. One of the key contributions of this dissertation is to extend the simple trigger model to support more sophisticated trigger functions and thresholds. In this section, we discuss the whole problem space of using distributed triggers for online anomaly detection, and introduce two type of important triggers – cumulative triggers and extended triggers.

1.4.1 The Problem Space

The problem space of protocol and algorithm design for online anomaly detection applications is shown in Figure 1.5, and the taxonomy of the design space is formed by the following three orthogonal axises: violation type, trigger function and system topology.

1. For the violation type, there are at least three distinct types of trigger conditions. *Instantaneous* triggers must fire when an aggregate threshold value is violated within a single time instance. On the other hand, *fixed-window* and *cumulative* triggers aim to catch persistent threshold violations over a window of time. Cumulative triggers, in particular, monitor aggregate signals for potential threshold violations over a time window *of any size* in the past, which is explained in more detail in Section 1.4.2.

2. The trigger functions supported in a triggering system can range from simple linear functions

like `SUM, AVG, MIN, MAX` to sophisticated anomaly detection functions like Top-k, PCA and Support Vector Machine (SVM) classification, etc. Rich types of function support enable the system to perform detection for a variety of problems, ranging from load-balancing, botnet attacks, volume anomalies on in ISP network, to electric power grid anomalies, etc.

3. For the system topology, there are triggers defined on a one-level tree topology, in which every monitor communicates with the coordinator directly; there are triggers defined on a multi-level tree structure, where monitors have a parent-child relationship, and are organized into a hierarchical tree structure with the coordinator as the root of the tree; finally, one could imagine evaluating and tracking triggers in a purely peer-to-peer fashion. In this dissertation, we only focus on one slice of the whole problem space: designing solutions for triggers defined on a one-level tree topology (the shaded region in Figure 1.5). It is interesting future work to extend the algorithms and protocols designed for a one-level tree topology to multi-level tree and peer-to-peer topologies.

1.4.2 Cumulative Triggers

While 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

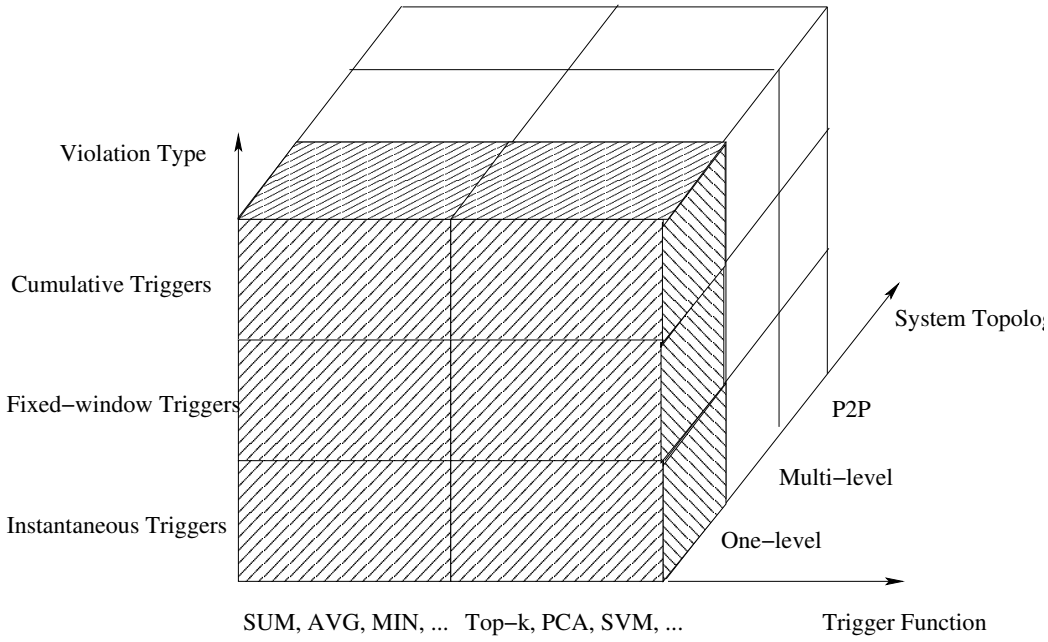


Figure 1.5: The whole problem space.

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 [36, 38]. 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.

1.4.3 Extended Triggers for Sophisticated Detections

Many existing online triggering and detection approaches have significant limitations: they only support simple thresholds on distributed aggregate functions like `SUM` and `COUNT`, which are insufficient for sophisticated detection in many networking applications. In this dissertation, we develop the support for more sophisticated detection functions for distributed triggers. We give some examples for detection schemes that may require checking non-linear threshold constraints, relative triggers, and so on. To illustrate these ideas more concretely, we use the example of a centralized PCA-based anomaly detection application [41]. This method detects anomalies in traffic volume levels by simultaneously examining the link load levels of all the links in a large network. It works by using a Principal Components Analysis (PCA) technique to decompose network traffic into normal and residual components, and then detects anomalies by applying a threshold function on the residual components. In this detection scheme, distributed monitors periodically ship all

observations to a central Network Operations Center (NOC), which in turn assembles and analyzes the data to perform anomaly detection.

However, such a "periodic push" approach suffers from scalability limitations. It is a central premise of this work that backhauling all distributed monitoring data may be unnecessary, depending upon the particular monitoring task, and thus smart data-filtering or data-reduction at the local monitoring sites should be employed. This approach would enable distributed monitoring systems to scale more gracefully both as the number of monitors increase and as the time scale for data collection and anomaly detection decreases. The promising effectiveness of the Lakhina, *et al.* technique provides strong motivation for designing a significantly more communication-efficient PCA-based scheme for real-time anomaly detection.

1.5 Contributions and Thesis Organization

1.5.1 Contributions

This dissertation work focuses on algorithm design and system development for D-Trigger, a general framework for efficient online monitoring and anomaly detection. It proposes a system framework for in-network anomaly detection by applying signal processing and machine learning techniques to analyze and model distributed data streams. The key accomplishments throughout the whole dissertation work have been to: 1) enable near real-time detection where the system's state is tracked continuously, so even the smallest anomalies will be exposed; 2) significantly reduce the data collected for anomaly detection, thus reducing the communication burden placed on the network; 3) guarantee desired detection accuracy even with the reduced amount of collected data. With the three achievements, this dissertation work has resulted in an efficient detection system that

is capable of detecting a wide-range of anomaly types in distributed systems in near real-time with bounded detection error. D-Trigger is a general purpose framework that can be applied to a wide variety of monitoring problems and domains, ranging from simple monitor functions (e.g., SUM, AVG, MIN, and MAX) to complex mathematical functions (see below), and spanning areas such as sensor networks, enterprise networks, and even power distribution networks.

Online Tracking of Distributed Triggers

We design and develop a general detection framework for efficient online detection with Distributed Triggers (D-Trigger), which makes the following contributions. First, we provide a mathematical definition of the distributed triggering problem with different constraint violation modes, along with the design of the supporting protocols. Our system enables users to tradeoff desired detection performance with communication overhead. Second, for cumulative triggers, we provide a principled queuing framework for analyzing the dynamic properties of protocols, and analytical solutions for finding effective queue sizes based upon the target detection accuracy. Third, for instantaneous triggers, we provide an adaptive protocol that exploits the specified trigger threshold to minimize communication while offering deterministic accuracy guarantees. Finally, we evaluate our schemes on real-life distributed data streams collected from PlanetLab IDS monitors, and clearly demonstrate the significant communication-efficiency gains that our algorithms can achieve. In all of our testing, the amount of monitor data that needed to be sent to the coordinator varied from 4% to 20% depending upon the test case, thus reducing the communication burden of the original signals by over 80%. We also significantly outperform previous streaming solutions for the instantaneous trigger case, particularly in regions where only very low errors can be tolerated.

Toward Sophisticated Online Detection

We propose a novel approach to extending simple threshold triggers for sophisticated anomaly detection problems. Through a set of examples, we show that D-Trigger is an efficient and extensible vehicle for advanced detection algorithms, and discuss our general extensions to existing triggering protocols to support wide-range of detection tasks. In particular, with certain assumptions, we show that D-Trigger can be extended to support distributed protocol for performing online detection of network-wide anomalies with modest communication overhead.

Online Detection of Network-Wide Anomalies

Based on the D-Trigger framework, we propose a novel approach for communication-efficient online detection of network-wide traffic anomalies. Our proposed solution is unusual in that it combines the existing PCA-based method with in-network processing ideas with new insights based on Stochastic Matrix Perturbation (SMP) theory. We develop an approximation technique in order to reduce the amount of data needed for anomaly detection. Having incomplete monitoring data can lead to errors that propagate through the computation, and this results in an anomaly detector that can make mistakes. We make use of SMP theory to derive analytic bounds on each of the terms affected by error propagation. We design an algorithm that derives filtering parameters for the monitors, such that the errors made by the detector are bounded. Our algorithms allows users to input a target error rate which gives them control over the tradeoff between communication cost and detection accuracy.

Our evaluation using real-world data streams collected from a well known ISP network shows that our methods work very well. While sending less than 10% of the original time-series

data (over an order of magnitude communication reduction), we guarantee that the detection error would be no more than 4% bigger than when using full data. In fact, our system performs much better than these bounds; we find that the actual error rates are nearly indistinguishable from the full data method. This results in a huge savings in communication overhead (e.g., typically 80 or 90% of the original data is no longer sent) with only a very small impact on errors.

1.5.2 Organization of Dissertation

The remainder of the thesis is organized as follows: We begin in Chapter 2 by providing context and discussing related work in distributed monitoring and anomaly detection systems. We then discuss the general framework of D-Trigger, and describe detailed protocols and algorithms for efficient tracking of distributed triggers in Chapter 3, followed by extended algorithms and mechanisms for tracking sophisticated triggers in Chapter 4. In Chapter 5 we present the PCA-based method in detail and give a comprehensive description of how to use D-Trigger framework to continuously detect network-wide traffic anomalies. Finally, we summarize lessons learned, outline future work and conclude in Chapter 6.

Chapter 2

Related Work

In this chapter, we summarize related work in both the networking and database domains, focusing on network monitoring and intrusion detection methods, streaming protocols for distributed query processing, and online data mining and decentralized machine learning algorithms.

2.1 Network Monitoring and Intrusion Detection

Network intrusions and large-scale attacks happen on the Internet daily, and range from stolen or corrupted data, widespread denial-of-service attacks, to disruption of essential services. Protecting networks from intrusions and attacks is challenging, and current best practice for protecting against intrusions is through the use of firewalls or network intrusion detection systems (NIDS) [44], which are usually placed at the edge of networks. Firewalls are choke points that filter traffic at network gateways based on local security policies [7]. NIDS passively observe the local network traffic and react to specific signatures (misuse detection) or statistical anomalies (anomaly detection). Examples of NIDS that employ misuse detection are Snort [54] and Bro [49].

Snort [54] is capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, amongst other features. The system can also be used for intrusion prevention purposes, by dropping attacks as they are taking place.

Bro [49] is an open-source intrusion detection system developed by Vern Paxson *et al*. Similar to SNORT, it passively monitors network traffic and detects suspicious activity by comparing network traffic against a set of rules describing attack signatures. Bro detects intrusions by first parsing network traffic to extract its application-level semantics and then executing event-oriented analyzers that compare the activity with patterns deemed troublesome. Its analysis includes detection of specific attacks (including those defined by signatures, but also those defined in terms of events) and unusual activities (e.g., certain hosts connecting to certain services, or patterns of failed connection attempts).

However, a misuse-detection based NIDS is incapable of detecting new types of intrusions. This fundamental weakness comes from their detection relying on existing signatures. For detection of new types of intrusions, anomaly detection techniques have been proposed [24], which establish statistical profiles of network traffic and flag any traffic deviating from the profile as anomalous. For example, a large component of the work on machine learning, signal processing and time-series analysis is devoted to detecting outliers or anomalies in time-series [6, 10, 31, 40, 61]. These methods range in sophistication from [10], which suggests the use of the standard Holt-Winters forecasting technique for network anomaly detection, to [6], which uses a sophisticated wavelet based method with great potential. It is interesting future to pursue a distributed wavelet

method for anomaly detection.

The high variability common in network traffic limits the effectiveness of anomaly detection techniques. In general, current NIDS suffer from two major drawbacks: high false alarm rates and perspective from a single vantage point, which limits their ability to detect distributed or coordinated attacks. One promising approach to addressing the abovementioned shortcomings is through the use of distributed and coordinated intrusion detection techniques. In this environment, measurements and alerts from different NIDS are combined and correlated to address above-mentioned shortcomings.

2.2 Coordinated Monitoring and Detection

Recent progress in network monitoring, profiling and intrusion detection [41, 48, 64, 67] aims to share information and foster collaboration between widely distributed monitoring boxes to offer improvements over isolated systems. Specifically, a number of techniques have been proposed to detect network traffic anomalies by correlating measurements from multiple vantage points. However, most existing approaches for network-wide anomaly detection mainly focus on off-line diagnosis instead of continuous online detection. These systems provide other examples of distributed monitoring applications for which our triggering tools would be useful.

Yegneswaran *et al*. [67] explored the possibility of improving detection speed and accuracy of isolated systems by sharing data between widely distributed intrusion detection systems. They proposed DOMINO as an architecture for a distributed intrusion detection system that fosters collaboration among heterogeneous nodes. Leveraging active-sink nodes which respond to and measure connections to unused IP addresses, and organizing them into overlay network, DOMINO

is capable of performing efficient detection of attacks from spoofed IP sources, reduces false positives, and enables attack classification and production of timely blacklists. This key observation motivates a set of work to develop distributed data mining techniques for coordinated anomaly detection using network-wide information.

Lakhina *et al*. [41], carried out pioneering work in detecting network-wide anomalies by correlating observations from multiple vantage points together. They proposed an anomaly detection scheme in which monitors ship observations to a central Network Operations Center (NOC), which in turn assembles and analyzes the data to perform anomaly detection. Specifically, they propose that local monitors continuously measure the total volume of traffic (in bytes) on each network link, and periodically push all recent measurements to the NOC. The NOC then performs Principal Component Analysis (PCA) on the assembled data matrix to reveal traffic anomalies that were not detectable in any single link-level measurements. Lakhina, *et al.* demonstrate that this technique is quite effective in detecting anomalies in traffic, in part due to the inherently low-dimensional nature of the underlying data. One of our contributions is to illustrate how to redesign this application in a distributed fashion, with far less communication overhead, by using distributed triggers as an underlying paradigm. Abstractly, this requires distributed triggers that support complex threshold functions, and can be composed to lend support to sophisticated distributed detection applications.

Zhang *et al*. [69], extended it further and proposed a general "anomography" framework to infer network-level anomalies in both spatial and temporal domains. They tackled the problem of inferring anomalies from indirect traffic measurement. With clear separation of traffic inference and anomaly detection, they proposed a framework and developed a whole families of new algorithms for network-wide anomalies detection, based on ARIMA modeling, the Fourier transform, Wavelets,

and Principal Component Analysis. They introduced a new dynamic anomography algorithm, which can effectively track routing and traffic changes, to alert with high fidelity on intrinsic changes in network-level traffic. It is interesting and challenging future work to redesign these timeseries analysis algorithms in a distributed online fashion for coordinated anomaly detection.

Network architecture support for large-scale coordinated monitoring and detection systems has been an important research field in recent years [14, 15, 33, 58]. One of the main focuses is how to efficiently deliver timely and relevant data about the state of the system to all the dispersed components of the system. Their vision articulates the need for distributed tools that monitor the overall activity of the system with efficient data collection and management techniques.

2.3 Continuous Query Processing and Triggering

Database research on continuous distributed query processing has considered efficient data management for coordinated monitoring systems. For example, a set of stream processing protocols [12, 16, 20, 35, 47] aim at "$\epsilon-$error" approximation of the aggregate signal computed from the continuous and distributed data streams. However, these early efforts always focused on the accurate estimation of the aggregate signal itself rather than catching a constraint violation. They are well-suited for answering approximate queries and continuously recording system state.

In a distributed online setting, the work of Olston *et al*. [47] shares many similarities with ours. They considered an environment where distributed data sources continuously stream updates to a centralized processor that monitors continuous queries over the distributed data. To reduce the communication overhead, they developed a system that allows users to register continuous queries with accuracy requirements at the central stream processor, which installs filters at remote data

sources; the filters adapt to changing conditions to minimize stream rates while guaranteeing that all continuous queries still receive the updates necessary to provide answers of adequate accuracy at all times. The proposed approach provides applications the tradeoff between approximation accuracy and communication overhead at a fine granularity. However, they only considered the instantaneous version of the problem and focus on aggregate approximation rather than accurate triggering.

Jain *et al*. [35] extended the filtering-based approach further. They recast query processing on distributed streaming data as a filtering problem, in which the objective is to filter out as much data as possible to reduce communication overhead, while still meet the accuracy standards specified by applications. They leveraged the Kalman Filter as a general and adaptive filtering solution for conserving resources, and realized a significant performance boost by switching from caching of static data to caching dynamic procedures that can predict data reliably at the centralized processor. The Kalman Filter method is very powerful in smoothing data (thus reducing communication cost) for streams with strong temporal correlation. However, it still remains unclear whether this method is applicable to detection problems, because necessary spiky anomalies might be smoothed out by the Kalman Filter.

In order to maintain system-wide invariants and catch unusual constraint violations, the database community has explored centralized triggering mechanisms [27, 63]; however, the goal of minimizing communication overhead in widely distributed environments introduces new challenges.

Jain *et al*. [36] argued convincingly that a critical component missing from network monitoring architectures is distributed triggering. They proposed to use 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 [22] studied the problem of detecting whether the sum of a set of numeric values from distributed sources exceeds a user-supplied threshold value. They proposed and analyzed novel techniques which allow to significantly reduce the amount of monitoring related traffic, based on a combination of aperiodic polling and asynchronous event reporting. They demonstrated how the combination of a central monitoring algorithm, with simple local constraint verification, can be used to save a significant amount of the monitoring overhead.

More recently, Keralapura *et al*. [38] proposed solutions to detect threshold violations on sum functions with specified accuracy while minimizing communication overhead. They study the "thresholded counts" problem in a monitoring environment where they must return the aggregate frequency count of an event that is continuously monitored by distributed nodes with a user-specified accuracy whenever the actual count exceeds a given threshold value. They explored algorithms with both static thresholds and adaptive thresholds algorithms, which involve setting local thresholds at each monitoring node and initiating communication only when the locally observed data exceeds these local thresholds.

To monitor distributed data streams with sophisticated functions, Sharfman *et al*. [56] proposed a novel geometric approach by which an arbitrary global monitoring task can be split into a set of constraints applied locally on each of the streams. The constraints are used to locally filter out data increments that do not affect the monitoring outcome, thus avoiding unnecessary communication. As a result, their approach enables monitoring of arbitrary threshold functions over distributed data streams in an efficient manner. However, their method assumes preset thresholds,

does not consider global matrix analysis queries, and cannot scale to large networks with high-speed data streams. Our approach goes further by providing both a firm detection guarantee for a wide range of trigger functions and conditions, as well as the flexibility for users to trade off communication overhead with detection accuracy.

2.4 Decentralized Data Mining and Machine Learning

In many networking environments, applications have to deal with different distributed sources of voluminous data, multiple compute nodes, and distributed user community. Analyzing and modeling data from distributed sources require data mining techniques designed for distributed applications.

For the PCA method, [5] and [53] proposed distributed algorithms to compute principal components of data matrices distributed across blocks of rows or columns. [53] used truncated singular value decompositions (SVDs) in the distributed locations to reduce communications costs. This approach is very effective when the local data matrices have low ranks and can be accurately approximated via a truncated SVD. Unfortunately, truncated SVDs introduce local approximation errors that could add up and adversely affect the accuracy of the final PCA. [5] introduces a new method to compute the PCA on a distributed block matrix without incurring local approximation errors, as well as updating the PCA when new data arrive at the various locations. However, these methods assume that a block of data are available at each local site, and are not applicable to our case where only a single column of data are available at each local site. Furthermore, neither [5] and [53] address the issue of continuously tracking principal components within a given error tolerance or the issue of implementing a communication/accuracy tradeoff; issues which are the main focus of

our work.

Data clustering is one of the fundamental techniques in scientific data analysis and data mining. It partitions a data set into groups of similar items, as measured by some distance metric. For large and distributed datasets, Forman *et al*. [18] proposes distributed clustering algorithms to reduce communication overhead and computation time. With a focus on one-shot clustering with statistic data, their central idea is to communicate only sufficient statistics, which yields linear speed-up with excellent efficiency. Cormode *et al*. [18] goes further to study the problem of maintaining a clustering of distributed data that is continuously evolving, with focus on minimizing the communication and computational cost while still providing guaranteed accuracy of the clustering. However, their method requires a pre-specified cluster number, and employs a deterministic data model for clustering. It is interesting but challenging future work to extend their method to handle noisy data without specifying a cluster number, using probabilistic data model [8].

Recent work in the machine learning literature considers distributed constraints in learning algorithms such as kernel-based classification [66] and graphical model inference [39]. (See [52] for a survey). In [66], Nguyen *et al*. considered the problem of decentralized detection under constraints on the number of bits that can be transmitted by each monitor. Without assuming the joint distribution of monitor observations to be known, they addressed the problem of when only a set of empirical samples is available. They proposed a novel algorithm using the framework of empirical risk minimization and marginalized kernels and analyzed its computational and statistical properties both theoretically and empirically. It will be of interest for future work to incorporate this approach into our D-Trigger framework to enable sophisticated online detection via classification methods.

Chapter 3

Online Tracking of Distributed Triggers

As introduced in Chapter 1, distributed triggering is a critical component in distributed monitoring and detection systems that can efficiently fire alarms when a global condition across a set of distributed machines exceeds acceptable levels. In this chapter, we describe the design, implementation and evaluation of D-Trigger, our general framework for distribution detection and triggering. We use `SUM` as the detection function in this chapter, and as shown later on in this thesis, the D-Trigger framework can be easily extended to support other sophisticated (nonlinear) detection functions.

Our D-Trigger system consists of n distributed *monitors* and one *coordinator* node. Monitors continuously produce time series signals on the monitored data, and send filtered versions of their signals to the coordinator, which is responsible for firing a trigger if the aggregate signal of the monitors exceeds a pre-specified threshold (See Fig. 1.4).

Work in the database community on data streaming has considered similar environments [12, 16, 20, 35, 47]. However, a critical difference between our work and these is that these efforts

are focused on accurate estimation of the aggregate signal itself. Conversely, we aim to fire the trigger accurately. This means that when our aggregate signal is far from the constraint violation, the coordinator can tolerate receiving less accurate updates from the monitors, which in turn lowers communication overhead. Only when the global state starts approaching the trigger condition do we need accurate information about the time series data. This key insight enables us to achieve a far greater amount of communication reduction than previous methods. We thus employ adaptive control based on how far we are from the triggering condition. We also exploit other avenues for overhead reduction: adapting to changing data statistics in the time series themselves and exploiting cancellation of data variations across different monitors.

Our primary goal in D-Trigger is to extend distributed monitoring systems to enable *continuous* tracking for anomaly detection applications while still allowing for a scalable system architecture. We believe that many anomaly detection applications can be successful without collecting *all* of the monitored data. Because our applications are focused on anomaly detection, we point out that most of the time the traffic will be "normal" and there is no need to send such data to the fusion center. The goal is thus to send only the monitoring data that is "needed" for the anomaly detector to work properly. To reduce the amount of data transmitted through the network, we install local triggers on the distributed monitors which allow the monitors to only send updates to the data fusion center when the local trigger fires.

Our system supports three distinct types of distributed triggers to enable a broad range of triggering conditions. *Instantaneous* triggers must fire when an aggregate threshold value is violated within a single time instance. On the other hand, *fixed-window* and *cumulative* triggers aim to catch persistent threshold violations over a window of time. Cumulative triggers, in particular,

monitor aggregate signals for potential threshold violations over a time window *of any size* in the past. To the best of our knowledge, our work is the first to address the detection of a constraint over such varying-window sizes. Moreover, our cumulative trigger conditions pose novel algorithmic problems that have not been addressed in earlier work on data streaming.

We design a solution for cumulative distributed triggers based on ideas from *queuing 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*. Selection of these queue sizes affects the resulting amount of communication overhead as well as the resulting false-alarm and missed-detection rates. We develop an analytical solution to determine the queue sizes based on user supplied target error rates for false alarms and missed detections. Large reductions in communication bandwidth used by these systems is of paramount importance for scaling. To avoid overwhelming a single coordinator when the number of monitors grow, it is critical that each monitor be able to limit the information it sends to a coordinator.

In summary, our first-step work toward a general detection framework for D-Trigger makes the following contributions. First, we provide a mathematical definition of the distributed triggering problem with different constraint violation modes, along with the design of the supporting protocols. Our system enables users to tradeoff desired detection performance with communication overhead. Second, for cumulative triggers, we provide a principled queuing framework for analyzing the dynamic properties of protocols, and analytical solutions for finding effective queue sizes based upon the target detection accuracy. Third, for instantaneous triggers, we provide an adaptive protocol that exploits the specified trigger threshold to minimize communication

while offering deterministic accuracy guarantees. Finally, we evaluate our schemes on real-life distributed data streams collected from PlanetLab IDS monitors, and clearly demonstrate the significant communication-efficiency gains that our algorithms can achieve. In all of our testing, the amount of monitor data that needed to be sent to the coordinator varied from 4% to 20% depending upon the test case, thus reducing the communication burden of the original signals by over 80%. We also significantly outperform previous streaming solutions for the instantaneous trigger case, particularly in regions where only very low errors can be tolerated. This shows that our idea of targeting trigger accuracy rather than aggregate signal estimation accuracy is powerful in reducing overhead. We also show that our system can perform well in general when there is little error tolerance.

Chapter Organization. The rest of the chapter is organized as follows: we define the problem and evaluation metrics in Chapter 3.1; we discuss our general framwork and approach in Chapter 3.2; we present solutions for three different triggers in Chapters 3.3–3.5; we evaluate the approach in Chapter 3.6; finally, we conclude in Chapter 3.7.

3.1 System Model and Problem Statement

As shown in Fig. 1.4, 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. 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. For example, if a number of machines had been compromised and were participating in a botnet, then the number of TCP connections per second coming from them could reach unacceptably high numbers. Our coordinator should identify this event, raise a flag and should indicate the machines involved. In general, such a coordinator can aggregate the incoming time series signals using any typical aggregation function, such as `SUM, AVG, MIN, MAX`, etc. In this chapter we focus on linear aggregators, using the `SUM` aggregator throughout the chapter as our main example. With this aggregator, the goal of the coordinator is to fire a trigger whenever the sum of the time series signals exceeds a preselected threshold.

In this dissertation, we assume that: 1) all communication happens only between monitoring nodes and the coordinator, and no communication happens among monitoring nodes; 2) all monitoring nodes want to send as less data to the coordinator as possible in order to reduce communication overhead; 3) there is no loss in the network, and 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 [47].

If all the monitors sent their time series signals continuously, 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, sending all the monitored signals all the time is extremely costly in terms of

communication overhead.

Our intent here is to enable the coordinator to fire its triggers 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.

3.1.1 Types of Constraint Violations

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. We now define three distinct types of violations.

Instantaneous Violation. Here, we consider threshold violations occurring in a single time instant t. More formally, we require our trigger to fire a violation for any time instant t such that $\sum_i^n r_i(t) > C + \epsilon$. Fig. 3.1, illustrates a sample aggregate signal as it evolves over time, along with the corresponding error-tolerance zone $[C, C + \epsilon)$. An instantaneous violation is shown at time slot 5. Let $V(t, 1)$ denote the size of the violation (also called a *penalty*) at time t; it is defined to be

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

Thus, the goal of the system is to fire the trigger whenever the penalty exceeds ϵ, namely when $V(t, 1) > \epsilon$.

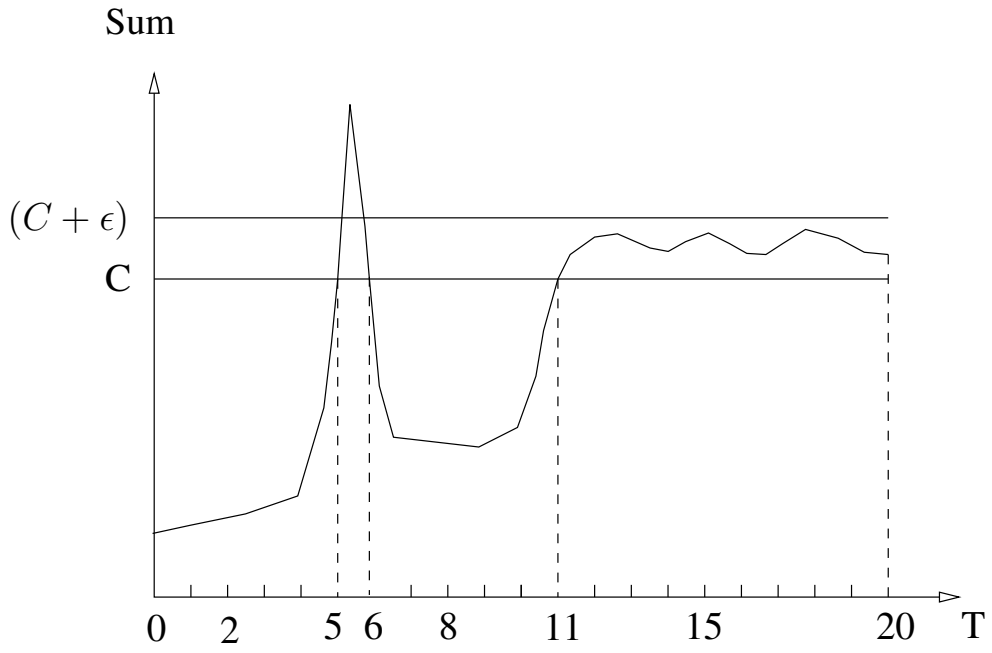


Figure 3.1: Threshold, error tolerance and violations.

Fixed-Window Violation. A threshold violation can also be defined as persistent cumulative violation of a threshold condition over a given window of time. Consider again the example in Fig. 3.1: During time slots 11 to 20, the signal remains in our error tolerance zone. However, if this situation persists over a long period of time, we may very well want to flag this as a violation. The coordinator can achieve this by computing a violation penalty that accrues over time, and fires the trigger condition when the penalty becomes excessive. Given a fixed time-window size τ, the penalty at time t accrued over the interval $[t - \tau, t]$ is defined to be

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

(We maximize this term with zero to keep the penalty non-negative.) Thus, with a fixed-window

trigger (with window size $= \tau$) our goal is to fire the trigger at any time t such that $V(t, \tau) > \epsilon$.

Cumulative Violation. In cumulative triggers, the threshold condition is defined in terms of the accumulated excess *area* of the aggregate signal over time: (bytes $\times$ time) or (number of connections $\times$ time). Abstractly, a cumulative trigger condition should fire when the excess area of the observed aggregate signal over a time window *of any size*, exceeds the pre-specified cumulative threshold. Such cumulative triggering conditions cannot be captured using existing `SUM`-trigger mechanisms based on instantaneous sums of local values [36, 38].

To capture temporally-persistent phenomena, we define a violation in terms of the accumulation of excess *area* of the underlying signal over windows of time. The coordinator computes a violation penalty that accrues over time, and fires the trigger condition when the penalty becomes excessive. During a time window of size $\tau = \tau(t)$, the penalty at time t accrued over the interval $[t - \tau, t]$ is defined to be

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

(We maximize this term with zero to keep the penalty non-negative.) Our cumulative triggering mechanism does not depend on any fixed window τ; instead, a cumulative trigger fires at time t if penalty $V(t, \tau) > \epsilon$ *for any* window size $\tau \in [1, t]$. Thus, intuitively, we fire the trigger if there is *some time window* that causes the cumulative penalty to exceed the ϵ constraint; or, more formally, if $\max_\tau\{V(t, \tau)\} > \epsilon$, where $\max$ is computed on all possible τ over the entire signal history. One of our key insights in this work is that, by exploiting an analogy to queuing theory, our system can track cumulative trigger conditions effectively, without having to retain the entire signal history or check the condition against all possible τ.

We allow the user or network operator to specify an error tolerance ϵ which indicates that

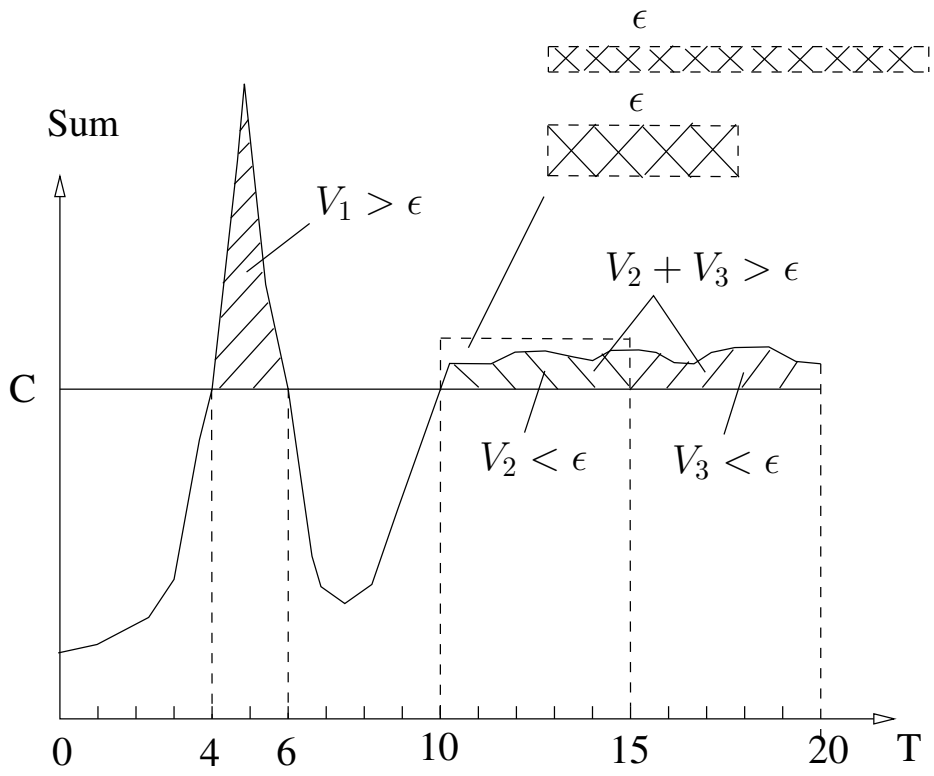


Figure 3.2: Cumulative violations.

it is sufficient to track the global state (i.e., the aggregated time series) approximately with an error bounded by ϵ. We will exploit this error tolerance to gain additional savings in communication overhead reduction.

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. 3.2, the persistent (but small) violation occurring in time

slots $[10, 20]$ could go undetected with a window size of $\tau = 5$ because the penalty (over any 5 time slots), V_2 or V_3, does not grow to exceed ϵ. If, on the other hand, the violation were short in duration but large in magnitude, the system would miss it if a large τ (long window) is used. In our example figure, a short but large violation occurs during the time period $[4, 6]$. With a window of size 5 time units, this violation is likely to get averaged out because the positive penalty in period $[4, 6]$ is canceled out by the negative contribution in period $[3, 4]$ (or, $[6, 7]$). This illustrates the difference between fixed sized windows and cumulative violations with varying window sizes. With a fixed window of size 5, both these violations would have been missed. However, with cumulative conditions, a window of size 10 would have caught the violation in $[10, 20]$ in our first example since the penalty $V_2 + V_3$ exceeds ϵ; a window size of 2 would detect the violation in $[4, 6]$ in our second example since the penalty V_1 does exceed ϵ. We can thus see the flexibility in not having to specify a priori the time window over which a potential violation is measured. Our cumulative triggering mechanism can thus capture a wider variety of persistent violation scenarios while avoiding the pitfalls of fixed time granularities. In several application scenarios, it is important to detect both types of violations regardless of the specific time window in which they occur.

3.1.2 Problem Statement

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

In our system, a *missed detection* occurs if $\max_\tau\{V(t, \tau)\} > \epsilon$ and the system does *not* fire the corresponding trigger. Conversely, a *false alarm* occurs whenever $\max_\tau\{V(t, \tau)\} \leq \epsilon$ and

the system fires a trigger. We define the *missed-detection rate* β as the fraction of missed detections over the total number of real violations, and the *false-alarm rate* η as the fraction of false alarms over the total number of triggers fired. Allowing β and η to be inputs, creates a flexible system in which different deployments can be tailored to their own needs. For example, some systems may consider minimizing false alarms more important than minimizing missed detections; other systems may take the opposite view.

The 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. The goal we have herein is to guarantee the trigger fires with (ϵ, β, η)-accuracy while simultaneously keeping communication overheads low.

Based on the violation types discussed earlier in this section, we now define three key instances of our triggering problem that we address in the remainder of this paper.

• **Problem 1: Instantaneous Trigger**: A system that guarantees a trigger will fire whenever the instantaneous violation $V(t, 1) > \epsilon$ occurs, with accuracy (ϵ, β, η), and using minimal communication overhead.

• **Problem 2: Fixed-Window Trigger**: Monitor and coordinator protocols that trigger if the penalty $V(t, \tau) > \epsilon$ occurs at *any time* t, *for a fixed* τ, with accuracy (ϵ, β, η), and using minimal communication overhead.

• **Problem 3: Cumulative Trigger**: Monitor and coordinator protocols that trigger if the

penalty $V(t,\tau) > \epsilon$ occurs at *any time* t, *for any* τ, with accuracy (ϵ, β, η), and using minimal communication overhead.

Within the D-Trigger framework, local monitors will do *continuous* filtering, and thus our coordinator also does *continuous* detection at any time t. If an unusual traffic pattern were detected at one monitor, but not at others, the coordinator will receive an update from that one monitor (and nothing from the others), thus allowing it to detect the problem immediately.

The protocol at the monitor sites needs to enable filtering, the use of local triggers, and communication with the coordinator. The protocol at the coordinator needs to oversee the global data it receives, decide on the level of accuracy it needs, and communicate to the monitors how they should do their filtering. 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.

3.2 The General Framework of D-Trigger

This section provides an overview of several key elements of our novel approach to distributed trigger tracking. Fig. 3.3 depicts the building blocks of our system. As mentioned earlier, $r_i(t)$ denotes the actual time series observed at monitoring node i. We use $R_i(t)$ to denote 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 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 predic-

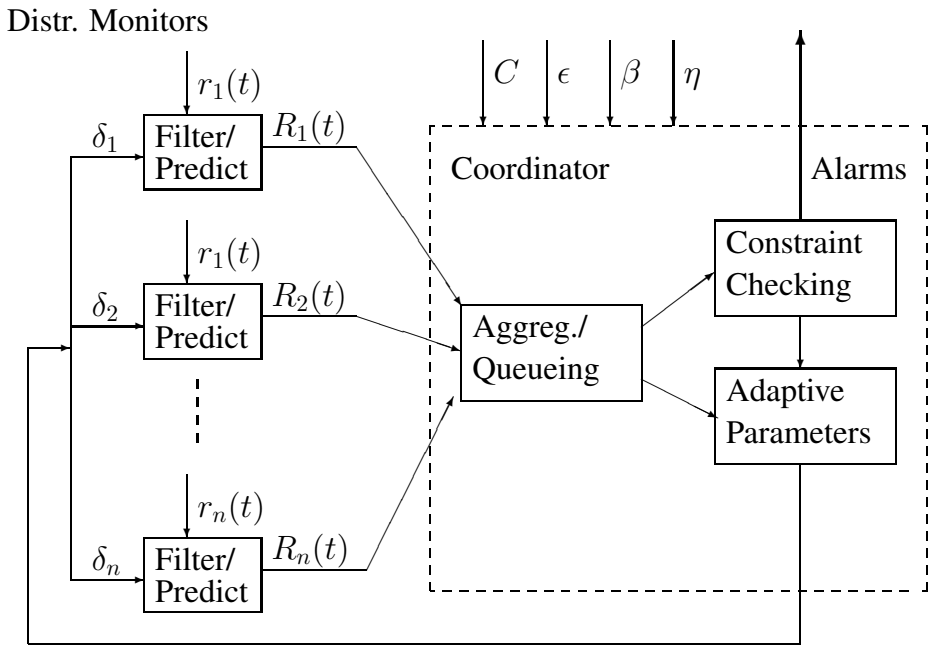


Figure 3.3: Our distributed trigger tracking framework.

tion models [16, 17] can be used. Our techniques remain applicable regardless of prediction-model specifics.

The key idea here is that, at any time t, $\sum_{i=1}^{n} R_i(t)$ captures the coordinator's view of the global state. On the other side, 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. Since monitors can track their own data continuously and send an update to the coordinator at *any* time (e.g., not on 5 minute boundaries), we can achieve continuous global tracking. Meanwhile, the coordinator continuously monitors its up-to-date global

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, $\Delta = \sum \delta_i$	Local and global monitor slack parameters
θ	Coordinator slack parameter
β	Miss detection (*i.e.*, false negative) rate
η	False alarm (*i.e.*, false positive) rate

Table 3.1: *Notation.*

prediction to ensure that it stays below the required trigger threshold (*e.g.*, $\sum_{i=1}^{n} R_i(t) \leq C$ in the instantaneous case), and fires the trigger when that condition is violated. We use $\Delta = \sum_{i=1}^{n} \delta_i$ to denote the *global monitor slack*. Table 3.1 summarizes some of the key notational conventions used in our discussion.

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) determining the amount of global slack Δ, and (2) splitting this global slack into individual local slacks δ_i for each monitor. In our system, both the global and local slacks can vary over time and, thus, are allocated in an *adaptive* manner in order to effectively maximize the effect of local filtering, and thus, to minimize overall communication. Note that, unlike earlier work in the data-streaming literature [20, 35, 47], we are *not* interested in maintaining a guaranteed ϵ-error aggregate at the coordinator at all times. While such a constraint trivially solves our problem, it is also overly restrictive, given that we care about accurately estimating the aggregate signal *only if* its value is close to the trigger threshold C. 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 the C threshold. This

observation is one of the key motivations for building adaptivity into our distributed trigger monitoring system, and, as our results demonstrate, can lead to very significant savings in communication costs.

3.3 Instantaneous Triggers

Our algorithms for tracking instantaneous trigger violations follow along the framework discussed in Sec. 3.2. 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.*, [35, 47]).

The framework for instantaneous triggers is illustrated in Fig. 3.4, which has global trigger threshold, C, and error tolerance, ϵ as user specified inputs. The time series data collected at monitor M_i is given by $r_i(t)$, while $R_i(t)$ denotes the filtered version of this data sent to the coordinator. The idea behind the filtering is to send a summary of $r_i(t)$ at some time t and then not to send anything at all until it is deemed necessary. From the point of view of the coordinator, $R_i(t)$ can be viewed as a prediction of $r_i(t)$ because when the coordinator is not receiving any data from monitor i, it assumes that its most recently received value of $R_i(t)$ accurately predicts $r_i(t)$ [35].

Each monitor M_i continuously tracks the (instantaneous) difference $d_i(t) = |r_i(t) - R_i(t)|$ between the true local signal and its (most recent) prediction. In order to upper bound this drift, monitor i uses a parameter δ_i and checks if $d_i(t) > \delta_i$. If this occurs at a time t, then M_i updates the coordinator by sending it the latest $r_i(t)$ value. Although in general, $R_i(t)$ can be any prediction function of $r_i(t)$, in our examples herein we simply use $R_i(t) = r_i(t)$. The parameter δ_i is called the *local monitor slack*, and this parameter indicates that no communication is needed as

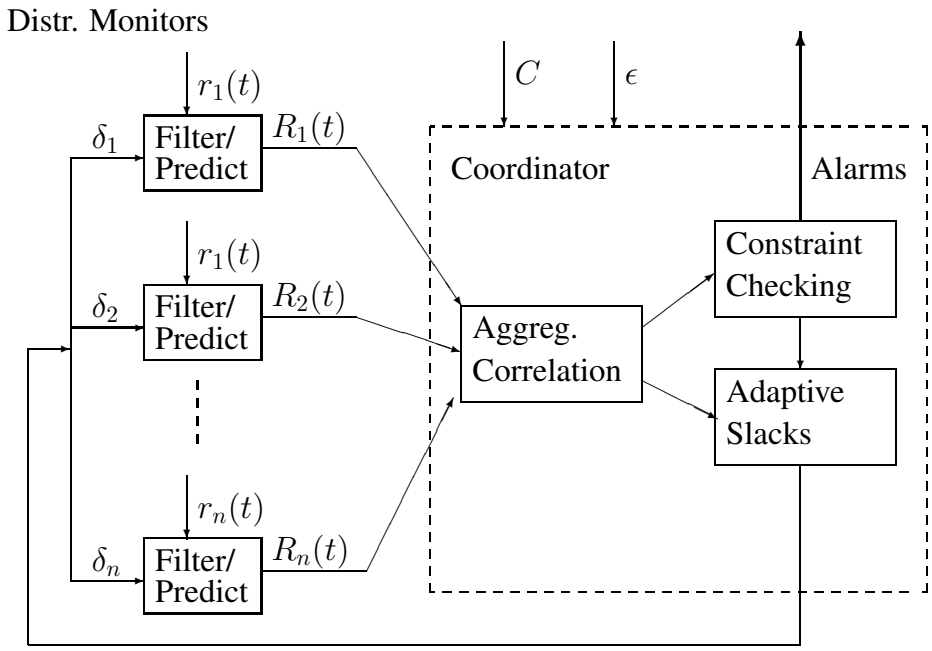


Figure 3.4: The tracking framework for instantaneous triggers.

long as the prediction captures $r_i(t)$ to within a δ_i bound.

The coordinator has two jobs. The first is to continuously track the value of the aggregation function based on the predictions it has received. Using the `SUM` as an aggregation function, this means tracking $\sum R_i(t)$. A violation occurs whenever $\sum_{i=1}^{n} R_i(t) > C + \epsilon$. The coordinator's second job is to continuously estimate a parameter $\Delta(t)$, called the *global monitor slack*, and partition this global slack into individual monitor slacks δ_i, such that $\sum_i \delta_i = \Delta$ and each δ_i is proportion to the variance of the local stream. These values are then sent to each of the monitors for local data filtering.

Based on the above protocols, it is not difficult to show (see, *e.g.*, [47]) 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 1 *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 1 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. 3.5. A straightforward application of the above theorem with $\Delta = \epsilon$ (essentially, directly applying the techniques of [35, 47]) would ensure that our algorithm tracks the instantaneous distributed trigger to within ϵ additive error (as discussed in Sec. 3.1). Similarly, the convex optimization algorithms of Olston et al. [47] (based on the idea of marginal gains) can be used to determine the optimal allocation of the (fixed) global 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. 3.2). We now discuss such an adaptive, threshold-based approach that provides the required ϵ-error guarantees — our experimental results in Sec. 3.6 clearly demonstrate the benefits of such adaptivity in practice.

Adaptive Instantaneous Trigger Tracking. The key idea of our adaptive scheme is quite simple. Unlike [35, 47], 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 [47]). (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

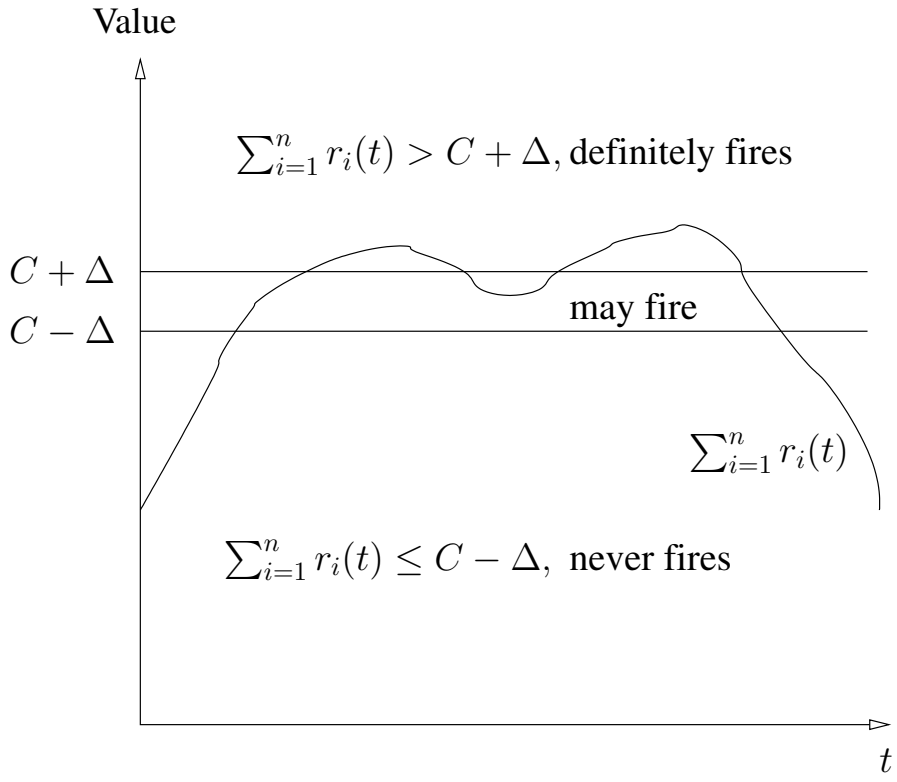


Figure 3.5: Instantaneous trigger tracking guarantees.

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 2 *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* $\sum_i r_i(t) < C - \epsilon$.

Theorem 2 asserts a "band of uncertainty" (of size 2ϵ) around the trigger threshold C, where our tracking algorithm may or may not fire a trigger violation (see Fig. 3.5). The key observation is that both global and local slacks vary over time, and can be allocated in an *adaptive* manner that maximizes the effect of local filtering, and thus minimizes overall communication. Unlike earlier data-streaming work [35, 47], we are *not* interested in continuously guaranteeing that the coordinator's estimate of the aggregate function is within an ϵ-error. Our focus instead is highly accurate trigger firing: we care about accurate aggregate signal estimation *only if* its value is close to the trigger threshold C. Our adaptive slack allocation schemes exploit the trigger condition to yield significant communication reductions by allowing for much "looser" (and thus, more effective) filters at monitors when the signal is well below the C threshold.

3.3.1 Heterogeneous and Adaptive Slack Allocation

Clearly, at any time instant t, Theorem (2) can be used at the coordinator to provide optimal settings $\Delta(t)$ for our system slack parameters. 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 $\Delta(t)$, as discussed in detail in this section.

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 [20, 47], in the case of a fixed global 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 global slack as well, and our implementation (discussed in Sec. 3.6) employs such a marginal-gains-based allocation scheme.

In this section, we briefly discuss non-uniform allocation of local monitor slacks, and discuss two simple schemes for filtering δ updates at the coordinator. They aim at limiting the sensitivity of the system to transient variations and the number of required δ-slack dissemination from the coordinator to the monitors.

Heterogeneous Local-Slack Allocation. One could use non-uniform slacks across monitors. Intuitively this could seem useful in that, for example, one could allow highly variable or heavy flows to send data either more often, or less often if the percentage change remains small. This could potentially achieve some kind of load balancing across monitors. Instead of using identical $\delta(t)$ for all monitors, the coordinator can compute and distribute non-uniform local 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. The slacks we compute are no longer guaranteed to meet the error bounds with this scheme (we leave the extension of the guarantees as future work). However in our simulations, all the errors were well within the target bounds.

When to Adapt? At any time instant t, the coordinator can use Theorem 2 to provide optimal settings $\delta(t)$ for our system slack parameters - thus allowing our system to adapt to the evolving time series being monitored. However, in practice, frequent re-calibration of the δ parameter is not necessary, and, in fact, could cause system instability and excessive communication. Below we discuss two simple schemes for disseminating δ updates from the coordinator to the monitors. Both of our dissemination schemes presented below for uniform slacks *are* safe, meaning that our error guarantees hold. It is not the adaptivity that affects the validity of the guarantee, but rather the heterogeneity.

Min-based $\delta(t)$ Dissemination. 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 dissemination 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

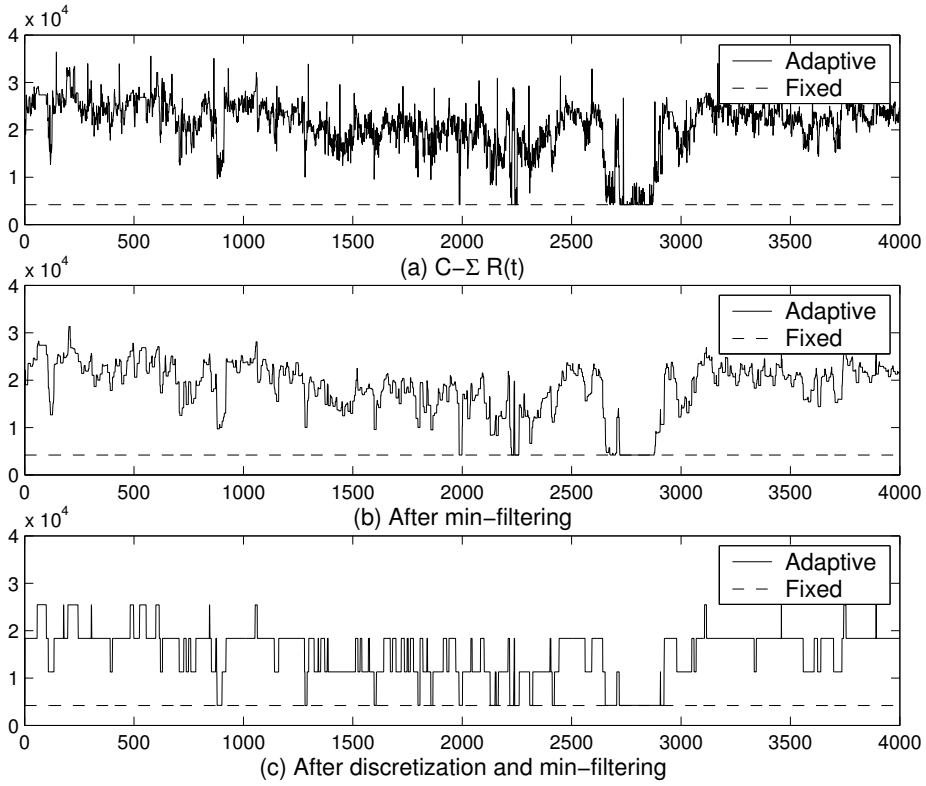


Figure 3.6: Effect of min- and discretization-based filtering.

be applied. Unlike `min`, however, these other aggregates are not "safe" and may cause more miss detections and/or false alarms.

Discretization-based $\delta(t)$ Dissemination. 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 $I_0, I_1, \ldots$, 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. Note that our two schemes can be combined into a single scheme.

As an example, Fig. 3.6 depicts the the smoothing effect of the two filtering steps on a real-life aggregate signal.

3.4 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 $s_i(t) = \int_{t-\tau}^{t} r_i(x)dx$ (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.

3.5 Cumulative Triggers

Th early efforts on distributed triggers, together with our schemes proposed in Sections 3.3, focused solely on *instantaneous* trigger conditions, where the goal is to fire the trigger as soon as the aggregate (typically, `SUM`) of observations (e.g., CPU utilization or number of messages sent) across distributed machines 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 and server load. 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*. Persistent violations are better observed by measuring activity over a window of time. However, the task of selecting a particular window size over which something is measured is a headache that has long plagued operators because a single window size cannot accommodate all of their needs. In addition to ISPs, managers of distributed server systems have found that measuring average server load using fixed sized windows is unsatisfactory in that it is insufficient to identify good or bad system behavior [68].

Our research goal is to enable broader and more flexible conditions for triggering. In this section, we introduce and formalize the concept of *distributed cumulative triggers*. These triggers allow one to detect cumulative violations that are persistent over time and are spread across a distributed set of machines. We 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 [16, 20] with insights based on *queueing theory*.

We achieve data reduction through smart filtering at the local monitors. Each monitor is assigned an amount of *slack* that carefully controls the discrepancy between its view of the data and that available at the coordinator. One of our key insights that drives our analysis is that this slack can be viewed as analogous to *queue sizes*. The challenge is to select the size of the distributed queues (or slack) so that the coordinator can still detect anomalies accurately. By only giving the coordinator a limited view of the data, the detector could make mistakes. We allow the users to input their tolerable false alarm rate and missed detection rate, as well as an error tolerance on the size of the violation. One of our key contributions is the development of an analytic solution to choose the slack values while guaranteeing that the three error tolerances are not exceeded. With our system we can perform anomaly detection with much less data than naïve "push-all" solutions, and yet simultaneously bound the error. These user-supplied target error rates enable the user to carefully control the tradeoff between reducing communication overhead and alarm detection accuracy.

3.5.1 A Queueing Perspective on Cumulative Triggers

Our approach to **supporting cumulative violations** without having to specify windows of time a priori is to use insights from queueing theory. Earlier work on data streaming uses *window-based* stream processing [19, 25] and 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 ahead of time. Instead, our key observation is that we can accurately model the monitoring of a cumulative trigger condition (see Chapter. 3.1) using a simple *queueing model* (see Fig. 3.7), as stated by the following theorem.

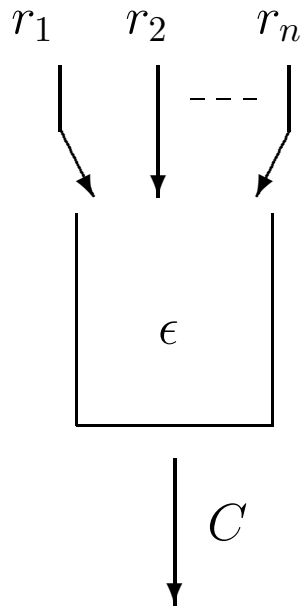


Figure 3.7: Cumulative violation and queue overflow.

Theorem 3 *Consider a queue of size ϵ with an arrival rate equal to the actual aggregate signal $\sum_{i=1}^{n} r_i(t)$ and a drain (i.e., service) rate equal to the trigger threshold C. A cumulative trigger should fire (i.e., $\exists \tau$ s.t. $V(t,\tau) > \epsilon$) if and only if the above queue overflows.*

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

As an example, the bottom half of Fig. 3.8 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

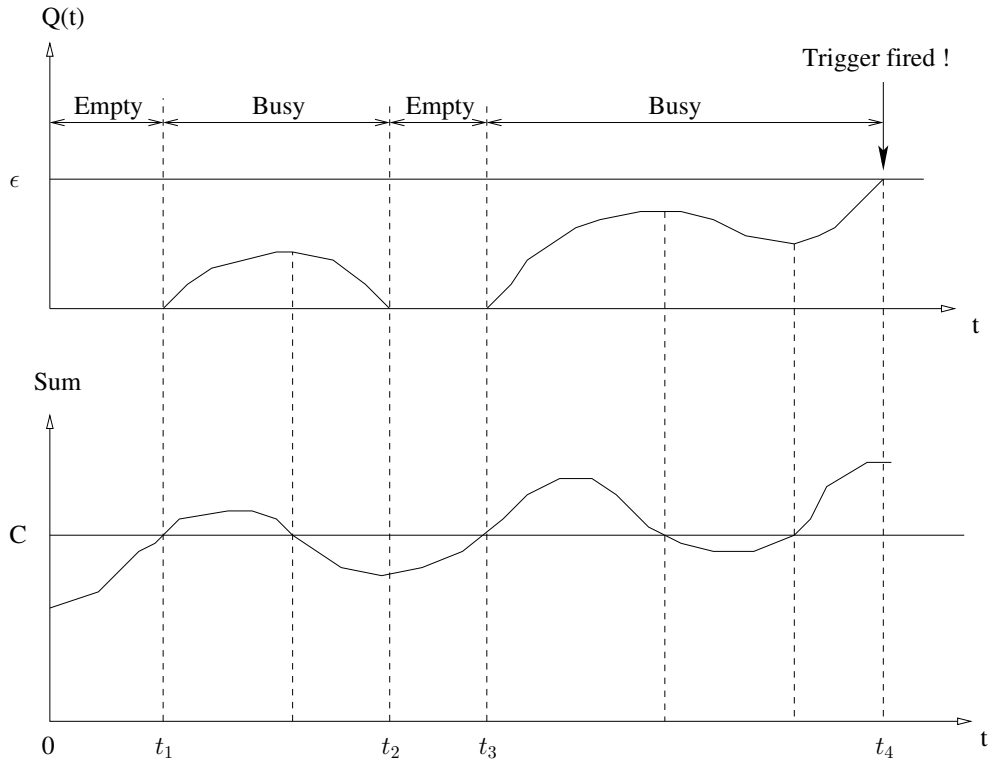


Figure 3.8: Queueing model for a cumulative trigger.

of a *busy period* $[t^s, t]$ (*i.e.*, a period during which the queue is persistently non-empty; that is, $t^s = \max\{x | x \leq t \text{ and } Q(x) = 0\}$) ending at t with a queue occupancy $Q(t) \geq \epsilon$. Fig. 3.8 shows two busy periods, $[t_1, t_2]$ and $[t_3, t_4]$, the second of which results in sufficient queue buildup to fire the trigger. It is not difficult to see that, by our queueing model, $Q(t) = V(t, t - t^s)$, so that $Q(t) > \epsilon$ (*i.e.*, a queue overflow) indeed implies that our trigger should fire. Similarly, for any time window $\tau \leq t$, $V(t, t - t^s) \geq V(t, \tau)$ (i.e., windows smaller or larger than the latest busy period can only reduce the cumulative size of the violation). In other words, $Q(t) = V(t, t - t^s) = \max_\tau\{V(t, \tau)\}$, implying the cumulative trigger should fire if and only if the queue overflows. The model in Fig. 3.7 captures the equivalence between an overflowing queue and a violation of the cumulative trigger constraint.

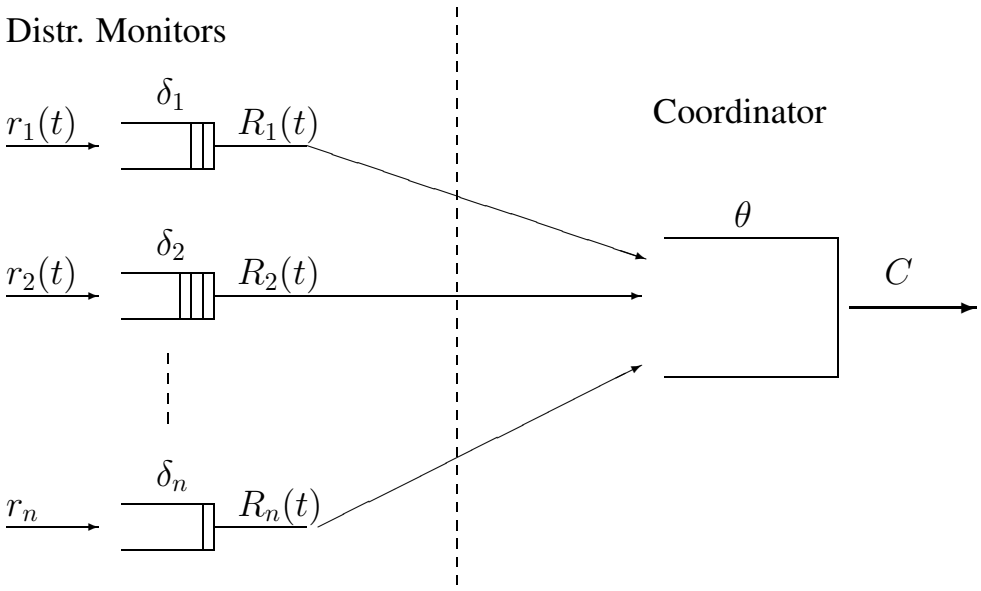


Figure 3.9: Distributed queueing model: cumulative triggers.

The simple queueing model discussed in Chapter 3.2 is ideal since it assumes the true aggregate $\sum r_i(t)$ feeds a single coordinator queue. However, in our distributed environment, the global coordinator only observes approximate predictions $R_i(t)$ of the local signals. We extend our queueing model to the distributed environment by placing queues at each of the monitors in addition to the one queue at the coordinator. This distributed queueing model is depicted in Fig. 3.9. Our task is then to design algorithms to convert the centralized queue model of size ϵ into a coordinator queue of size θ and a set of local monitor queues of size $\delta_1, \ldots, \delta_n$, while still guaranteeing the necessary false alarm and missed detection rates.

3.5.2 The Distributed Tracking Protocols

The Local Monitor Protocol. In our distributed model, each local queue has an arrival rate or $r_i(t)$, a drain rate of $R_i(t)$ and a size of δ_i. Let t_i^{prev} denote the time of the last update message from m_i

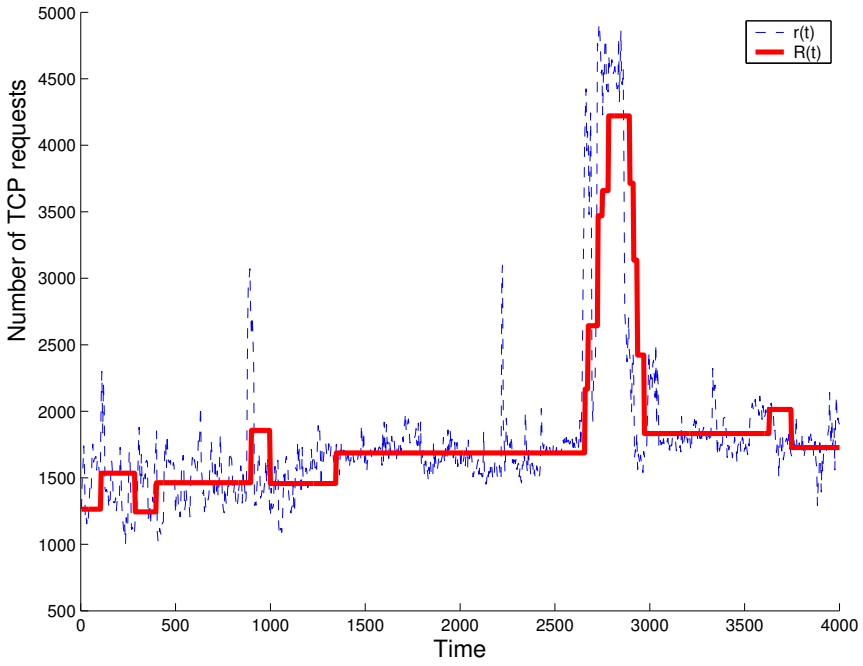


Figure 3.10: Local prediction-based filtering.

to the coordinator. At any time t, the size of the monitor's queue captures the cumulative deviation of $r_i(t)$ from its most recent prediction $R_i(t^{prev})$ over the interval $[t_i^{prev}, t]$, namely $d_i(t) = \int_{t_i^{prev}}^{t}(r_i(x) - R_i(x))dx$. Should the local queue overflow, i.e., when $|d_i(t)| > \delta_i$, this means the drift has exceeded the allowed slack. At this time the monitor sends the coordinator an update on its time series. It sends the current value $r_i(t)$, a prediction $R_i(t)$ for near-term future values, and the current value $d_i(t)$. The amount $d_i(t)$ corresponds to the cumulative deviation of $r_i(t)$ from its most recent prediction. At the time of the update, the local queue also resets $d_i(t)$ to zero. Note that, unlike traditional queueing, local monitor queue occupancies are allowed to become negative, if predictions consistently overestimate the true local signals. Such conditions are important to detect and bring to the coordinator's attention since they also capture excessive drift and thus lead to more

updates. Sending underflow information to the coordinator can also enable cross-site variations to cancel out (thus avoiding false alarms).

We point out that the queues we are using here are models, not actual physical queues. In an implementation, a queue that stores data is not needed. Instead only a counter is needed that is incremented and decremented according to the queueing models herein. We use these models for our analysis that enables us to compute slack sizes while meeting guarantees for upper bounds on errors.

As an example, Fig. 3.10 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).

The Coordinator Protocol. In our distributed queueing model, the coordinator's queue has an arrival rate of $\sum_{i=1}^{n} R_i(t)$, a drain rate equal to the trigger threshold C, and is of size θ, as in (Fig. 3.9). In addition to the continuous "arrivals" at rate $\sum_{i=1}^{n} R_i(t)$ to the coordinator queue, each update from monitor m_i also introduces a *chunk* of $d_i(t)$ arrivals into the queue. Note that if the queue underflows (drops below zero), then $d_i(t)$ is negative. The coordinator continuously tracks this complex arrival process at its queue and fires a trigger violation if its queue overflows. A high-level pseudo-code description of both the local-monitor and coordinator protocols is depicted in Fig. 3.11.

Intuitively, the local slacks δ_i at the remote monitors aim to filter out local variations in individual $r_i(t)$ signals, while the *coordinator slack* θ is useful for canceling out variations *across*

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

Procedure Monitor(i, δ_i)

Input: Monitor index i, local slack parameter δ_i.

1. **while** (**true**) **do**
2. t := current time
3. t_i^{prev} := time of last update to coordinator
4. $d_i(t) := \int_{t_i^{prev}}^{t} (r_i(x) - R_i(x))dx$
5. **if** ($|d_i(t)| > \delta_i$) **then**
6. Send update message $(i, d_i(t), R_i(t))$ to coordinator
7. Set $d_i(t) := 0$
8. **if** (new slack δ_i^* is received) **then**
9. 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. 3.5.3)*
8. **if** (adaptive allocation) **then disseminate**($\{\delta_i\}$)

Figure 3.11: Procedures for (a) local monitor update processing, and (b) distributed trigger tracking at the coordinator.

3.5.3 Queueing Analysis for Slack Estimation

We now present an analysis of a simplified variant of our distributed queueing model (Fig. 3.9), and discuss the application of our results to estimating effective settings for the monitor and coordinator slack parameters in our system. The existence of the local δ_i filters obviously reduces communication costs by allowing monitors to "absorb" updates with no communication to the coordinator. At the same time, however, this local filtering also makes the arrival process at the coordinator queue more *bursty* by introducing bursts of queue arrivals and departures when the filter constraints at local monitors are violated. Thus, abstractly, the role of the coordinator queue (of size θ) is to allow for such bursts to be effectively absorbed (or, cancel each other out) as long as the cumulative trigger bound is not exceeded.

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

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

We make two key assumptions to make the analysis tractable. First, we assume uniform local slack parameters, where $\delta_i = \delta$ for all i (in Sec. 3.5.4 we briefly discuss non-uniform parameters). Second, we assume an $M/M/1$ queueing model for the coordinator queue[1]. Under the $M/M/1$ assumption, let λ_r and λ_R denote the mean "arrival rates" for the true signal and predicted signal, respectively (*i.e.*, the estimated averages of $\sum_i r_i(t)$ and $\sum_i R_i(t)$ over time). Similarly, let λ_e and λ_d be the mean arrival rates for enqueue and dequeue chunks (respectively) at the coordinator. Note that, the λ_R, λ_e, and λ_d rates are directly observable at the coordinator, and can be computed empirically (*e.g.*, through averaging over a time window of recent queueing activity). Since the overall "mass" of the true aggregate signal is preserved over time, the coordinator can also accurately estimate λ_r as $\lambda_r = \lambda_R + (\lambda_e - \lambda_d) \cdot \delta$. [2]

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

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

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

result relating the miss-detection rate to ϵ, θ, and δ, under the assumption of normally-distributed local "queue" sizes.

Theorem 4 *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*

$$\int_{x=0}^{\infty} \left[1 - F\left(\frac{\epsilon - \theta}{\delta} + x + 1\right)\right] \rho^x (1-\rho) dx = \beta \tag{3.1}$$

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

The assumption of a zero mean for the aggregate occupancy of all local monitor queues is motivated by the fact that, over a large enough window of time, the true and predicted signal rates are approximately equal (*i.e.*, $\lambda_R \approx \lambda_r$). Similarly, the normality assumption can be justified under the assumption of *independent updates* across local monitors and the law of large numbers (for large enough n)[3]. To estimate the aggregate variance σ^2 in our system, each local monitor m_i continuously tracks the up-to-date variance σ_i^2 of its local occupancy and ships that information to the coordinator in its update messages if there is a significant change with respect to the most recent measurement; the coordinator then estimates the aggregate variance as $\sigma^2 = \sum_{i=1}^{n} \sigma_i^2$. Note that Theorem (4) has the ability to support adaptivity through its dependence on $\rho = \frac{\lambda_r}{C}$. As the rate λ_r evolves, so will ρ, and the resulting value computed for δ.

Now, consider the false alarm rate η. Observe that, in our distributed queueing model, the arrival and drain rates at the coordinator queue can be naturally approximated as $\lambda_R + \lambda_e \cdot \delta$ and $C + \lambda_d \cdot \delta$ (respectively), whereas the corresponding rates for the idealized (centralized) case

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

are simply λ_r and C. Based on this observation and our $M/M/1$ assumption, we can prove the following result (see the Appendix for details).

Theorem 5 *Assume an $M/M/1$ model for the coordinator queue. Then, setting:*

$$1 - \left(\frac{\lambda_r}{C}\right)^{\frac{\epsilon}{\delta}+1} \Big/ \left(\frac{\lambda_R + \lambda_e \cdot \delta}{C + \lambda_d \cdot \delta}\right)^{\frac{\theta}{\delta}+1} = \eta \tag{3.2}$$

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

Given a triple of trigger-tracking requirements (ϵ, β, η), our coordinator algorithms use the derived system of two non-linear equations (Theorems 4 and 5) to solve for the optimal (under our assumptions) coordinator- and monitor-slack values θ and δ (Step 7 in Fig. 3.11(b)). The local slacks δ are then distributed to the monitors. Both Eqn (3.1) and Eqn (3.2) depend on queue input rates, and can be solved again as often as desired; as the time series change, the queue input and drain rates will evolve and thus delta can be updated. Thus supporting adaptivity with our scheme is straightforward.

3.5.4 Heterogeneous and Adaptive Slack Allocation

Clearly, at any time instant t, Theorems (3.1) and (3.2) 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. Our coordinator algorithms achieve this through min-based and discretization-based filtering steps on $\theta(t)$, $\delta(t)$, similar to those described in Sec 3.3.1.

3.6 Evaluation

In this section, we use our protocol implementation and protocol simulator with a real wide-area network activity dataset, and present results for performance metrics, and cumulative and instantaneous triggers.

3.6.1 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, which 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 [50], 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. Thus, 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).

3.6.2 Performance Metrics

In evaluations of cumulative triggers, the target performance level is specified by the usual triplet parameters (ϵ, β, η). We start with experiments in which the monitors are given uniform slacks $\delta_i = \delta$, and evaluate the effect of heterogeneous slack allocation 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 4 and 5. 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.

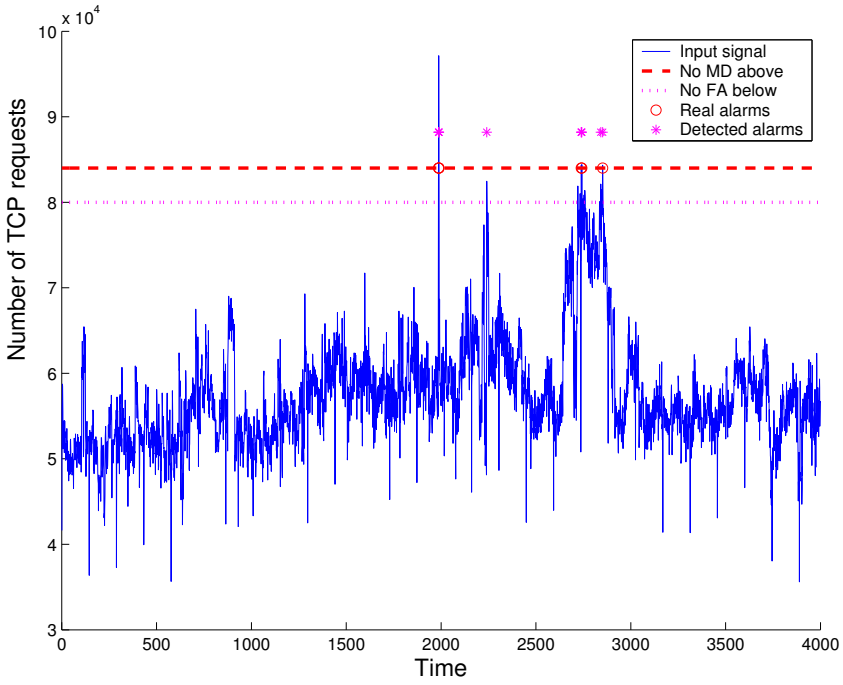


Figure 3.12: Triggering on instantaneous violation.

3.6.3 Results for Instantaneous Triggers

Recall that our guarantee for instantaneous triggers is slightly different from varying-window 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.

Model Validation

We first show in Fig. 3.12 a sample of aggregate time series used in experiments and the triggering performance guaranteed by Theorem 2. 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

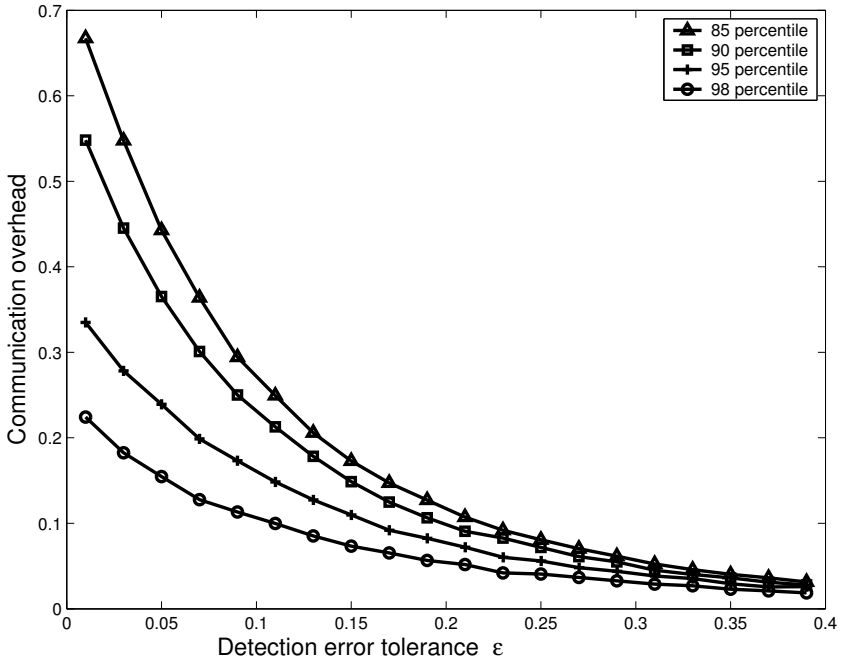


Figure 3.13: Impact of target C on communication overhead.

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 uncertainty 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 the uncertainty zone, no trigger is missed, as indicated by one star for each circle; if aggregate signals are below uncertainty zone, no trigger is fired, as indicated by no star for signals under the dotted line. Inside the uncertainty zone, our protocol has one false alarm, as indicated by the second star in the graph.

Experiment Configuration

The 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 tolerances, followed by the comparison of our work with existing approaches and the scalability of our system.

Fig. 3.13 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 importantly, for any given ϵ, the communication overhead decreases as the target threshold C increases. This is expected, when C is as 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 as 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 a relatively high cost to achieve the desired detection accuracy. However, when the error tolerance is big enough, the protocol pays a low cost even when C is small. This demonstrates that our ideal of letting total slack adapt according to aggregate signals is very effective in reducing communication overhead.

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 gains 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

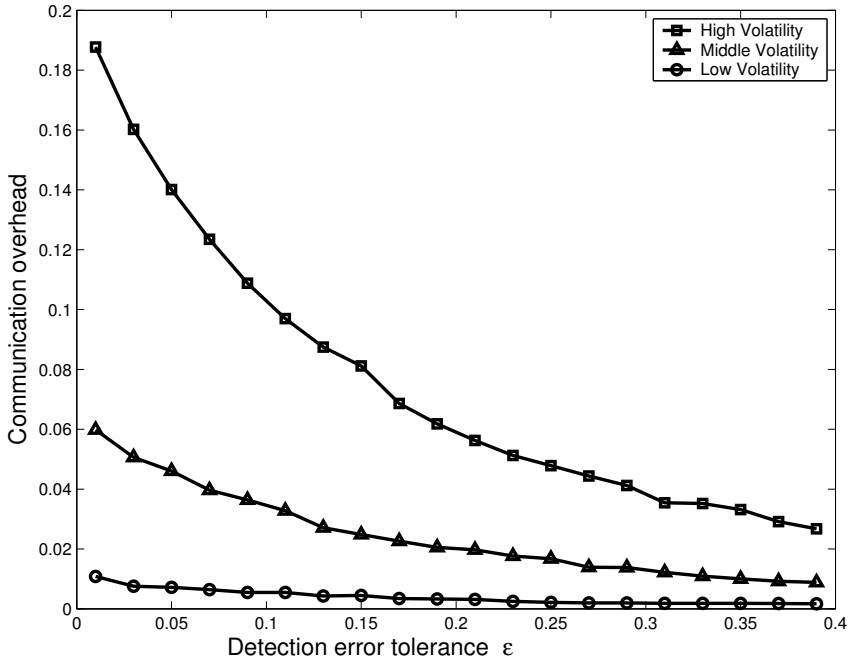


Figure 3.14: Impact of volatility on communication overhead.

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. 3.14. 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.

Detection Performance vs. Communication Overhead

We now compare our work with previous work. The only adaptive filtering approach similar to our problem is that proposed by Olston *et al.* in [47] in the context of data streaming. We also compare our solution with a naïve 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 $r_i(t) > \frac{(C+\epsilon)}{n}$. [4] We consider two versions of our protocol. In one, called *uniform adaptive detection*, the δ_i's are homogeneous but change based on the 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 [47] 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. 3.15. 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 2 essentially guarantees no false alarms/missed detections outside the "uncertainty" zone for our instantaneous triggering scheme.)

[4] We do not compare with [36] since, unlike our schemes and [47], they offer no strict error guarantees.

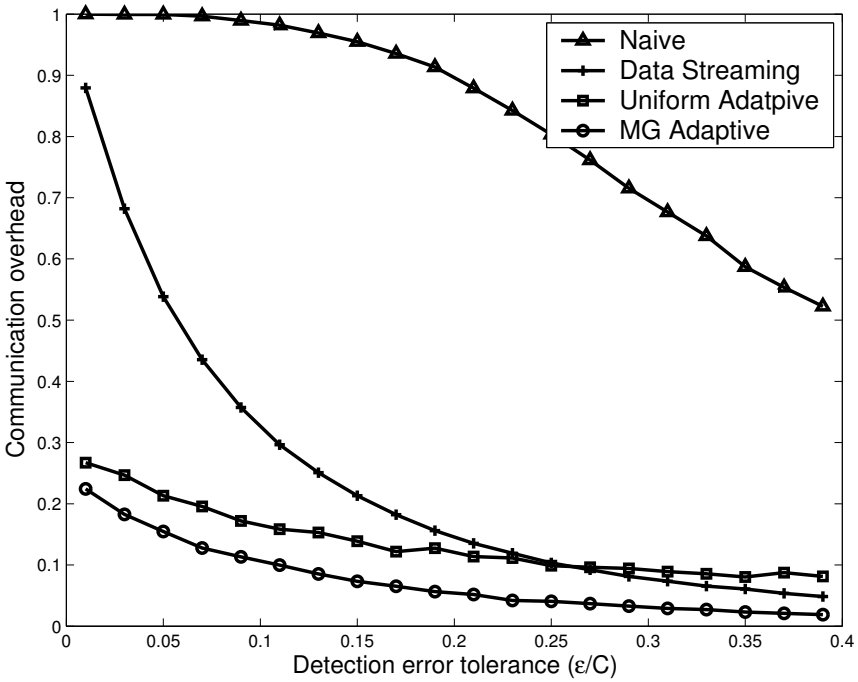


Figure 3.15: Comparing our approach to existing approaches.

We see that our MG adaptive scheme substantially outperforms both the naïve 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 illustrates 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

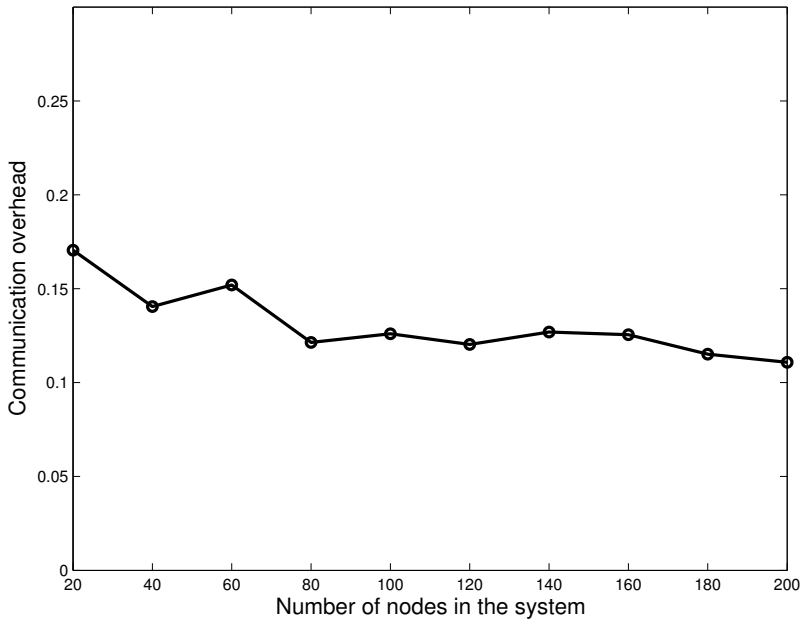


Figure 3.16: Communication overhead versus system size.

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. 3.16 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 to that shown in Fig. 3.20. As seen from the results, the (per-node) communication overhead is relative stable and slightly decreasing when the

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

Table 3.2: *Desired vs. achieved detection performance.*

system size increases, indicating that our protocol is scalable to large systems.

3.6.4 Results for Cumulative Triggers

Model Validation

In Table 3.2, 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 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.

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 anomalous 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. 3.17 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 0.1-0.2 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.

Fig. 3.18 shows the impact of data volatility on communication overhead. The experiment setup is similar as that in Fig. 3.14. 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

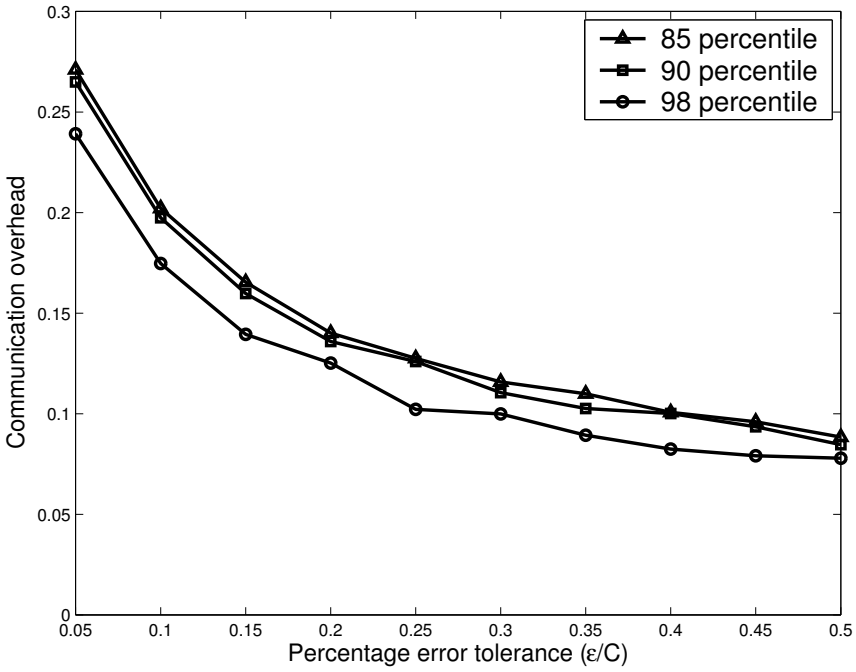


Figure 3.17: Impact of constraint violation threshold C.

most volatile set we considered, we still achieve efficient communication. In the remainder of our experiments, we use the middle volatility set.

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. 3.19(a) shows the communication overhead tradeoff, (b) and (c) show the monitor queue and coordinator queue sizes used for each achieved (β^*, η^*) pair. Note that to facilitate viewing of the 3-dimensional plots, the order of increasing β^* and η^* in Fig. 3.19(a) differs from that in (b) and (c).

In Fig. 3.19(a) we see that the communication overhead decreases quickly as β and η

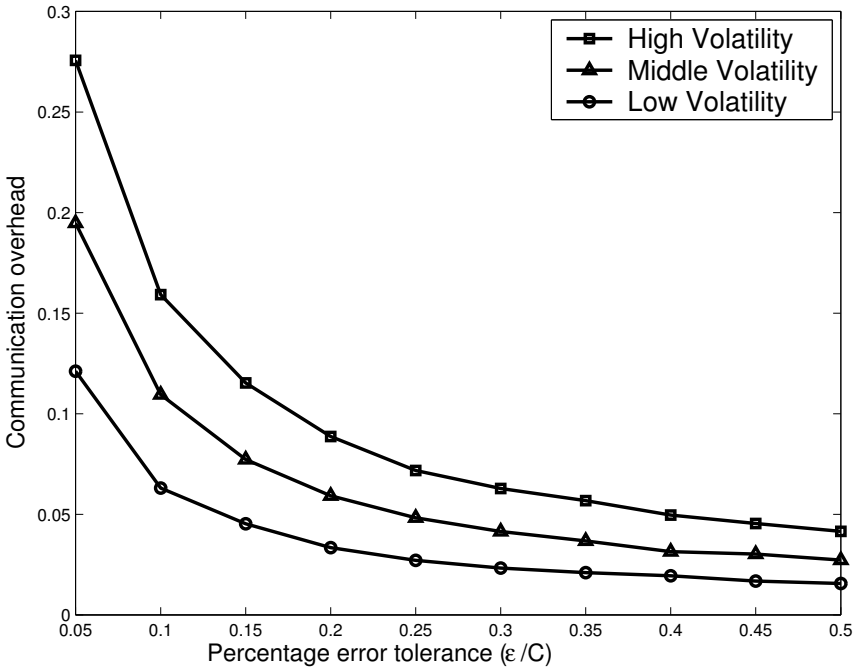


Figure 3.18: Impact of volatility on overhead.

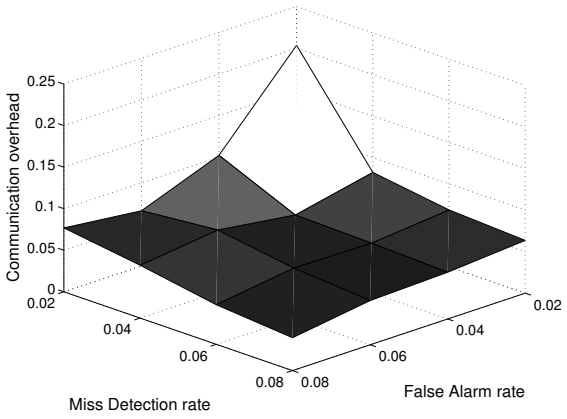
increase. The basic phenomenon here is that for any error type (ϵ, β, and η are different error types), the communication overhead can be reduced if we can tolerate higher errors. In this sense, Fig. 3.19(a) is consistent with Figs. 3.17 and 3.18. What is surprising is that the range of communication 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. 3.17, 3.18, and 3.19(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 to-

day 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.

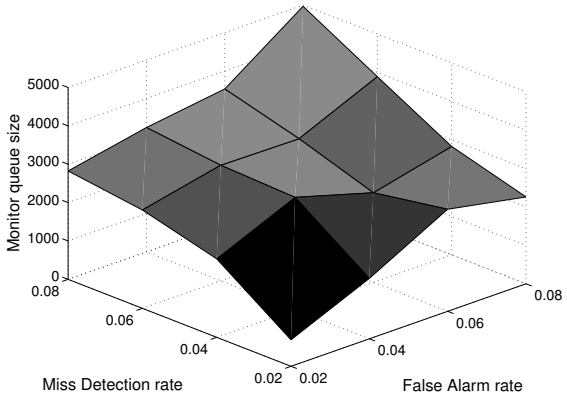
Fig. 3.19(b) shows that as the tolerable false alarm rate increases, the local queues increase in size because we can do more filtering at the monitors, which in turn brings down the overhead. This explains why the overhead decreases with increasing false alarm rate. A similar behavior occurs when the tolerable missed detection rate is raised. 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.

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.

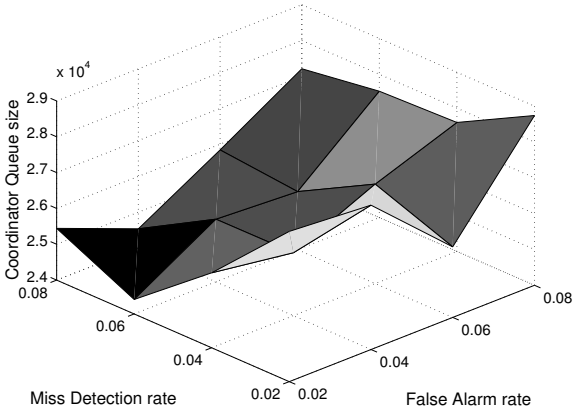
We plot the communications cost as a function of the number of monitors in Fig. 3.20. We varied the number of monitors from 40 to 200, and used the target performance triplet $(\epsilon, \beta, \eta) =$



(a) *Communication overhead*



(b) *Monitor queue size*



(c) *Coordinator queue size*

Figure 3.19: Parameters design and tradeoff between false alarm, miss detection and communication overhead.

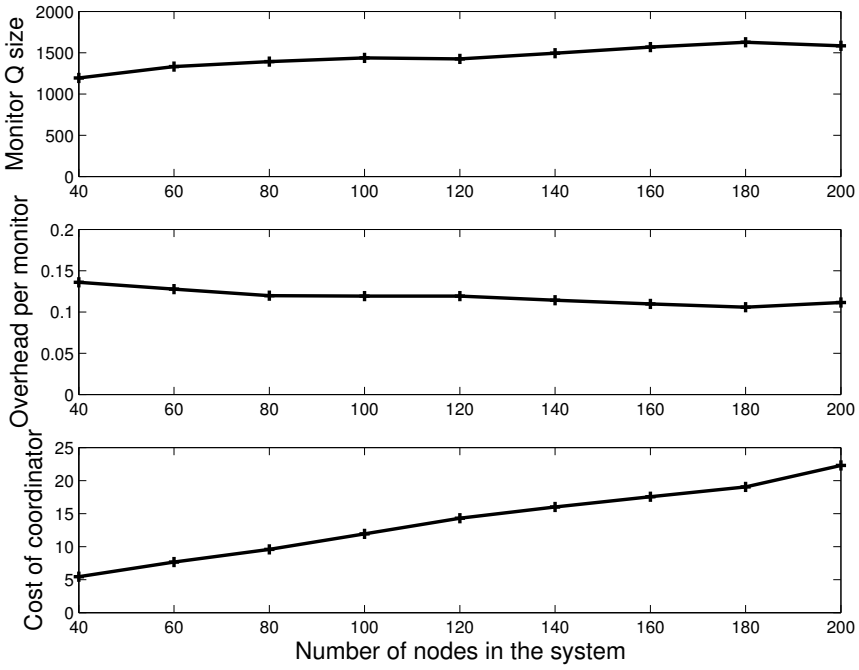


Figure 3.20: Communication overhead versus system size.

$(0.2C, 0.06, 0.06)$. 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 sub-linearly 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

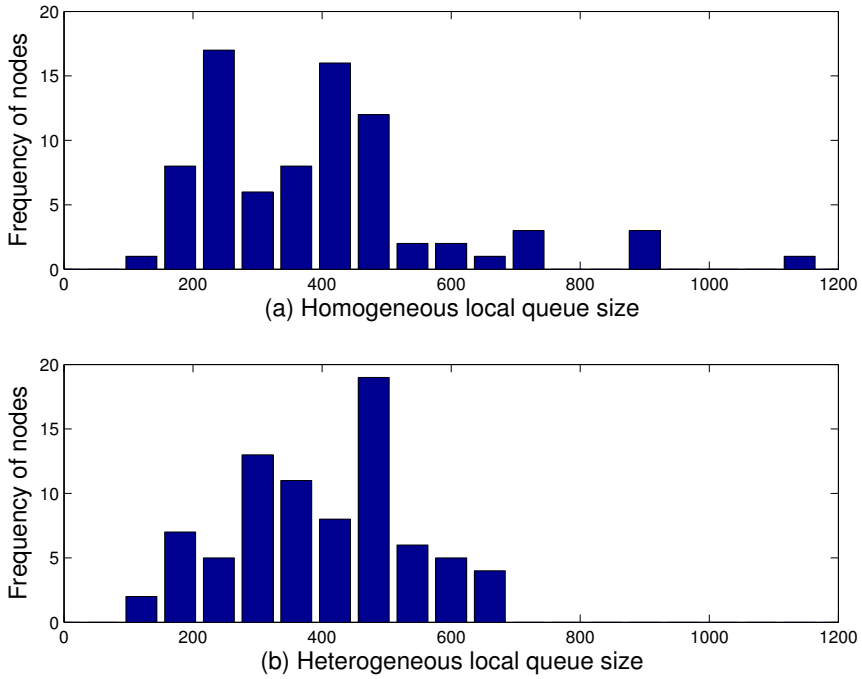


Figure 3.21: Number of messages per node.

to implement actual buffers; instead the queues are implemented as counters, and the queue sizes correspond to maximum counter values.

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 assigned 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. 3.21. 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 shows 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.

3.7 Chapter Summary

We have presented a novel solution and proposed the D-Trigger framework to the problem of efficient online detection and triggering on aggregate constraint conditions in a distributed monitoring system. We believe our work is the first to address the constraint detection over a time-varying window. Our work relies on a key insight: focus on accurate triggering, and not ϵ-accurate aggre-

gate value reporting. This insight enables a greater than 85% reduction communication overhead, while preserving high detection accuracy.

Our contributions include: providing a mathematical definition of distributed triggering with different constraint violation modes; using a queuing theory-based problem definition, which makes analytical solutions possible; providing strong guarantees for the instantaneous case; 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. Our general and efficient detection framework, D-Trigger, is the foundation and extensible vehicle for the sophisticated detection approaches discussed in the following chapters.

Chapter 4

Toward Sophisticated Online Detection

To support the continuous detection and triggering functionalities in distributed monitoring system, a set of approaches, including our D-Trigger framework discussed in Chapter 3, have been proposed recently for communication-efficient tracking of distributed triggers [38, 56, 32]. However, existing approaches have significant limitations: they only support simple thresholds on distributed aggregate functions like `SUM` and `COUNT`, which are insufficient for sophisticated detection applications.

In this chapter, we present our initial efforts to support more sophisticated triggering functionalities in D-Trigger system. We give some examples for detection schemes that may require checking non-linear threshold constraints, relative triggers, and so on. To illustrate these ideas more concretely, we use the example of a centralized PCA-based anomaly detection application [41]. This method detects anomalies in traffic volume levels by simultaneously examining the link load levels of all the links in a large network. It works by using a Principal Components Analysis (PCA) technique to decompose network traffic into normal and residual components, and then detects

anomalies by applying a threshold function on the residual components. One of our contributions is to illustrate how to redesign this application in a distributed fashion, with far less communication overhead, by using D-Trigger as an underlying paradigm. Abstractly, this requires D-Trigger that support complex threshold functions, and can be composed to lend support to sophisticated distributed detection applications. We describe, for example, a simple approximation to a non-linear threshold constraint that works well in our example application: our method achieves roughly the same level of accuracy as the centralized scheme, while communicating only around 20% of the underlying data.

4.1 Extensions to Simple Triggers

We believe that more general constraints, that include varied aggregation functions, threshold functions, that use subsets of monitors, and that are defined over varying time period, can all be incorporated into a more general triggering protocol. We briefly introduce these ideas here.

4.1.1 Supporting Complex Constraints

In detection systems, the constraints are typically made up of an aggregation function, a threshold and a set of nodes whose data is used to determine whether or not the constraint has been violated. Our previous work used linear aggregation functions. In this work, we introduce methods for dealing with quadratic aggregation functions, which is useful in detecting network-wide anomalies. We now discuss how a variety of complex constraints can be built by varying these elements of the constraint(s). Each of these variations can be expressed in the terms of our framework, and incorporated into a generalized triggering protocol.

In our anomaly detector application we used a Taylor series expansion (for the aggregation function on $f[R_1, \ldots, R_n]$), dropping higher order terms, to deal with a quadratic constraint. This approach can be applied more generally. For any such continuous function $f(t) = f[r_1(t), \ldots, r_2(t)]$, the Taylor Expansion is

$$f[R_1, \ldots, R_n] - f[r_1, \ldots, r_n] = \sum_{i=1}^{n} \frac{\partial f}{\partial r_i} \cdot (R_i - r_i) + \mathcal{O}\left[\sum_{i,j=1}^{n} (R_i - r_i) \cdot (R_j - r_j)\right].$$

Then, if $(R_i - r_i), i = 1, \ldots, n$, are small and independent, and we ignore all second and higher order terms, we can linearize this continuous function, and the distributed simple triggers can track this function based on its first order components. We define $g_i \equiv \frac{\partial f}{\partial r_i}$ as the marginal impact of local value $r_i(t)$ on the global function. This is incorporated into our system by the monitor which performs filtering using $|r_i(t) - R_i(t)| > \frac{\delta_i}{|g_i|}$. Note that the marginal factor g_i can be computed by M_i itself (simple constant) or computed by the coordinator (time-varying distributed values).

So far we have used a constant trigger threshold C. However, this threshold could, for example, vary in time. It could be determined or computed as a user specified input, or be a function of the data - computed using all the data (as in our anomaly detector), or using data from only a subset of the monitors. Other constraints can be built when the set of monitoring nodes are split into different subsets. Let A and B be two subsets of monitors of interest, with n_1 and n_2 monitors, respectively. To denote variables from monitors in the set, we use set labels in the superscript.

More complex threshold functions and the ideas of using subsets of monitoring data can be incorporated into the coordinator's protocol. The coordinator can track one or more global functions that are defined on subsets of distributed data streams. For example, let $f(t)$ be the function defined

on set A, and $C(t)$ on set B. The coordinator computes

$$f(t) = f\left[R_1^A, \ldots, R_{n_1}^A\right], \quad C(t) = C\left[R_1^B, \ldots, R_{n_2}^B\right],$$

and triggers an alarm if $f(t) > C(t)$. Based on its view of global information, the coordinator computes a set of parameters and sends them to monitors when necessary as:

$$\max(\epsilon, C(t) + \epsilon - f(t)) = \delta_1^A + \cdots + \delta_{n_1}^A + \delta_1^B + \cdots + \delta_{n_2}^B,$$
$$g_i^A = \frac{\partial f}{\partial r_i^A}, \quad g_j^B = \frac{\partial C}{\partial r_j^B}.$$

Various optimization algorithms can be used to compute δ_i's that minimize communication, however our protocol remains applicable regardless of algorithm choice.

It is also possible to incorporate the notion of time, in case one seeks to detect constrain violations that occur over a period of time. To support such constraints, we can extend the form into cumulative triggers as discussed in Chapter 3, which track the relationship between $f(t)$ and $C(t)$, and only trigger alarms when $f(t)$ exceeds $C(t)$ over time.

4.1.2 Advanced Queries For Load Balancing

We now demonstrate how a generalized triggering protocol can support advanced queries for hot spot detection in distributed systems. Consider: 1) **relative triggers** that fire an alarm if the total workload of servers in set A is β times more than that of set B; 2) **any-set triggers** that fire an alarm if the total workload of any $\alpha\%$ servers is more than C; 3) **composite triggers** that fire an alarm if the total workload of any $\alpha\%$ servers is more than β portion of the total system workload.

Tracking relative triggers. We view relative triggers as normal ones with time-varying threshold $C(t)$ as follows: 1) the coordinator has threshold $C(t) = \beta \cdot \sum R_j^B(t)$, and 2) it triggers whenever

$\sum R_i^A(t) > C(t)$. Monitors and the coordinator have to track both values of $\sum r_i^A(t)$ and $\sum r_j^B(t)$. One can easily extend the instantaneous trigger to detect unbalanced load and guarantee a "2ϵ-band" of detection accuracy.

Tracking any-set and composite triggers. One can prove that detecting whether the workload sum from any subset of k servers is above a threshold is equivalent to detecting whether the sum of the top-k workload is above the threshold. So by composing distributed top-k monitoring [3] with our triggering protocols, we can efficiently track both any-set and composite triggers with guaranteed accuracy. We leave as future work to extend and customize triggers for top-k monitoring.

4.2 Extended Triggers for Network-Wide Traffic Anomaly Detection

In this section, we first summarize a centralized algorithm [41] for doing network-wide volume anomaly detection. Network volume anomalies are unusual and significant changes in end-to-end traffic flows typically caused by worms, DoS attacks, device failures, misconfigurations, etc. We then show how this problem can be mapped onto our D-Trigger system and explain the functionality required at our monitors and coordinator to implement this approach.

4.2.1 Problem description

We consider a monitoring system that includes a set of *distributed monitor nodes* $M_1, \ldots, M_n$, each of which collects a locally-observed time-series data stream (Fig. 4.1). A central *coordinator node* seeks to observe the ensemble of these time series (i.e., the global network-wide data), and make global decisions such as those concerning matters of network-wide health. The application of detecting *volume anomalies* across a large network employs such a distributed monitoring infras-

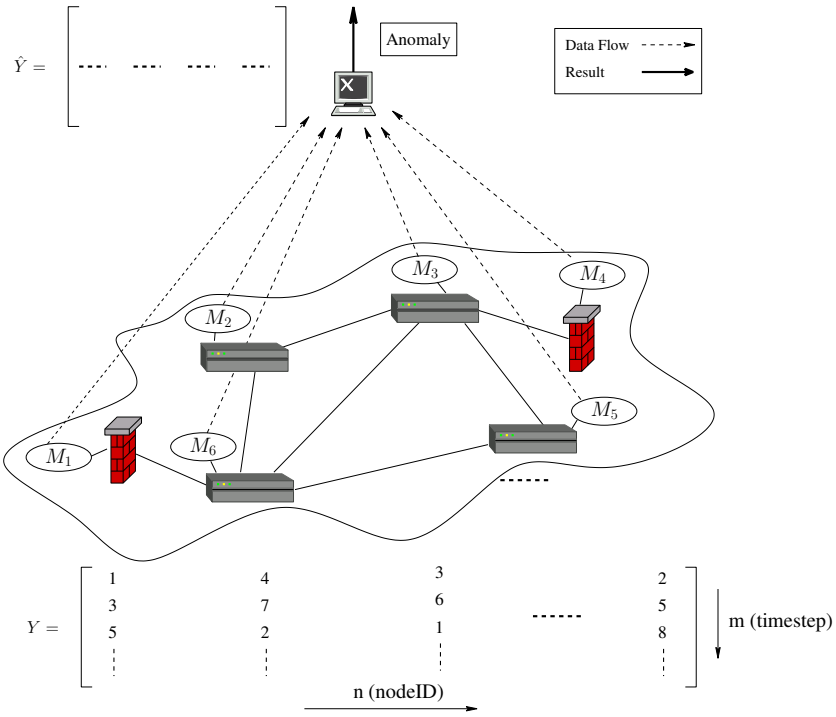
tructure. A volume anomaly refers to unusual traffic load levels in a network that are caused by anomalies such as DDoS attacks, flash crowds, device failures, misconfigurations, and so on.

Each monitor M_i collects a new data point $r_i(t)$ at every time step, and assuming a naïve "continuous push" protocol, sends the new point to the coordinator. Based on these updates, the coordinator keeps track of a sliding time window of size m (i.e., the m most recent data points) for each monitor's time series, organized into a matrix $\mathbf{Y}$ of size $m \times n$ (where the i^{th} column $\mathbf{Y}_i$ captures the data from monitor i, see Fig. 4.1). The coordinator then makes its decisions based on this global $\mathbf{Y}$ matrix.

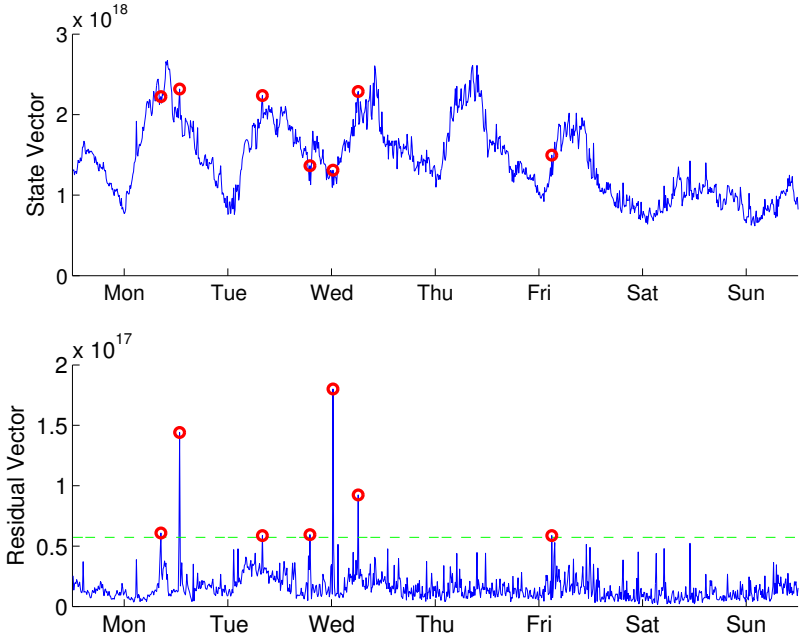
In the network-wide volume anomaly detection algorithm of [41], the local monitors measure the total volume of traffic (in bytes) on each network link, and periodically (e.g., every 5 minutes) centralize the data by pushing all recent measurements to the coordinator. The coordinator then performs PCA on the assembled $\mathbf{Y}$ matrix to detect volume anomalies. This method has been shown to work remarkably well, presumably due to the inherently low-dimensional nature of the underlying data [42]. However, such a "periodic push" approach suffers from inherent limitations: To ensure fast detection, the update periods should be relatively small; unfortunately, small periods also imply increased monitoring communication overheads, which may very well be unnecessary (e.g., if there are no significant local changes across periods).

4.2.2 Using PCA for Centralized Detection

Detecting anomalies is the first, critical step for network diagnostics, however they are usually hidden in large amounts of high-dimensional, noisy data. Volume anomalies usually propagate through the network and are observable on all links they traverse. As observed by Lakhina et al., although, the measured data is of seemingly high dimensionality (n = number of links), nor-



(a) *The system setup*



(b) *Abilene network traffic data*

Figure 4.1: (a) The distributed monitoring system; (b) Data sample ($\|\mathbf{y}\|^2$) collected over one week (top); its projection in residual subspace (bottom). Dashed line represents a threshold for anomaly detection.

mal traffic patterns actually lie in a very low-dimensional subspace; furthermore, separating out this normal traffic subspace using PCA (to find the principal traffic components) makes it much easier to identify volume anomalies in the remaining subspace. Lakhina *et al.* [41] proposed a solution to uncovering volume anomalies within a network by examining the traffic on all links inside a network simultaneously. Their approach assumes a protocol such as SNMP is available to collect link counts on every link and ship these statistics to a central network operations center (NOC). Their technique performs PCA on these link traffic measurements, and decomposes the high-dimensional space occupied by a set of network traffic measurements into disjoint subspaces corresponding to normal and anomalous network conditions. By performing statistical analysis on traffic signals in anomalous subspaces, they can effectively detect, identify, and quantify network-wide traffic anomalies.

Their method is summarized as follows. Consider a network with n links each of which has a monitor, M_i, where $i = 1, ..., n$, that measures the traffic load every measurement interval. After m measurement intervals, the monitor effectively has a time series of link counts for its link. The times series data from all of the monitors is sent to the NOC where it is assembled in a matrix $\mathbf{Y}$, in which each column i denotes the timeseries measurements of the i-th link and each row t represents an instance of all the links at time t (where $t = 1, ..., m$). We use $\mathbf{y}$ to denote a vector of measurements of all the links from a single timestep, which is an arbitrary row of $\mathbf{Y}$, transposed to a column vector,

$$\mathbf{y} = \begin{bmatrix} r_1 & r_2 & \dots & r_n \end{bmatrix}^T,$$

where $r_i = r_i(t)$, link i's value at time t, for $i = 1, \dots, n$.

PCA is a coordinate transformation method that maps a given set of data points onto principal components, which are ordered by the amount of data variance that they capture. Applying

PCA to $\mathbf{Y}$ yields a set of n principal components, $\{\mathbf{v}_i\}_{i=1}^{n}$, which are computed as:

$$\mathbf{v}_k = \arg\max_{\|\mathbf{x}\|=1} \|(\mathbf{Y} - \sum_{j=1}^{k-1} \mathbf{Y}\mathbf{v}_j\mathbf{v}_j^T)\mathbf{x}\|.$$

They are essentially the n eigenvectors of the estimated covariance matrix $\mathbf{A} := \frac{1}{m}\mathbf{Y}^T\mathbf{Y}$. As shown in [41], PCA reveals that the Origin-Destination (OD) flow traffic matrices (i.e., the complete traffic demand across an entire network) of ISP backbones have low intrinsic dimensionality. Because the link traffic and the end-to-end traffic demands are linearly related, it turns out that the ensemble of all link traffic in a backbone network also exhibits low dimensionality. For example, in the Abilene network with 41 links, most data variance can be captured by the first $k = 4$ principal components. Thus, the underlying normal OD flows effectively reside in a (low) k-dimensional subspace of $\mathbb{R}^n$. This subspace is referred to as the *normal* traffic subspace $\mathcal{S}_n$. The remaining $(n - k)$ principal components constitute the *abnormal* traffic subspace $\mathcal{S}_a$.

Detecting volume anomalies relies on the decomposition of link traffic $\mathbf{y} = \mathbf{y}(t)$ at any time step into normal and abnormal components, $\mathbf{y} = \mathbf{y}_n + \mathbf{y}_a$, such that (a) $\mathbf{y}_n$ corresponds to modeled normal traffic (the projection of $\mathbf{y}$ onto $\mathcal{S}_n$), and (b) $\mathbf{y}_a$ corresponds to residual traffic (the projection of $\mathbf{y}$ onto $\mathcal{S}_a$). Mathematically, $\mathbf{y}_n(t)$ and $\mathbf{y}_a(t)$ can be computed as

$$\mathbf{y}_n(t) = \mathbf{P}\mathbf{P}^T\mathbf{y} = \mathbf{C}_n\mathbf{y} \text{ and } \mathbf{y}_a(t) = (\mathbf{I} - \mathbf{P}\mathbf{P}^T)\mathbf{y} = \mathbf{C}_a\mathbf{y},$$

where $\mathbf{P} = [\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}]$, is formed by the first k principal components which capture the dominant variance in the data. The matrix $\mathbf{C}_n = \mathbf{P}\mathbf{P}^T$ represents the linear operator that performs projection onto the normal subspace $\mathcal{S}_n$, and, $\mathbf{C}_a$ projects onto the abnormal subspace $\mathcal{S}_a$.

As observed in [41], a volume anomaly typically results in a large change to $\mathbf{y}_a$; thus, a useful metric for detecting abnormal traffic patterns is the squared prediction error (SPE): $\mathbf{SPE} \equiv$

$\|\mathbf{y}_a\|^2 = \|\mathbf{C}_a\mathbf{y}\|^2$ More formally, their proposed algorithm signals a volume anomaly if

$$\mathbf{SPE} = \|\mathbf{C}_a\mathbf{y}\|^2 > Q_\alpha, \tag{4.1}$$

where Q_α denotes the threshold statistic for the $\mathbf{SPE}$ residual function at the $1-\alpha$ confidence level. Such a statistical test for the $\mathbf{SPE}$ residual function, known as the Q-statistic [34], can be computed as a function $Q_\alpha = Q_\alpha(\lambda_{k+1}, \ldots, \lambda_n)$, of the $(n-k)$ non-principal eigenvalues of the covariance matrix $\mathbf{A}$. With the computed Q_α, this statistical test can guarantee that the false alarm probability is no more than α (under certain assumptions).

4.2.3 Distributed Detection

Shifting from a centralized to a distributed anomaly detection algorithm raises some important issues. First, we want to understand which functionality can be pushed from the coordinator to the monitors, so as to engage them more actively in the detection process. In our system, this will involve "smart" filtering. The second issue is to ensure that even in this distributed system, and with a discrepancy between its view and the true network state, the coordinator can still fire the trigger on the global system aggregate accurately. There are at least two significant obstacles to extending the subspace method to a distributed setting:

1. With minimal communication overhead, maintain projection matrix $\mathbf{C}_a$ while matrix $\mathbf{Y}$ (formed by distributed link measurements) evolves over time.

2. With minimal communication overhead, track and fire triggers to indicate anomalies when $\|\mathbf{C}_a\mathbf{y}\|^2 > Q_\alpha$.

Maintaining the subspace projection matrix $\mathbf{C}_a$ in a distributed way is difficult, because computing $\mathbf{C}_a = \mathbf{I} - \mathbf{P}\mathbf{P}^T$ is equivalent to solving the symmetric eigenvalue problem for the

covariance matrix $\mathbf{Y}^T\mathbf{Y}$, which involves quadratic terms of measurement data from all links. The stability of matrix $\mathbf{P}$ is a function of the stability of the network's traffic matrix, and impacts how often $\mathbf{C}_a$ needs to be updated. There is some indication that traffic matrices are stable for up to 5 day periods (weekdays) [42]. However, in general, it is assumed that the matrix $\mathbf{P}$ will need to be updated frequently (although the exact frequency is unclear). In this chapter, we assume a stable $\mathbf{P}$ and choose to focus on obstacle 2. We design a novel method for tracking the principal components in Chapter 5.

Given a relatively stable projection matrix $\mathbf{C}_a$, it is still not easy to compute the distributed function $\|\mathbf{C}_a\mathbf{y}\|^2$ and check whether it is above the threshold Q_α in a communication-efficient way. This is because $\|\mathbf{C}_a\mathbf{y}\|^2$ is a quadratic function and involves the cross-product of measurements from different links (i.e., it has terms like $r_i \cdot r_j$ for $i \neq j$). It is unclear how local link measurement r_i impacts $\|\mathbf{C}_a\mathbf{y}\|^2$ without knowing the measurements from other links. Our way to tackle this issue is to use the first order approximation of the quadratic function. One can compute the partial derivative of $\|\mathbf{C}_a\mathbf{y}\|^2$ w.r.t. r_i, which is the marginal factor of r_i on $\|\mathbf{C}_a\mathbf{y}\|^2$

$$g_i := \frac{\partial\|\mathbf{C}_a\mathbf{y}\|^2}{\partial r_i} = \frac{\partial\left(\sum_{j=1}^{n}\left(\mathbf{y^T\tilde{c}_j}\right)^2\right)}{\partial r_i} = 2\mathbf{y}^T\mathbf{C}_a\mathbf{\tilde{c}_i}, \tag{4.2}$$

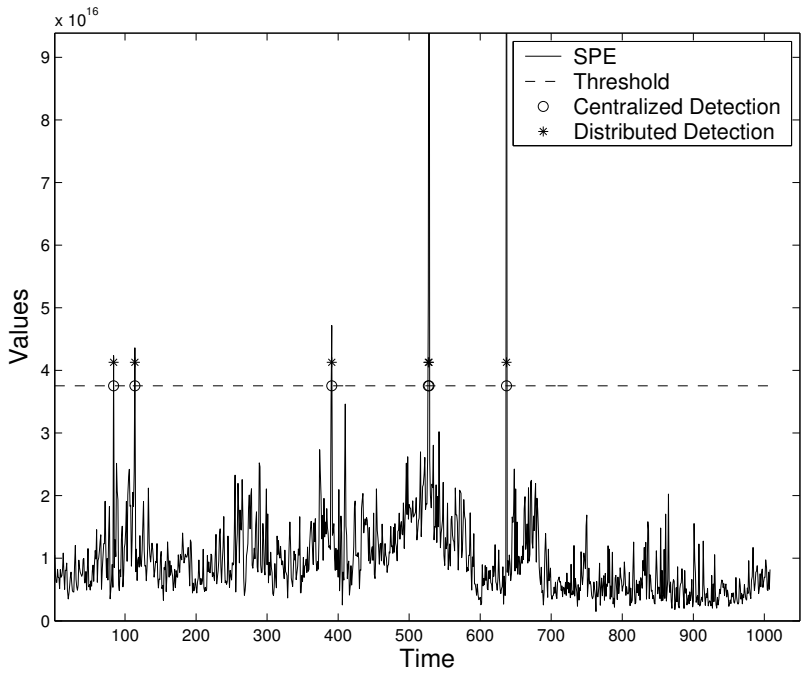
where $\tilde{\mathbf{c}}_i$ is the i-th column of matrix $\mathbf{C}_a$. If we ignore second order terms, we can see that if r_i changes by 1 unit, then $\|\mathbf{C}_a\mathbf{y}\|^2$ would change by a factor of $g_i = 2\mathbf{y}^T\mathbf{C}_a\mathbf{\tilde{c}_i}$ units. The coordinator can compute these derivatives g_i, because it has all information needed to do so. The coordinator sends each monitor i, its partial derivative g_i at the same time it sends the local slack δ_i. We can

now describe the functionality at the monitors and coordinator as follows.

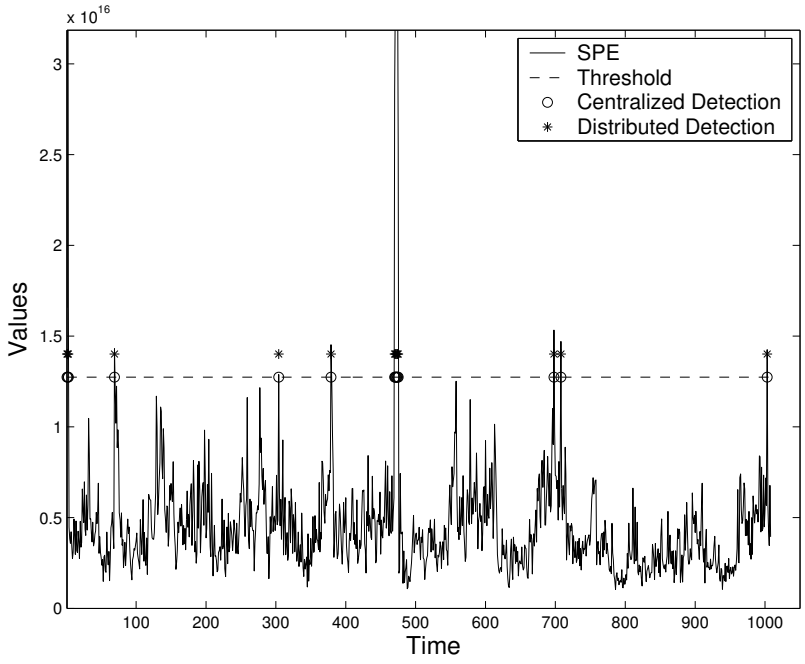
Each monitor M_i tracks the change of its $r_i(t)$ and sends the coordinator updates whenever $|r_i(t) - R_i(t)| > \frac{\delta_i}{|g_i|}$. Even though each monitor cannot see the data from other monitors, in this way, it can filter its traffic not only based upon how far the data is from its own most recent prediction (as in earlier triggers), but also according to the relative impact of a particular monitor's data on the aggregation function, relative to other monitors.

The coordinator computes $\|\mathbf{C}_a\mathbf{y}\|^2$ and triggers an alarm indicating an anomaly if $\|\mathbf{C}_a\mathbf{y}\|^2 > Q_\alpha$. Second, it continuously computes $\Delta = \max(\epsilon, Q_\alpha + \epsilon - \|\mathbf{C}_a\mathbf{y}\|^2)$, based on which it computes δ_i's. Third, it computes the partial derivatives g_i's according to Eqn.4.2, and disseminates the parameters (δ_i, g_i) the monitors. To maintain system stability, the coordinator uses low-pass filtering techniques (e.g., based on discretization intervals as discussed in Chapter 3) to avoid disseminating new parameters for small, transient changes in Δ.

We justify using a first order approximation as follows: 1) $\|\mathbf{C}_a\mathbf{y}\|^2$ is a quadratic function of $r_i(t)$'s and has terms only up to the second order; 2) the approximation is only used to determine when to send $r_i(t)$ values to the coordinator, thus bounding the difference between $r_i(t)$ and $R_i(t)$. Once $r_i(t)$ is updated, the coordinator uses $\|\mathbf{C}_a\mathbf{y}\|^2$ for calculations; 3) when δ_i is small, which is the case as $\|\mathbf{C}_a\mathbf{y}\|^2$ approaches the threshold, $|r_i(t) - R_i(t)|$ is small and its high order is even smaller. Thus, the accuracy is sufficient for our detection purposes when using only the first order of $\|\mathbf{C}_a\mathbf{y}\|^2$ to control updates. Our experiments show that this approximation is accurate and does not introduce detection errors.



(a) *Week a*



(b) *Week b*

Figure 4.2: Distributed detection on the timeseries of $\mathbf{SPE} = \|\tilde{\mathbf{C}}\mathbf{y}\|^2$ with $\epsilon = 0$.

ϵ	Missed Detections		False Alarms		Comm. Overhead	
	week a	week b	week a	week b	week a	week b
0.00	0	0	0	0	0.13	0.30
0.05	0	1	1	0	0.12	0.24
0.10	0	1	0	0	0.10	0.21
0.15	0	1	0	0	0.10	0.19

Table 4.1: *Detection error vs. communication overhead. Week a has 6 anomalies and week b has 15 anomalies.*

4.2.4 Evaluation

For a preliminary validation for our approach, we used two one-week long Abilene network traffic matrices, collected in 10 minute intervals on 41 individually monitored links. We set the threshold Q_α to a $1 - \alpha = 99.5\%$ confidence level, and set $\epsilon = 0$. The results are shown in Figure 4.2. The solid curve is **SPE**, the timeseries of $\|\mathbf{C}_a\mathbf{y}\|^2$, and the dashed line is the threshold Q_α. Note that our distributed algorithm (star points) detects all anomalies (6 in week a and 15 in week b) that are detected by the centralized algorithm (circle points). Examining the timeseries values of $\|\mathbf{C}_a\mathbf{y}\|^2$, we find that the signal values of anomalies computed by our distributed algorithm are exactly the same as those computed by the centralized algorithm, when setting $\epsilon = 0$. These results demonstrate that our approximation up to the first order of the **SPE** function is accurate.

Table 4.1 shows the tradeoff between triggering accuracy ϵ, and missed detections, false alarms, and communication overhead (including messages from monitors to coordinator and from coordinator to monitors). When varying ϵ from 0.00 to 0.15, our distributed algorithm has low detection error (at most 1 missed detection/false alarm), and incurs modest communication overhead, ranging from 13% to 10% of original data for week a, and 30% to 19% of original data for week b. While using only 13% (or 30%) of original data, our distributed algorithm is as equally effective as the centralized algorithm. We hypothesize that this per-node communication overhead remains

stable as the network size increases.

4.3 Chapter Summary

We have presented our novel approach to extending simple threshold triggers for sophisticated anomaly detection problems. Through a set of examples, we have shown that D-Trigger is an efficient and extensible vehicle for advanced detection algorithms, and discussed our general extensions to existing triggering protocols to support wide-range of detection tasks. In particular, we designed a distributed protocol that can perform online detection of network-wide anomalies with modest communication overhead, with the assumption that the dominant traffic pattern does not change over time. In the next chapter, we further extend this line of research to design a concrete solution to efficiently update traffic principal components in a dynamic environment, and perform network-wide anomaly detection without any assumption about the dominant traffic pattern.

Chapter 5

Online Detection of Network-Wide Anomalies

Today's large distributed systems (e.g., server clusters, large Internet Service Providers (ISPs), and enterprise networks) employ distributed monitoring infrastructures to collect and aggregate information describing system status and performance. An example application is one that seeks to detect network-wide traffic anomalies. Recent work by Lakhina *et al.* [41] proposes an anomaly detection scheme in which monitors ship observations to a central Network Operations Center (NOC), which in turn assembles and analyzes the data to perform anomaly detection. Instead of performing detection at local nodes for node-level anomalies, they developed a network-wide anomaly detection scheme based on Principal Component Analysis (PCA). They showed that the minor components of PCA (the subspace obtained after removing the components with largest eigenvalues) revealed anomalies that were not detectable in any single node-level trace.

However, this work assumes that local monitors continuously measure the total volume of

traffic (in bytes) on each network link, and periodically push all recent measurements to the NOC for anomaly detection. Such a solution cannot scale either for networks with a large number of monitors nor for networks seeking to track and detect anomalies at very small time scales. Our solution proposed in Chapter 4 only touches the surface of decentralized anomaly detection using the PCA-based method: it built upon the strong assumption that principal components of network traffic do not change over time, which is not realistic in many situations; neither does it proposed feasible solution for updating principal components when network traffic evolves over time.

We are thus motivated to study how well principal components can be approximated and traffic anomalies can be detected if only a portion of the monitored data is shipped to the NOC. Using D-Trigger as an efficient and extensible vehicle for advanced anomaly detection, we aim to recast the ideas of Lakhina, *et al.* in our D-Trigger framework and design a solution that detects anomalies at a desired accuracy level with low communication cost. In the D-Trigger framework, we engage the monitors in *local* filtering so that they only send data to the NOC on an "as-needed" basis. The NOC (often referred to as a *coordinator* hereafter) guides the monitors in how to do the filtering because it sees the global data and knows, via the triggering condition, the extent of dependencies across different monitors. With the guideline, the distributed monitors collect data continuously but each monitor only updates the coordinator with new data as needed (determined by the filtering parameter). Monitors can do so at any moment in time, and are not restricted to time window boundaries (such as every 5 minutes). Because the NOC will find out anything it "needs" to know immediately (ignoring network delays), the NOC is effectively doing continuous tracking, which in turn enables real-time detection.

Our approach demonstrates a classical example for applications of machine learning meth-

ods in distributed networking systems. One of the most interesting aspects of such applications is that learning algorithms that are embedded in a distributed computing infrastructure are themselves part of that infrastructure and must respect its inherent local computing constraints (e.g., constraints on bandwidth, latency, reliability, etc.), while attempting to aggregate information across the infrastructure so as to improve system performance (or availability) in a global sense. Designing a scalable solution presents several algorithmic challenges. Viable solutions need to process data "in-network" to intelligently control the frequency and size of data communications. The key underlying problem is that of developing a mathematical understanding of how to trade off quantization arising from local data filtering against fidelity of the detection analysis. We also need to understand how this tradeoff impacts overall detection accuracy. Finally, the implementation needs to be simple if it is to have impact on developers.

Our approach for communication-efficient online detection is novel and unusual. It extends the power of the PCA-based method by coupling insights from *Stochastic Matrix Perturbation (SMP) theory* together with in-network processing ideas [16, 47]. Because we filter locally at the distributed monitors, the NOC's view of the global data (captured in a matrix) is approximate since elements in the matrix can become out-of-date. Thus the computation of the principle components is done on a *perturbed* data matrix. We appeal to Matrix Perturbation theory as it helps to quantify the effect of such perturbations on the computation of eigenvectors and eigenvalues. Out-of-date data can lead to errors that propagate through the anomaly detector including not only the eigenvalues, but also the anomaly trigger thresholds – because all of these are data-driven. This results in an anomaly detector that can make mistakes. Using SMP theory, we derive analytic bounds on the terms affected by error propagation. We design an algorithm that derives filtering parameters for

the monitors, such that the errors made by the detector are bounded. Our algorithm combines many techniques together, SMP theory, binary search, and Monte Carlo simulation.

Our evaluation using real-world data streams collected from a well known ISP network shows that our methods work very well. While sending less than 10% of the original time-series data (over an order-of-magnitude reduction in communication), we guarantee that the detection error would be no more than 4% bigger than when using full data. In fact, our system performs much better than these bounds; we find that the actual error rates are nearly indistinguishable from the full-data method. This results in a huge savings in communication overhead (e.g., typically 80 or 90% of the original data is no longer sent) with only a very small impact on errors. Put another way, within the same (fixed) communication budget, our algorithms can allow for a ten-fold increase in the time granularity of network-statistics collection. Finally, we show that our system can indeed scale gracefully as the number of monitors grows.

Chapter Organization. The rest of the chapter is organized as follows: we present the problem description in Chapter 5.1; we describe our protocols for efficient online tracking PCA and detecting network anomalies Chapter 5.2; we discuss our algorithm for system parameter design and error guarantee in Chapter 5.3; we evaluate our approach in Sec. 5.4; finally, we summarize the chapter in Chapter 5.5.

5.1 Communication Efficient Detection Problem

The Centralized PCA method for network-wide traffic volume anomaly detection is summarized in Chapter 4. The problem we address here is how to do the filtering at the monitors, so as to send as little data as possible through the network but still allow the anomaly detector to work

accurately. The idea is that monitors should send a description of their time series signal, and then not send any more measurements (or summaries) until a change happens that is either "sufficiently large" or likely to impact the global trigger condition being monitored. In our application, the trigger condition being monitored by the coordinator is that in Eqn (4.1). Because the monitors send data less frequently to the coordinator, the coordinator's view of the global network data can be out-of-date and perturbed. Thus, the statistics it computes for anomaly detection, such as the eigenvalues and projection matrix, will deviate from those of the true global state. This implies that the detection error at the coordinator (when triggering on condition (4.1)) will differ from that achieved using the full data.

Our solution includes the design of protocols used by the monitors and coordinator, and an algorithm to determine how to do the appropriate filtering. We allow the user (network operator) to input the tolerable *deviation μ of the false alarm probability*, a parameter that specifies how much the false alarm probability achieved by our approximation technique is permitted to deviate from the false alarm probability achieved by the centralized-data solution. We provide an algorithm for computing each monitor's filtering parameter that guarantees that our false alarm probability does not deviate by more than the specified deviation μ. In order to guarantee an error performance within μ, we need to track and limit the perturbations in the system caused by the local filtering at the monitors. This amounts to bounding the perturbations of the eigenvalues λ_i, projection matrix $\mathbf{C}_a$ and trigger threshold Q_α (all of which get perturbed due to error propagation that occurs with out-dated measurement data). In this paper, we show that all of these system component errors can be bounded, and thus excellent detection can still be achieved, even with a substantial reduction in data transmitted to the coordinator.

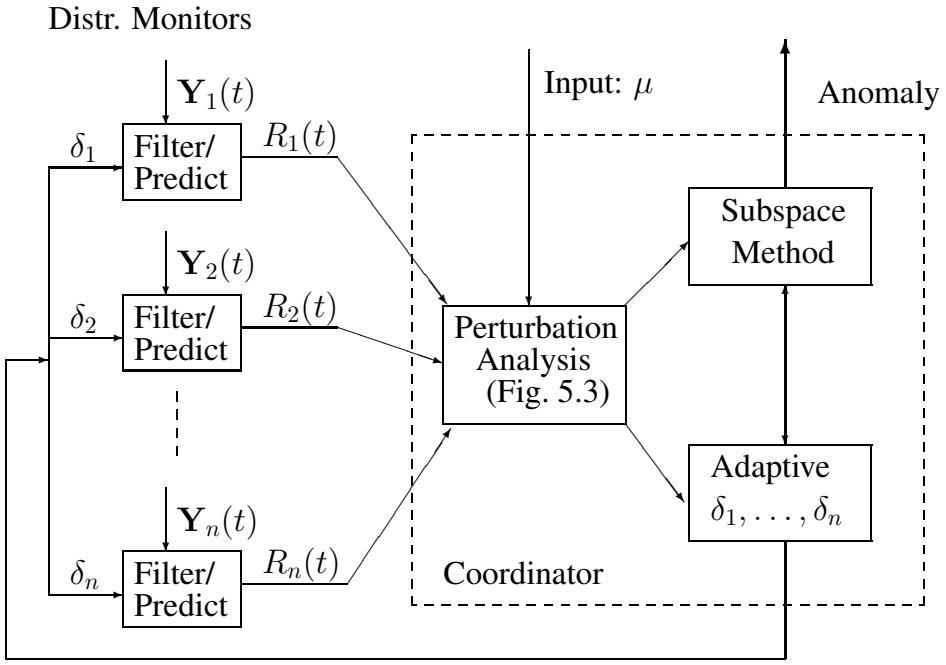


Figure 5.1: Distributed detection system.

Our filtering parameters are both heterogeneous (across different monitors) and adaptive in time. Intuitively, the selection of the filtering parameter at a monitor should take into account two things: the variability of the local time series data itself, and the marginal impact this particular data has on the global trigger condition relative to other data streams. We thus aim to do filtering locally at monitors using parameters that are derived based upon global correlations across different data streams and their joint impact on the trigger condition being tracked.

5.2 Our Approach

The architecture of our system is depicted in Fig. 5.1. Our approach consists of two parts: (1) the monitors process their collected data by applying local filtering to suppress unnecessary message updates to the coordinator; and (2) the coordinator makes global decisions and provides

feedback to the monitors (e.g., local filter parameter settings) based on the observed updates.

We use $\mathbf{Y}_i(t)$ to denote the actual time series observed at monitoring node M_i, which is one column vector of data matrix $\mathbf{Y}$. We use $R_i(t)$ to denote the approximate representation of $\mathbf{Y}_i(t)$ that is sent to the coordinator. If no further data is sent shortly after time t, the coordinator assumes that $R_i(t)$ serves as a *prediction* of the true data at these latter time instances. A simple prediction model might set $R_i(t)$ to the latest $\mathbf{Y}_i(t)$ value communicated from the site, or an average of recent communications, but more sophisticated prediction models [16, 35] can be used. Our techniques remain applicable regardless of prediction-model specifics.

Each monitor i maintains a filtering window $F_i(t)$ of size $2\delta_i$ centered at a value R_i (i.e., $F_i(t) = [R_i(t) - \delta_i, R_i(t) + \delta_i]$). At each time t, the monitor sends both $\mathbf{Y}_i(t)$ and $R_i(t)$ to the coordinator only if $\mathbf{Y}_i(t) \notin F_i$, otherwise it sends nothing. The window parameter δ_i is called the *slack*; it captures the amount the time series can drift before an update to the coordinator needs to be sent. Let t^* denote the time of the most recent update. The monitor needs to send both $\mathbf{Y}_i(t^*)$ and $R_i(t^*)$ to the coordinator when it does an update, because the coordinator will use $\mathbf{Y}_i(t^*)$ at time t^* and $R_i(t^*)$ for all $t > t^*$ until the next update arrives. For any subsequent $t > t^*$ when the coordinator receives no update from that monitor, it will use $R_i(t^*)$ as the prediction for $\mathbf{Y}_i(t)$.

The coordinator has two principal tasks: (1) to carry out anomaly detection, based on the PCA subspace method, using the inputs $R_i(t)$ it receives, and (2) to compute the filtering slacks δ_i for each monitor. The inputs to the coordinator are the deviation of false alarm probability μ, and the filtered time series. The outputs of the coordinator are a trigger that is fired whenever the condition in Eqn (4.1) is true, and the filtering parameters δ_i, that are sent to the monitors whenever they change. We will informally call the filtering parameter at a node the "slack" for

Symbol	Meaning
M_i	Monitor sites ($i = 1, \ldots, n$)
$\mathbf{Y}_i(t), \hat{\mathbf{Y}}_i(t)$	Data at monitor i and its approx. at the coordinator
$\mathbf{Y}, \hat{\mathbf{Y}}$	Data matrix at monitors and its approx. at the coordinator
$\mathbf{A}, \hat{\mathbf{A}}$	Cov. matrix at monitors and its approx. at the coordinator
$\lambda_i, \hat{\lambda}_i, \bar{\lambda}$	Eigenvalues of $\mathbf{A}$, $\hat{\mathbf{A}}$, and $\bar{\lambda} = \sum \hat{\lambda}_i / n$
$R_i(t)$	Most recent prediction model for $\mathbf{Y}_i(t)$
$\mathbf{W}_i(t), \mathbf{W}$	Filtering error time-series and matrix, $\mathbf{W} = \mathbf{Y} - \hat{\mathbf{Y}}$
$\mathbf{y}, \hat{\mathbf{y}}$	One time-step data from all monitors (one row of $\mathbf{Y}, \hat{\mathbf{Y}}$)
$\mathbf{C}_a, \hat{\mathbf{C}}_a$	The projection matrix of residual subspace
$\alpha, \hat{\alpha}$	False alarm (*i.e.*, false positive) probability
$Q_\alpha, \hat{Q}_\alpha$	The detection threshold
ϵ, ϵ^*	Tolerable and actual aggregate eigen-error
μ	Tolerable deviation of false alarm probability
δ, δ_i	Local monitor slack parameters

Table 5.1: *Notation.*

that node. The monitors use slacks when tracking the drift between the actual time series signal and the prediction function; whenever this drift exceeds the allowed slack, the monitor sends the coordinator an updated prediction, $R_i(t)$. Intuitively, these slacks are used to upper bound the difference between the coordinator's view of the data and the actual data.

The Local Monitor Protocol. Given a slack parameter δ_i, the protocol that runs at each monitor site M_i is fairly straightforward. Let $R_i(t)$ be the most recent prediction model for $\mathbf{Y}_i(t)$ sent to the coordinator. At any time t, monitor M_i continuously tracks the deviation of $\mathbf{Y}_i(t)$ from its prediction $R_i(t)$ as $\mathbf{W}_i(t) = \mathbf{Y}_i(t) - R_i(t)$, and checks the condition $|\mathbf{W}_i(t)| \leq \delta_i$. Whenever $|\mathbf{W}_i(t)| > \delta_i$, the monitor sends an update message to the coordinator that includes $\mathbf{Y}_i(t)$ and an up-to-date prediction $R_i(t)$, and resets $\mathbf{W}_i(t)$ to zero. (Table 5.1 summarizes our notation.)

The Coordinator Protocol. The global detection task at the coordinator is the same as in the centralized scheme. However, the coordinator does not have an exact version of the raw data matrix $\mathbf{Y}$; it has the approximation $\hat{\mathbf{Y}}$ instead. The connection between the slacks and the coordinator's detec-

tion accuracy comes from the following. The PCA at the coordinator is performed on a perturbed version of the covariance matrix, $\hat{\mathbf{A}} := \frac{1}{m}\hat{\mathbf{Y}}^T\hat{\mathbf{Y}} = \mathbf{A} + \mathbf{\Delta}$. The magnitude of the perturbation matrix $\mathbf{\Delta}$ is determined by the slack parameters δ_i ($i = 1, \ldots, n$). We can thus bound the perturbation of the covariance matrix through the control of the slack parameters.

The coordinator protocol works as follows. Each time t, if a new input arrives at the coordinator from some or all of the monitors, it carries out the following steps:

1. Makes a new row of data $\hat{\mathbf{y}}$ as $\hat{\mathbf{y}} = [\ \hat{\mathbf{Y}}_1(t)\ \ \hat{\mathbf{Y}}_2(t)\ \ \ldots\ \ \hat{\mathbf{Y}}_n(t)\ ]$, where $\hat{\mathbf{Y}}_i(t)$ is defined as either the update received from monitor i (if one exists), or the corresponding prediction $R_i(t)$ otherwise.

2. Updates its view of the global data $\hat{\mathbf{Y}}$, by replacing the oldest row of $\hat{\mathbf{Y}}$ using $\hat{\mathbf{y}}$.

3. Re-computes PCA on $\hat{\mathbf{Y}}$, the residual projection matrix $\hat{\mathbf{C}}_a$, and the trigger threshold $\hat{Q}_\alpha$.

4. Performs anomaly detection using $\hat{\mathbf{C}}_a, \hat{Q}_\alpha$ and $\hat{\mathbf{y}}$; fires an alarm if $\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2 > \hat{Q}_\alpha$.

The coordinator can recompute the monitor slacks either periodically or upon each monitor update. The coordinator only sends new slacks to the monitors if there is a substantial change. A high-level pseudo-code description of both the local-monitor and coordinator protocols is depicted in Fig. 5.2.

5.3 Algorithm for Filtering Parameter Selection

We now describe our method for determining the parameters used for filtering δ_i (also called *slacks*) by the local monitors. Let α denote the false alarm probability that is guaranteed by the Q_α-statistic condition in Eqn (4.1) in the original push-all solution. Similarly, $\hat{\alpha}$ denotes the

Procedure Monitor(i, δ_i)

Input: Monitor index i, local slack parameter δ_i.

1. **while** (**true**) **do**
2. t := current time
3. $\mathbf{W}_i(t) := \mathbf{Y}_i(t) - R_i(t)$
4. **if** ($|\mathbf{W}_i(t)| > \delta_i$) **then**
5. Send update message $(i, \mathbf{Y}_i(t), R_i(t))$ to coordinator
6. Set $\mathbf{W}_i(t) := 0$
7. **if** (new slack δ_i^* is received) **then**
8. Set $\delta_i := \delta_i^*$

Procedure Coordinator(μ)

Input: Deviation μ on false alarm probability.

1. **while** (**true**) **do**
2. Make a new row of data $\hat{\mathbf{y}} = \begin{bmatrix} R_1(t) & \dots & R_n(t) \end{bmatrix}$
3. Replace the oldest row of $\hat{\mathbf{Y}}$ using $\hat{\mathbf{y}}$, pointed to by $\hat{\mathbf{Y}}_i(t)$
4. **for each** (monitor update $(i, \mathbf{Y}_i(t), R_i^*(t))$ received) **do**
5. Set local prediction $R_i(t)$:= $R_i^*(t)$
6. Set $\hat{\mathbf{Y}}_i(t)$:= $\mathbf{Y}_i(t)$
7. Re-compute PCA on $\hat{\mathbf{Y}}$
8. Re-compute threshold $\hat{Q}_\alpha$, matrix $\hat{\mathbf{C}}_a$ and residual $\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2$
9. **if** ($\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2 > \hat{Q}_\alpha$) **then** **fire**("anomaly");
10. Compute new optimal settings for local slacks $\{\delta_i\}$ based on μ and maintained statistics *(Sec. 5.3)*
11. **if** (adaptive allocation) **then disseminate**($\{\delta_i\}$)

Figure 5.2: Procedures for (a) local monitor update processing, and (b) distributed detection at the coordinator.

false alarm probability of our approximation algorithm. The false alarm deviation, μ, specifies the extent to which the $\hat{\alpha}$ is allowed to increase compared to α. In particular, our goal is to determine δ_i values such that the false alarm probability $\hat{\alpha}$ of our technique satisfies $\hat{\alpha} - \alpha < \mu$, while minimizing communication cost on the network[1]. To determine the δ_i values minimizing communication for a given μ, we need to be able to quantify the effects of local monitor filtering on the observed false alarm probability.

We remind the reader that because the monitors filter their data and thus often do not send updates to the coordinator, the coordinator's matrix of the global data can have elements that are out-of-date. This perturbed view of the data propagates errors forward through a PCA-based detector as follows. First there will be errors in the computation of the eigenvalues of the covariance matrix, and second there will be errors introduced into the projection matrix (projecting onto the anomalous subspace) as well as to the Q_α threshold. The errors in these last two terms, cause errors in the trigger condition, thus causing errors in the detection accuracy. The direction of error propagation is depicted in (Fig. 5.3(a)) via the dashed lines.

Because the goal of our algorithm is to take the tolerable deviation μ of false alarm probability as input, and produce the δ_i parameters as output, we need a model of error propagation in the inverse direction to which it naturally flows. This turns out to be a non-trivial task due to the complex dependencies across different parameters in our monitoring framework. The errors in the eigenvalues are critical in our methodology as they impact all parts of the PCA-based detector. We thus elect to control the errors introduced into the eigenvalues. Let λ_i and $\hat{\lambda}_i$ ($i = 1, \ldots, n$) denote the eigenvalues of the covariance matrix $\mathbf{A} = \frac{1}{m}\mathbf{Y}^T\mathbf{Y}$, and its perturbed version $\hat{\mathbf{A}} = \frac{1}{m}\hat{\mathbf{Y}}^T\hat{\mathbf{Y}}$, re-

[1]Even though condition (4.1) is only a one-sided test, our experimental results demonstrate that our methods achieve very small missed-detection rates, similar to [41].

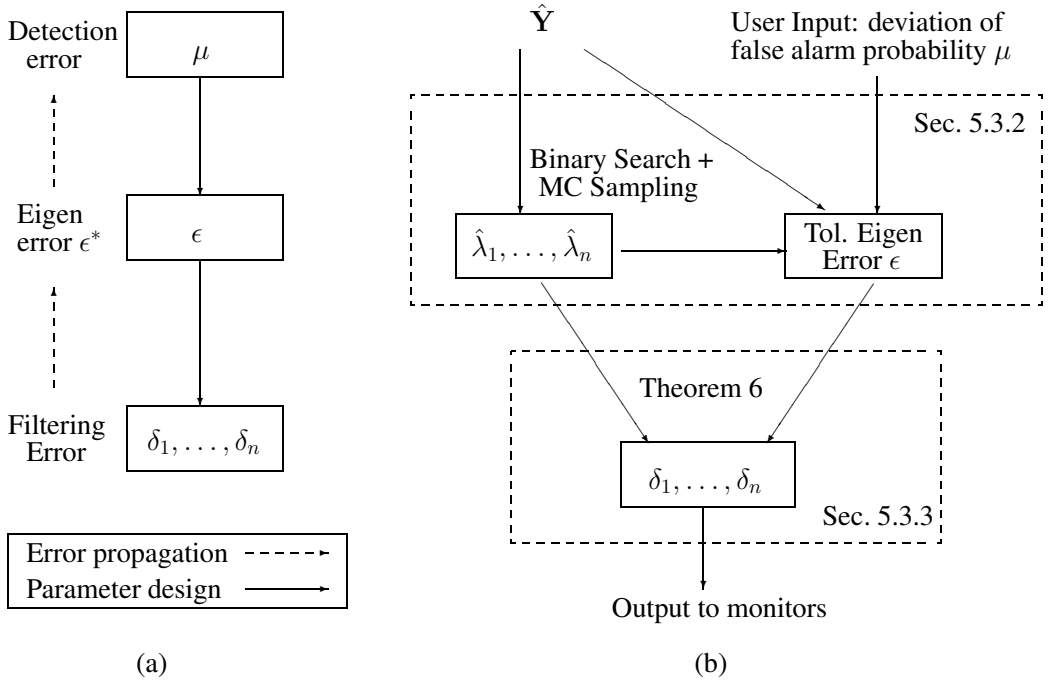


Figure 5.3: Perturbation analysis: from deviation of false alarm to monitor slacks.

spectively. We use the L_2 *aggregate eigen-error* ϵ^*, defined formally as $\epsilon^* := \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{\lambda}_i - \lambda_i)^2}$, as a metric of the errors across the set of eigenvalues. By limiting this quantity, we can limit error propagation. Our approach thus consists of a two step method: 1) given a false alarm deviation bound μ, determine an upper bound on eigen-error ϵ^*; 2) then for a given eigen-error ϵ^*, find monitor slacks δ_i such that eigen-error does not exceed its bound.

5.3.1 Perturbation Analysis

Our perturbation analysis goes as follows. We measure the size of a perturbation using a norm on $\mathbf{\Delta}$. We derive an upper bound on the changes to the eigenvalues λ_i and the residual subspace $\mathbf{C}_a$ as a function of $\|\mathbf{\Delta}\|$. We choose δ_i to ensure that an approximation to this upper bound on $\mathbf{\Delta}$ is not exceeded. This in turn ensures that λ_i and $\mathbf{C}_a$ do not exceed their upper bounds.

Controlling these latter terms, we are able to bound the false alarm deviation.

Recall that the coordinator's view of the global data matrix is the perturbed matrix $\hat{\mathbf{Y}} = \mathbf{Y} + \mathbf{W}$, where all elements of the column vector $\mathbf{W}_i$ are bounded within the interval $[-\delta_i, \delta_i]$. Let λ_i and $\hat{\lambda}_i$ $(i = 1, \ldots, n)$ denote the eigenvalues of the covariance matrix $\mathbf{A} = \frac{1}{m}\mathbf{Y}^T\mathbf{Y}$ and its perturbed version $\hat{\mathbf{A}} := \frac{1}{m}\hat{\mathbf{Y}}^T\hat{\mathbf{Y}}$. Applying the classical theorems of Mirsky and Weyl [60], we obtain bounds on the eigenvalue perturbation in terms of the Frobenius norm $\|.\|_F$ and the spectral norm $\|.\|_2$ of $\mathbf{\Delta} := \frac{1}{m}(\mathbf{A} - \hat{\mathbf{A}})$, respectively:

$$\epsilon^* := \sqrt{\sum_{i=1}^{n} \frac{1}{n}(\hat{\lambda}_i - \lambda_i)^2} \leq \|\mathbf{\Delta}\|_F/\sqrt{n} \quad \text{and} \quad \max_i |\hat{\lambda}_i - \lambda_i| \leq \|\mathbf{\Delta}\|_2, \tag{5.1}$$

where

$$\|\mathbf{\Delta}\| = \frac{1}{m}\|\mathbf{Y}^T\mathbf{W} + \mathbf{W}^T\mathbf{Y} + \mathbf{W}^T\mathbf{W}\|. \tag{5.2}$$

Applying the sin theorem and results on bounding the angle of projections to subspaces [60] (see the Appendix for more details), we can bound the perturbation of the residual subspace $\mathbf{C}_a$ in terms of the Frobenius norm of $\mathbf{\Delta}$:

$$\|\mathbf{C}_a - \hat{\mathbf{C}}_a\|_F \leq \frac{\sqrt{2}\|\mathbf{\Delta}\|_F}{\nu}. \tag{5.3}$$

where ν denotes the eigengap between the k^{th} and $(k+1)^{th}$ eigenvalues of the approximated covariance matrix $\hat{\mathbf{A}}$.

To obtain a practical (i.e., computable) bound on the norms of $\mathbf{\Delta}$, we make the following assumptions on the error matrix $\mathbf{W}$:

1. The column vectors $\mathbf{W}_1, \ldots, \mathbf{W}_n$ are independent and radially symmetric m-vectors.

2. For each $i = 1, \ldots, n$, all elements of column vector $\mathbf{W}_i$ are i.i.d. random variables with mean 0, variance $\sigma_i^2 := \sigma_i^2(\delta_i)$ and fourth moment $\mu_i^4 := \mu_i^4(\delta_i)$.

Note that the independence assumption is on the single-monitor errors only – this by no means implies that the signals received by different monitors are statistically independent. The *error* variance $\sigma_i^2 := \sigma_i^2(\delta_i)$ is a function of the corresponding monitor slack because the slack determines the size of the allowed drift (or discrepancy), between the true data and the coordinator's view of the data, before the coordinator needs an update. Under the above assumption, we can show that $\|\mathbf{\Delta}\|_F/\sqrt{n}$ is upper bounded in expectation by the following quantity:

$$\epsilon = 2\sqrt{\frac{1}{mn}\sum_{i=1}^{n}\lambda_i \cdot \sum_{i=1}^{n}\sigma_i^2} + \sqrt{\left(\frac{1}{m}+\frac{1}{n}\right)\sum_{i=1}^{n}\sigma_i^4 + \frac{1}{mn}\sum_{i=1}^{n}\left(\mu_i^4 - \sigma_i^4\right)}. \tag{5.4}$$

Similar results can be obtained for the spectral norm as well. In practice, these upper bounds are very tight because $\sigma_1, \ldots, \sigma_n$ tend to be small compared to the top eigenvalues.

5.3.2 Step 1: From false alarm deviation to eigen-error

Unfortunately, there is no closed-form solution for determining the tolerable eigen-error ϵ given a desired bound on the false alarm probability $\hat{\alpha}$. As mentioned earlier, errors in the eigenvalues and eigenvectors propagate through $\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2$ and the threshold $\hat{Q}_\alpha$, thus affecting the trigger condition $\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2 > \hat{Q}_\alpha$, which determines the false alarm deviation μ.

From our observations, μ is typically monotonically increasing in ϵ; this matches our intuition as larger perturbations to eigenvalues naturally imply higher false alarm probabilities. Thus, given an efficient method for computing μ for a given tolerable eigen-error ϵ, our strategy is to determine ϵ for a given μ using a binary search strategy. Our search starts with an initial guess for a tolerable ϵ, and then computes our estimate for the resulting μ^*. If this is too far from our target μ, then a standard binary search procedure can be used to iteratively find a better ϵ value that satisfies our requirements on μ. A pseudo-code description of our method for estimating the eigen-error ϵ

Procedure FalseAlarmToEigenError(μ, err)

Input: Deviation μ of false alarm probability; desired approximation

factor (err) for eigen-error ϵ.

1. $\epsilon_l := 0.0$; $\epsilon_u := \bar{\lambda}$ *// search range for* ϵ
2. **while** ($(\epsilon_u - \epsilon_l) >$ err $\cdot\ \epsilon_l$) **do**
3. $\epsilon := 0.5 \cdot (\epsilon_l + \epsilon_u)$
4. $\eta_X :=$ MonteCarloSampling(ϵ)
5. $\mu^* := \mathbf{Pr}\left[c_\alpha - \eta_X < N(0,1) < c_\alpha\right]$
6. **if** ($\mu^* > \mu$) **then** $\epsilon_u := \epsilon$ **else** $\epsilon_l := \epsilon$
7. **return**(ϵ)

Figure 5.4: Procedure for estimating eigen-error given a false alarm probability deviation μ using binary search.

corresponding to a desired μ is given in Fig. 5.4. Detailed analysis is as follows.

The false alarm deviation μ is closely related to the perturbation on the trigger condition $\|\mathbf{C}_a\mathbf{y}\|^2 > Q_\alpha$. We can compute an upper bound on the perturbation of $\|\mathbf{C}_a\mathbf{y}\|^2$ as follows. First, note that

$$\begin{aligned} |\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\| - \|\mathbf{C}_a\mathbf{y}\|| &\leq \|(\hat{\mathbf{C}}_a - \mathbf{C}_a)\hat{\mathbf{y}}\| + \|\mathbf{C}_a(\mathbf{y} - \hat{\mathbf{y}})\| \leq \frac{\sqrt{2}\|\boldsymbol{\Delta}\|_F\|\hat{\mathbf{y}}\|}{\nu} + \|\mathbf{C}_a\|_2\sqrt{\sum_{i=1}^{n}\delta_i^2} \\ &\leq \frac{\sqrt{2}\|\boldsymbol{\Delta}\|_F\|\hat{\mathbf{y}}\|}{\nu} + \left(\|\hat{\mathbf{C}}_a\| + \frac{\sqrt{2}\|\boldsymbol{\Delta}\|_F}{\nu}\right)\sqrt{\sum_{i=1}^{n}\delta_i^2} \quad =: \eta_1(\hat{\mathbf{y}}). \end{aligned}$$

$$|\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2 - \|\mathbf{C}_a\mathbf{y}\|^2| \leq \eta_1(\hat{\mathbf{y}})(2\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\| + \eta_1(\hat{\mathbf{y}})) =: \eta_2(\hat{\mathbf{y}}). \tag{5.5}$$

To estimate μ for a particular ϵ, we consider the following random variables:

$$X = \frac{\phi_1[(\|\mathbf{C}_a\mathbf{y}\|^2/\phi_1)^{h_0} - 1 - \phi_2 h_0(h_0-1)/\phi_1^2]}{\sqrt{2\phi_2 h_0^2}}, \tag{5.6}$$

where $h_0 = 1 - \frac{2\phi_1\phi_3}{3\phi_2^2}$, $\phi_p = \sum_{j=k+1}^{n} \lambda_j^p$ for $p = 1, 2, 3$. The X random variable essentially normalizes the random quantity $\|\mathbf{C}_a\mathbf{y}\|^2$ and is known to approximately follow a standard normal distribution [37]. To perform detection on $\|\mathbf{C}_a\mathbf{y}\|^2$ with false alarm α, the threshold Q_α can be determined as a high-order complex function of $\lambda_{k+1}, \ldots, \lambda_n$ [34]:

$$Q_\alpha \;=\; \phi_1 \left[\frac{c_\alpha \sqrt{2\phi_2 h_0^2}}{\phi_1} + 1 + \frac{\phi_2 h_0(h_0-1)}{\phi_1^2} \right]^{\frac{1}{h_0}}, \tag{5.7}$$

where c_α is the $(1-\alpha)$-percentile of the standard normal distribution. Based on (5.6), we can express the false alarm probability (of the original PCA-based detector) as

$$\mathbf{Pr}\left[\|\mathbf{C}_a\mathbf{y}\|^2 > Q_\alpha\right] = \mathbf{Pr}\left[X > c_\alpha\right] = \alpha,$$

where c_α denotes the $(1-\alpha)$-percentile of a standard normal distribution.

In our approximation setting, the normalized quantity of $\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2$ is denoted by $\hat{X}$ rather than X. Let η_X denote an upper bound on $|\hat{X} - X|$. Then, the deviation of the false alarm probability in our approximate detection scheme can be estimated as

$$\mu = \mathbf{Pr}\left[c_\alpha - \eta_X < N(0,1) < c_\alpha\right], \tag{5.8}$$

where $N(0,1)$ denotes a standard normal random variable. A key issue here is how to estimate the η_X upper bound on $|\hat{X} - X|$. Our approach is to use a *Monte Carlo (MC) sampling* technique to obtain observations of the $|\hat{X} - X|$ random variable, and use the maximum of these observations as an estimate of η_X.

Monte Carlo Sampling For Estimating η_X. In principle, η_X, the perturbation of X, could be analytically derived from the perturbation of the eigenvalues and $\|\mathbf{C}_a\mathbf{y}\|^2$ based on Eqn (5.6).

However, such an analytic approach would be cumbersome – instead, we have adopted a simple and computationally efficient alternative using Monte Carlo sampling of the perturbed values of the (non-principal) eigenvalues and $\|\mathbf{C}_a\mathbf{y}\|^2$ to extract the upper bound η_X. The procedure is summarized as follows:

1. For each non-principal eigenvalues λ_i, we randomly generate samples from interval $[\hat{\lambda}_i - \epsilon, \hat{\lambda}_i + \epsilon]$.

2. We may bound the perturbation on the $\|\mathbf{C}_a\mathbf{y}\|^2$ term. In our data set, compared to the eigenvalue perturbation, this perturbation has much less impact on X. For computational simplicity, in place of this term we simply use the average value of timeseries $\|\hat{\mathbf{C}}_a\hat{\mathbf{y}}\|^2$ for the estimation.

3. Plug the above generated elements into Eqn (5.6) to compute the perturbed statistics $\hat{X}$. This procedure of sample generation and plug in calculation is repeated 1000 times to generate 1000 instances of $\hat{X}$, based on which we compute $\eta_X = \max\{|\hat{X} - X|\}$.

5.3.3 Step 2: From tolerable eigen-error to monitor slacks

After we derive an upper bound on eigen-error ϵ^*, we need to find a set of monitor δ_i such that the eigen-error does not exceed its bound. This can be achieved by analyzing the norms of $\mathbf{\Delta}$ discussed in Section 5.3.1 Let $\bar{\lambda} := \frac{1}{n}\sum\hat{\lambda}_i$ denote the average of the perturbed eigenvalues of $\hat{\mathbf{A}}$. Based on the Equation 5.4, we can further prove the following theorem relating monitor slacks δ_i to an upper bound of the aggregate eigen-error ϵ^*.

Theorem 6 *Under the independent assumptions on filtering errors, setting $\delta_1, \ldots, \delta_n$ to satisfy*

$$2\sqrt{\frac{\bar{\lambda}}{m} \cdot \sum_{i=1}^{n} \sigma_i^2} + \sqrt{\left(\frac{1}{m} + \frac{1}{n}\right) \sum_{i=1}^{n} \sigma_i^4} = \epsilon \tag{5.9}$$

guarantees that $\epsilon^* \leq \epsilon$ with probability $\geq 1 - o(\frac{1}{m^3})$.

(We refer to ϵ as the *tolerable eigen-error* in what follows.) Given a tolerable eigen-error ϵ as input, we can then solve for the slacks δ_i using the equation in Theorem 6. However to do so, we need to quantify the relationship between error variances σ_i and local slacks δ_i. We now discuss different techniques employed in our system for this purpose.

Homogeneous Slack Allocation: Uniform Distribution Method. A simple method that often works well in practice is to assume that filtering errors are independently and uniformly distributed in $[-\delta_i, \delta_i]$. This gives a closed form for local variances: $\sigma_i = \frac{\delta_i^2}{3}$. Assuming homogeneous slack allocation, that is, all monitors share the same slack $\delta_i = \delta$, we can directly solve Eqn. (5.9) for δ:

$$\delta = \frac{\sqrt{3\bar{\lambda}n + 3\epsilon\sqrt{m^2 + m \cdot n}} - \sqrt{3\bar{\lambda}n}}{\sqrt{m+n}}.$$

Homogeneous Slack Allocation: Local Variance Estimation Method. In some cases, the uniform-distribution assumption for filtering errors may be unrealistic. A more accurate method is to estimate local error variances $\sigma_i(\delta)$ directly from the data. Variance estimation is performed locally (at each monitor) by fitting a (e.g., quadratic) function of δ using a recent window of observations. These local functions are sent to the coordinator (either periodically or on-demand), and plugged into Eqn. (5.9) to solve for a new δ. While imposing some additional overhead on the network and local monitors, this method avoids possibly unrealistic uniformity assumptions on the monitor data.

Heterogeneous Slack Allocation. We now consider allowing the local slacks $\delta_1, \ldots, \delta_n$ to differ from one another, and to dynamically adapt to local stream characteristics. Let the message update

frequency (a direct measure of communication cost) of each monitor M_i be a function $f_i(\delta_i)$. Then, assuming each slack takes on a random value uniformly in the range $[-\delta_i, \delta_i]$, we can formalize heterogeneous slack allocation as the following optimization problem:

$$\text{Minimize} \quad \sum_{i}^{n} f_i(\delta_i) \quad \text{such that} \quad 2\sqrt{\frac{\bar{\lambda}}{m} \cdot \sum_{i=1}^{n} \frac{\delta_i^2}{3}} = \epsilon,$$

where the second summand in Eqn. (5.9) is ignored, since it is typically an order of magnitude smaller than the first. We used the method in [47], based on Lagrangian multipliers to solve for the optimal slack allotments. Although heterogeneous slack values are intuitively appealing, they often bring marginal benefit over homogeneous allocations. We will see this to be the case in our evaluations as well.

5.4 Evaluation

5.4.1 Evaluation Methodology and Metrics

We implemented our system and developed a trace-driven simulator to validate our methods. The real world traffic, used as input to our simulator, comes from the Abilene network. We used four one-week traces of router-to-router origin-destination (OD) traffic matrices. The traces contain OD-flow traffic loads measured every 10 minutes, for all 121 flows of the Abilene network, from which we can compute the per-link traffic loads for all 41 links, using its provided routing matrix. Using a time unit of 10 minutes, data was collected for 1008 time units for each week.

To evaluate the detection accuracy of our approach, we synthetically injected 60 anomalies and 60 non-malicious bursts[2] into the dataset using the method described in [41], so that we

[2]In [41] the authors use the term "small anomaly" to refer to events that should be ignored (not flagged) and whose detection counts as false alarms. While we use their same method for synthetic anomalies, we change the terminology to be more intuitive.

would have sufficient anomaly data to compute error rates. We used a threshold Q_α corresponding to a $1 - \alpha = 99.5\%$ confidence level. In the detection process, when any anomaly is missed, we count it as a missed detection; when any non-malicious burst is detected, we count it as a false alarm. To make the results intuitive, we define the *false alarm rate* as the fraction of false alarms over the total number of injected bursts, which is α (defined in Sec. 5.2) re-scaled as a rate rather than a probability. We define *missed detection rate* as the fraction of missed detections over the total number of injected anomalies.

In order to evaluate the scalability of our method, we had to generate synthetic traffic matrices because no traffic matrix datasets with thousands of links and tens of thousands of OD flows exist. We used the BRITE topology generator [43] to generate both sample topologies and their associated routing matrices. We considered a number of networks with anywhere from 100 to 1000 links, and up to 500×500 pairs of OD flows. For each of the $250,000$ OD flows, we generate four weeks of data based on the method discussed in [46], by extracting the relevant statistics (e.g., mean distribution, noise level, etc.) from the Abilene network traffic matrices.

5.4.2 Model Validation

For model validation, we set the input parameter for our algorithm to be the tolerable relative error of the eigenvalues ("relative eigen-error" for short), which acts as a tuning knob. (Precisely, it is $\epsilon/\sqrt{\frac{1}{n}\sum \lambda_i^2}$, where ϵ is defined in Eqn (5.9).) Given this parameter and the input data we can compute the filtering slack δ for the monitors using Eqn (5.9). We then feed in the data to run our protocol in the simulator with the computed δ. The simulator outputs a set of results including: 1) the actual relative eigen errors and the relative errors on the detection threshold Q_α; 2) the missed detection rate, false alarm rate and communication cost achieved by our method.

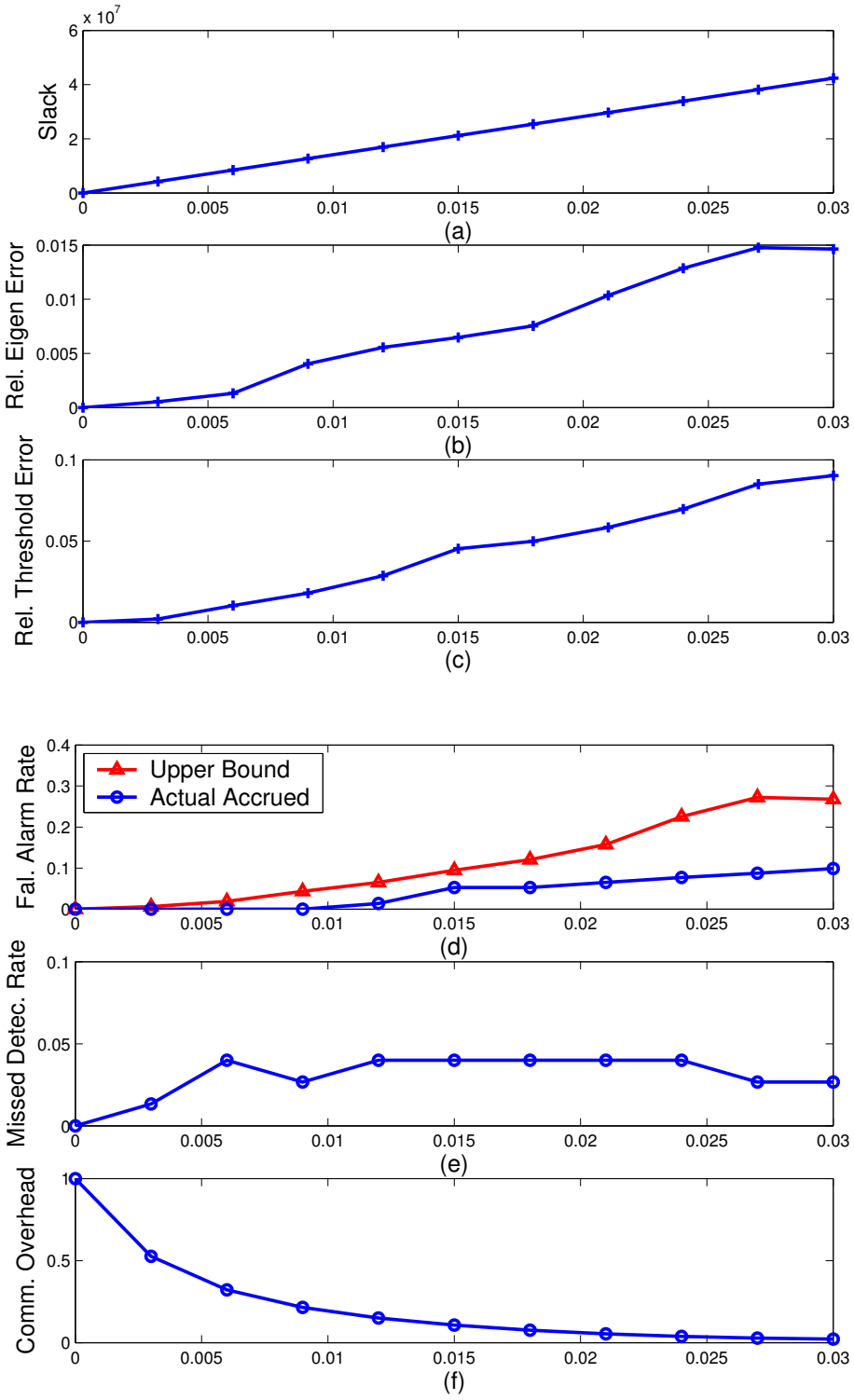


Figure 5.5: In all plots the x-axis is the relative eigen-error. (a) The filtering slack. (b) Actual accrued eigen-error. (c) Relative error of detection threshold. (d) False alarm rates. (e) Missed detection rates. (f) Communication overhead.

The results are shown in Fig. 5.5. In all plots, the x-axis is the relative eigen-error. In Fig. 5.5(a) we plot the relationship between the relative eigen-error and the filtering slack δ when assuming filtering errors are uniformly distributed on the interval $[-\delta, \delta]$. With this model, the relationship between the relative eigen-error and the slack is determined by a simplified version of Eqn (5.9) (with all $\sigma_i^2 = \frac{\delta^2}{3}$). The results make intuitive sense. As we increase our error tolerance, we can filter more at the monitor and send less to the coordinator. The slack increases almost linearly with the relative eigen-error because the first term in the right hand side of Eqn (5.9) dominates all other terms.

In Fig. 5.5(b) we compare the relative eigen-error to the actual accrued relative eigen-error (defined as $\epsilon / \sqrt{\frac{1}{n}\sum \lambda_i^2}$, where ϵ is defined in Eqn (5.1)). These were computed using the slack parameters δ as computed by our coordinator. We can see that the real accrued eigen-errors are always less than the tolerable eigen errors. The plot shows a tight upper bound, indicating that it is safe to use our model's derived filtering slack δ. In other words, the achieved eigen-error always remains below the requested tolerable error specified as input, and the slack chosen given the tolerable error is close to being optimal. Fig. 5.5(c) shows the relationship between the relative eigen-error and the relative error of detection threshold Q_α[3]. We see that the threshold for detecting anomalies decreases as we tolerate more and more eigen-errors. In these experiments, an error of 2% in the eigenvalues leads to an error of approximately 6% in our estimate of the appropriate cutoff threshold.

We now examine the false alarm rates achieved by the procotol. In Fig. 5.5(d) the curve with triangles represents the upper bound on the false alarm rate as estimated by the coordinator. The curve with circles is the actual accrued false alarm rate achieved by our scheme. Note that the upper

[3]Precisely, it is $1 - \hat{Q}_\alpha / Q_\alpha$, where $\hat{Q}_\alpha$ is computed from $\hat{\lambda}_{k+1}, \ldots, \hat{\lambda}_n$.

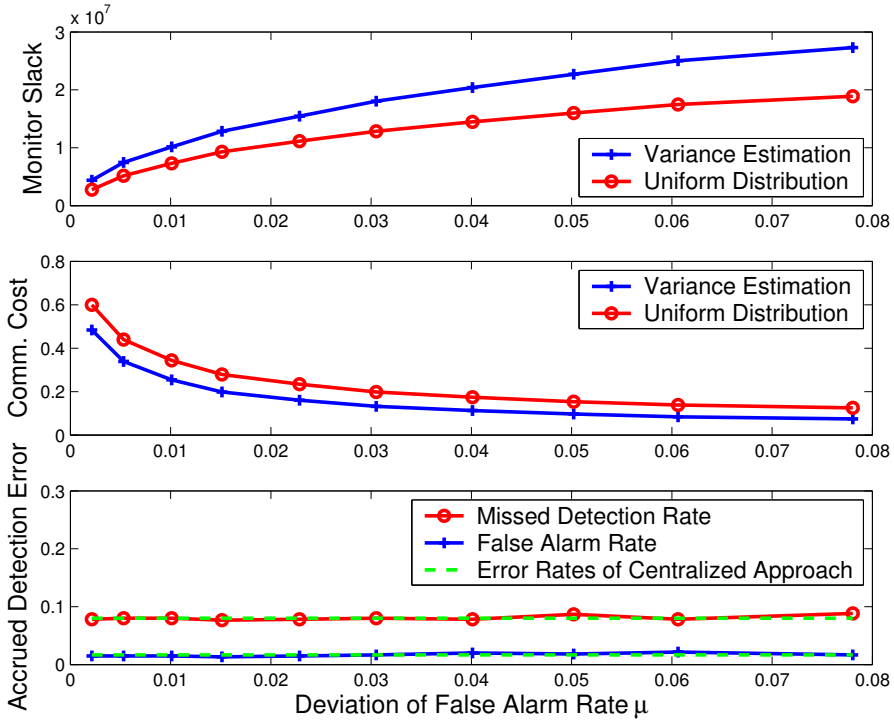
bound on the false alarm rate is fairly close to the true values, especially when the slack is small. The false alarm rate increases with increasing eigen-error because as the eigen-error increases, the corresponding detection threshold Q_α will decrease, which in turn causes the protocol to raise an alarm more often. (If we had plotted $\hat{Q}$ rather than the relative threshold difference, we would obviously see a decreasing $\hat{Q}$ with increasing eigen-error.) We see in Fig. 5.5(e) that the missed detection rates remain below 4% for various levels of communication overhead.

5.4.3 Detection accuracy vs. communication cost

We now evaluate the performance and tradeoffs of our protocols and algorithm for computing the monitor slacks. We implemented both methods of homogeneous allocation for computing the monitor slack δ: the closed-form solution relying on uniform assumptions and the variance measurement solution.

In Fig. 5.6 we consider a whole range of possible inputs on the tolerable false alarm rate deviation μ (the probability Eqn (5.8) is re-scaled to a rate). We show in the top plot the relationship between μ and the filtering slack δ, and in the middle plot the relationship between μ and communication cost. These results make intuitive sense. As we allow more error tolerance μ, we can use larger slack values and filter out more data at the monitors, and consequently reduce the amount of data sent to the coordinator. For example, when the tolerable deviation of false alarm is 5%, our algorithm reduces the data sent through the network by more than 90% when using the variance estimation method.

The bottom plot shows the actual accrued detection errors. The curve with circles depicts the missed detection rate; the curve with pluses depicts the false alarm rate; the dashed lines depict the corresponding detection errors of the centralized approach. First we point out that in all cases,



μ	Communication Cost		
	Uniform Distr.	Variance Est.	Difference
0.006	0.430	0.330	0.100
0.018	0.266	0.186	0.080
0.032	0.185	0.119	0.066
0.064	0.141	0.086	0.055
0.080	0.111	0.064	0.047

Figure 5.6: Monitor slacks, communication cost and accrued detection. The dashed line is the detection error of centralized approach with complete data.

the actual false alarm rate with our protocols is always smaller than the guaranteed bound. In other words, although we may input that we can tolerate an additional $\mu = 5\%$ errors, in fact we actually do not incur reduced accuracy, as the lower plot illustrates that our method performs nearly identically to the original subspace method in terms of false alarms and missed detections. Moreover, this nearly identical error performance can be achieved with far less data; values such as 80% or 90% less data (depending upon the particular value of μ) are typical. These results show, that for our

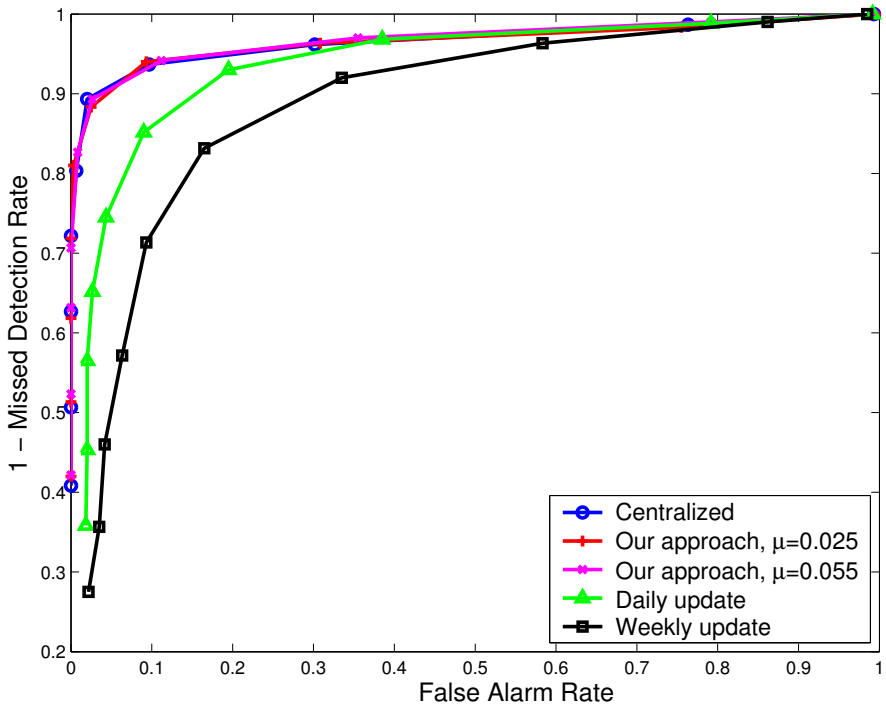
μ	Relative Eigen Error		Communication Cost		
	Homo.	Hetero.	Homo.	Hetero.	Difference
0.006	0.004	0.006	0.430	0.420	0.010
0.018	0.015	0.013	0.266	0.253	0.013
0.032	0.029	0.039	0.185	0.169	0.014
0.064	0.052	0.049	0.141	0.121	0.020
0.080	0.075	0.070	0.111	0.092	0.019

Table 5.2: *Homogeneous vs. heterogeneous slack allocation.*

dataset, the reduction in communication costs can be enormous whereas the tradeoff in terms of increased detection error is very small. These promising results confirm our hypothesis that it is not necessary to back-haul all the data for an anomaly detection problem such as [41].

In comparing the variance estimation and the uniform distribution methods for slack estimation, we see from the table in Fig. 5.6 that the measurement-based variance estimation method always performs better. The absolute difference in communication cost varied from 5% to 10% for tight requirements on μ (with μ=0.006 to 0.08, respectively). The advantage of the closed-form method is its simplicity and low computational overhead. Since, for this dataset, its performance is quite close to the measurement-based method, we conclude that such solutions might be "good enough" for many datasets.

We also implemented our heterogeneous slack allocation and compared its performance to that of the homogeneous slack allocation. In the experiment we assume filtering errors are uniformly distributed on the interval $[-\delta_i, \delta_i]$. The result is shown in Table 5.2. We found that the performance did not differ greatly (at most 3% in terms of communication costs) between the two. This indicates that the simpler solution may be good enough for the data type we consider. However the benefits of having the more general solution using heterogeneous slacks would need to be evaluated for each data type and application.



Approach	Communication Cost
Centralized, daily and weekly update	1.000
Our approach, $\mu = 0.025$	0.159
Our approach, $\mu = 0.055$	0.097

Figure 5.7: ROC curve: benefit and cost of data update approaches.

We now compare our method and the original subspace method using an ROC curve [57]. The y-axis plots the true positives (one minus the missed detections) and the x-axis depicts the false alarms. ROC curves allow one to compare two methods over a range of detection thresholds; each point on each curve corresponds to a different cutoff threshold for signaling an alarm. In general, if one curve lies entirely above and to the left of another [57], then that method is superior in that it handles the tradeoff between missed detections and false alarms better.

Because in [41] they do not indicate how often they update their PCA transform, we tested 3 variants of their method. The "centralized" version updates the principal components each time

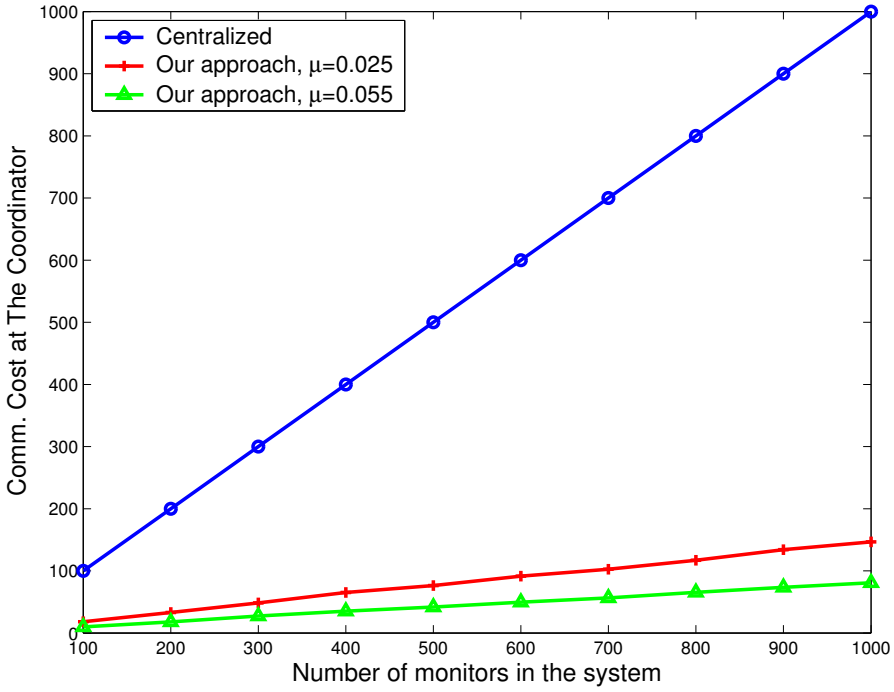
interval (upon the arrival of new data). The "daily update" version updates the principal components once a day (based on the previous 24 hours); the "weekly update" version updates the components once a week (based on the previous week). The results are shown in Fig. 5.7.

We can see from the plot that the ROC curve and detection accuracy of our approximation technique (either $\mu = 0.015$ or $\mu = 0.045$) are extremely close to that of the centralized approach. It is surprising, that using only 10% to 20% of the data, our technique has a detection ability that is essentially as good as the fully centralized approach.

This figure also indicates that it is important to keep the principal components up to date because the performance drop-off is considerable for either the daily or weekly data update cases. We point out that in our technique, the recomputation of the principle components is done less frequently than in the original algorithm (we refer here to the version in which the PCA transform is updated every time interval). This is because in any time interval (e.g., 5 minutes in this example), if none of our monitors send anything to the coordinator, then the principle components are not recomputed. There may be additional ways to reduce this computation overhead such as checking the norm of the covariance matrix and only doing updates if the change to this norm "is large enough". We leave that for future work.

5.4.4 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 cost metric we have been using until now (namely $num/n \cdot m$) is an average value for the cost *per monitor*. The communication cost coming into the coordinator is the sum of costs of all monitors, which is can be



System Size	Communication Cost (per Monitor)	
	$\mu = 0.025$	$\mu = 0.055$
200	0.155	0.088
400	0.160	0.088
600	0.152	0.083
800	0.146	0.082
1000	0.146	0.080

Figure 5.8: System Scalability.

computed from num/m. This captures the average number of messages the coordinator receives in one time slot.

We plot the communications cost at the coordinator as a function of the number of monitors in Fig. 5.8. We varied the number of monitors from 100 to 1000, and used tolerable deviation of false alarm rate $\mu = 0.025$ and $\mu = 0.055$. For each system size n, we run 5 rounds of experiments, each of which runs on n randomly picked monitors. In the Figure, in our approach, as the system

size increases: 1) the communication cost of each monitor roughly remains constant (which is the slope of the line); and 2) the communication cost at the coordinator increases linearly with system size with the slopes roughly being 0.150 ($\mu = 0.025$) and 0.088 ($\mu = 0.055$), which are far less than 1.0, the slope of the centralized approach. This result indicates that the communication cost increases slowly as system size increases, and that our system thus scales gracefully.

5.5 Chapter Summary

In this chapter we developed an approximate online scheme for PCA-based anomaly detection method, and incorporated it into our D-Trigger framework, by leveraging ideas from "in-network" processing (to engage local monitors to filter based on global conditions) combined with ideas from stochastic matrix perturbation theory. Perturbation theory is used to derive bounds on the terms in the anomaly detector that are affected by error propagation when limited data is used. We designed an algorithm to select filtering parameters so that that monitors only send data to the central tracking site "when necessary". The necessity is determined from individual traffic behaviors, correlations across traffic streams, and the global trigger tracking condition. We show that anomaly detection can still be done very accurately even when 80 or 90% of the original data is never sent to the coordinator. Thus the tradeoff between detection accuracy and communication savings is very lopsided - there is a huge reduction in communication overhead accompanied by a very small increase in errors. Moreover we illustrated that this data reduction leads to a system that scales gracefully as the number of monitors grows. In particular, we showed that the coordinator's input data rate grows more than an order of magnitude more slowly than a system that back-hauls all monitoring data for volume anomaly detection.

Chapter 6

Conclusion and Future Work

In this chapter, we conclude the dissertation by summarizing our contributions and proposing several directions for future work.

6.1 Summary of Contributions

In this dissertation, we present the D-Trigger system for continuous online anomaly detection, which gracefully integrates a variety of approximation and optimization algorithms to address the inefficiency and inflexibility in today's distributed monitoring and anomaly detection systems. Our key contributions in this work can be summarized as follows:

- In order to design and develop an efficient detection system which is capable of detecting anomalies in near real-time with bounded error requirement, our work on the D-Trigger framework makes the following contributions. First, we provide a mathematical definition of different constraint violation modes for D-Trigger, along with the design of the supporting protocols. Our system enables users to tradeoff desired detection performance with commu-

nication overhead. Second, for instantaneous and fixed-window triggers, we provide an adaptive protocol that exploits the specified trigger threshold to minimize communication while offering deterministic accuracy guarantees. Third, for cumulative triggers, we provide a principled queuing framework for analyzing the dynamic properties of protocols, and analytical solutions for finding effective queue sizes based upon the desired target detection accuracy.

- To enable D-Trigger to detect a wide range of anomaly types in distributed systems, we present our novel approach to extending simple threshold triggers for sophisticated anomaly detection problems. Through a set of examples, we have shown that D-Trigger is an efficient and extensible vehicle for advanced detection algorithms, and discuss our general extensions to existing triggering protocols to support wide-range of detection tasks. In particular, we design a distributed protocol that can perform online detection of network-wide anomalies with modest communication overhead, using PCA-based method with a strong assumption.

- Using the D-Trigger framework, we propose a novel approach for communication-efficient online detection of network-wide traffic anomalies. Our solution is unusual in that it extends the power of the PCA-based method by coupling insights from *Stochastic Matrix Perturbation (SMP) theory* together with in-network processing ideas. Because we process data locally at the distributed monitors, the NOC's view of the global data (captured in a matrix) is approximate since elements in the matrix can become out-of-date. Thus the computation of the principle components is done on a *perturbed* data matrix. We appeal to Matrix Perturbation theory as it helps to quantify the effect of such perturbations on the computation of eigenvectors and eigenvalues. Out-of-date data can lead to errors that propagate through the anomaly detector including not only the eigenvalues, but also the anomaly trigger thresholds – because

all of these are data-driven. This results in an anomaly detector that can make mistakes. Using SMP theory, we derive analytic bounds on the terms affected by error propagation. We design an algorithm that derives filtering parameters for the monitors, such that the errors made by the detector are bounded. Our algorithm combines many techniques together, SMP theory, binary search, and Monte Carlo simulation.

In summary, D-Trigger is designed with a focus on data collection for anomaly detection, and brings together the best techniques from continuous data streaming, online machine learning, and distributed signal processing. D-Trigger combines in-network processing at distributed local sites, and decision making at the NOC. The combination of distributed local processing strategies, sophisticated detection algorithms, and theoretical analysis tools enables D-Trigger to perform in-network tracking with very high detection accuracy and low communication overhead.

6.2 Future Directions

Our work to date has demonstrated that D-Trigger is an efficient, general and extensible framework for online anomaly detection. In order to enable D-Trigger to support a broader range of applications in large-scale dynamic systems, there are several interesting extensions to our work, including using a multi-level tree structure or a Peer-to-Peer topology to further reduce the processing and communication workload at the coordinator, support for more types of detection algorithms, and development of resilient monitoring infrastructures, to name a few. They are discussed in detail as follows.

6.2.1 Going Beyond a One-Level Tree Structure

In this dissertation, we develop efficient online anomaly detection schemes for monitoring systems with a one-level tree topology, which is a subset of the whole problem space. With a one-level tree structure, D-Trigger has a single coordinator responsible for firing triggers, which could possibly be a single point of failure. 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 [62] or peer-to-peer topology management [70]. Regardless of the choice of fault-tolerance mechanism, our D-Trigger scheme offers benefits and remains applicable.

Our approach can be extended to a multi-level tree structure with the following benefits: 1) further 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. However, extending our detection protocols to a multi-level tree is not easy. It is challenging to analyze the detection performance and guarantee the user-specified detection accuracy on the multi-level tree structure in the face of limited data. It would be interesting and useful future work to extend the algorithms and protocols designed for one-level tree topology on to multi-level tree and peer-to-peer topologies.

6.2.2 Supporting a Broader Range of Detection Functions

In this dissertation, we propose a decentralized detection approach that is capable of detecting anomalies identified by thresholding either simple (linear) functions, or complicated residual

components in the subspace method. Our solution detects a set of anomaly types with user-specified accuracy while minimizing communication overhead, as well as providing the flexibility for users to trade off communication overhead with detection accuracy. We believe that our model of efficient and extensible D-Trigger 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.

To tackle different detection problems under a variety of situations, the D-Trigger system needs to accommodate a wide range of correlation and decision functions. There are large sets of data mining and machine learning algorithms for sophisticated anomaly detections [23, 28], including clustering [18], classification [66], entropy analysis [65], and sequential hypothesis testing methods [45]. It is interesting but challenging future work to develop decentralized data mining and machine learning algorithms for online continuous anomaly detection. These algorithms would improve D-Trigger's detection capabilities and make it applicable to a wider range of applications.

6.2.3 Toward Fault-Tolerant and Self-Adaptive Network Systems

Monitoring and detection are not the ultimate goals. Instead, they should provide service for the target systems and help them react, adapt and evolve according to the system status. One interesting and challenging research direction is to explore the design space of fault-tolerant, self-adaptive, large-scale network systems. This would require bridging the best techniques from overlay networking protocols, machine learning algorithms, and efficient detection approaches, amongst all other things. One area we would like to explore is the construction of self-diagnosing and self-repairing networks, where member nodes use distributed protocols to perform network measurements and fault diagnosis. Nodes in the system would continuously perform self-checks on con-

sistency, securely collaborate to detect abnormal behavior, and produce human-readable reports at each administrative level of the network. Such a network would be useful for detecting and resolving failures, and providing protection mechanisms against a variety of attacks ranging from Internet worms to distributed denial of service attacks and targeted network intrusions. A key characteristic of this problem is that it requires the creation of a global view in the presence of distributed, incomplete data, which could be achieved by leveraging D-Trigger-like infrastructures.

Appendix A

Background: Matrix Perturbation Theory

This dissertation addresses the problems of approximate but accurate online anomaly detection with low overhead. In this appendix, we review a set of mathematical tools for system performance analysis and parameter calculation. Based on these set of tools, we can derive mathematical bounds on the detection errors in the face of limited available data, so that we can provide users with a fine-grained tradeoff between detection accuracy and system overhead.

We start with a few basic concepts of matrices, and then introduce matrix perturbation theory, which measures the impact of small perturbation on matrices on relevant quantities, such as the eigenvalues and eigenvectors.

A.1 Eigenvalues and Eigenvectors

For an $n \times n$ matrix $\mathbf{A}$, scalars λ and vectors $\mathbf{x}_{n\times 1} \neq 0$ satisfying $\mathbf{Ax} = \lambda\mathbf{x}$ are called *eigenvalues* and *eigenvectors* of $\mathbf{A}$, respectively, and any such pair, $(\lambda, \mathbf{x})$, is called an eigenpair for $\mathbf{A}$. The set of distinct eigenvalues, denoted by $\sigma(\mathbf{A})$, is called the *spectrum* of $\mathbf{A}$:

$$\lambda \in \sigma(\mathbf{A}) \Longleftrightarrow \mathbf{A} - \lambda\mathbf{I} \text{ is singular } \Longleftrightarrow det(\mathbf{A} - \lambda\mathbf{I}) = 0.$$

where $det(\cdot)$ denotes the determinant of a matrix.

A.2 Matrix Norms

The *Frobenius norm* of matrix $\mathbf{A}$ is defined by the equation

$$\|\mathbf{A}\|_F = \sum_{i,j} |a_{ij}|^2 = trace(\mathbf{A}^T \cdot \mathbf{A}),$$

where a_{ij} is the element at i^{th} row and j^th column in matrix $\mathbf{A}$.

The *2-norm* of matrix $\mathbf{A}$ is defined by the equation

$$\|\mathbf{A}\|_2 = \max_{\|\mathbf{x}\|_2=1} \|\mathbf{Ax}\|_2 = \sqrt{\lambda_{max}},$$

where $\mathbf{x}$ is a Euclidean vector, λ_{max} is the largest number λ such that $\mathbf{A}^T \cdot \mathbf{A} - \lambda\mathbf{I}$ is singular, i.e., λ_{max} is the largest eigenvalue of matrix $\mathbf{A}^T \cdot \mathbf{A}$.

A.3 Eigenvalue perturbation bounds

The basic perturbation bounds for eigenvalues of a matrix are due to Weyl and Mirsky given in the following two theorems [59]. Let matrix $\mathbf{A}$ have eigenvalues λ_i, and its perturbated matrix, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{\Delta}$, have eigenvalues $\hat{\lambda}_i$, for $i = 1, \ldots, n$. We have:

Theorem 7 ***(Weyl)*** $\max_i|\hat{\lambda}_i - \lambda_i| \leq \|\mathbf{\Delta}\|_2$.

Theorem 8 ***(Mirsky)*** $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{\lambda}_i - \lambda_i)^2} \leq \frac{\|\mathbf{\Delta}\|_F}{\sqrt{n}}$.

Here $\|.\|_2$ and $\|.\|_F$ denote the spectral 2-norm and the Frobenius norm (cf. [60]).

A.4 Invariant subspace perturbation

While eigenvalues are quite stable under matrix perturbation, the individual eigenvectors are not. Instead one needs to look at the perturbation of subspaces spanned by the eigenvectors. Subspaces spanned by eigenvectors are an example of *invariant* subspaces, which are known to be stable [1].

Let $\mathcal{L}(\cdot)$ denote the set of eigenvalues of a matrix, $\mathcal{S}(\cdot)$ denote the subspace spanned by a matrix, and $\mathbf{\Theta}$ denote the matrix of canonical angle between two subspaces (cf. [60]). Then the perturbation of an invariant subspace spanned by eigenvectors can be quantify by the sin of the canonical angle by the following $sin\ \Theta$ theorem [60]:

Theorem 9 *Let* $\mathbf{A}$ *have the spectral resolution*

$$\begin{bmatrix} \mathbf{X_1^T} \\ \\ \mathbf{X_2^T} \end{bmatrix} \mathbf{A} \begin{bmatrix} \mathbf{X_1} & \mathbf{X_2} \end{bmatrix} = \begin{bmatrix} \mathbf{L_1} & \mathbf{0} \\ \\ \mathbf{0} & \mathbf{L_2} \end{bmatrix},$$

where $\begin{bmatrix} \mathbf{X_1} & \mathbf{X_2} \end{bmatrix}$ *is unitary with* $\mathbf{X_1} \in \mathbf{C^{n\times k}}$. *Let* $\mathbf{Z} \in \mathbf{C^{n\times k}}$ *have orthonormal columns, and for any symmetric* $\mathbf{M}$ *of order* k*, let*

$$\mathbf{R} = \mathbf{AZ} - \mathbf{ZM},$$

[1] A subspace $\mathcal{X}$ is invariant of transformation A if $A\mathcal{X} \subset \mathcal{X}$.

and suppose that $\mathcal{L}(\mathbf{M}) \subset [\mathbf{a}, \mathbf{b}]$ *and for some eigengap* $\nu > 0$*,*

$$\mathcal{L}(\mathbf{L_2}) \subset \mathbb{R} \backslash [\mathbf{a} - \nu, \mathbf{b} + \nu],$$

then for any unitarily invariant norm

$$\| \sin \Theta \left[\mathcal{S}(\mathbf{X_1}), \mathcal{S}(\mathbf{Z}) \right] \| \leq \frac{\|\mathbf{R}\|}{\nu}.$$

Note that this theorem applies to any unitarily invariant norm such as the spectral norm $\|.\|_2$ and Frobenius norm $\|.\|_F$. Applying this result to the eigen subspaces for (symmetric) covariance matrix $\mathbf{A}$ and its perturbed version $\hat{\mathbf{A}}$, assume that $\hat{\mathbf{A}}$ has the following the spectral resolution

$$\begin{bmatrix} \mathbf{Z_1^T} \\ \mathbf{Z_2^T} \end{bmatrix} \hat{\mathbf{A}} \begin{bmatrix} \mathbf{Z_1} & \mathbf{Z_2} \end{bmatrix} = \begin{bmatrix} \mathbf{M_1} & 0 \\ 0 & \mathbf{M_2} \end{bmatrix},$$

where $\begin{bmatrix} \mathbf{Z_1} & \mathbf{Z_2} \end{bmatrix}$ is unitary with $\mathbf{Z_1} \in \mathbf{C^{n \times k}}$. Then we have $\mathbf{Z_1}^{\mathbf{T}} \hat{\mathbf{A}} \mathbf{Z_1} = \mathbf{M_1}$ and $\hat{\mathbf{A}} \mathbf{Z_1} = \mathbf{Z_1} \mathbf{M_1}$. Let $\mathbf{R} = \mathbf{A} \mathbf{Z_1} - \mathbf{Z_1} \mathbf{M_1} = \mathbf{A} \mathbf{Z_1} - \hat{\mathbf{A}} \mathbf{Z_1} = \mathbf{\Delta} \mathbf{Z_1}$. For any unitarily invariant norm, there holds $\|\mathbf{R}\| = \|\mathbf{\Delta} \mathbf{Z_1}\| = \|\mathbf{\Delta}\|$. As a result, we have:

$$\| \sin \Theta \left[\mathcal{S}(\mathbf{X_1}), \mathcal{S}(\mathbf{Z_1}) \right] \| \leq \frac{\|\mathbf{R}\|}{\nu} = \frac{\|\mathbf{\Delta}\|}{\nu}.$$

Finally, there is a close relationship between the perturbation of the projection operator onto invariant subspaces and the canonical angle of the subspace perturbation. Let $\mathbf{P}_X$ and $\mathbf{P}_Z$ be the orthogonal projections onto $\mathcal{S}(\mathbf{X})$ and $\mathcal{S}(\mathbf{Z})$. There holds [60]:

$$\|\mathbf{P}_X - \mathbf{P}_Z\|_F \quad = \quad \sqrt{2} \| \sin \mathbf{\Theta} \left[\mathcal{S}(\mathbf{X}), \mathcal{S}(\mathbf{Z}) \right] \|_F \leq \frac{\|\mathbf{\Delta}\|_\mathbf{F}}{\nu}.$$

In summary, in order to assess the perturbation in eigenvalues and eigensubspace, we need to estimate the upper bounds given in terms of the Frobenius norm and the spectral norm of Δ.

Appendix B

Proofs for Instantaneous Triggers

We present here the proofs of theorems for instantaneous triggers in Chapter 3, for both the simple protocol and the adaptive protocol.

B.1 Proof of Theorem 1

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.

B.2 Proof of Theorem 2

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 up-bounded 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 1.

- 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

$$DR \;=\; (C - \sum_i R_i(t) + \epsilon) - (\sum_i r_i(t) - \sum_i R_i(t)) = C - \sum_i r_i(t) + \epsilon.$$

This "non-update" would not cause constraint violation, because the value of $\sum_i r_i(t')$ unknown to the coordinator is at most

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

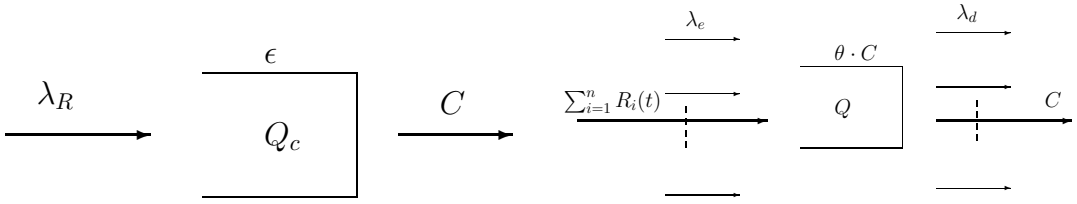
Appendix C

Proofs for Cumulative Triggers

Here we present the proofs of theorems for the cumulative triggers in Chapter 3. For the cumulative trigger, both the centralized ideal model and the distributed solution model are shown in Figure C.1. 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.

C.1 Proof of Theorem 4

Missed 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 $F(\cdot)$ denotes the CDF of $N(0, \sigma^2)$, then the probability that monitor queues have aggregate occupancy more than x is $1 - F(x)$.



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

Figure C.1: Queuing model for slack estimation.

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 the 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 [51]

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

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 missed 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 missed 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 [51]

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

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

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

C.2 Proof of Theorem 5

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

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

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

Appendix D

Perturbation Analysis for PCA-Based Method

In this appendix we develop a more detailed analysis of the impact of the slack parameter $(\delta_1, \ldots, \delta_n)$ on the eigenvalues and eigen subspaces on the principal components using matrix perturbation theory. Some of the main results presented herein are summarized in Chapter 5.3. Based on the brief background description of known results from matrix perturbation theory in Appendix A, we here proceeds to its application on our problem, and present bounds and estimation of the Frobenius norm and spectral norm of the perturbation.

D.1 Error Matrix and Assumptions

Recall that $\mathbf{A} = \frac{1}{m}\mathbf{Y}^T\mathbf{Y}$ and $\hat{\mathbf{A}} = \frac{1}{m}\hat{\mathbf{Y}}^T\hat{\mathbf{Y}}$, where $\hat{\mathbf{Y}} = \mathbf{Y} + \mathbf{W}$. $\mathbf{W}_i$ is a column vector of filtering error at each monitor i and $\mathbf{W}$ is the filtering (perturbation) error on the distributed matrix $\mathbf{Y}$. Each element e_{ji} of vector $\mathbf{W_i}$ is bounded within $[-\delta_i, \delta_i]$.

The norm of the perturbation error matrix $\mathbf{\Delta} = \frac{1}{m}(\mathbf{A} - \hat{\mathbf{A}})$ can be bounded as follows:

$$\|\mathbf{\Delta}\| = \frac{1}{m}\|\mathbf{Y}^T\mathbf{W} + \mathbf{W}^T\mathbf{Y} + \mathbf{W}^T\mathbf{W}\| \leq \frac{1}{m}\left(\|\mathbf{Y}^T\mathbf{W}\| + \|\mathbf{W}^T\mathbf{Y}\| + \|\mathbf{W}^T\mathbf{W}\|\right).$$

Our strategy is to obtain bounds for each terms in the RHS of this inequality. It is possible to derive absolute bounds in terms of the absolute error $\delta_i (i = 1, \ldots, n)$. However, such bounds would be too loose for practical purposes. Instead, we appeal to *stochastic* perturbation theory. The basic idea is to assume that the error matrix $\mathbf{W}$ is random according to a certain distribution with estimated mean and higher-order moments. In order to estimate the absolute upper bound for $\|\Delta\|$, we start with estimating or bounding $\mathbb{E}\|\Delta\|$. This is done by bounding the expectation of the terms on the RHS of the above inequality.

Our assumption on the random distribution of $\mathbf{W}$ is given as follows:

1. The column vectors $\mathbf{W_1}, \ldots, \mathbf{W_n}$ are independent and radially symmetric m-dim vectors.

2. For each $i = 1, \ldots, n$, all elements of column vector $\mathbf{W_i}$ are i.i.d. random variables with mean 0, variance $\sigma_i^2 := \sigma_i^2(\delta_i)$ and the fourth moment $\mu_i^4 := \mu_i^4(\delta_i)$.

Note that the independence assumption is on the single-monitor errors only – this by no means implies that the signals received by different monitors are statistically independent. The *error* variance $\sigma_i^2 := \sigma_i^2(\delta_i)$ and the fourth moment $\mu_i^4 := \mu_i^4(\delta_i)$ are functions of the corresponding monitor slack because the slack determines the size of the allowed drift (or discrepancy), between the true data and the coordinator's view of the data, before the coordinator needs an update.

D.2 Analysis of Frobenius norm

Computation of $\mathbb{E}\|\mathbf{Y}^T\mathbf{W}\|_{\mathbf{F}}^{\mathbf{2}}$. We exploit results from [9]: For any m-dimensional random vector $\mathbf{v}$ uniformly distributed on the unit sphere $\mathbb{S}^{m-1}$, and given a $m \times n$ matrix $\mathbf{Y}$, there hold:

$$\mathbb{E}(\|\mathbf{Y}^T\mathbf{v}\|^2) = \frac{\|\mathbf{Y}^T\|_F^2}{m}, \quad \mathrm{Var}(\|\mathbf{Y}^T\mathbf{v}\|^2) \leq \frac{2}{m+2}.$$

As observed in [55], since $\mathbf{W}_i$ is assumed to be radially symmetric m-dimensional random vector, its projection on the unit sphere as $\mathbf{W_i} = \mathbf{v_i} \cdot \|\mathbf{W_i}\|$, where $\mathbf{v}_i$ is uniformly distributed on $\mathbb{S}^{m-1}$, and is independent with $\|\mathbf{W_i}\|$. Then we have

$$\begin{aligned}
\mathbb{E}(\|\mathbf{Y}^T\mathbf{W}_i\|^2) &= \mathbb{E}(\|\mathbf{Y}^T\mathbf{v}_i\|^2 \cdot \|\mathbf{W}_i\|^2) = \mathbb{E}(\|\mathbf{Y}^T\mathbf{v}_i\|^2) \cdot \mathbb{E}(\|\mathbf{W}_i\|^2) \\
&= \|\mathbf{Y}\|_F^2 \cdot \frac{\mathbb{E}(\|\mathbf{W}_i\|^2)}{m} = \|\mathbf{Y}\|_F^2 \cdot \sigma_i^2, \\
\mathbb{E}(\|\mathbf{Y}^T\mathbf{W}\|_F^2) &= \mathbb{E}(\|\mathbf{Y}^T\mathbf{W}\|_F^2) = \mathbb{E}(\sum_{i=1}^{n} \|\mathbf{Y}^T\mathbf{W}_i\|_F^2) = \sum_{i=1}^{n} \mathbb{E}(\|\mathbf{Y}^T\mathbf{W}_i\|_F^2) \\
&= \sum_{i=1}^{n} \|\mathbf{Y}\|_F^2 \cdot \sigma_i^2 = \|\mathbf{Y}\|_F^2 \cdot \sum_{i=1}^{n} \sigma_i^2 \\
&= \mathrm{tr}(\mathbf{Y}^{\mathrm{T}}\mathbf{Y}) \cdot \sum_{\mathrm{i}=1}^{\mathrm{n}} \sigma_{\mathrm{i}}^2 = \mathrm{m} \sum_{\mathrm{i}=1}^{\mathrm{n}} \lambda_{\mathrm{i}} \cdot \sum_{\mathrm{i}=1}^{\mathrm{n}} \sigma_{\mathrm{i}}^2 = \mathrm{m} \sum_{\mathrm{i}=1}^{\mathrm{n}} \lambda_{\mathrm{i}} \cdot \sigma,
\end{aligned}$$

where $\lambda_i' s$ are eigenvalues of covariance matrix $\mathbf{A} = \frac{1}{m}\mathbf{Y}^T\mathbf{Y}$.[1]

Computation of $\mathbb{E}(\|\mathbf{W}^T\mathbf{W}\|_F^2)$ This is a high order term, and its value is generally dominated by $\mathbb{E}\|\mathbf{Y}^T\mathbf{W}\|_{\mathbf{F}}^{\mathbf{2}}$. Our computation relies on the assumption that the error vectors $\mathbf{W}_1, \ldots, \mathbf{W}_n$ are independent. In addition, we use the following fact from [30]: if $\mathbf{u}$, $\mathbf{v}$ are independently and uniformly distributed column vectors on $\mathbb{S}^{m-1}$, then there hold:

$$\mathbb{E}(\mathbf{u}^T \cdot \mathbf{v}) = 0, \quad \mathbb{E}[(\mathbf{u}^T \cdot \mathbf{v})^2] = \frac{1}{m}, \quad \mathrm{Var}[(\mathbf{u}^T \cdot \mathbf{v})^2] = \frac{2(m-1)}{m^2(m+2)}.$$

[1] For simplicity, we typically suppress the dependence on δ in our notations, such as using σ instead of $\sigma(\delta)$.

For $i \neq j$, we have

$$\begin{aligned}\mathbb{E}[(\mathbf{W}_i^T\mathbf{W}_j)^2] &= \mathbb{E}\left[\left(\frac{\mathbf{W}_i^T}{\|\mathbf{W}_i\|}\cdot\frac{\mathbf{W}_j}{\|\mathbf{W}_j\|}\right)^2\cdot\|\mathbf{W}_i\|^2\cdot\|\mathbf{W}_j\|^2\right] = \frac{1}{m}\cdot\mathbb{E}(\|\mathbf{W}_i\|^2\cdot\|\mathbf{W}_j\|^2)\\ &= \frac{1}{m}\cdot\mathbb{E}(\|\mathbf{W}_i\|^2)\cdot\mathbb{E}(\|\mathbf{W}_j\|^2) = \frac{m^2\sigma_i^2\sigma_j^2}{m} = m\sigma_i^2\sigma_j^2.\end{aligned}$$

Define $z_i := \mathbf{W}_i^T\mathbf{W}_i = \sum_{j=1}^m e_{ji}^2$. We have

$$\mathbb{E}(e_{ji}^2) = \sigma_i^2, \quad \mathrm{Var}(e_{ji}^2) = \mathbb{E}(e_{ji}^4) - (\mathbb{E}(e_{ji}^2))^2 = \mu_i^4 - \sigma_i^4.$$

Then we have

$$\begin{aligned}\mathbb{E}(z_i) &= \mathbb{E}(\sum_{j=1}^m e_{ji}^2) = \sum_{j=1}^m \mathbb{E}(e_{ji}^2) = m\sigma_i^2,\\ \mathrm{Var}(z_i) &= \mathrm{Var}(\sum_{j=1}^m e_{ji}^2) = \sum_{j=1}^m \mathrm{Var}(e_{ji}^2) = m(\mu_i^4 - \sigma_i^4),\\ \mathbb{E}(z_i^2) &= (\mathbb{E}(z_i))^2 + \mathrm{Var}(z) = m^2\sigma_i^4 + m(\mu_i^4 - \sigma_i^4).\end{aligned}$$

In sum, we have

$$\begin{aligned}\mathbb{E}(\|\mathbf{W}^T\mathbf{W}\|_F^2) &= \sum_{i=1}^n \mathbb{E}[(\mathbf{W}_i^T\mathbf{W}_i)^2] + 2\sum_{i=1}^n\sum_{j=i+1}^n \mathbb{E}[(\mathbf{W}_i^T\mathbf{W}_j)^2]\\ &= m^2\sum_{i=1}^n \sigma_i^4 + m\sum_{i=1}^n(\mu_i^4 - \sigma_i^4) + 2\sum_{i=1}^n\sum_{j=i+1}^n m\sigma_i^2\sigma_j^2.\end{aligned}$$

Expectation bounds An application of Jensen's inequality yields $\mathbb{E}(x) \leq \sqrt{\mathbb{E}(x^2)}$. Then we can upper bound $E(\|\mathbf{\Delta}\|_F)$ as follows

$$\begin{aligned}\mathbb{E}(\|\mathbf{\Delta}\|_F) &\leq \frac{2}{m}\mathbb{E}(\|\mathbf{Y^T W}\|_F) + \frac{1}{m}\mathbb{E}(\|\mathbf{W^T W}\|_F)\\ &\leq \frac{2}{m}\sqrt{\mathbb{E}(\|\mathbf{Y^T W}\|_F^2)} + \frac{1}{m}\sqrt{\mathbb{E}(\|\mathbf{W^T W}\|_F^2)}\\ &= \frac{2}{m}\sqrt{m\sum_{i=1}^n\lambda_i\cdot\sum_{i=1}^n\sigma_i^2} + \frac{1}{m}\sqrt{m^2\sum_{i=1}^n\sigma_i^4 + m\sum_{i=1}^n(\mu_i^4-\sigma_i^4) + 2\sum_{i=1}^n\sum_{j=i+1}^n m\sigma_i^2\sigma_j^2}\\ &\leq 2\sqrt{\frac{1}{m}\sum_{i=1}^n\lambda_i\cdot\sum_{i=1}^n\sigma_i^2} + \sqrt{\sum_{i=1}^n\sigma_i^4 + \frac{1}{m}\sum_{i=1}^n(\mu_i^4-\sigma_i^4) + \frac{n}{m}\sum_{i=1}^n\sigma_i^4} := \sqrt{n}\cdot Tol_F\end{aligned}$$

Combining with Mirsky's theorem, we have that

$$\mathbb{E}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{\lambda}_i - \lambda_i)^2} \leq \mathbb{E}\left(\frac{\|\mathbf{\Delta}\|_F}{\sqrt{n}}\right) \leq Tol_F,$$

where Tol_F is given by our foregoing analysis.

Proof of Theorem 6 We now have the tool to prove Theorem 6. Recall that

$$\epsilon^* \leq \frac{1}{\sqrt{n}}\|\mathbf{\Delta}\|_F = \frac{1}{m\sqrt{n}}\|\mathbf{Y}^T\mathbf{W} + \mathbf{W}^T\mathbf{Y} + \mathbf{W}^T\mathbf{W}\|_F,$$

where $\mathbf{W}^T\mathbf{W}$ is a high order term, and $\|\mathbf{Y}^T\mathbf{W}\|_F = \|\mathbf{W}^T\mathbf{Y}\|_F$. So the deviation of $\|\mathbf{\Delta}\|_F$ from its expectation is dominated by the deviation of $\|\mathbf{Y}^T\mathbf{W}\|_F$ from its expectation. Using Chebyshev Inequality, we have

$$\begin{aligned}
& \mathbf{Pr}\left[\|\mathbf{Y}^T\mathbf{W}\|_F - \sqrt{\mathbb{E}\|\mathbf{Y}^T\mathbf{W}\|_F^2} \geq m\sqrt{n}\zeta\right] \\
\leq\ & \mathbf{Pr}\left[|\|\mathbf{Y}^T\mathbf{W}\|_F^2 - \mathbb{E}\|\mathbf{Y}^T\mathbf{W}\|_F^2| \geq m\sqrt{n}\zeta(2\sqrt{\mathbb{E}\|\mathbf{Y}^T\mathbf{W}\|_F^2})\right] \\
\leq\ & \frac{\mathrm{Var}(\|\mathbf{Y}^T\mathbf{W}\|_F^2)}{m^2n\zeta^2(2\sqrt{m\sum\lambda_i\sum\sigma_i^2})^2} \approx O\left(\frac{\frac{1}{m}m^2n^2n}{m^2n\zeta^2mmnn}\right) \\
=\ & O\left(\frac{mn^3}{\zeta^2m^4n^3}\right) = O\left(\frac{1}{\zeta^2m^3}\right).
\end{aligned}$$

Thus with probability $1 - O(1/\zeta^2m^3)$, we have

$$\begin{aligned}
\epsilon^* &\leq \frac{1}{m\sqrt{n}}\|\mathbf{\Delta}\|_F \leq \frac{2}{m\sqrt{n}}\sqrt{\mathbb{E}\|\mathbf{Y}^T\mathbf{W}\|^2} + \frac{1}{m\sqrt{n}}\sqrt{\mathbb{E}\|\mathbf{W}^T\mathbf{W}\|^2} + 3\zeta \\
&\leq 2\sqrt{\frac{1}{mn}\sum_{i=1}^{n}\lambda_i\cdot\sum_{i=1}^{n}\sigma_i^2} + \sqrt{\frac{1}{n}\sum_{i=1}^{n}\sigma_i^4 + \frac{1}{mn}\sum_{i=1}^{n}(\mu_i^4 - \sigma_i^4) + \frac{1}{m}\sum_{i=1}^{n}\sigma_i^4} + 3\zeta \\
&\approx 2\sqrt{\frac{1}{mn}\sum_{i=1}^{n}\lambda_i\cdot\sum_{i=1}^{n}\sigma_i^2} + \sqrt{\left(\frac{1}{m} + \frac{1}{n}\right)\sum_{i=1}^{n}\sigma_i^4} + 3\zeta,
\end{aligned}$$

where the last step is achieved by ignoring the high order term $\frac{1}{mn}\sum_{i=1}^{n}(\mu_i^4 - \sigma_i^4)$.

Computation of variances The variances of the terms analyzed above can also be computed analytically. Using the following identity for independent variables X and Y that

$$\mathrm{Var}(XY) = \mathrm{Var}(X)\mathrm{Var}(Y) + (\mathbb{E}Y)^2\mathrm{Var}(X) + (\mathbb{E}X)^2\mathrm{Var}(Y),$$

we obtain

$$\begin{aligned}
& \mathrm{Var}(\|\mathbf{Y^T W}_i\|^2) = \mathrm{Var}(\|\mathbf{Y^T v}_i\|^2\|\mathbf{W}_i\|^2) \\
= \;& \mathrm{Var}(\|\mathbf{Y^T v}_i\|^2)\mathrm{Var}(\|\mathbf{W}_i\|^2) + (\mathbb{E}\|\mathbf{W_i}\|^{\mathbf{2}})^{\mathbf{2}}\mathrm{Var}\|\mathbf{Y^T v_i}\|^{\mathbf{2}} + (\mathbb{E}\|\mathbf{Y^T v_i}\|^{\mathbf{2}})^{\mathbf{2}}\mathrm{Var}\|\mathbf{W_i}\|^{\mathbf{2}} \\
\leq \;& \frac{2}{m+2}\mathrm{Var}(\|\mathbf{W_i}\|^{\mathbf{2}}) + \frac{\mathbf{2}}{\mathbf{m+2}}(\mathbb{E}\|\mathbf{W_i}\|^{\mathbf{2}})^{\mathbf{2}} + \frac{\|\mathbf{Y^T}\|_{\mathbf{F}}^{\mathbf{4}}}{\mathbf{m^2}}\mathrm{Var}(\|\mathbf{W_i}\|^{\mathbf{2}}) \\
= \;& \frac{2m}{m+2}\mathrm{Var}(e_{1i}^2) + \frac{2m^2}{m+2}(\mathbb{E}e_{1i}^2)^2 + \frac{1}{m}\|\mathbf{Y}\|_F^4\mathrm{Var}(e_{1i}^2).
\end{aligned}$$

Noting that $\mathbf{W_1}, ..., \mathbf{W_n}$ are independent, each element e_{ji} has the forth moment μ_i^4, then we have $\mathrm{Var}(e_{ji}^2) = E(e_{ji}^4) - (E(e_{ji}^2))^2 = \mu_i^4 - \sigma_i^4$. Thus,

$$\begin{aligned}
\mathrm{Var}(\|\mathbf{Y^T W}\|_F^2) &= \mathrm{Var}\left(\sum_{i=1}^n \|\mathbf{Y^T W}_i\|_F^2\right) = \sum_{i=1}^n \mathrm{Var}(\|\mathbf{Y^T W}_i\|^2) \\
&\leq \frac{2m}{m+2}\cdot\sum_{i=1}^n \mathrm{Var}(e_{1i}^2) + \frac{2m^2}{m+2}\sum_{i=1}^n \sigma_i^4 + \frac{1}{m}\|\mathbf{Y}\|_F^4\sum_{i=1}^n \mathrm{Var}(e_{1i}^2) \\
&= \frac{2m}{m+2}\cdot\sum_{i=1}^n (\mu_i^4 - \sigma_i^4) + \frac{2m^2}{m+2}\sum_{i=1}^n \sigma_i^4 + \frac{1}{m}\|\mathbf{Y}\|_F^4\sum_{i=1}^n (\mu_i^4 - \sigma_i^4).
\end{aligned}$$

The variance of $\|\mathbf{W}^T\mathbf{W}\|_F^2$ can also be computed analytically using result from [30]. The computation is tedious, so we omit the procedure here.

Note that our computation of means and variances can be simplied signficantly by using further assumption on the distribution of the error elements e_{ji} of matrix $\mathbf{W}$, so that the result depend directly on the slack parameters $\delta_i (i = 1, \ldots, n)$. For example, if e_{1i} is uniformly distributed

on $[-\delta_i, \delta_i]$, we have $\mathrm{Var}(e_{1i}^2) = \mu_i^4(\delta_i) - \sigma_i^4(\delta_i) = \frac{\delta_i^4}{5} - \frac{\delta_i^4}{9} = \frac{4\delta_i^4}{45}$, and so on. On the ther hand, if $e_{1i} \sim N(0, \sigma_i^2(\delta_i))$, we have $\mathrm{Var}(e_{1i}^2) = \mu_i^4(\delta_i) - \sigma_i^4(\delta_i) = 3\sigma_i^4(\delta_i) - \sigma_i^4(\delta_i) = 2\sigma_i^4(\delta_i)$ and so on.

D.3 Analysis of spectral norm

In this section, we turn to the estimation of the spectral norm of the perturbation error matrix $\mathbf{\Delta}$. This quantity provides a tighter upper bound for the eigenvalue perturbation (via Weyl's theorem). Unfortunately, it is also difficult to bound. For many applications, it suffices to replace a bound on $\|.\|_2^2$ by its expectation $\mathbb{E}\|.\|_2^2$. In the following derivations, we rely on the concentration of eigenvalues of random symmetric matrices [2]. This result is applicable to matrices whose elements are independent or weakly correlated.

Let $\mathcal{L}_{max}(\cdot)$ denote the maximum eigenvalue of a matrix. Then we have

$$\begin{aligned}
\mathbb{E}(\|\mathbf{W^T Y}\|_2^2) &= \mathbb{E}(\mathcal{L}_{max}(\mathbf{Y^T W W^T Y})) \approx \mathcal{L}_{max}(\mathbb{E}(\mathbf{Y^T W W^T Y})) \\
&= \mathcal{L}_{max}(\mathbf{Y}^T \mathbb{E}(\mathbf{WW})^T \mathbf{Y}) = \mathcal{L}_{max}(\mathbf{Y}^T [\sum_{i=1}^{n} \sigma_i^2 \mathbf{I}] \cdot \mathbf{Y}) \\
&= = \mathcal{L}_{max}(\mathbf{Y}^T \mathbf{Y}) \cdot \sum_{i=1}^{n} \sigma_i^2 \\
&= \lambda_{max} \cdot \sum_{i=1}^{n} \sigma_i^2.
\end{aligned}$$

Likewise, we have

$$\begin{aligned}
\mathbb{E}(\|\mathbf{Y^T W}\|_2^2) &= \mathbb{E}(\mathcal{L}_{max}(\mathbf{W^T Y Y^T W})) \approx \mathcal{L}_{max}(\mathbb{E}(\mathbf{W^T Y Y^T W})) \\
&= \mathcal{L}_{max}\left(\mathbb{E}\left[\mathbf{W}_i^T \mathbf{Y}\mathbf{Y}^T \mathbf{W}_j\right]_{1 \le i,j \le n}\right) \\
&= = \mathcal{L}_{max}\left(\mathbb{E}\left[\sum_{k,l}^{m} e_{ik} (\mathbf{Y}\mathbf{Y}^T)_{kl} e_{jl}\right]\right).
\end{aligned}$$

Because the elements e_{ji} of matrix $\mathbf{W}$ are independent with mean 0, the matrix inside $\mathcal{L}_{max}$ is a diagonal matrix. As a result,

$$\begin{aligned}\mathbb{E}(\|\mathbf{Y^T W}\|^2) &= \mathcal{L}_{max}\left(\mathbb{E}\left[\sum_{k=1}^{m}\sigma_i^2(\mathbf{YY}^T)_{kk}\right]_{1\le i\le n}\right)\\ &= \max_i\left\{\sigma_i^2\sum_{k=1}^{m}(\mathbf{YY}^T)_{kk}\right\} = \sigma_{max}^2\sum_{k=1}^{m}(\mathbf{YY}^T)_{kk}\\ &= \sigma_{max}^2\mathrm{tr}(\mathbf{YY}^{\mathrm{T}}).\end{aligned}$$

A remaining term is $\mathbb{E}\|\mathbf{W^T W}\|_{\mathbf{2}}$, which is generally dominated by $\mathbb{E}(\|\mathbf{Y^T W}\|_2) + \mathbb{E}(\|\mathbf{W^T Y}\|)$ and is omitted in our analysis. Thus we have the following *approximate* upper bound on expected spectral norm of the perturbation error matrix:

$$\mathbb{E}\|\mathbf{\Delta}\|_2 \lesssim Tol_2,$$

where

$$Tol_2 = \sqrt{\lambda_{max}\cdot\sum_{i=1}^{n}\sigma_i^2} + \sqrt{\sigma_{max}^2\mathrm{tr}(\mathbf{YY}^{\mathrm{T}})}.$$

By Weyl's theorem, there holds

$$\mathbb{E}\max_i|\lambda_i - \hat{\lambda}_i| \lesssim Tol_2.$$

Bibliography

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

[2] ALON, N., KRIVELEVICH, M., AND VU, V. H. On the concentration of eigenvalues of random symmetric matrices. *Israel Journal of Mathematics*, 131 (2002), 259–267.

[3] BABCOCK, B., AND OLSTON, C. Distributed top-k monitoring. In *Proceedings of SIGMOD* (2003).

[4] BABU, S., AND WIDOM, J. Continuous queries over data streams. *ACM SIGMOD Record 30*, 3 (September 2001), 109–120.

[5] BAI, Z.-J., CHAN, R., AND LUK, F. Principal component analysis for distributed data sets with updating. In *Proceedings of International workshop on Advanced Parallel Processing Technologies (APPT)* (2005).

[6] BARFORD, P., KLINE, J., PLONKA, D., AND RON, A. A signal analysis of network traffic anomalies. In *Proceedings of Internet Measurement Workshop* (November 2002).

[7] BELLOVIN, S., AND CHESWICK, W. Network firewalls. *IEEE Communications Magazine* (September 1994).

[8] BLEI, D. M., AND LAFFERTY, J. D. Dynamic topic models. In *Proceedings of ICML* (2006).

[9] BOTTCHER, A., AND GRUDSKY, S. The norm of the product of a large matrix and a random vector. *Electronic Journal of Probability*, 8 (2003), 1–29.

[10] BRUTAG, J. D. Aberrant behavior detection and control in time series for network monitoring. In *Proceedings of 14th Systems Administration Conference (LISA)* (2000).

[11] CHEN, J., DEWITT, D. J., TIAN, F., AND WANG, Y. Niagaracq: A scalable continuous query system for internet databases. In *Proceedings of SIGMOD* (Dallas, Texas, May 2000), pp. 379–390.

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

[13] CHU, D., DESHPANDE, A., HELLERSTEIN, J., AND HONG, W. Approximate data collection in sensor networks using probabilistic models. In *Proceedings of ICDE* (Atlanta, GA, 2006).

[14] CHUN, B., HELLERSTEIN, J., HUEBSCH, R., MANIATIS, P., AND ROSCOE, T. Design considerations for information planes. In *Proceedings of 1st Workshop on Real, Large Distributed Systems (WORLDS)* (December 2004).

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

[16] CORMODE, G., AND GAROFALAKIS, M. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of VLDB* (2005), pp. 13–24.

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

[18] CORMODE, G., MUTHUKRISHNAN, S., AND ZHUANG, W. Conquering the divide: Continuous clustering of distributed data streams. In *Proceedings of ICDE* (2007).

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

[20] DELIGIANNAKIS, A., KOTIDIS, Y., AND ROUSSOPOULOS, N. Hierarchical in-network data aggregation with quality guarantees. In *Proceedings of the 9th International Conference on Extending DataBase Technology (EDBT)* (Heraklion, Crete, March 2004).

[21] DESHPANDE, A., GUESTRIN, C., MADDEN, S., HELLERSTEIN, J. M., AND HONG, W. Model-driven data acquisition in sensor networks. In *Proceedings of VLDB* (2004).

[22] DILMAN, M., AND RAZ, D. Efficient reactive monitoring. In *Proceedings of InfoCom* (2001).

[23] DUDA, R. O., HART, P. E., AND STORK, D. G. *Pattern Classification*, second ed. John Wiley and Sons, 2001.

[24] ESTEVEZ-TAPIADOR, J. M., GARCIA-TEODORO, P., AND DIAZ-VERDEJO, J. E. Anomaly detection methods in wired networks: a survey and taxonomy. *Computer Communications 27*, 16 (October 2004), 1569–1584.

[25] GAROFALAKIS, M., GEHRKE, J., AND RASTOGI, R. Querying and mining data streams: You only get one look. Tutorial in VLDB, Aug. 2002.

[26] GUESTRIN, C., THIBAUX, R., BODIK, P., PASKIN, M., AND MADDEN, S. Distributed regression: an efficient framework for modeling sensor network data. In *Proceedings of IPSN* (2004).

[27] HANSON, E. N., BODAGALA, S., AND CHADAGA., U. Trigger condition testing and view maintenance using optimized discrimination network. *IEEE Transaction on Knowledge of Data Engineering 14*, 2 (2002).

[28] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The Elements of Statistical Learning*. Springer, 2001.

[29] HOHLT, B., DOHERTY, L., AND BREWER, E. Flexible power scheduling for sensor networks. In *Proceedings of IPSN* (2004).

[30] HOLMES, R. B. On random correlation matrices. *SIAM J. Matrix Anal. Appl. 12*, 2 (1991).

[31] HOOD, C., AND JI, C. Proactive network fault detection. *IEEE Transaction on Reliability 46*, 3 (1997), 333–341.

[32] HUANG, L., GAROFALAKIS, M., JOSEPH, A. D., AND TAFT, N. Communication-efficient tracking of distributed cumulative triggers. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)* (2007).

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

[34] JACKSON, J. E., AND MUDHOLKAR, G. S. Control procedures for residuals associated with principal component analysis. *Technometrics 21*, 3 (1979), 341–349.

[35] JAIN, A., CHANG, E. Y., AND WANG, Y.-F. Adaptive stream resource management using kalman filters. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (Paris, France, 2004), ACM Press.

[36] JAIN, A., HELLERSTEIN, J. M., RATNASAMY, S., AND WETHERALL, D. A wakeup call for internet monitoring systems: The case for distributed triggers. In *Proceedings 3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)* (San Diego, CA, November 2004).

[37] JENSEN, D. R., AND SOLOMON, H. A gaussian approximation for the distribution of definite quadratic forms. *Journal of the American Statistical Association 67*, 340 (1972), 898–902.

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

[39] KREIDL, O. P., AND WILLSKY, A. Inference with minimal communication: A decision-theoretic variational approach. In *Proceedings of Neural Information Processing Systems (NIPS)* (2005).

[40] KRISHNAMURTHY, B., SEN, S., ZHANG, Y., AND CHEN, Y. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of Internet Measurement Conference* (Octobor 2003).

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

[42] LAKHINA, A., PAPAGIANNAKI, K., CROVELLA, M., DIOT, C., KOLACZYK, E. D., AND TAFT, N. Structural analysis of network traffic flows. In *Proceedings of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, p. 2004.

[43] MEDINA, A., LAKHINA, A., MATTA, I., AND BYERS, J. Brite: an approach to universal topology generation. In *Proceedings of MASCOTS* (2001).

[44] MUKHERJEE, B., HEBERLEIN, L. T., AND LEVITT, K. Network intrusion detection.

[45] NGUYEN, X., WAINWRIGHT, M. J., AND JORDAN, M. I. On optimal quantization rules in sequential decision problems. In *Proceedings of IEEE International Symposium on Information Theory (ISIT)* (2006).

[46] NUCCI, A., SRIDHARAN, A., AND TAFT, N. The problem of synthetically generating ip traffic matrices: initial recommendations. In *Proceedings of ACM CCR* (2005).

[47] OLSTON, C., JIANG, J., AND WIDOM, J. Adaptive filters for continuous queries over distributed data streams. In *Proceedings of SIGMOD* (2003), pp. 563–574.

[48] PADMANABHAN, V. N., RAMABHADRAN, S., AND PADHYE, J. Netprofiler: Profiling wide-area networks using peer cooperation. In *Proceedings of the Fourth International Workshop on Peer-to-Peer Systems (IPTPS)* (Ithaca, NY, USA, 2005).

[49] PAXSON, V. Bro: A system for detecting network intruders in real time. In *Proceedings of the 7th USENIX Security Symposium* (1998).

[50] PAXSON, V., AND FLOYD, S. Wide-area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking 3*, 3 (1995), 226–244.

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

[52] PREDD, J. B., KULKARNI, S. B., AND POOR, H. V. Distributed learning in wireless sensor networks. In *IEEE Signal Processing Magazine* (2006), vol. 23, pp. 56–69.

[53] QU, Y. M., OSTROUCHOVZ, G., SAMATOVAZ, N., AND GEIST, A. Principal component analysis for dimension reduction in massive distributed data sets. In *Proceedings of IEEE International Conference on Data Mining (ICDM)* (2002).

[54] ROESCH, M. Snort network intrusion detection system. `http://www.snort.org/`.

[55] RUBINSTEIN, R. Generating random vectors uniformly distributed inside and on the surface of different regions. *European Journal of Operations Research*, 10 (1982), 205–209.

[56] SHARFMAN, I., SCHUSTER, A., AND KEREN, D. A geometric approach to monitoring threshold functions over distributed data streams. In *Proceedings of ACM SIGMOD* (2006).

[57] SOULE, A., SALAMATIAN, K., AND TAFT, N. Combining filtering and statistical methods for anomaly detection. In *Proceedings of IMC* (2005).

[58] SPRING, N., WETHERALL, D., AND ANDERSON, T. Scriptroute: A facility for distributed internet measurement. In *Proceedings Fourth USENIX Symposium on Internet Technologies and Systems (USITS)* (March 2003).

[59] STEWART, G. W. Perturbation theory for the sigular value decomposition. Tech. rep., UMIACS-TR-90-123, 1990.

[60] STEWART, G. W., AND GUANG SUN, J. *Matrix Perturbation Theory*. Academic Press, 1990.

[61] THOTTAN, M., AND JI, C. Proactive anomaly detection using distributed intelligent agents. *IEEE Network* (September/Octobor 1998).

[62] VAN RENESSE, R., AND KENNETH, B. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS 21*, 2 (May 2003).

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

[64] XIE, Y., KIM, H.-A., O'HALLARON, D. R., REITER, M. K., AND ZHANG, H. Seurat: A pointillist approach to anomaly detection. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)* (Sophia Antipolis, France, 2004).

[65] XU, K., ZHANG, Z.-L., AND BHATTACHARYYA, S. Profiling internet backbone traffic: Behavior models and applications. In *Proceedings of ACM SIGCOMM* (2005).

[66] XUANLONG NGUYEN, MARTIN J. WAINWRIGHT, M. I. J. Nonparametric decentralized detection using kernel method. In *EEE Transactions on Signal Processing* (2005), vol. 53, pp. 4053–4066.

[67] YEGNESWARAN, V., BARFORD, P., AND JHA, S. Global intrusion detection in the domino overlay system. In *Proceedings of Network and Distributed Security Symposium (NDSS)* (2004).

[68] ZHANG, S., AND HUANG, L. Personal communication, November 2006.

[69] ZHANG, Y., GE, Z., GREENBERG, A., AND ROUGHAN, M. Network anomography. In *Proceedings of IMC* (2005).

[70] ZHAO, B. Y., HUANG, L., STRIBLING, J., KUBIATOWICZ, J. D., AND JOSEPH, A. D. Exploiting routing redundancy via structured peer-to-peer overlays. In *Proceedings of ICNP* (November 2003).