2013

Data Streaming Algorithms for Rapid Cyber Attack Detection

Yang Liu
Iowa State University

Data Streaming Algorithms for Rapid Cyber Attack Detection

by

Yang Liu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:

Yong Guan, Major Professor

Thomas E Daniels

Arun K Somani

Johnny S Wong

Lei Ying

Iowa State University

Ames, Iowa

2013

DEDICATION

I would like to dedicate this thesis to my wife Liping Chen without whose support I would not have been able to complete this work. I would also like to thank my mother Baihong Yang, my father Jide Liu, and my sister Xia Liu, for their endless love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, I am deeply grateful to my advisor, Dr. Yong Guan. Without his knowledge, perceptiveness, patience, guidance and support, I would have never finished my thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education.

I would like to thank my committee members for their efforts and contributions to this work: Dr. Thomas E Daniels, Dr. Arun K Somani, Dr. Akhilesh Tyagi, Dr. Johnny S Wong and Dr. Lei Ying. Their suggestions inspired me and improved the quality of my thesis. I would additionally like to thank all the lecturers in Department of Electrical and Computer Engineering and other departments, from whom I learnt a lot.

I would like to thank all my labmates and friends at Iowa State University. I am only able to list a few here: Leonardo Aguilera, Wenji Chen, Bibudh Lahiri, Yanlin Peng, Yawen Wei, Ka Yang, Zhen Yu, Linfeng Zhang, and Gang Xu.

This work was partially supported by NSF under grants No. CNS-0644238 and CNS-0831470.

ABSTRACT

Internet has become an essential part of the daily life for billions of users worldwide. However, everyday we are still reading news stories about major security breaches, new polymorphic worm/virus spreading, identity theft, DDoS or phishing emails. Attackers are using botnets as an effective tool to conduct a wide range of criminal activities even more efficiently than before. It becomes very challenging to detect/defend cyber attacks in high-speed networks nowadays. Firstly, the Internet traffic exhibits huge fluctuations and long range dependence, which makes the attack traffic often be hidden by large volumes of normal traffic. In order to get reliable information for the malicious traffic, we must be able to process each packet in a wire speed. Secondly, ISPs want to detect the attacks when they are still at a low-profile volume in order to reduce the damage as much and early as possible. Therefore, the detection algorithm should be able to correlate multiple data sources and identify the root of causes efficiently.

In this dissertation, we propose data streaming algorithms to detect/defend cyber attacks in high-speed networks. We propose a novel data structure to detect click frauds in the online advertising network, and track the command-and-control communications in the botnets. Next, we introduce a fast sketch for the aggregate queries in high-speed network traffic, which can preserve some critical information for root cause analysis, like IP addresses, port numbers, etc. This sketch can also be extended to track super spreaders. Lastly, we apply the low-rank matrix approximation to monitor network-wide traffic anomalies and traffic activity graphs.

CHAPTER 1. OVERVIEW

1.1 Introduction

An important problem for the network security is how to effectively monitor network traffic and detect cyber attacks in real time. Internet has become essential parts of the daily life for billions of people worldwide. People would not be able to afford the lost and unavailability of such services or applications. Maintaining the healthiness of the Internet services and applications requires not only the robust design and construction of such networs/applications, but also effective monitoring/recovery countermeasures in response to any failures or cyber attacks. One can predict that the major security breaches, polymorphic worm/virus spreading, identity theft, DDoS or phishing emails that we are facing today would be the ones or some even tougher emerging ones that we (will) have in the future.

- *Click Frauds*: Online advertising has become a trillion dollar market per year. Currently, the pay-per-click model is the state-of-the-art model between the advertiser and the publisher. If an advertisement link is clicked by target users, the advertiser will pay the publisher (or the search engine) for each click. Fraudulent clicks are the clicks without actual advertising impression, which can be produced by cheap labors, automated scripts, or botnets, etc. A possible solution for the click fraud problem is that both advertisers and publishers keep on auditing the click stream and reach an agreement on the determination of valid clicks. A reasonable countermeasure is that identical clicks will be considered as invalid if they are within short intervals, and valid if they happen sparsely. We want to detect any duplicated clicks in a sliding window model.

- *Botnet C&C Connections*: Traffic from the botnet C&C connections differentiate themselves significantly from normal traffic and those generated from traditional attacks such as DDoS attack and worm spreading. Firstly, bots only send/recieve a small amount of malicious traffic on the C&C channels in order to avoid to be detected. It is very difficult to detect botnet C&C connections in high-speed networks, because normal traffic has much higher volumes than the malicious traffic from the botnets. Secondly, bots tend to maintain persistent connections, i.e., long duration flows. A flow is considered to be active if the inter-arrival time between any two conjunctive packets in the flow is shorter than some timeout interval, and the long duration flows (LDFs) are the ones that can keep active for a relatively long period of time, i.e., 30 minutes or even longer. The botnet C&C connections tend to keep alive as long as possible in order to maintain the connectivity of the botnets. Therefore, the LDFs can be used as a simple and effective traffic feature to identify the botnet C&C connections, and thereby discover the membership, structure, and communication patterns of the botnets.

- *Heavy-change Flows*: Due to the rapid increase in the traffic volume, it is often infeasible to monitor every individual flow in the backbone network due to space and time constraints. Instead, we are often required to aggregate packets into a small number of flows and develop the detection methods with aggregated flows, namely aggregate queries. Although it enables ISPs to detect network problems in a timely manner, the flow aggregation cannot preserve certain critical information in network traffic, e.g., IP addresses, port numbers, etc. Due to such missing information, it becomes very difficult (or often infeasible) for ISPs to identify the sources of network attacks or the causes of traffic anomalies, which are important to resolve the network problems effectively. We are willing to identify any flows that have a sudden change in their traffic volumes.

- *Super Spreaders*: Super spreaders are the hosts with a large number of connections to distinct hosts. For example, a compromised host doing fast scanning for worm propagation often creates an unusually high number of connections within a short time. By identifying in real-time any super spreaders, we can identify hosts corresponding to many security problems, e.g. DDoS, spam, botnets, etc. A super spreader may not have be a heavy hitter, because it may send only a few packets on each connection. We cannot simply detect the hosts with large traffic volumes as super spreaders, which have also been widely studied as the heavy hitter problem. It is more difficult to detect super spreaders than heavy hitters.

- *Coordinated Traffic Anomalies*: Traffic anomalies can occur due to a variety of problems. Firstly, security threats like DDoS, worms, and botnets, can generate extremely large-volume anomalous traffic. Secondly, unusual events can cause traffic anomalies, like equipment failures, vendor implementation errors, and software bugs. Thirdly, abnormal user behaviors can change the traffic patterns, for example, flash crowds, non-malicious large file transfers, etc.. In the early days, traffic anomalies often involve unusual large-volume traffic, i.e., high-profile traffic, which are mainly caused by traditional DoS, worm, or flash crowds. In recent years, new threats like botnets introduce low-profile but in a coordinated manner, which only generate a small amount of traffic but follow specific coordinated traffic patterns. Besides these, there are also some traffic anomalies that are low-profile and non-coordinated, e.g. Black mails and spam voice IP calls.

- *Traffic Activity Graphs*: Recently, Traffic Activity Graphs (TAGs) have been proposed to understand, analyze, and model network-wide communication patterns. The topological properties of the TAGs have been shown to be very helpful for malware analysis, anomaly detection, and attack attribution. In a TAG, nodes represent hosts in the network and edges are observed flows that indicate certain communication relations or interactions of interest among the hosts. The challenge is how to capture and analyze TAGs which are usually large, sparse and complex and often have overly large space and computation requirements.

In this dissertation, we mainly focus on the above security problems. We design efficient and effective algorithms to monitor traffic and detect such large-scale cyber attacks. We hope that our work can be used to improve the practice for network security and various Internet applications.

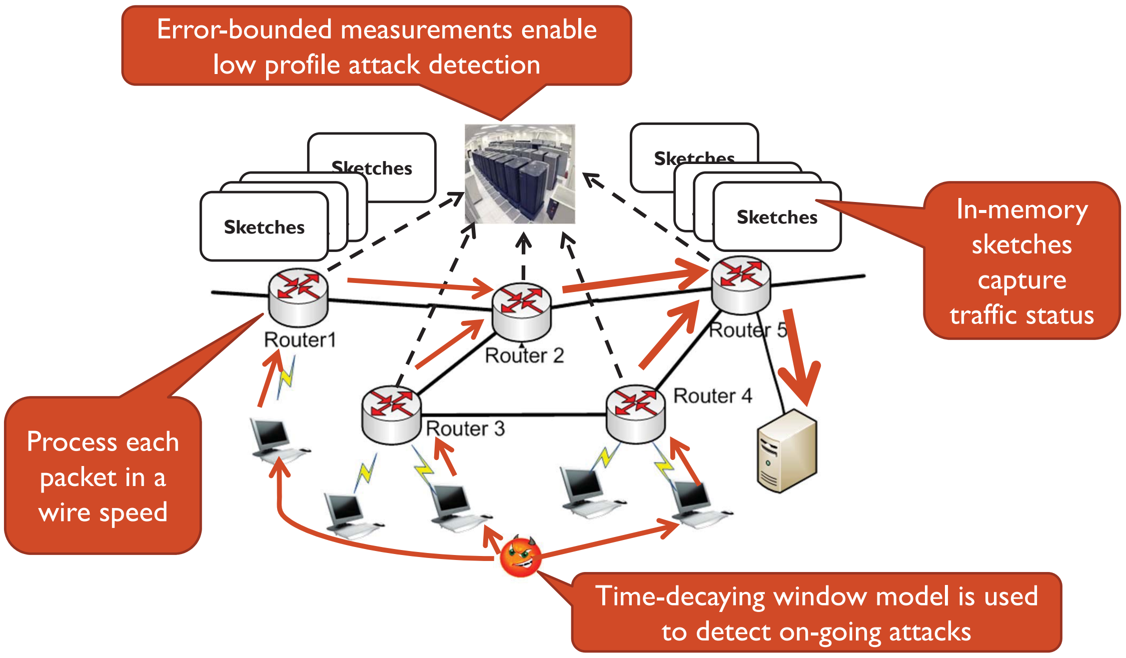

Error-bounded measurements enable low profile attack detection
Sketches
Sketches
Sketches
Sketches
In-memory sketches capture traffic status
Router1
Router 2
Router 5
Router 4
Router 3
Process each packet in a wire speed
Time-decaying window model is used to detect on-going attacks

Figure 1.1 A Motivating Scenario

1.2 A Motivating Scenario

For the current high-speed networks, results from centralized monitoring would not make much sense, or be even effective and feasible. It is often common that Internet Service Providers (ISPs) choose to take a distributed monitoring architecture. Usually in it, a Network Operation Center (NOC) is responsible for collecting data from a large number of local monitors that capture the local view about "things" of interest, mining certain set of characteristics and measurements of interest, and identifying the happening problems and the root causes thereof. The local monitors are deployed to capture/measure the "things" of interest about entities in the network and sometimes aggregate the local observations in real time to the NOCs.

In a distributed monitoring system, we have multiple local monitors and one network operation center (NOC). Each local monitor has a bi-direction connection with the NOC, which is used to send their measurements to the NOC. The NOC is responsible to answer various quires regarding to the network traffic. In this system, the measurements from a local monitor are processed one by one, which come in a high speed rate. Quires over real-time measurements are different from previous data processing, due to the following requirements and design constraints:

- It is infeasible to save or transfer all measurements to the NOC due to the limited space and bandwidth.
- Different measurements may have different weights (i.e., importance), for example, according to time. There are several types of processing models, e.g. the probabilistic data stream model, the sliding window model, etc.
- The NOC must be able to answer queries in real time due to the critical importance of the Internet applications to the society. The NOC should detect any failures and attacks to the network as soon as their happening.

The problems of processing continuous data updating are studied by the data stream computation model [3] [71]. A data stream is a sequence of data items,

$$x_1, x_2, \ldots, x_i, \ldots \tag{1.1}$$

where each item is derived from the same universe, i.e., $x_i \in \mathcal{U}$. In a basic model, all of the items are considered with equally importance, and the computation is over the entire history. For the traffic monitoring, data should decay over time, i.e., the recent data is considered as relevant for the computation. Datar et. al. [32] proposed a strict model to extend the basic data stream model to handle recent data, i.e. the sliding window model. In this model, only the recent N items are considered as important for the computation, and the expired items should be excluded for the computation.

The set of all active items is denoted by

$$\mathcal{A}_i = \{x_{i-N+1}, \ldots, x_i\} \tag{1.2}$$

where N denotes the size of the sliding window. In general, we cannot save all the active items in the memory due to the limitation in computation and space. If an item is not saved in the memory, we cannot know when it will become expired. The requirement to remove expired information from the query result makes the information aggregation problems over sliding windows more difficult than those over the entire history. If all active items are saved in the memory or on the disk, we can compute any function over the active set $\mathcal{A}_i$. Let $f(\mathcal{A}_i)$ denote the exact result which can be computed by saving all active items. Given an error bound $0 < \epsilon < 1$ and a failure probability $0 < \delta < 1$, a data streaming algorithm with bounded space and running time can only find an approximated result $\hat{f}(\mathcal{A}_i)$ such that

$$\|f(\mathcal{A}_i) - \hat{f}(\mathcal{A}_i)\| \leq \epsilon \|f(\mathcal{A}_i)\| \tag{1.3}$$

with a probability at least $1 - \delta$.

The advantages of the data stream algorithm is that the space and running time are much smaller than the requirement of the computation of the exact result $f(\mathcal{A}_i)$. Usually, both the approximation and the randomization are required to find an efficient algorithm. We expect that the data stream algorithm only requires sub-linear space and constant running time,

$$\begin{aligned} Space &= O\left(poly(\log N, \log R)\right), & (1.4)\\ UpdateTime &= O(1), & (1.5)\\ QueryTime &= O(1), & (1.6) \end{aligned}$$

where $poly(\cdot)$ denotes a polynomial function and R denotes the size of the universe, i.e. $R = |\mathcal{U}|$. The running time has two parts: update time and query time. The update time is used to insert a new item into the data structure and remove expired items. And the query time is used to compute a approximate result from the data structure. In a data streaming algorithm, the update time is more important for the computation.

1.3 Objectives of Research

In our research, we propose data streaming algorithms to detect cyber attacks in an efficient and effective ways. We have the following research goals:

- Our algorithms process the data when they become available, which enable the NOC to response to the attacks before or shortly after they happen.

- Our algorithms can bound the errors with limited space. In this way, the false positives are at an acceptable rate, and the NOC has enough operators to investigates detected attacks.

To achieve these goals, we study three fundamental techniques as follows:

1.3.1 Dynamic Membership Data Structure

We want a data structure to support approximate membership in the time-decaying window model. In this model, items are inserted one-by-one over a data stream, and we want to determine whether an item is among the most recent w items for any given window size $w \leq n$. The data structure only allows for a small false positive δ but no false negative, using small (limited) space.

1.3.2 Reversible Sketches

Due to the rapid increase in link speeds and traffic volumes, it is often unfeasible to monitor every individual host or flow in the backbone network. Instead, we are forced to aggregate the packets into a small number of groups and monitor some KPIs (Key Performance Indicators) on the aggregated flow for each group. Although the flow aggregation enables ISPs to detect traffic anomalies in a timely manner, it cannot preserve some critical information like IP addresses, port numbers, etc. Due to such missing information, it becomes very difficult (or often infeasible) for ISPs to identify the root causes of the traffic anomalies, and further resolve the network problems efficiently. We want to design an algorithm to help ISPs to recover these key information efficiently.

1.3.3 Low-rank Matrix Approximation

Spatial analysis methods have been proved to be effective in detecting coordinated low-profile cyber attacks that are not detectable at a single monitor. However, existing solutions have scalability problems in that they require high running time and space to analyze the traffic measurements, which makes it often infeasible to be deployed for monitoring large-scale high-speed networks. We design some algorithm to utilize the low-rank properties in the network traffic, and furthermore detect network-wide traffic anomalies and analyze the traffic activity graphs in an efficient way.

1.4 Contribution of Research

The contributions of our research are as follows:

- We propose a novel data structure to support the approximate membership query in the time-decaying window model. In this model, items are inserted one-by-one over a data stream, and we want to determine whether an item is among the most recent w items for any given window size $w \leq n$. Our data structure only requires $O(n(\log 1/\delta + \log n))$ bits and $O(1)$ running time. We also prove a non-trivial space lower bound, i.e. $(n - \delta m)\log(n - \delta m)$ bits, which guarantees that our data structure is near-optimal. Our data structure has been evaluated using both synthetic and real data sets.

- We propose two data streaming algorithms based on Cukoo-hashing Dictionary for tacking LDFs with only false negatives but no false positives. There is no short-lived flows that can be falsely detected in our algorithms. One of our algorithms is a deterministic algorithm which can provide a strong bound for the flow duration estimation. The other is a randomized algorithm which provides a best-effort solution for the flow duration estimation when the space is less than the lower bound of the deterministic algorithm. Their performance is evaluated using real network traces. We expect that the results from our work will improve the practice and ways of network monitoring and measurement.

- We propose an efficient data structure, namely the *fast* sketch, which can both aggregate packets into a small number of aggregated flows, and further enable ISPs to identify the causes of detected anomalies. Using the fast sketch, the number of aggregated flows can achieve the lower bound of the heavy-change detection, and the running time to either process each packet or identify anomalous keys is sub-linear, which enable a real-time anomaly detection. Our work can significantly improve the efficiency and the reliability of the traffic anomaly detection.

- We propose a new data streaming algorithm to identify high-cardinality hosts over the network-wide traffic measurements. Our algorithm introduces a new mergeable and reversible data structure for the distributed network monitoring system, which is designed by Noisy Group Testing. We have theoretically analyzed our algorithm and evaluated it against both synthetic and real-world data sets.

- We study the network-wide traffic anomaly detection problem. Our algorithm achieves $O(w \log n)$ running time and $O(w \log^2 n)$ space at local monitors. The NOC could run PCA-based detection method with $O(m^2 \log n)$ running time and $O(m \log n)$ space. Our algorithm also makes the ISPs be able to implement the detection method by paying careful consideration about the space requirement, the communication cost, and other resources over a distributed computing environment.

- We present a new sampling-based low-rank approximation method for monitoring TAGs. The resulted solution can reduce the computation complexity for the communication pattern analysis from $O(mn)$ to $O(m+n)$, where m and n denote the number of sources and destinations, respectively. The experimental results with real-world traffic traces show that our method outperforms existing solutions in terms of efficiency and the capability of processing and identifying unknown TAGs.

1.5 Dissertation Organization

The rest of this dissertation is structured as follows: Chapter 2 provides a review of literature for the problems targeted in our research. We propose a dynamic dictionary data structure for the problems of the click fraud and the botnet C&C connections in Chapter 3. Some reversible sketches are designed to identify heavy-change flows or high-cardinality hosts (super spreaders) in Chapter 4. In Chapter 5, we utilize the low-rank matrix approximation to detect coordinated traffic anomalies and monitor traffic activity graphs. We finally summarize our research and discuss our future work in Chapter 6.

CHAPTER 2. LITERATURE REVIEW

2.1 Approximate Membership Query over Time-decaying Windows for Click Fraud Detection

The approximate membership query has been widely studied in different research areas such as database system [46], computer architecture [17; 33], network security [69; 94], etc. Formally, a randomized data structure for the membership query stores a function $f(\cdot) : \mathcal{U} \rightarrow \{true, false\}$ for a set $\mathcal{A} \subset \mathcal{U}$, such that it will return true if a given item $x \in \mathcal{U}$ is present in $\mathcal{A}$, and otherwise false. Bloom [14] proposed the first data structure (Bloom filter) for this problem in the 1970's, which used a bit vector to represent $\mathcal{A}$. All the bits are initialized to 0 at the beginning. Each item in $\mathcal{A}$ is hashed using b independent hash functions and the corresponding bits are set to 1. Given a query whether an item x is present, it is hashed by the same hash functions. If all the corresponding bits equal to 1, it will return true and otherwise false. Due to the hash collisions, Bloom filter allows a few false positives but no false negative. By using $\log 1/\delta$ hash functions and a bit vector of the size $n \log 1/\delta \log_2 e$ ($\simeq 1.44 n \log 1/\delta$), Bloom filter can guarantee that the false positive error is less than a given small probability δ.

If no false negative is allowed, Bloom filter is quite close to the entropy lower bound [22] (i.e., $n \log 1/\delta$ bits). This lower bound has been achieved by using the matrix solving technique for a static set [80]. This method constructs a one-sided error dictionary which maps n items in $\mathcal{A}$ to n keys. Given an item x, a dictionary can retrieve its key if this item is present in $\mathcal{A}$. Otherwise, the dictionary may return any value. Such dictionary can be computed by solving sparse linear equations. By using a pairwise independent hash function to compute a $\log 1/\delta$-bits key for each item in $\mathcal{A}$, a dictionary can answer the approximate membership query in $O(1)$ running time. However, the computation of the dictionary requires at least $\Omega(n)$ running time.

In many real-world applications, $\mathcal{A}$ can be changed dynamically in a high speed. In this case, we cannot use Bloom filter or the optimal data structure. Currently, most data structures for the dynamic membership query are based on Bloom filter, which extends the bit vector to support additional operations on $\mathcal{A}$, e.g., Counting Bloom Filter (CBF) [43], Stable Bloom Filter (SBF) [33], Invertible Bloom Filter (IBF) [41], Timing Bloom Filter (TBF) [94], Variable-Increment CBF [82] and so on.

Metwally *et al.* [69] considered the problem of detecting duplicates in a time-decaying window model for the click fraud problem in the online advertisement. They applied the CBF to detect duplicates in a data stream, which replaced each bit in Bloom filter by a counter to record the count of the items hashed to each position. When a new item x comes, they will first query whether x is a duplicate, and then insert it into the CBF. They keep all recent n items in the memory in order to remove the oldest item from the CBF when a new item comes. Because all recent n items need to be kept in the memory, their method have to require a large memory.

Zhang *et al.* [94] introduced the TBF, in which each bit in Bloom filter was replaced by a $\log n$-bits time stamp for the last updating. In this way, old items can be removed from the TBF without maintaining the set $\mathcal{A}$ explicitly in the memory. We can detect any duplicated items using TBF with the same error bound as Matwally's method. But the space requirement of the TBF can be reduced to $\Omega(n \log 1/\delta \log n)$ bits, which is close to the lower bound.

Existing data structures for the time-decaying window model still require much more space than the entropy lower bound of the approximate membership query. The optimal data structure for the static set cannot handle dynamic sets in this model. There may exist a better lower bound for the approximate membership query in the time-decaying window model, which has not been studied before. In this paper, we attempt to fill the gap between the practice and the theory for the approximate membership query in the time-decaying window model.

2.2 Tracking Long Duration Flows for Botnet C&C Communications

Chen *et al.* [25] recently proposed the first algorithm for tracking LDFs in the network traffic. They implemented two Counting Bloom Filters (CBF) [43], denoted by $B_i[\cdot]$ for $i = 1, 2$, to estimate the flow durations, and further identify LDFs. The duration of a flow f was estimated as

$$B_i(f) = \min_{k=1,\ldots,K} \{B_i[h_{ik}(f)]\} \tag{2.1}$$

where $\{h_{i1}, \ldots, h_{iK}\}$ was a set of hash functions associated with B_i for $i = 1, 2$, and $B_i[h_{ik}(f)]$ denoted the value of the counter at the location $h_{ik}(f)$ in B_i. Here, B_1 was used to estimate the duration of each flow until the previous interval, and B_2 was used to estimate the duration of each flow until the current interval. Both were initialized with all 0s at the beginning. When a new packet (f, t) arrived, B_2 would be updated as following.

- If $B_1(f) = 0$, this flow was new and inserted into B_2 by updating $B_2[h_{2k}(f)]$ with $\max(1, B_2[h_{2k}(f)])$ for $k = 1, \ldots, K$.

- If $0 < B_1(f) < d - 1$, this flow already existed. Its counters $B_2[h_{2k}(f)]$ were updated with $\max(B_1(f) + 1, B_2[h_{2k}(f)])$ for $k = 1, \ldots, K$.

- If $B_1(f) = d - 1$, this flow was identified as a LDF and added to the hash table of the LDFs. Its counters were updated as the same as the second case.

- If $B_1(f) \geq d$, this flow was an existing LDF. They checked whether f was in the hash table of the LDFs. If so, its counters were updated as the same as before. Otherwise, this was an error due to the collisions in the CBF and this packet would be skipped.

At the end of each interval, B_1 was replace by B_2, and B_2 was set to be all 0s.

Chen's algorithm can guarantee that the estimated duration $B_1(f)$ for any flow f can be bounded by εD with a probability at least $1 - \delta$,

$$d_f \leq B_1(f) \leq d_f + \varepsilon D \tag{2.2}$$

where d_f is the true duration of f, and D denotes the sum of the durations. The space requirement in Chen's algorithm is $O(\frac{1}{\varepsilon}\log(\frac{1}{\delta}))$ and the running time per packet is $O(\log(\frac{1}{\delta}))$.

Because Chen's algorithm always overestimates the flow duration, there are only false positives but no false negative. Short-lived flows can be falsely detected as LDFs. However, a false negative algorithm only inserts true LDFs into the hash table for monitoring, which is much more space-efficient than Chen's algorithm. If the number of flows is very large in high-speed networks, Chen's algorithm becomes unpractical due to the large space used for tracking LDFs. We propose data streaming algorithms for tracking LDFs with only false negatives, which are more space-efficient for tracking LDFs in high-speed networks.

If we only want to estimate the traffic volume v_f for a give flow f, the Count-Min (CM) sketch [30] provides the best-known result so far. By using $O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\right)$ counters, the estimate $\hat{v}_f$ from the CM sketch for a given flow f can be bounded by a small error ϵ,

$$|\hat{v}_f - v_f| \leq \epsilon V, \tag{2.3}$$

with a probability at least $1-\delta$. Although, the CM sketch can be used to find the heavy hitters, we have to query each flow key f, and keep the top-k keys with the largest estimated traffic volume. The problem is that the recovery algorithm takes $\Omega\left(n\log(1/\delta)\right)$ running time at least, which will be impractical if the range of the keys n is large. Unfortunately, this bottleneck is a practical problem for the network traffic anomaly detection. We have a huge number of flows in the current Internet, and it will become much larger in the foreseeable future due to the implementation of the Internet Protocol version 6. Therefore, we must find a data structure which can recover the frequent items efficiently, e.g., in a sub-linear running time.

2.3 A Fast Sketch for Heavy-change Detection over High-Speed Network Traffic

In order to efficiently identify IP addresses with a heavy change in traffic volumes, the *combinatorial group testing* (CGT) sketch [31] has been proposed by the same authors of the CM sketch. There are at most k IPs with $|v_{IP}| > \frac{1}{k+1}V$, which are unknown for us. If we put an IP with a number of IPs having small traffic, we can identify it as a majority by the group testing technique [38]. The CGT sketch divides the whole IP addresses into $2k$ groups randomly, and picks out the majority from each group if there exists one. By re-separating the addresses $\log(k/\delta)$ times, we can guarantee that all top-k IPs can be picked up as a majority with a probability at least $1-\delta$. For each group, the CGT sketch maintains $(1+\log N)$ counters, each of which is corresponding to one bit in the binary representation of the IP addresses, where $N = 2^{32}$ is the size of the range of the IPv4 addresses. The CGT sketch picks out the top-k IPs by finding the counters larger than a threshold $\theta = \Delta V/(k+1)$. However, the CGT sketch will maintain a large number of counters, which becomes impractical in a high-speed network.

In order to handle high-speed networks, Schweller et. al. proposed the *Reversible* sketch (RevSketch) [83] for the change detection, and Guan et. al. proposed a sketch based on *Chinese Reminder Theorem* (CRT) [47] for the super-spreader detection. Both of them utilize the position information to recover the keys in their sketch. A similar recovery method has also been proposed for the traffic anomaly detection with the Principle Component Analysis (PCA) [66]. However, the running time to recover keys in them cannot achieve the sub-linear bound, which may be up to $\Omega(n)$ in various cases.

2.4 Identifying High-Cardinality Hosts (Super Spreaders) from Network-wide Traffic Measurements

For the cardinality estimation, there has been a lot of work done for the centralized data processing [9; 39; 44; 58]. D. Kane *et al.* [58] developed an optimal algorithm which can guarantee a half decent $(1\pm\epsilon)$-approximation of the cardinality at all time points, using $O(\frac{1}{\epsilon^2} + \log n)$ space and $O(1)$ updating and reporting times. An efficient algorithm for the cardinality estimation requires both approximation and randomness. In order to provide a small false positive/negative error δ, Kane's algorithms have to be repeated $\log(1/\delta)$ times independently. If Kane's algorithm is used in the reversible sketch [83; 31; 67] designed for the change detection, the space and the running time would be very high for the detection of the high-cardinality hosts. Therefore, we propose a more efficient data structure than trivial solutions in this paper, which considers the errors in the cardinality estimation.

Venkataraman *et al.* [91] proposed the first efficient algorithm to detect super-spreaders, which sampled packets from a set of distinct source-destination pairs. A super-spreader detection algorithm contains two steps: the cardinality estimation and the host identification. Similar sampling method was also used by Cao *et al.* [21] to detect hosts with moderately large number of the connections in the network. Zhao *et al.* [97] proposed a sketch-based algorithm for the super-spreader detection. Their method was based on the Bloom Filter, which consisted of two dimension bit arrays. Each packet was hashed by using independent hash functions into different positions in the bit arrays, and the bit on each position was set to 1. The number of connections at each host can be estimated based on the number of 1s in the bit arrays. Their algorithm was improved by Guan *et al.* [47], who used Chinese Remainder Theorem to speed up the super-spreader identification.

The reversible sketch may be firstly proposed for the frequent item detection [31]. They applied the *combinatorial group testing* [38] to efficiently identify the top-k frequent items. There are at most k items with a frequency larger than the threshold in the data stream, which are unknown for us. If we put an item with a number of items having small frequencies, we can identify it as a majority by the group testing technique. The CGT sketch [31] divides the whole universe into $2k$ groups randomly, and picks out the majority from each group if there exists one. By re-separating the universe $\log(k/\delta)$ times, we can guarantee that all top-k frequent items can be picked up as a majority with a probability at least $1-\delta$. Schweller *et al.* proposed another *Reversible* sketch [83] for the change detection in the network traffic. Both sketches utilize the position information to identify interested hosts in the network. A similar recovery method has also been proposed for the traffic anomaly detection with the Principle Component Analysis (PCA) [66] and the aggregate queries in the high-speed network [67].

2.5 Sketch-based Network-wide Coordinated Traffic Anomaly Detection

Traffic anomaly detections have become an important issue for the network management in the Internet, which have obtained considerable research interests [66; 50; 51; 26; 60; 15]. For the high-profile traffic anomalies, researchers can apply signal analysis methods to detect them [10]. To deal with the low-profile coordinated traffic anomalies, Lakhina et al. [62; 63] proposed PCA-based detection methods by utilizing traffic measurements from multiple links. The packets were aggregated into origin-destination (OD) flows and the traffic volume for each

OD flow was updated to the NOC for every 5-minutes interval. Lakhina's method required $O(mn)$ space to maintain traffic volumes and $O(m^2n)$ running time to compute PCA, where n was the number of intervals and m was the number of OD flows. Li et al. [66] aggregated flows into sketch subspaces and also detected traffic anomalies based on the PCA. Their method can help ISPs to identify the IP addresses related to the traffic anomalies but it involved a large number of aggregated flows which required at least as much computation as Lakhina's method. Due to the high communication cost in Lakhina's method, several methods have been proposed. Huang et al. [50] designed a local algorithm to filter data at the local monitor in order to avoid excessive use of the network-wide communication. A local monitor would send its data to the NOC only if the local error exceeded a user-specified tolerance. Chhabra et al. [26] also proposed a method to reduce the communication cost, which used the generalized quantile sets (GQSs) to identify a set of candidate anomalies at each local monitor. Then a local monitor communicated its detection results with other local monitors to finally detect traffic anomalies. However, the computation overhead at the NOC cannot be reduced by using the above methods. Kline et al. [60] utilized Bayes Net to identify potential anomalous traffic from traffic volumes and correlations between ingress/egress packet and bit rates. Also, the temporal correlation was introduced to improve the accuracy of the PCA methods [15]. Such methods used more traffic information than Lakhina's method, and also required higher computation complexity.

Data streaming algorithms have been widely deployed for the traffic measurement and monitoring in real time [4]. Such algorithms usually use limited memory space to compute some functions with continuous traffic updating and without retrieving previous measurements [97]. PCA has also been applied for mining multiple data streams [48; 78; 79]. A more challenging problem is the sliding window model [32], where the streaming algorithm only focuses on the recent elements within a time window. According to our knowledge, there has been no work about the PCA computation in the sliding window model. This paper provides a novel distributed streaming algorithm for the PCA computation over traffic measurements in the sliding window model, which can used to detect network-wide traffic anomalies in real time.

2.6 Monitoring Traffic Activity Graphs with Low-rank Matrix Approximation

Packet sampling has been widely used for the network monitoring and management. Uniform packet sampling, e.g., Cisco's NetFlow, has been implemented in most of the routers. Because the uniform packet sampling cannot provide a high accuracy about the network usage, Estan *et. al.* [42] proposed a sample-and-hold method which only focused on heavy hitters to improve the measurement accuracy. However, the above methods can only provide a poor estimation of the TAGs, because they mainly focused on traffic volumes rather than communication patterns. *FlexSample* implemented a sketch-guided sampling method to monitor low-volume flows, which can be used to monitor the TAGs for some given traffic subpopulation. However, such methods cannot be used to study the communication patterns for unknown applications or malwares, because their subpopulation were unknown during the packet sampling.

In order to remain the usability of the TAGs, a sampling method that can provide enough information for all possible traffic subpopulation is required. Sekar *et. al.* [85] proposed cSAMP to provide a high flow coverage for all traffic subpopulation. Each router used a key derived from the packet header, and computed the hash of the key. If the hash fell in the range assigned to this router, this packet would be sampled. Otherwise, it would be skipped. The cSAMP can remain more information about the TAGs than the other existing methods. However, the sampled TAGs generated by cSAMP become more sparse, it becomes even more difficulty to discover the communication patterns. Our sampling method follows the cSAMP but aims to provide a high *host* coverage to preserve communication patterns and community structures.

Due to the limitation of the sampling methods, we are willing to improve the measurement accuracy and get better estimations based on extra knowledge about the network traffic. For any given low-rank matrix with random missed values, Cai *et. al.* [19] introduced the singular value thresholding algorithm to recover missing values, which was mainly used to recovery dense matrices. Zhang *et. al.* [96] used compressive sensing to recover missing values in the traffic matrices. Their method depended on the sparse and low-rank nature of the real-world traffic matrices. But the computation cost in [96] was almost the same as the tNMF method, which was unpractical for monitoring the TAGs. Clarkson *et. al.* [27] considered the rank-k approximation problem in the data streaming model. Their method maintained sketches for a given matrix. If the TAGs are known before sampling, we can use this method to maintain sketches for the TAGs. However, this method cannot be used to monitoring the TAGs for unknown applications or malwares.

The low-rank approximation of a large matrix has been studied for more than decades. A popular method is the CUR decomposition [37], which approximates a matrix $\mathbf{A}$ by three matrices, i.e. $\mathbf{C}$, $\mathbf{U}$, and $\mathbf{R}$, where $\mathbf{C}$ and $\mathbf{R}$ contain a set of scaled columns and rows sampled from $\mathbf{A}$. This decomposition can preserve the sparse of the matrix, which are useful for many applications. Sun *et al.* [88] improved the CUR decomposition by removing the duplicated columns and rows. Furthermore, Tong *et al.* [89] showed that the CUR decomposition can be further improved by eliminating linearly dependent columns and rows. Nguyen *et. al.* [75] provided a fast and efficient algorithm to sample rows and columns from preprocessed sets in order to spread out information of every columns and rows. However, existing CUR decomposition methods require the whole matrix saved in the disk, and multiple passes to sample rows and columns depending on a specific distribution. Our problem is different from previous ones in [37; 75; 88; 89]. We can only sample elements in the matrix $\mathbf{A}$ in one pass, and cannot retrieve previous elements if they are not sampled.

CHAPTER 3. DYNAMIC DICTIONARY DATA STRUCTURE

3.1 Approximate Membership Query over Time-decaying Windows for Click Fraud Detection

There has been a long history of finding a space-efficient data structure to support approximate membership queries, started from Bloom's work in the 1970's. Given a set $\mathcal{A}$ of n items and an additional item x from the same universe $\mathcal{U}$ of a size $m \gg n$, we want to distinguish whether $x \in \mathcal{A}$ or not, using small (limited) space. The solutions for the membership query are needed for many network applications, such as cache directory, load-balancing, security, etc. If $\mathcal{A}$ is static, there exist optimal algorithms to find a randomized data structure to represent $\mathcal{A}$ using only $(1+o(1))n \log 1/\delta$ bits, which only allows for a small false positive δ but no false negative. However, existing optimal algorithms are not practical for many Internet applications, e.g., social network services, peer-to-peer systems, network traffic monitoring, etc. They are too space- and time-expensive due to the frequent changes in the set $\mathcal{A}$, because all items are needed to recompute the optimal data structure for each change using a linear running time. In this section, we propose a novel data structure to support the approximate membership query in the time-decaying window model. In this model, items are inserted one-by-one over a data stream, and we want to determine whether an item is among the most recent w items for any given window size $w \leq n$. Our data structure only requires $O(n(\log 1/\delta + \log n))$ bits and $O(1)$ running time. We also prove a non-trivial space lower bound, i.e. $(n - \delta m) \log(n - \delta m)$ bits, which guarantees that our data structure is near-optimal. Our data structure has been evaluated using both synthetic and real data sets.

3.1.1 Introduction

Membership query (or duplication detection) is a common query in a wide range of applications, which looks up whether a given item x is present in a large set $\mathcal{A}$ or not. In this section, we propose a space-efficient randomized data structure to support approximate membership queries over a time-decaying window model, which can determine whether an item x is among the most recent w items or not for any $w \leq n$. Such data structure can provide an efficient solution for various Internet applications, such as:

- *Caching and replication [17; 43]:* On a cache miss, a proxy will try to get a desired web page from other proxies before obtaining this page from the Internet. Furthermore, it can fetch the web page from a proxy who has the most recently updated web page.
- *Web crawling [18]:* A search engine may use multiple independent crawlers to harvest URLs from the Internet. A data structure can be shared by crawlers to determine whether an URL has been fetched recently or not.
- *Network security [57; 70]:* Many network security problems can be solved by detecting and removing duplicated packets within a short interval over the incoming traffic. For example, duplicated requests from a denial-of-service attack can be detected and blocked by a web server.
- *Click frauds [69; 94]:* In a pay-per-click model, an advertiser pays the syndicator/publisher for each click on their advertisement pages. An attacker can earn extra incomes or deplete competitor's budget by simply clicking the advertisement pages. An important issue for the online advertisement is to validate each click and detect duplicates in a click stream.
- *Peer-to-peer application:* A data structure for the membership query can be used to provide a distributed search engine for the peer-to-peer application. Each peer can determine whether it has the keywords in the search queries efficiently.
- *Social networks [92]:* When a user login into a social network, the server needs to find recent "News feeds" or "Tweets" from his/her friends.In order to reduce the database

accesses, the server wants to determine whether a friend added “News feeds” or “Tweets” recently or not using an efficient data structure for the membership query.

In these systems and applications, we are not only interested in whether an item is present in the set $\mathcal{A}$ or not, but also its order based on the arrival/updating time. Such information can be used to remove redundancy in the massive data streams resulting in resource and compute efficiency for downstream processing [40]. The set $\mathcal{A}$ will change dynamically in a stream of events. A data structure for the approximate membership query needs to be able to provide the most recent updated information for each item as well.

In this section, we study the problem of the approximate membership query in a time-decaying window model. Our main contributions are summarized as following.

- We propose a novel data structure using Cuckoo hashing [77] to answer approximate membership queries in a time-decaying window model.

- Our data structure only requires $O(n(\log 1/\delta + \log n))$ bits and $O(1)$ running time, which provides a better bound in both space and running time than existing solutions [69; 94].

- We also provide the first space lower bound for the approximate membership query in the time-decaying window model, i.e. $(n - \delta m)\log(n - \delta m)$ bits, which guarantees that our data structure is near-optimal.

- We evaluate our data structure with both synthetic and real data sets, which shows the efficiency of our data structure in various applications.

3.1.2 Problem Definition

We consider a data stream of items [3; 8; 45; 71],

$$x_1, x_2, \ldots, x_i, \ldots \tag{3.1}$$

which are coming one-by-one at high speed. Let $\mathcal{A}_w$ denote the set that only includes the most recent w items,

$$\mathcal{A}_w = \{x_{i-w+1}, \ldots, x_i\}, \tag{3.2}$$

where $w = 1, \ldots, n$, and n is the maximum possible window size for the approximate membership query. The problem of the approximate membership query in a time-decaying model can be defined as follows:

Definition 1. Given any item $x \in \mathcal{U}$ and a window size $w \leq n$, a data structure for the approximate membership query in a time-decaying window model can determine whether there is an identical item among the most recent w items with the following guarantees,

- If $x \in \mathcal{A}_w$, it will always return true.
- If $x \notin \mathcal{A}_w$, it will return false with a probability at least $1 - \delta$, and return true with a probability at most δ.

3.1.3 Approximate Membership Queries over Time-decaying Windows

In this section, we propose a novel data structure to support approximate membership queries in a time-decaying window model. TBF [94] is the only existing data structure that can work in this model, which was proposed to answer approximate membership queries in a sliding window model as CBF [69]. TBF replaces each bit in Bloom filter by a counter to maintain a timestamp. Because each timestamp requires $\Omega(\log n)$ bits and each item is associated with $\Omega(\log 1/\delta)$ timestamps, the space of the TBF is at least $\Omega(n \log 1/\delta \log n)$ bits. To reduce the space requirement, we design a data structure to reduce the number of timestamps for each item to one, which is similar to the optimal algorithm for the static set [80].

In this section, we first describe an ideal algorithm to show the main ideas in our data structure and its difference to existing algorithms. Next, a detailed description of our data structure is provided in the algorithm part. Last, we prove its performance in the theoretical analysis part.

3.1.3.1 An Ideal Algorithm

Our main idea is to use a dictionary structure to support approximate membership queries in the time-decaying window model. Given an item x, the dictionary can retrieve its key $K(x)$ if x is among recent n items in the stream. Otherwise, the dictionary just returns any value.

Each key $K(x)$ consists of two parts: a $\log 1/\delta$-bits signature $S(x)$ and a $\log n$-bits timestamp $T(x)$. The signature is used to determine whether $K(x)$ is the correct key for the given item x, and the timestamp is used to determine whether x is among the recent w items if the key is correct. In order to process a stream of items, we need two additional operations over the dictionary:

- *Insertion:* Once a new item comes, we need to insert its key including a signature and a timestamp into the dictionary.
- *Deletion:* And at the same time, we need to remove the oldest key from the dictionary because its item has been expired.

Our primary goal is that each operation only requires $O(1)$ running time in the ideal algorithm. In addition, the space of the dictionary can be bounded by $O(n(\log 1/\delta + \log n))$, where each key requires $\log 1/\delta + \log n$ bits and there are at most n keys in the dictionary.

The dictionary-based solution provides a better guarantee in both running time and space than TBF [94]. Because the dictionary only maintains one key for each item, the space is much smaller than TBF. And the query is also simplified. The dictionary only needs to retrieve one key for each query, but TBF needs to check $\log 1/\delta$ timestamps.

3.1.3.2 Our Data Structure

Our data structure uses the Cuckoo Hashing [77] to design a dictionary to maintain the keys for each item. Two hash tables are used to resolve hash collisions. In order to provide a constant running time for each operation, the size of the hash table should be twice as large as the number of items. This may cause a poor space utilization compared to a conventional hash table with linked overflow chains. To solve this problem, we use the variable-bit-length array (VLA) [13] to implement the hash tables. The VLA is a space-efficient data structure that can maintain an array $C[\cdot]$ of the length ℓ using only $O(\ell+\sum len(C[\cdot]))$ bits, where $len(C[\cdot])$ denotes the bit length of the value stored at a counter $C[\cdot]$. In other words, an empty position in the hash table will only cost $O(1)$ bits. Because chained hashing needs to maintain an additional link for each hash collision, our algorithm can provide a better space utilization. Furthermore,

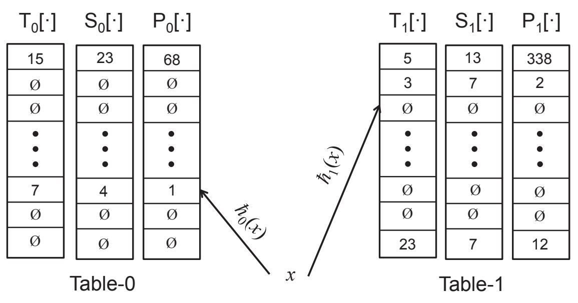


Figure 3.1 Cuckoo-Hashing Dictionary

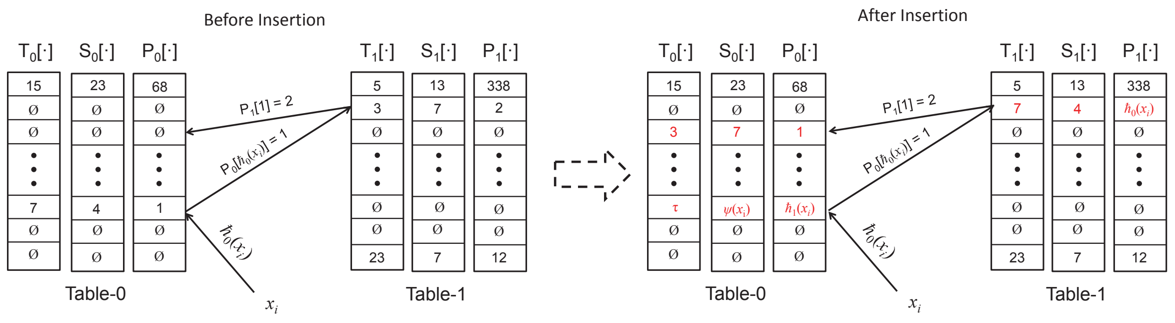


Figure 3.2 Insertion in Cuckoo-Hashing Dictionary

the running time of Cuckoo Hashing has been shown to be better than chained hashing and other conventional methods [7; 98].

There are two hash tables as shown in Fig.3.1, denoted by Table-0 and Table-1. Each hash table is associated with an independent universal hash function for the lookups, denoted by $\hbar_0(\cdot)$ and $\hbar_1(\cdot)$. The information of each item is maintained in either Table-0 or Table-1, but never both of them. At each position in both hash tables, we maintain three counters, i.e., the timestamp $T_j[\cdot]$, the signature $S_j[\cdot]$, and the position in the other table $P_j[\cdot]$, where $j = 0, 1$ denotes the index of the hash table.

- $T_j[\cdot]$: Each timestamp is chosen from $\{0, 1, 2, \ldots, 2n\}$, and the empty entry is associated with $\emptyset = 0$.
- $P_j[\cdot]$: Each position $P_j[\cdot]$ is in the range $\{0, \ldots, \eta - 1\}$, where η denotes the size of the hash table. We can use $P_j[\cdot]$ to move the information for each item between two hash tables to resolve hash collisions.
- $S_j[\cdot]$: Each signature is computed by an independent hash function $\psi(\cdot)$, and the bit

length of a signature is chosen as $max\{0, \log(1/\delta) - \log\eta + 1\}$.

Both the position and the signature are used together to determine whether this is a correct key for a given item.

Besides these two hash tables, we also maintain a linked list, denoted by $\mathcal{L}$, which is used to store items that cannot be inserted into either hash table due to hash collisions. The standard Cuckoo hashing will rehash all items if we cannot insert an item into the hash table. Because we lose the information of original items due to the space limitation, we cannot rehash them into a new hash table. Therefore, if an item cannot be inserted into either hash table, we will maintain this item and its timestamp in the linked list $\mathcal{L}$. If the size of the hash table η is large enough, the insertion failure can only happen with a small probability.

When an item x_i comes, we first update the current timestamp τ as a wrapround counter,

$$\tau = i \bmod (2n) + 1. \tag{3.3}$$

We compute its positions in two tables and check whether one of them has the same signature $S_j[\hbar_j(x_i)] = \psi(x_i)$ and the same position $P_j[\hbar_j(x_i)] = \hbar_{j'}(x_i)$. If so, this item has already been maintained and we only update its timestamp to τ. Otherwise, we check the distance between the timestamps at the position of the new item x_i in both tables and the current timestamp τ. The distance $d(T_j[\hbar_j(x_i)], \tau)$ equals to the number of timestamps from $T_j[\hbar_j(x_i)]$ to τ.

$$d(T_j[\hbar_j(x_i)], \tau) = \begin{cases} \tau - T_j[\hbar_j(x_i)] + 1 & \text{if } \tau \geq T_j[\hbar_j(x_i)] \\ \tau + 2n - T_j[\hbar_j(x_i)] + 1 & \text{if } \tau < T_j[\hbar_j(x_i)] \end{cases} \tag{3.4}$$

If $d(T_j[\hbar_j(x_i)], \tau) > n$, we know that this timestamp has already been outside the maximum time-decaying window, and we consider it as expired. If one of the timestamps is expired $(d(T_j[\hbar_j(x_i)], \tau) > n)$ or empty $(T_j[\hbar_j(x_i)] = 0)$, we can insert x_i into this table as

$$\begin{aligned} T_j[\hbar_j(x_i)] &= \tau \\ S_j[\hbar_j(x_i)] &= \psi(x_i) \\ P_j[\hbar_j(x_i)] &= \hbar_{j'}(x_i) \end{aligned} \tag{3.5}$$

where $j, j' \in \{0, 1\}$ and $j = \neg j'$.

If there is no position expired or empty, we will randomly select one hash table to insert x_i by moving timestamps and their signatures between two hash tables. We insert x_i into Table-j by moving the previous timestamp $T_j[\hbar_j(x_i)]$ and its signature $S_j[\hbar_j(x_i)]$ into the other table at the position $P_j[\hbar_j(x_i)]$. If the timestamp $T_{j'}[P_j[\hbar_j(x_i)]]$ in the other table is expired or empty, we can set them as

$$\begin{aligned} T_{j'}[P_j[\hbar_j(x_i)]] &= T_j[\hbar_j(x_i)] \\ S_{j'}[P_j[\hbar_j(x_i)]] &= S_j[\hbar_j(x_i)] \\ P_{j'}[P_j[\hbar_j(x_i)]] &= \hbar_j(x_i) \end{aligned} \tag{3.6}$$

and stop. Otherwise, we move the timestamp $T_{j'}[P_j[\hbar_j(x_i)]]$ and its signature $S_{j'}[P_j[\hbar_j(x_i)]]$ to the position $P_{j'}[P_j[\hbar_j(x_i)]]$ in Table-j, and set them in a similar way as Eq.(3.6). We can continue the move operation until we find an expired or empty timestamp, as shown in Fig.3.2.

However, there is a small probability that the insertion procedure may fail. If we cannot find an expired or empty timestamp within $O(\log n)$ move operations, we will keep all timestamps and their signatures in their original positions, and insert this new item x_i and its timestamp $\tau_{x_i} = \tau$ into the linked list $\mathcal{L}$,

$$\mathcal{L} = \mathcal{L} \cup \{(x_i, \tau_{x_i})\}. \tag{3.7}$$

The linked list $\mathcal{L}$ is implemented as a doubly linked list where the head is the newest item and the tail is the oldest one.

Given an item $x \in \mathcal{U}$ and a time-decaying window size w, we use two hash functions $\hbar_0(\cdot)$ and $\hbar_1(\cdot)$ to find its positions in both hash tables, as shown in Fig.3.1. If one of them, e.g. Table-j, has the following properties,

- $T_j[\hbar_j(x)]$ is not expired or empty,
- $S_j[\hbar_j(x)] = \psi(x)$,
- $P_j[\hbar_j(x)] = \hbar_{j'}(x)$ for $j' = \neg j$,

we get a correct key for this item. Next, we will compute the distance between $T_j[\hbar_j(x)]$ and τ to determine whether this item is within the time-decaying window. If $d(T_j[\hbar_j(x)], \tau) \le w$,

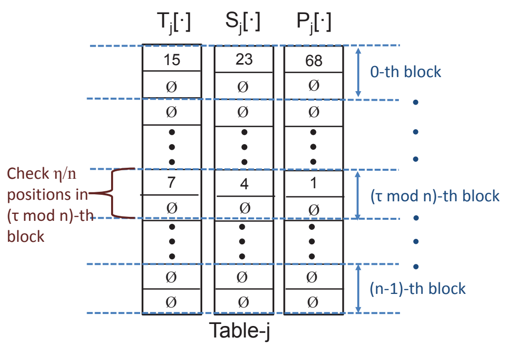


Figure 3.3 Deletion in Cuckoo-Hashing Dictionary

we will return true, and otherwise false. If there is no position with the above properties, we check whether x is maintained in the linked list $\mathcal{L}$. If $x \in \mathcal{L}$ and $d(\tau_x, \tau) \leq w$, we will return true, and otherwise false.

After the insertion procedure, we check $\lceil \eta/n \rceil$ positions in each hash table, and for each of them, we reset it to empty if its timestamp gets expired. To do this, we can simply divide the hash table into n blocks and each block contains $\lceil \eta/n \rceil$ positions. At each step, we check the positions in the $(\tau \bmod n)$-th block, as shown in Fig.3.3. In this way, we can reset an expired timestamp to empty before τ counts back to the same value. We also check the last item in $\mathcal{L}$ and delete it if its timestamp gets expired.

3.1.3.3 Theoretical Analysis

We first prove that our data structure can provide a false positive error bound for the approximate membership query in the time-decaying window model.

Theorem 1. Given the bit length of a signature as $len(S_j[\cdot]) = max\{0, \log(1/\delta) - \log \eta + 1\}$, our data structure can answer the approximate membership query in the time decaying window model with no false negative and a false positive error at most δ.

Proof. Given an item $x \in \mathcal{U}$ and the size of the time-decaying window w, we want to determine whether $x \in \mathcal{A}_w$ or not. If $x \in \mathcal{A}_w$, there is an item $x_{i'}$ in $\mathcal{A}_w$ such that $x_{i'} = x$. The timestamp of $x_{i'}$ must be maintained in either one of the hash tables or the linked list. The distance from its timestamp to τ must also be at most w. If $x_{i'}$ is maintained in one of the hash table, we can always find its signature and return true. Otherwise, we can find $x_{i'}$ in the linked list and also return true. Therefore, there is no false negative.

If $x \notin \mathcal{A}$, we will return true only if there are the same signature $\psi(x)$ and the same position $P_j[\hbar_j(x)]$ at either $\hbar_0(x)$ or $\hbar_1(x)$. Because the size of the signature is at least $\log(1/\delta)-\log\eta+1$, there are $2^{\log(1/\delta)-\log\eta+1+\log\eta} = 2/\delta$ possible pairs of the position and the signature. Thus, the probability that two items has both the same signature and the same position is at most $\delta/2$. Therefore, the probability that one of the two hash tables has the same signature and the same position as x is at most δ. In sum, the false positive error is δ at most . $\square$

The standard Cuckoo hashing [77] has the worst case constant lookup time and amortized expected constant time for updates with only $O(n)$ space. We can choose the size of the hash table as $\eta = 2n$ to provide a constant running time. Our data structure has the following property.

Theorem 2. Given $\eta = 2n$, our data structure only uses $O(1)$ running time and $O(n(\log 1/\delta + \log n))$-bits space for the approximate membership query in the time-decaying window model.

Proof. Running Time:

For the query, we only need to check two positions in the hash table and all items in the linked list. Because the size of $\mathcal{L}$ is $O(1)$ in average, the running time for the approximate membership query is bounded by $O(1)$ in average.

For the insertion, we first try to insert an item into one of the hash tables using Cuckoo hashing. Based on the property of Cuckoo hashing [77], the expected number of moves is bounded by $O(1)$ with the size of the hash table $\eta = 2n$ in our data structure. If the insertion fails, we will insert this item into the linked list $\mathcal{L}$, which will only take $O(1)$ running time. Therefore, the running time to insert an item into our data structure is $O(1)$ in average.

For the deletion of expired timestamp, we need to check two positions in each hash table, which takes $O(1)$ running time. For the linked list, we only need to check whether the item at the tail is expired or not, which also takes $O(1)$ running time. Therefore, it only takes $O(1)$ running time to remove the information of the expired items from our data structure.

Space:

The bit length of each timestamp is $\log(2n)$. The total bit length of a position and a signature is at most $\max\{\log 1/\delta + 1, \log 2n\}$. There are $2n$ positions in two hash tables. Therefore, the total space of the hash tables in our data structure is $O(n(\log n + \log 1/\delta))$.

Based on the property of the Cuckoo hashing [77], the probability that an insertion fails is at most $O(1/n)$. We will insert an item into the linked list if and only if there is an insertion failure in the Cuckoo hashing. Thus, the size of the linked list $\mathcal{L}$ is $n \times O(1/n) = O(1)$ in average. Therefore, the space of the linked list is $O(\log m + \log n)$, which should be much smaller than n.

In sum, the space requirement of our data structure is bounded by $O(n(\log n+\log 1/\delta))$. $\square$

3.1.4 Lower Bound

If $\log n = O(\log 1/\delta)$, the space requirement of our data structure is already within a constant factor of the entropy lower bound $n \log 1/\delta$. Our data structure may not be optimal only if n is sufficiently large. In this section, we provide a non-trivial lower bound for the approximate membership query in the time-decaying window model, when n is sufficient large and δ is small. As far as we know, this is the first work to provide a lower bound for this problem in the time-decaying window. This lower bound can guarantee that our data structure is still near-optimal if n is very large.

We consider the randomized data structures which have only false positives but no false negative. The space requirement is only related to the bound of the false positive error δ, the size of the maximum time-decaying window n, and the size of the universe m. Our lower bound is provided in the following theorem.

Theorem 3. Given a maximum window size n, a universe of the size m, and a false positive error bound δ, a randomized data structure for the approximate membership query in the time-decaying window model must require at least

$$(n - \delta m)\log(n - \delta m) \tag{3.8}$$

bits in space.

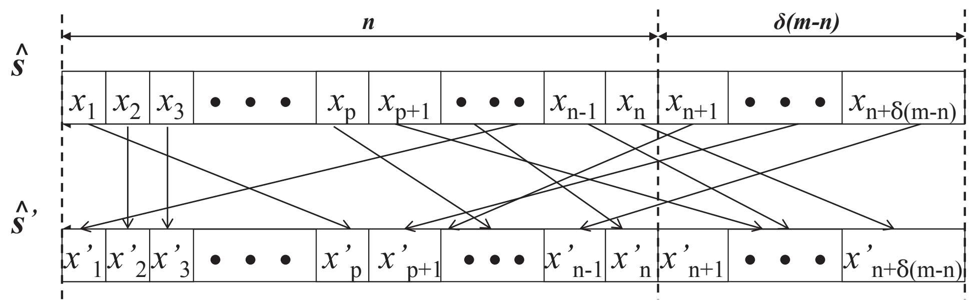

n
δ(m-n)
ŝ
ŝ'
x1 x2 x3 • • • xp xp+1 • • • xn-1 xn xn+1 • • • xn+δ(m-n)
x'1 x'2 x'3 • • • x'p x'p+1 • • • x'n-1 x'n x'n+1 • • • x'n+δ(m-n)

Figure 3.4 An example of the permutation from $\mathbf{s}$ to $\mathbf{s}'$

In a time-decaying window model, a randomized data structure needs to maintain the timing information in order to remove expired items. As a consequence, the same set of items may require several binary representations, i.e. the instances in the randomized data structure, to record their orders, and thus the space requirement will also become larger than that for the static set. Based on this property, our main idea to find a new lower bound is that an adversary can also observe the items in the data stream, and make queries using an observed item rather than a randomly selected item.

3.1.4.1 Notation

Let $\mathbf{s}$ denote a sequence with n distinct items,

$$\mathbf{s} = < x_1, \ldots, x_p, \ldots, x_n >, \tag{3.9}$$

where $x_p \in \mathcal{U}$ for $p = 1, \ldots, n$, and $x_p \neq x_q$ for $\forall p \neq q$.

Let $\mathcal{A}_\mathbf{s}$ denote the set of items in the sequence $\mathbf{s}$, and $\mathcal{S}$ denote the set of such sequences with n distinct items,

$$|\mathcal{A}_\mathbf{s}| = n \text{ for } \forall \mathbf{s} \in \mathcal{S}. \tag{3.10}$$

The total number of the sequences with n distinct items is

$$|\mathcal{S}| = m(m-1)\cdots(m-n+1). \tag{3.11}$$

For two sequences $\mathbf{s}, \mathbf{s}' \in \mathcal{S}$, the Hamming distance is defined as the number of positions at which the corresponding items are different,

$$\|\mathbf{s} - \mathbf{s}'\| = |\{p \,|\, x_p \neq x'_p\}|. \tag{3.12}$$

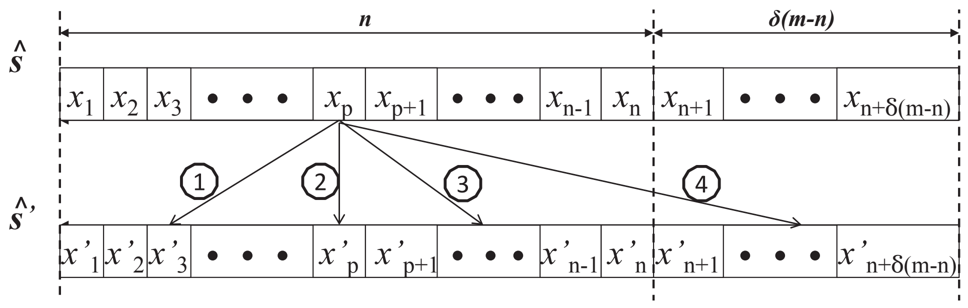


Figure 3.5 Possible permutations of an item

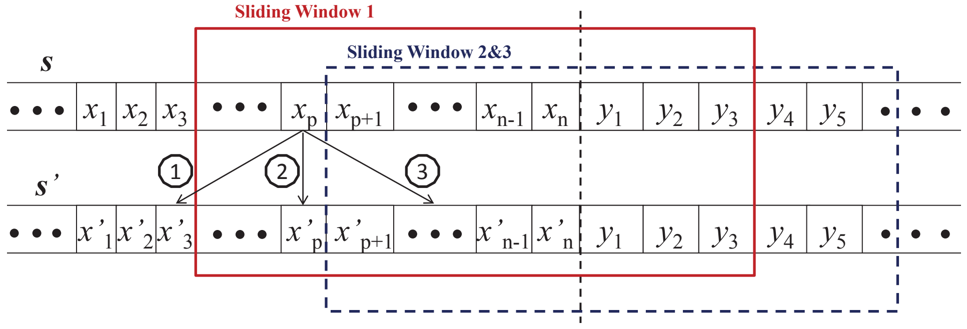


Figure 3.6 False positive errors when $p, q \leq n$

Given a sequence $\mathbf{y} =< y_1, \ldots, y_g, \ldots, y_n >$ in $\mathcal{S}$, let $\mathcal{S}_{\notin \mathbf{y}}$ denote the set of sequences that do not include any item in $\mathbf{y}$,

$$\mathcal{S}_{\notin \mathbf{y}} = \{\mathbf{s} \in \mathcal{S} \,|\, \mathcal{A}_{\mathbf{s}} \cap \mathcal{A}_{\mathbf{y}} = \emptyset\} \tag{3.13}$$

The total number of such sequences in $\mathcal{S}_{\notin \mathbf{y}}$ is

$$|\mathcal{S}_{\notin \mathbf{y}}| = (m-n)(m-n-1)\cdots(m-2n+1). \tag{3.14}$$

Let I denote an instance in the randomized data structure, and $\mathcal{U}_I$ denote a subset of items such that the data structure with the instance I will return true for each item $x \in \mathcal{U}_I$. Let $\mathcal{I}$ denote the set of all instances in a randomized data structure. And the space requirement for any data structure is bounded by $\log(|\mathcal{I}|)$ bits, because each instance requires an unique binary representation.

Because there is no false negative, there must be at least one instance I for each sequence $\mathbf{s} \in \mathcal{S}$ such that

$$\mathcal{A}_{\mathbf{s}} \subset \mathcal{U}_I. \tag{3.15}$$

Furthermore, because the data structure only allows a false positive error δ, there must be at least one instance with $\mathcal{A}_{\mathbf{s}} \subset \mathcal{U}_I$ such that

$$|\mathcal{U}_I| \leq \delta(m-n)+n. \tag{3.16}$$

For each sequence $\mathbf{s}$, the data structure must find an instance I with the above two properties, i.e. Eq.(3.15) and Eq.(3.16). If two sequences $\mathbf{s}$ and $\mathbf{s}'$ share the same instance I, we must have

$$\mathcal{A}_{\mathbf{s}}, \mathcal{A}_{\mathbf{s}'} \subset \mathcal{U}_I \text{ and } |\mathcal{U}_I| \leq \delta(m-n)+n. \tag{3.17}$$

For the static set, we can use these two properties to get an entropy lower bound for the membership query [22], i.e. $n\log(1/\delta)$ bits.

If two sequences share the same instance I, we can extend them to the length $n+\delta(m-n)$ in order to include all items in $\mathcal{U}_I$. The extended sequence $\hat{\mathbf{s}}'$ can be viewed as a random permutation of the extended sequence $\hat{\mathbf{s}}$, as shown in Fig.3.4. Each arrow from x_p to x'_q represent that the item at the position p in the sequence $\hat{\mathbf{s}}$ moves to the position q in the sequence $\hat{\mathbf{s}}'$,

$$x_p = x'_q. \tag{3.18}$$

3.1.4.2 Property

We can prove the following property for two sequences sharing the same instance in the randomized data structure.

Lemma 1. *Given a sequence $\mathbf{y} \in \mathcal{S}$, let $\mathbf{s}$ and $\mathbf{s}'$ be two randomly selected sequences from $\mathcal{S}_{\notin \mathbf{y}}$ that share the same instance I in the randomized data structure and $\mathbf{s} \neq \mathbf{s}'$. For a false positive error bound δ, we have the following property,*

$$E(\|\mathbf{s}-\mathbf{s}'\|) < \delta(m+n) \tag{3.19}$$

where $E(\|\mathbf{s}-\mathbf{s}'\|)$ denotes the average Hamming distance between $\mathbf{s}$ and $\mathbf{s}'$.

Proof. Let $\mathbf{y} \in \mathcal{S}$ be a sequence of n distinct items in the current time-decaying window of a size n over a data stream. We consider all possible sequences $\mathbf{s} \in \mathcal{S}_{\notin \mathbf{y}}$ that may appear before

$\mathbf{y}$ in the data stream,

$$\underbrace{x_1, \ldots, x_p, \ldots, x_n}_{\mathbf{s}}, \underbrace{y_1, \ldots, y_g, \ldots, y_n}_{\mathbf{y}}, \ldots \tag{3.20}$$

where $p, g = 1, \ldots, n$.

Let $\mathbf{s}$ and $\mathbf{s}'$ denote two different sequences in $\mathcal{S}_{\notin \mathbf{y}}$, which appear before $\mathbf{y}$ in two data streams, respectively. As the item in $\mathbf{y}$ comes, we can get two sequence chains in the time-decaying window. At the beginning, we have $\mathbf{s}_0 = \mathbf{s}$ and $\mathbf{s}'_0 = \mathbf{s}'$. When the first item y_1 in $\mathbf{y}$ comes, x_1 gets expired. In the time-decaying window, we have

$$\mathbf{s}_1 = < x_2, \ldots, x_n, y_1 >, \tag{3.21}$$

$$\mathbf{s}'_1 = < x'_2, \ldots, x'_n, y_1 > . \tag{3.22}$$

When the g^{th} item y_g comes, $x_1, \ldots, x_g$ get expired. In the time-decaying window, we have

$$\mathbf{s}_g = < x_{g+1}, \ldots, x_n, y_1, \ldots, y_g >, \tag{3.23}$$

$$\mathbf{s}'_g = < x'_{g+1}, \ldots, x'_n, y_1, \ldots, y_g > . \tag{3.24}$$

At last, we get the same sequence in the time-decaying window between two data streams, i.e. $\mathbf{s}_n = \mathbf{s}'_n = \mathbf{y}$.

If two sequences $\mathbf{s}$ and $\mathbf{s}'$ share the same instance I in the randomized data structure, all the following sequences in two sequence chains will share the same instance chain, denoted by I_g. This is because the data structure starts with the same instance $I_0 = I$, and updates its state based on the same coming item y_g at each time step between two data streams.

An adversary is always willing to make queries in order to increase the false positive error as much as possible. We consider an adversary who can also observe the data stream. Based on his/her observation, the adversary always make a query of the item that just gets expired at each time step. For example, when y_g comes, the adversary will make a query for whether x_g (or x'_g) is present in the time-decaying window. We consider all possible queries that an adversary can make over a data stream started with $\mathbf{s}$ (or $\mathbf{s}'$). There will be n possible queries in a data stream.

Because $\hat{\mathbf{s}}'$ can be viewed as a permutation of $\hat{\mathbf{s}}$, we consider all possible moves for an item x_p in the permutation from $\hat{\mathbf{s}}$ to $\hat{\mathbf{s}}'$, where $p = 1, \ldots, n$. There are four possible moves of the same item x_p in the permutation, as shown in Fig.3.5.

- $p > q$: When x'_q gets expired, the data structure must return true if an adversary makes a query of x_p, because x_p is still in the time-decaying window of the first data stream. However, the data structure must also return true if an adversary makes a query of x'_q, because the data structure has the same instance chain between two data streams. Therefore, there must be a false positive.

- $p = q$: The data structure can return false in both data streams, when the adversary makes a query of x_p or x'_q after it gets expired. This is because they get expired when the same item y_p comes. Therefore, there will be no false positive if the data structure updates its instance properly.

- $p < q \leq n$: When x_p gets expired, the data structure must return true if an adversary makes a query of x_p, because x_p is still in the time-decaying window of the second data stream. Therefore, there must also be a false positive as the first case.

- $q > n$: In this case, x_p does not appear in the second data stream. If the adversary makes a query of x_p, the data structure can return false when x_p gets expired. There will be no false positive if the data structure updates its instance properly.

If x_p moves to some position $1 \leq q \leq n$ rather than p from $\mathbf{s}$ to $\mathbf{s}'$, there must be a false positive, as shown in Fig.3.6. The number of items that move to a different position equals to the Hamming distance between $\mathbf{s}$ and $\mathbf{s}'$. If $\mathbf{s}$ and $\mathbf{s}'$ share the same instance I, there will be at most $n + \delta(m-n)$ items in $\mathcal{U}_I$. And, there are at most $\delta(m-n)$ items in $\mathbf{s}$ which can move to a position $q > n$ and do not cause any false positives.

Let $e(\mathbf{s}, \mathbf{s}')$ denote the number of false positives when an adversary makes queries for each item in a pair of sequences sharing the same instance. We can know that the number of false positives can be bounded by

$$e(\mathbf{s}, \mathbf{s}') \geq \|\mathbf{s} - \mathbf{s}'\| - \delta(m-n). \tag{3.25}$$

If we consider all pairs of sequences in $\mathcal{S}_{\notin \mathbf{y}}$ that share the same instance in the randomized data structure, the average number of false positives can be bounded by

$$E(e(\mathbf{s},\mathbf{s}')) \geq E(\|\mathbf{s}-\mathbf{s}'\|) - \delta(m-n). \tag{3.26}$$

Given two sequences, there are $2n$ queries if the adversary makes queries for each item in a pair of sequences. If the false positive error can be bounded by δ, we must have

$$E(e(\mathbf{s},\mathbf{s}')) < 2\delta n. \tag{3.27}$$

Thus, we have

$$E(\|\mathbf{s}-\mathbf{s}'\|) < \delta(m-n) + 2\delta n = \delta(m+n). \tag{3.28}$$

□

3.1.4.3 Our Proof for Lower Bound

If $n < \delta(m+n)$, we have the same restriction as the entropy lower bound. In this section, we provide a bound for the average number of sequences sharing the same instance in the randomized data structure when n is sufficiently large,

$$m \gg n > \delta(m+n). \tag{3.29}$$

Lemma 2. *Given a sequence $\mathbf{y} \in \mathcal{S}$, let $\mathcal{S}_I^{\mathbf{y}} \subset \mathcal{S}_{\notin \mathbf{y}}$ denote the set of the sequences that share the same instance I. Given an error bound δ, we have*

$$|\mathcal{S}_I^{\mathbf{y}}| \quad < \quad \frac{2^{\delta m} n^n}{(n-\delta m)^{n-\delta m}}, \tag{3.30}$$

when $m \gg n > \delta(m+n)$.

Proof. Given a sequence $\mathbf{s} \in \mathcal{S}_I^{\mathbf{y}}$, let $\mathcal{S}_{I,d}^{<\mathbf{s},\mathbf{y}>} \subset \mathcal{S}_I^{\mathbf{y}}$ denote the set of sequences $\mathbf{s}'$ with a Hamming distance to $\mathbf{s}$ not larger than d,

$$\mathcal{S}_{I,d}^{<\mathbf{s},\mathbf{y}>} = \{\mathbf{s}' \in \mathcal{S}_I^{\mathbf{y}} \mid \|\mathbf{s}-\mathbf{s}'\| \leq d\}. \tag{3.31}$$

For each sequence $\mathbf{s}' \in \mathcal{S}_{I,d}^{<\mathbf{s},\mathbf{y}>}$, there are at most d positions in $\mathbf{s}$ at which the item is replaced by an item from $\mathcal{U}_I \setminus \mathcal{A}_{\mathbf{s}}$ or moves to a different position in $\mathbf{s}'$. For such position p that the corresponding items are different between two sequences, there are at most $d+\delta(m-n)-1$ items which can appear at this position in $\mathbf{s}'$. Therefore, the number of sequences in $\mathcal{S}^{\mathbf{y}}_{\mathbf{s},d}$ can be bounded by

$$|\mathcal{S}^{<\mathbf{s},\mathbf{y}>}_{I,d}| \leq \frac{n!}{d!(n-d)!}(d+\delta(m-n)-1)^d. \tag{3.32}$$

Based on lemma 1, we know that

$$E(\|\mathbf{s}-\mathbf{s}'\|) < \delta(m+n) \tag{3.33}$$

for two randomly selected sequences from $\mathcal{S}^{\mathbf{y}}_{I}$. Because

$$E(E(\|\mathbf{s}-\mathbf{s}'\| \,|\, \mathbf{s})) = E(\|\mathbf{s}-\mathbf{s}'\|), \tag{3.34}$$

the data structure must bound the average distance of a sequence $\mathbf{s}' \in \mathcal{S}^{\mathbf{y}}_{I}$ to $\mathbf{s}$ by $\delta(m+n)$ for any given sequence $\mathbf{s} \in \mathcal{S}^{\mathbf{y}}_{I}$.

If $\mathbf{S}^{\mathbf{y}}_{I}$ can include more sequences with a small distance to $\mathbf{s}$, its size will become larger, and thus the number of instances in the data structure can become smaller. We assume that there are enough sequences $\tilde{\mathbf{s}} \in \tilde{S}_{\mathbf{s}} \subset \mathcal{S}_{\notin \mathbf{y}}$ such that

$$\|\mathbf{s}-\tilde{\mathbf{s}}\| = \delta(m+n)+1. \tag{3.35}$$

Because there will be fewer sequences if $\mathbf{S}^{\mathbf{y}}_{I}$ includes any sequences with a distance to $\mathbf{s}$ larger than $\delta(m+n)+1$, we assume that $\mathbf{S}^{\mathbf{y}}_{I}$ only includes the sequences with a distance to $\mathbf{s}$ not larger than $\delta(m+n)+1$.

If there is a sequence $\mathbf{s}'$ with $\|\mathbf{s}-\mathbf{s}'\| = d < \delta(m+n)$, we can add at most $\delta(m+n)-d$ sequences from $\tilde{\mathcal{S}}_{\mathbf{s}}$ into $\mathcal{S}^{\mathbf{y}}_{I}$, and still bound the average distance to $\mathbf{s}$ smaller than $\delta(m+n)$. Therefore, the size of the set $\mathbf{S}^{\mathbf{y}}_{I}$ can be bounded by

$$\begin{aligned} |\mathcal{S}^{\mathbf{y}}_{I}| &< \sum_{d=1}^{\delta(m+n)} |\mathcal{S}^{<\mathbf{s},\mathbf{y}>}_{I,d}| \\ &< \sum_{d=1}^{\delta(m+n)} \frac{n!}{d!(n-d)!}(d+\delta(m-n)-1)^d \\ &< \frac{\delta(m+n)(n!)}{(\delta(m+n))!(n-\delta(m+n))!}(2\delta m-1)^{\delta(m+n)} \end{aligned} \tag{3.36}$$

Based on Stirling's approximation [1],

$$n! \simeq \sqrt{2\pi n}\left(\frac{n}{2}\right)^{n}, \tag{3.37}$$

and $m \gg n$, we have

$$|\mathcal{S}_I^{\mathbf{y}}| < \frac{\delta m 2^{\delta m} n^{n}}{(n-\delta m)^{n-\delta m}}. \tag{3.38}$$

$\square$

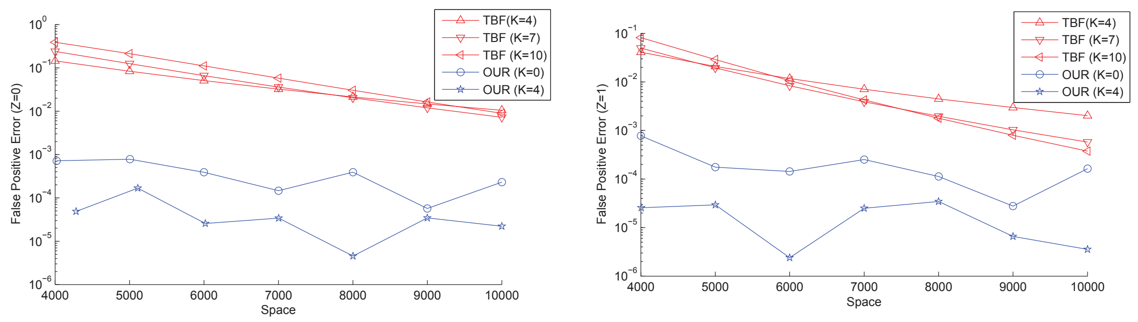


Figure 3.7 False Positives in Zipfian Distributions

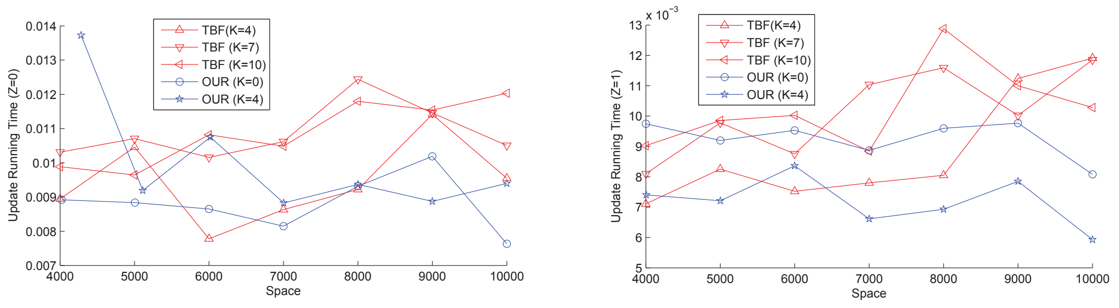


Figure 3.8 Update Time in Zipfian Distributions

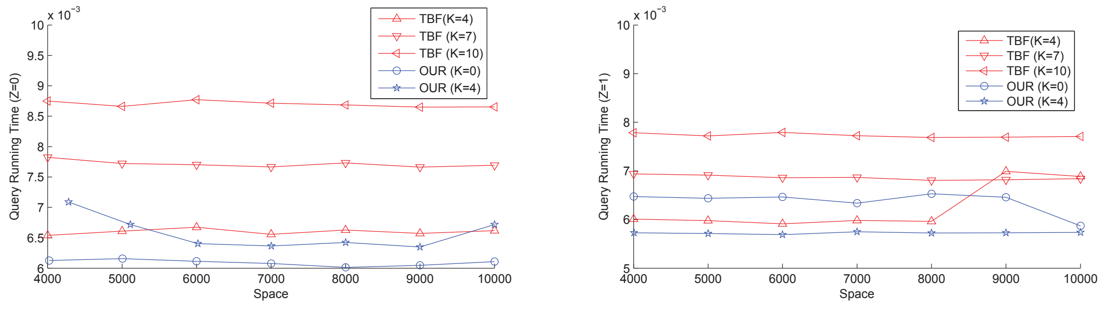


Figure 3.9 Query Time in Zipfian Distributions

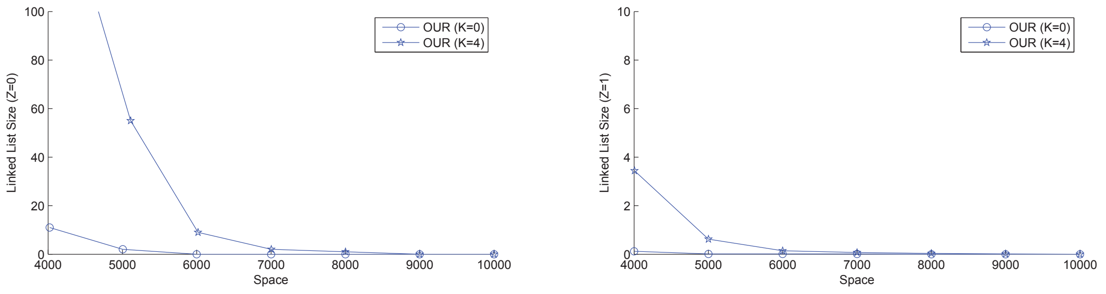


Figure 3.10 Maximum Linked List Sizes in Zipfian Distributions

3.1.5 Evaluation

We evaluate our data structure using both synthetic and real data sets in this section. All experiments run on a Linux (Fedora 13) machine with 3.2 GHz Pentium dual-core processor and 2GB RAM. Both algorithms are implemented in a single-threaded version. We compare the performance of our data structure with the TBF which is the only existing data structure for the time-decaying window model. Both data structures are implemented with C++. Each data structure is provided with the same number of counters and a small additional space for the hash functions and the linked list. Let ℓ denote the total number of counters in the TBF. In our data structure, we use two different configurations based on the error bound δ and the

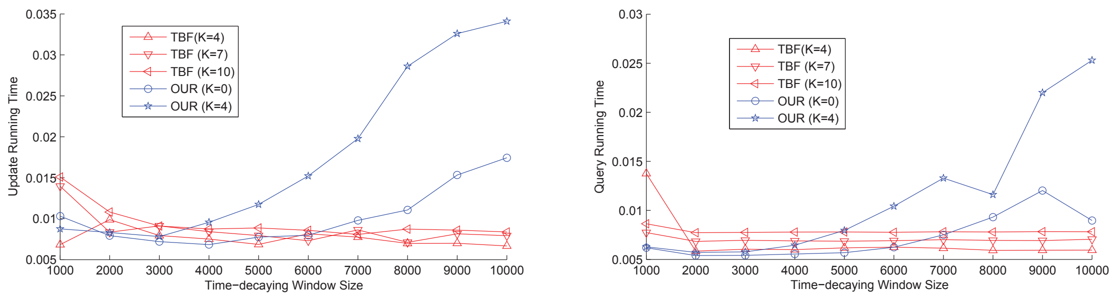


Figure 3.11 Running Time with Different Window Sizes

maximum window size n. If δ is large, i.e. $\delta > 1/n$, we do not need to maintain the signatures, and can bound the errors only based on the positions. In this configuration, we set the size of the hash table as $\eta = \ell/4$. Otherwise, we also maintain a signature at each position in the hash table. And we set the size of the hash table as $\eta = \ell/6$. Let K denote the size of the signature in our data structure, and the number of hash functions in TBF. In this way, we guarantee a fair comparison between the TBF and our data structure.

3.1.5.1 Synthetic Data

In this part, we compare two data structures with synthetic data sets produced by a Zipfian distribution, which is the state-of-the-art model of the HTTP requests [68]. We used integers as data items, and the size of the universe is chosen as 100,000. In each evaluation, we compare their false positive errors and running time for update and query.

We choose two different parameters in the Zipfian distribution: $z = 0$ (uniform data) and $z = 1$ (skewed data) to evaluate the performance of our data structure. The window size is chosen as 1000. Therefore, each hash table in our data structure should be at least 2000. We evaluate both data structure with $\ell = 4000, ..., 10000$. The false positive errors are shown in Fig.3.7. The y-axis is in logarithmic scale, and the x-axis is the total space in the TBF or our data structure. We can see that the errors in our data structure are much smaller than those in the TBF. The average running times for update and query are shown in Fig.3.8 and Fig.3.9,

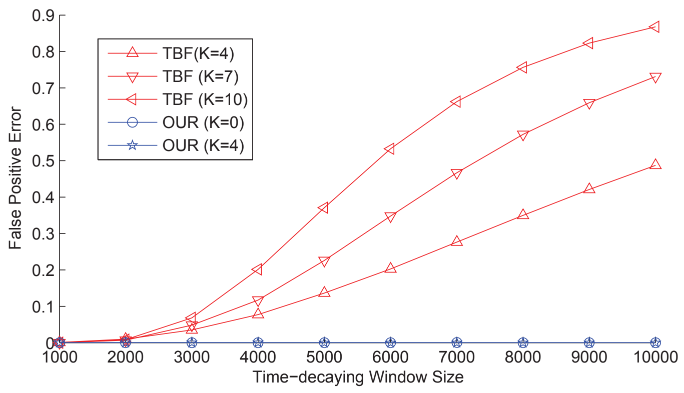


Figure 3.12 False Positive Error with Different Window Sizes

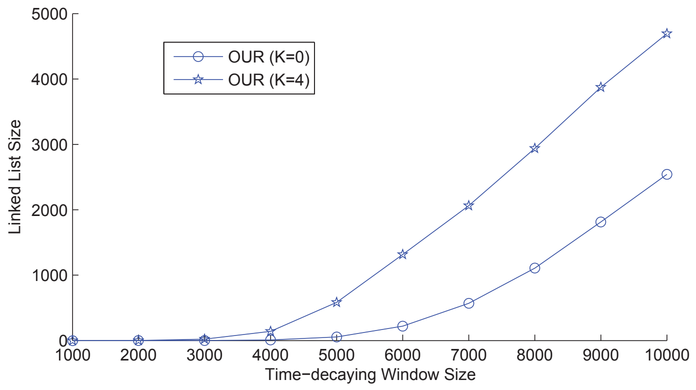
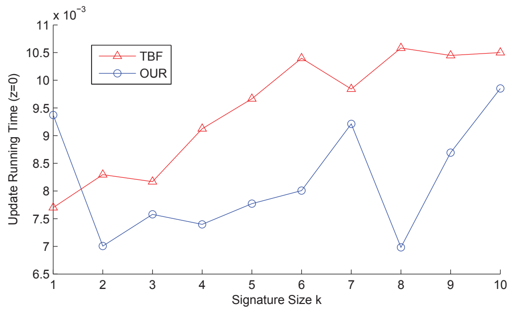


Figure 3.13 Linked List Size with Different Window Sizes



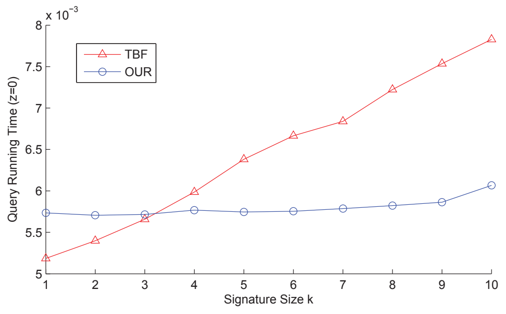


Figure 3.14 Running Time with Different Signature Sizes

respectively, which are measured in seconds. When the size of the hash tables is small, our data structure maintains many items in the linked list, as shown in Fig.3.10. In this case, the running time will become large, but the false positive error can still be bounded by δ. There are not significant differences between the performances with uniform or skewed data sets. In the following parts, we choose $z = 0$ and there will be more distinct items in each time-decaying window than that with $z > 0$.

We evaluate our data structure with different sizes of the time-decaying window. We change the time-decaying window size n from 1000 to 10000. The size of the hash tables in our data structure is chosen as $\eta = 5000$ if $K = 0$, and $\eta = 3333$ if $K = 4$. The size of the TBF is chosen as $\ell = 20000$ for a fair comparison. The false positive error in the TBF and our data structure is shown in Fig.3.12 (left). When the time-decaying window size increases, our data structure will maintain more items in the linked list whose size will increase as shown in Fig.3.13. The running time of the TBF will remain the same, and the running time in our data structure will increase as the size of the linked list increases, as shown in Fig.3.11.

We evaluate our data structure with different sizes of the signatures K, which changes from 1 to 10. The number of hash functions in TBF is also chosen as K. The size of the TBF is chosen as $\ell = 10000$ and the size of the hash tables in our data structure is chosen as $\eta = 1667$. The maximum size of the sliding window is chosen as $n = 1000$. The false positive errors are shown in Fig.3.15, and the running times are shown in Fig.3.14. When K increases, the false

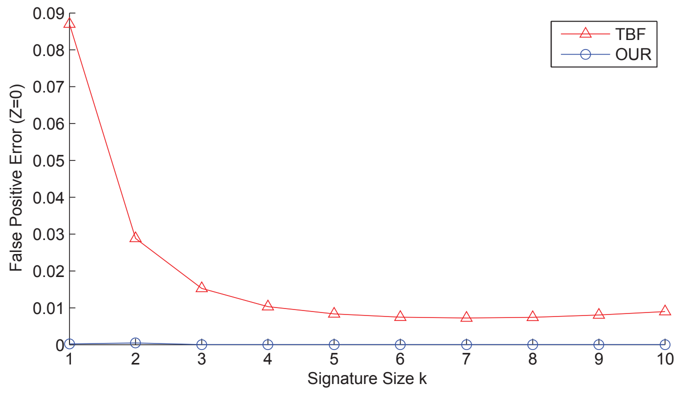


Figure 3.15 False Positives with Different Signature Sizes

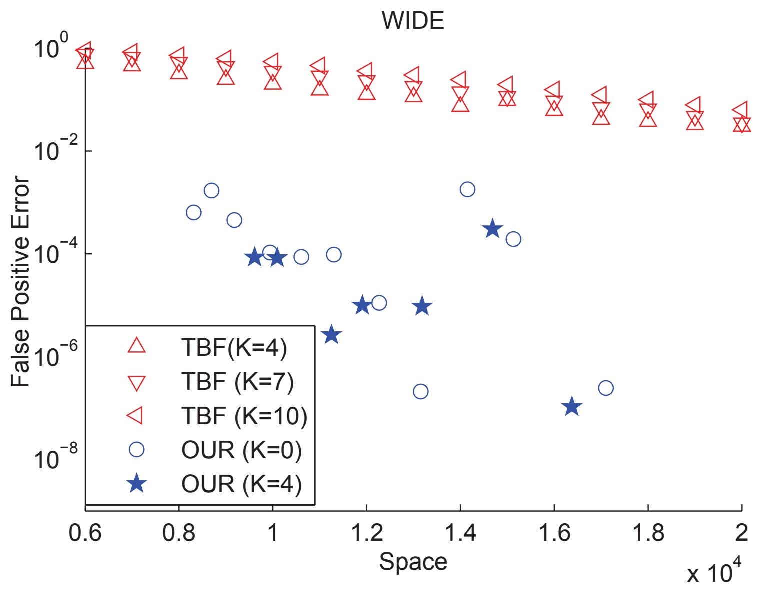


Figure 3.16 False Positives with WIDE Trace

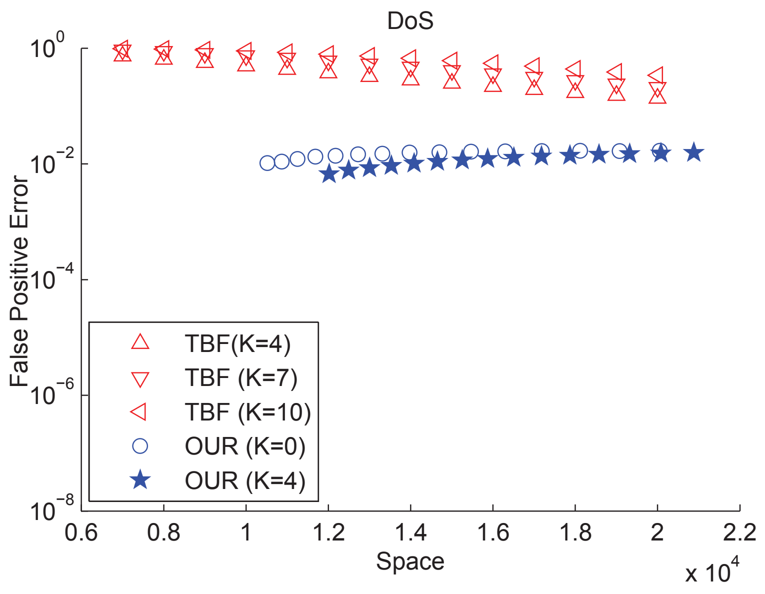


Figure 3.17 False Positives with DoS Attack

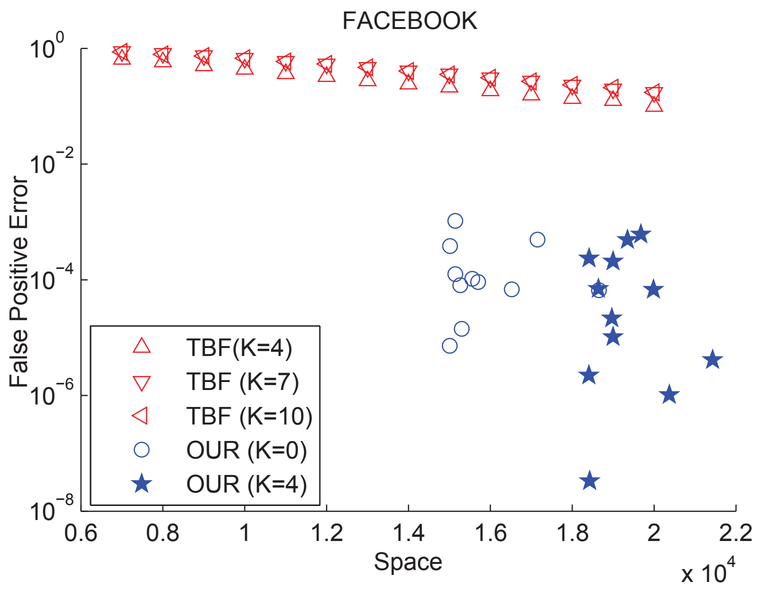


Figure 3.18 False Positives with Facebook Wall Post

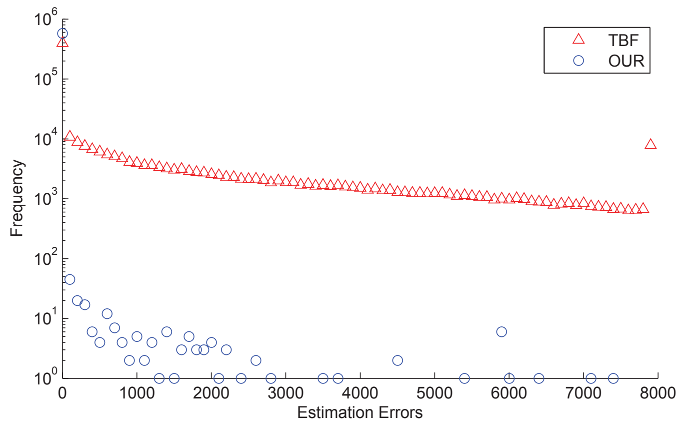


Figure 3.19 Estimation Error Distribution

positive error in our data structure will decrease very quickly. But the false positive error in the TBF will increase if K is too large. And the running time of the query is mainly determined by the number of memory accesses. The query running time in the TBF will increase but it will remain the same in our data structure as K increases.

3.1.5.2 Real Data

We use three different data sets to evaluate our data structure in different applications. The first one is a packet trace from a backbone link of a national ISP, which is used for the caching and replication application. The second one contains the HTTP request logs from a busy WWW server. We evaluate our data structure for the load balancing and DoS attack defending using this data set. The last one is a log of the user activities on a popular social network (FACEBOOK) [92], which can capture some new trends in the Web applications. We mainly focus on the false positive errors. The running time follows a similar pattern as the synthetic data set, which is skipped due to the page limitation.

The WIDE trace was collected on Mar 18, 2008, which consisted of packets on a 150 Megabit Ethernet external link between WIDE backbone and its upstream. The first 96 bytes were captured per each packet, which can be used to extract all HTTP packets and the URLs after the GET command in HTTP packets. When each URL comes, we first query whether it is maintained in the proxy by using the TBF or our data structure. If so, we consider that this web page is maintained at this proxy and skip this URL. Otherwise, we need to fetch the web page from the Internet and insert the URL into our data structure for a new available web page. A web page will become expired after $n = 10000$ packets. The false positive errors in both data structure are shown in Fig.3.16. We can see that the errors in our data structures are much smaller than the TBF.

In this data set, each item is an HTTP request with a host name, a timestamp, a URL, a HTTP reply code, and bytes in the reply. Based on the URL, the requests can be divided into seven categories: HTML, Images, Sound, Video, Dynamic, Formatted, and Other. We consider the HTML requests as a data stream in our evaluation. A detailed analysis of this data set can be found in [6]. As a typical application of the membership query, we simulate a DoS attack on this server by replaying duplicated requests randomly to the server in the data stream. The TBF and our data structure are implemented to filter these duplicated requests from the normal requests. In our evaluation, when a request x_i comes, we first query whether x_i is a duplicated request, and then update the data structures for it. If a request is a DoS request, the data structure should always determine it as a duplicated request and return true. Otherwise, it should return false with a probability at least $1 - \delta$. We use the same configurations as the WIDE trace. The false positive errors are shown in Fig.3.17. The false positive errors are a litter higher than the WIDE trace, which is because there are more distinct requests in this trace.

We also use a trace of all of the wall posts from the Facebook New Orleans networks from [92]. Each item contains two anonymized user IDs, which means the second user posted on the first user's wall. And a UNIX timestamp of the wall post is also included. When each item comes, we insert the first user ID into the TBF or our data structure, and query whether the second user has some new posts from his friend in the time-decaying window. The false positive errors with the same configuration as the Facebook trace are also shown in Fig.3.18. We also estimate the last updating time for the second user, i.e., the smallest w such that this item is still active. The distribution of the estimation error in the TBF ($\ell = 20000$ and $k = 4$) and our data structure ($\eta = 5000$ and $k = 0$) is shown in Fig.3.19. We can see that the estimation error in our data structure is much smaller.

3.1.6 Conclusion

We provide a novel data structure for the approximate membership query over the time-decaying window model. Our data structure has been proved to be near-optimal in both space and running time for a wide range of parameters. The space can be bounded by $O(n(\log 1/\delta + \log n))$ and the running time is bounded by $O(1)$ for both update and query. We hope that our work can be used to improve the practice for various real-world applications, i.e., user behavior monitoring, web crawling, network security, etc.

3.2 Tracking Long Duration Flows for Botnet C&C Communications

Recently, flow duration has been proposed as a new traffic feature for network monitoring and measurement. Most web browsing flows are short-lived. There are more and more applications communicating each other with long-lived flows over the Internet, e.g., streaming video, online chatting, multiplayer games, social networks, etc. The information about flow duration in the network becomes very useful for the purpose of traffic classification and network management. For example, we can easily determine what type of a flow is related based on its duration. Additionally, the flow duration feature can also be used by the intrusion detection system. One interesting observation made from our recent experimental study as well as several other work on P2P and IRC-based botnets is that bots often maintain persistent connections among themselves (in a peer-to-peer structure), and between these bots and their Command and Control servers (C&Cs) (in a IRC-like coordinated structure). In this section, we study the problem of how to efficiently track long duration flows (LDFs) in high-speed networks with small space and bounded errors. A flow is considered to be active if the inter-arrival time between any two consecutive packets is shorter than some timeout interval. LDFs are the ones that can keep active for a relatively long period of time, i.e., an hour or even longer, but may or may not have high traffic volumes. Existing algorithms for the LDF detection may detect short-lived flows as LDFs, i.e. false positives, which are not space-efficient. We propose two data streaming algorithms based on Cukoo-hashing Dictionary for tacking LDFs with only false negatives but no false positives. There is no short-lived flows that can be falsely detected in our algorithms. One of our algorithms is a deterministic algorithm which can provide a strong bound for the flow duration estimation. The other is a randomized algorithm which provides a best-effort solution for the flow duration estimation when the space is less than the lower bound of the deterministic algorithm. Their performance is evaluated using real network traces. We expect that the results from our work will improve the practice and ways of network monitoring and measurement.

3.2.1 Introduction

Previous studies for the traffic measurement and monitoring mainly focus on measuring traffic volumes and flow numbers. Network administrators usually detect device failures and network attacks based on the current traffic volume. If there is an sudden change in the traffic volume, we can know that there must be some traffic anomalies in the network. Also, the number of active flows is important for the network management. Many denial-of-service attacks can generate a large number of flows to the target server, which may only contribute a small amount of traffic to the network. Specifically, Internet Service Providers (ISPs) are interested in detecting any heavy hitters or super spreaders in their network. Heavy hitters are the hosts with high traffic volumes, and super spreaders are the ones with a high cardinality, i.e., a large number of flows. Such hosts are often corresponding to traffic anomalies and security problems. For quite a long time, the flow duration has not attracted many research interests or been used for the network management.

Recently, long duration flows (LDF) under different applications have been observed in the Internet [25]. Online streaming video often creates a large number of long-lived flows between service providers and home users, which often have high traffic volumes. Social network users can maintain a flow for more than an hour to chat with their friends, which only involve little traffic. The flow duration information can help ISPs and service providers to detect network problems and improve their user experiences. For example, by monitoring the flow durations from the streaming video applications, we can estimate the service qualities for each user. If most flows are short-lived, we can know that the user must have large lags during their video watching. The service provider can detect such conditions and response to any problems in their servers or the network links.

Especially, many LDFs in the Internet are closely related to the botnet activities based on our experimental study. As shown in Fig.3.20, each bot usually maintains a connection to the Command and Control (C&C) center or multiple connections with other bots in order to receive commands and updates from its controller, (i.e. the botmaster). Traffic from the botnet Command and Control channels differentiate themselves significantly from normal traffic and

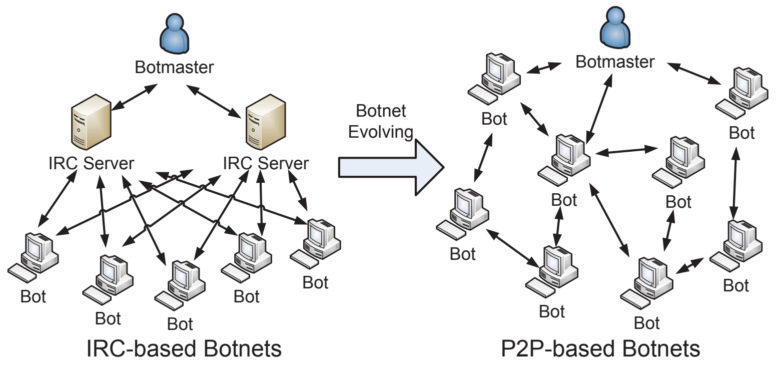


Figure 3.20 Botnet Communications

those generated from traditional attacks such as DDoS attacks and worm spreading. Firstly, bots only send/recieve a small amount of malicious traffic on the C&C channels in order to avoid to be detected. Secondly, bots tend to maintain persistent connections, i.e., long-duration flows, to keep the connectivity of the botnets. By monitoring the flow durations, we can identify the botnet C&C connections, and thereby discover the membership, structure, and communication patterns of the botnets.

In general, the hosts with LDFs do not necessarily have a large amount of traffic or a large number of flows. The flow duration should be considered as a new traffic feature for network management and monitoring, which requires different data streaming algorithms to process high-speed packets from previous ones used to detect heavy hitters or super spreaders. In this section, we present two data streaming algorithms for tracking LDFs in high-speed networks, which can guarantee that no short-lived flow can be falsely detected as LDFs. In this way, we only focus on true LDFs in the network, which can reduce the space requirement for monitoring LDFs significantly. Our algorithms are more space-efficient than existing algorithms which have false positives. Furthermore, our algorithms can provide a strong error bound for the flow duration estimation. We experimentally evaluate our algorithms with real network traces, and the experimental results have shown their effectiveness.

Our main contributions are summarized as follows:

- We propose two data streaming algorithms for tracking LDFs in high-speed networks, i.e. deterministic or randomized, which can detect LDFs with only few false negatives but no false positive.

- Our deterministic algorithm can provide a strong error bound for the flow duration estimation, which is also space-optimal.
- Our randomized algorithm provides an alternative space-efficient solution when the available space is much less than the lower bound of the deterministic algorithm.
- We evaluate two algorithms with a traffic trace including botnet communication flows, and show that our algorithms can detect LDFs with constant running time per packet.

3.2.2 Problem Definition

In this section, we assume that the flow ID is known before the traffic monitoring. For example, (SrcIP, SrcPort, DstIP, DstPort, Protocol) is one of the most common flow ID. A packet is denoted by its flow ID and its arrival time, i.e., (f, t), where f is the flow ID and t is the arrival time.

Let Δ denote the length of the timeout interval, which is predefined by the Internet applications. A flow is considered to be active if it has at least one packet in each timeout interval since its first packet, as shown in Fig.3.21. The inter-arrival time between two consecutive packets in an active flow should be less than Δ.

However, it is not practical to maintain the exact arrival time of each packet in a flow due to overly large space requirement. Alternatively, we can determine that a flow is terminated in the γ-th interval if there is no packet from this flow in this interval. For example, the flow f in Fig.3.21 can be determined as terminated in the 7th interval. In this way, we can use the index of the arrival interval rather than the exact arrival time for each packet to calculate the flow duration.

The duration of a flow is measured by the number of continuous timeout intervals in which the flow is still active since its start. For example, the duration of the flow f in Fig.3.21 is 6. A LDF can be defined as following.

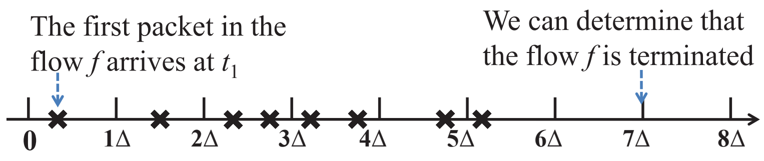

The first packet in the flow f arrives at t_1
We can determine that the flow f is terminated
0
1Δ
2Δ
3Δ
4Δ
5Δ
6Δ
7Δ
8Δ

Figure 3.21 Flow Duration Definition Example

Definition 2. *A flow f is considered as a LDF, if*

$$d_f \geq d, \tag{3.39}$$

where d_f denotes the duration of the flow f, and d is a given threshold.

The decision on the parameter d can be critical. For some small d, there may be too many flows that are considered as LDFs, which in turn requires overly large memory space for tracking LDFs in the network traffic. Given some specified and limited space, we are interested in identifying those flows with the longest duration. So the space available for tracking LDFs can be translated into the maximum number of LDFs that can be maintained, which is denoted by k. The larger the value of k, the more space it would require. So the problem of the LDF detection can be formulated into a problem that for a given space requirement, how to detect the k longest duration flows.

In the ideal situations, all the k longest duration flows are logged. However, it would be hard to bound the space requirement. Given limited memory space, we can only get an approximation for the k longest duration flows. In other words, we can only identify a subset of the k longest duration flows whose durations are much longer than the other flows. Let $\mathcal{F}$ denote the set of all flows presented in the stream. Give the threshold $\theta = 1/(k+1)$, the top-k LDFs are the flows with durations longer than θD,

$$d_f > \theta D, \tag{3.40}$$

where D is the sum of the durations of all flows,

$$D = \sum_{f \in \mathcal{F}} d_f. \tag{3.41}$$

The threshold d is given based on the sum of the durations of all flows, $d = \theta D$. It is easy to show that there are at most k flows that are among the top-k LDFs. If we want to get the exact top-k LDFs, we need to maintain all the flows in the memory which is infeasible in current high-speed networks. So we are usually interested in efficient algorithms that are capable of performing real-time traffic monitoring but can only provide an approximation result. We consider two types of algorithms in this section, i.e., deterministic and randomized.

3.2.2.1 Problem Formulation for Deterministic Algorithm

The deterministic algorithm can provide the strongest error bound for the LDF detection. We sort all flows in a decreasing order of their durations,

$$d_{f_1} \geq d_{f_2} \geq \ldots \geq d_{f_n}, \tag{3.42}$$

where $n = |\mathcal{F}|$ is the number of the flows. Let $D^{res(\ell)}$ denote the sum of the durations of the flows after the ℓ-th longest flow, i.e., the ℓ-tail duration,

$$D^{res(\ell)} = \sum_{i=\ell+1}^{n} d_{f_i}. \tag{3.43}$$

The deterministic algorithm can provide the following guarantee for the LDF detection, i.e. the ℓ-tail bound,

Definition 3. *Give the threshold $d = \theta D$, a deterministic algorithm with the ℓ-tail bound can guarantee that any flow $f \in \mathcal{F}$ whose*

$$d_f > d + \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)} \tag{3.44}$$

is reported as a LDF, and any flow $f' \in \mathcal{F}$ whose

$$d_{f'} < d \tag{3.45}$$

is not reported as a LDF.

If the duration of a flow is large enough, i.e., $d_f > d + \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)}$, the deterministic can always detect this flow as LDF without any errors. If the duration of a flow is between d and $d + \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)}$, there is no guarantee in the deterministic algorithm. The false negative errors are determined by the number of LDFs with a duration less than $d + \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)}$.

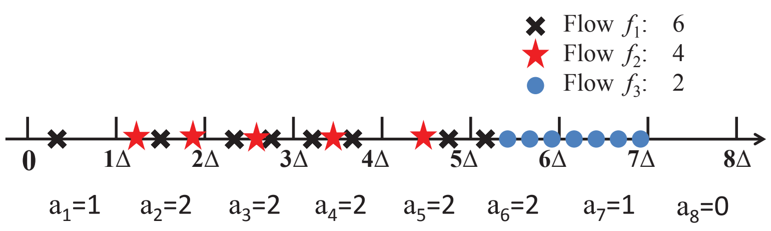


Figure 3.22 Example of Number of Active Flows

The error in the deterministic algorithm is mainly determined by the sum of the durations D. Let a_γ denote the number of the active flows in the γ-th timeout interval, i.e., all active flow with a packet (f, t) such that

$$(\gamma - 1)\Delta \leq t < \gamma\Delta, \tag{3.46}$$

where t is the arrival time of the packet (f, t). We have the following relation between the flow duration and the number of active flows.

Lemma 3. *The sum of the durations equals to the sum of the number of active flows in each interval,*

$$D = \sum_{\gamma=1}^{\Gamma} a_\gamma, \tag{3.47}$$

where Γ *denotes the number of the timeout intervals in the traffic stream.*

As shown in Fig.3.22, there are three flows in the traffic stream. The sum of their durations is

$$D = d_{f_1} + d_{f_2} + d_{f_3} = 6 + 4 + 2 = 12. \tag{3.48}$$

There are eight intervals in the traffic stream, and the sum of the number of active flows also equals to 12.

Let $\bar{a}$ denote the mean of the number of active flows in a timeout interval. If we want to detect a flow with the duration longer than a given threshold d, we must have

$$D > \bar{a}d. \tag{3.49}$$

This is due to that the length of the traffic stream must be larger than d. In order to detect the LDFs with the error bound of the deterministic algorithm, we must ensure that the error is less than the threshold,

$$d > \varepsilon D. \tag{3.50}$$

Thus, we have $1/\varepsilon > \bar{a}$. Thus, we get the following result.

Lemma 4. *In order to detect the LDFs with the durations longer than a given threshold d, the space in the deterministic algorithm should be larger than the mean of the number of the active flows.*

However, there may not be enough space if there are too many active flows in high-speed networks. There is a space lower bound for the flow duration estimation in the deterministic algorithm. When the space is less than the lower bound, there is no guarantee for the LDF detection, and we may need to find an alternative bound for the algorithms with a small space.

3.2.2.2 Problem Formulation for Randomized Algorithm

The randomized algorithms provide a weak bound for the flow duration estimation. It can detect LDFs with a high probability, but still may miss some LDFs even if their durations are very long. Our randomized algorithm can provide the following guarantee for the LDF detection,

Definition 4. *Give the threshold $d = \theta D$, the probability that a randomized algorithm can report a flow $f \in \mathcal{F}$ as a LDF approaches to 1 if its duration approaches to infinite.*

$$\lim_{d_f \to \infty} P(f \textit{ is detected}) = 1. \tag{3.51}$$

The above definition only guarantees that the randomized algorithm prefers LDFs to short-lived flows. The randomized algorithm does not provide any deterministic bound for the flow duration estimation. It only provides a best-effort solution for the LDF detection, when the available space is much less than the lower bound of the deterministic algorithm. If the duration of a flow is longer, the randomized algorithm can detect it with a higher probability. When there are too many flows with a duration between d and $d + \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)}$, we prefer to use the randomized algorithm than the deterministic algorithm.

```
Initialize $\mathcal{C} \leftarrow \emptyset$, $\tau_s \leftarrow 0$, $\tilde{c} \leftarrow 0$, and $\tilde{\tau} \leftarrow 0$.

for(each coming packet $(f, t)$)
    $\tau \leftarrow \lfloor \frac{t}{\Delta} \rfloor \bmod (R+1)$;
    /* Remove out-of-date information */
    if($\tau \neq \tau_s$)
        foreach($f' \in \mathcal{C}$ and $\tau_{f'} \neq \tau_s$)
            $\mathcal{C} \leftarrow \mathcal{C} \setminus \{f'\}$;
        end
        if($\tilde{\tau} \neq \tau_s$)
                $\tilde{c} \leftarrow 0$;
        end
        $\tau_s \leftarrow \tau$;
    end
    /* Update counters for the current packet $(f, t)$ */
    if($f \in \mathcal{C}$)
        if ($\tau_f = (\tau - 1) \bmod (R+1)$)
            $c_f \leftarrow c_f + 1$;
            $\tau_f \leftarrow \tau$;
        end
    else if ($\tilde{c} = 0$ or $\tilde{\tau} = (\tau - 1) \bmod (R+1)$)
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{f\}$;
        $c_f \leftarrow \tilde{c} + 1$;
        $e_f \leftarrow \tilde{c}$;
        $\tau_f \leftarrow \tau$;
    end
    /* Prune short-lived flows if $\mathcal{C}$ is full */
    if($|\mathcal{C}| = m + 1$)
        $f_{min} \leftarrow \arg\min_{f' \in \mathcal{C}, \tau_{f'} = \tau}\{c_{f'}\}$;
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{f_{min}\}$;
        $\tilde{c} \leftarrow c_{f_{min}}$;
        $\tilde{\tau} \leftarrow \tau$;
    end
    /* Report the flow $f$ if $c_f - e_f > d$ */
    if($f \in \mathcal{C}$ and $c_f - e_f > d$)
        Report the packet $(f, t)$;
    end

end
```

Figure 3.23 Deterministic Algorithm

3.2.3 Deterministic Algorithm

We provide our deterministic algorithm and its theoretical analysis in this section, which applies a similar idea from the SpaceSaving algorithm [69] for the heavy hitter detection. We also maintain a candidate list, this is the same as SpaceSaving. If the candidate list is full, we attempt to remove the flows with short durations from the candidate list. We utilize wrap-around counters to maintain the timing information for recent packets. Based on these information, we can skip duplicated packets in the same interval, and estimate the duration for each flow with bounded errors. An outline of our algorithm is shown in Fig.3.23.

3.2.3.1 Algorithm

In the deterministic algorithm, we maintain a list of the candidate flows, denoted by $\mathcal{C}$, which contains four variables for each flow, i.e., the flow ID f, the duration counter c_f, the error counter e_f, and the wrap-around counter τ_f. Besides the candidate list, we also maintain a counter $\tilde{c}$ for the maximum possible duration of any flow $f \notin \mathcal{C}$, and a wrap-around counter $\tilde{\tau}$ for its timing information. In order to detect LDFs, our deterministic algorithm always guarantees that the estimated duration of the candidate flows in $\mathcal{C}$ is larger than or equal to $\tilde{c}$.

When a new packet (f, t) arrives, we first convert t into an integer τ between 0 and R by using

$$\tau = \left\lfloor \frac{t}{\Delta} \right\rfloor \bmod (R+1), \tag{3.52}$$

where *mod* denote the modulus operation. We scan the whole candidate list at the beginning of each time interval such that the terminated flow can be removed as soon as it is terminated.We also check the counter $\tilde{c}$ at the beginning of each interval. If $\tilde{\tau} \neq (\tau - 1) \bmod (R+1)$, there would be no packet in any flows $f' \notin \mathcal{C}$ in the previous interval. And we can set the counter $\tilde{c}$ back to 0.

If a flow f is maintained in $\mathcal{C}$, we can use the wrap-around counter τ_f to update c_f without any errors. Because we remove terminated flows at the beginning of each interval, there are only two possible cases.

- If $\tau_f = \tau$, we have updated this counter in this interval and should skip this packet.

- If $\tau_f = (\tau - 1) \bmod (R+1)$, the counter was updated in the previous interval and has not been updated in the current interval. We should increase its duration for the new packet and update its wraparound counter correspondingly,

$$\begin{aligned} c_f &= c_f + 1, && (3.53) \\ e_f &= e_f, && (3.54) \\ \tau_f &= \tau. && (3.55) \end{aligned}$$

If $f \notin \mathcal{C}$, we estimate its duration based on $\tilde{c}$ and $\tilde{\tau}$, and insert this flow into $\mathcal{C}$ as a new flow if its estimated duration is longer than the minimum duration of the candidate flows in $\mathcal{C}$.

- If $\tilde{c} = 0$, all active flows are maintained in the candidate list $\mathcal{C}$. Thus, this packet is the first packet in the flow f. We insert it into $\mathcal{C}$ as

$$\begin{aligned} c_f &= 1, && (3.56) \\ e_f &= 0, && (3.57) \\ \tau_f &= \tau. && (3.58) \end{aligned}$$

- If $\tilde{c} > 0$ and $\tilde{\tau} = (\tau - 1) \bmod (R+1)$, the flow f may be missed in the previous interval. The duration of the flow f must be between 1 and $\tilde{c}+1$. This flow may have a longer duration than the candidate flows in $\mathcal{C}$. We insert f into $\mathcal{C}$ as

$$\begin{aligned} c_f &= \tilde{c} + 1, && (3.59) \\ e_f &= \tilde{c}, && (3.60) \\ \tau_f &= \tau. && (3.61) \end{aligned}$$

- If $\tilde{c} > 0$ and $\tilde{\tau} = \tau$, we have updated the counter $\tilde{c}$ to the minimum duration of the candidate flows in $\mathcal{C}$. This flow should be skipped because its estimated duration cannot be longer than the minimum duration of the candidate flows in $\mathcal{C}$.

If there are too many flows in $\mathcal{C}$, i.e., $|\mathcal{C}| = m + 1$, we scan the candidate list to prune a short-lived flow from $\mathcal{C}$. Let $f_{min} \in \mathcal{C}$ denote one of the active flows with the minimum duration, which should be inserted into $\mathcal{C}$ in this interval,

$$f_{min} = \arg \min_{f' \in \mathcal{C}, \tau_{f'} = \tau} \{c_{f'}\}. \tag{3.62}$$

We remove this flow f_{min} from $\mathcal{C}$ and update $\tilde{c}$ according to $c_{f_{min}}$,

$$\begin{aligned} \mathcal{C} &= \mathcal{C} \setminus \{f_{min}\}, && (3.63) \\ \tilde{c} &= c_{f_{min}}, && (3.64) \\ \tilde{\tau} &= \tau. && (3.65) \end{aligned}$$

We update $\tilde{\tau}$ to the current time stamp, and the scan process to find the flow with the minimum duration can be run at most once in each interval.

At last, if $c_f - e_f > d$, we report f as a LDF. Because $c_f - e_f$ is the number of intervals that the flow is maintained in the candidate list, we always have $d_f \geq c_f - e_f$.

3.2.3.2 Theoretical Analysis

In this section, we provide an error bound for the flow duration estimation in our deterministic algorithm. In the deterministic algorithm, we always prune the flow with the smallest estimated duration from the list. Therefore, the flows that are not maintained in the candidate list should have short durations. We show that the duration of a flow $f \in \mathcal{C}$ can be bounded by c_f and e_f.

Lemma 5. *For any flow $f \in \mathcal{C}$, we have*

$$c_f - e_f \leq d_f \leq c_f. \tag{3.66}$$

Proof. When the flow is first inserted into the candidate list $\mathcal{C}$, we always set $e_f = c_f - 1$. Because the duration of the flow is at least 1, we have $d_f \geq c_f - e_f$. For the following packets, we can update c_f without any errors by using the wrap-around counter τ_f. Thus we always have

$$d_f \geq c_f - e_f. \tag{3.67}$$

For each new flow, we estimate its duration when it is insert into the candidate list $\mathcal{C}$.

- If $\tilde{c} = 0$, we always insert a new flow into the candidate list. Because $\tilde{c} = 0$, there is no flow removed or skipped from $\mathcal{C}$ in the previous interval. All active flows should be maintained in $\mathcal{C}$. Thus, the duration of the new flow should be 1. We set $c_f = 1 = d_f$.

- If $\tilde{c} > 0$ and $\tilde{\tau} = (\tau - 1) \bmod (R+1)$, we also insert the new flow f into the candidate list. If this flow is missed in the previous interval, it has an estimated duration at most $\tilde{c}+1$. Otherwise, it should be kept in $\mathcal{C}$ because we only remove the flow with the minimum counter from $\mathcal{C}$. We set $c_f = \tilde{c} + 1 \geq d_f$.

- If $\tilde{c} > 0$ and $\tilde{\tau} = \tau$, we should remove an active flow with the minimum duration from $\mathcal{C}$ and set $\tilde{c} = c_{f_{min}}$ in the current interval. If f is a new flow, its duration is at most 1. We should skip this flow. If f is removed from $\mathcal{C}$ in one of previous intervals, it must have the minimum counter c_f when it is removed. Because all candidate flows have a larger counter than it, its estimate duration must be less than or equal to $\tilde{c}$. We should also skip it. We can ensure that the skipped flow $\tilde{f} \notin \mathcal{C}$ has a duration $d_{\tilde{f}} \leq \tilde{c}$.

Later, we can update its duration without errors if it is maintained in $\mathcal{C}$. Therefore, we always have

$$c_f \geq d_f. \tag{3.68}$$

□

Next, we show that the counter $\tilde{c}$ can be bounded by the ℓ-tail duration $D^{res(l)}$.

Lemma 6. *Let m denote the size of the candidate list $\mathcal{C}$. The counter $\tilde{c}$ can be bounded by*

$$\tilde{c} \leq \frac{D^{res(l)}}{m - \ell}, \tag{3.69}$$

if the top-ℓ LDFs exist in the traffic stream.

Proof. Because the counter $\tilde{c}$ can only be increased when $\mathcal{C}$ is full, we always have at least m candidate flows. The m packets in the candidate flows increase the sum of the durations D by m. If the top-ℓ LDFs exist in the traffic stream, we have at least ℓ candidate flows with $d_{f_i} > \tilde{c}$. Therefore,

$$\sum_{f \in \mathcal{F}} d_f \geq (m - \ell)\tilde{c} + \sum_{i=1}^{\ell} d_{f_i}. \tag{3.70}$$

We can get

$$\tilde{c} \leq \frac{D^{res(\ell)}}{m-\ell}. \tag{3.71}$$

□

At last, we show that our deterministic algorithm can provide the ℓ-tail bound for the LDF detection.

Theorem 4. *Given $m = \lceil \frac{1}{\varepsilon} \rceil$, our deterministic algorithm can provide the ℓ-tail bound for the LDF detection.*

Proof. Based on the above two lemmas, we have

$$d_f - \frac{D^{res(\ell)}}{m-\ell} \leq c_f - e_f \leq d_f. \tag{3.72}$$

Because $m = \lceil \frac{1}{\varepsilon} \rceil$, we have

$$d_f - \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)} \leq c_f - e_f \leq d_f. \tag{3.73}$$

Therefore, our deterministic algorithm can always determine f as a LDF if $d_f > d + \frac{\varepsilon}{1-\varepsilon\ell} D^{res(\ell)}$ and do not determine any flow f' as a LDF if $d_{f'} < d$. □

Our deterministic algorithm can use $O(1/\varepsilon)$ space to provide the ℓ-tail bound. If $\ell = 0$, our deterministic algorithm provides a deterministic ε bound for the flow duration estimation. Thus, the ℓ tail bound is at least as strong as the error bound in Chen's algorithm. If the distribution of the flow durations is skewed, D^{ℓ} is much smaller than D. In this case, the ℓ-tail bound provide a much stronger bound for the flow duration estimation.

In fact, there is a lower bound for the flow duration estimation with the ℓ-tail bound based on the studies of the heavy hitter problem.

Theorem 5. *Any algorithm that provides the ℓ-tail bounds for the LDF detection must require at least $O(1/\varepsilon)$ space.*

Proof. If each flow has only one packet in each interval, the LDF detection problem is equivalent to the heavy hitter problem. Based on the results of the heavy hitter analysis [12], we need at least $O(1/\varepsilon)$ counters for the LDF detection. In the general cases, we only consider the first packet for each flow in each interval, and skip the following packets by using the wraparound counters. Therefore, the LDF detection problem is also converted to the heavy hitter problem. The same lower bound is still true. □

Based on the above theorem, our deterministic algorithm is space-optimal for the flow duration estimation with the ℓ-tail bound.

3.2.4 Randomized Algorithm

In order to detect LDFs when the available space is less than the mean of the number of active flows, we must find some space-efficient algorithms that can detect LDFs with a certain probability but require much less space than the deterministic algorithm. In this section, we provide a randomized algorithm which utilizes the priority sampling to maintain the candidate list. We generate a new hash function for the priority computation in each interval and ensure that the flow with a longer duration will also have a smaller priority in our randomized algorithm. In this way, the LDFs can be detected with a probability as high as possible, even though the space is much less than the deterministic algorithm.

3.2.4.1 Algorithm

In our randomized algorithm, we maintain a candidate list, which contains four variables for each flow, i.e. the flow ID f, the duration counter c_f, the priority counter p_f, and the wrap-around counter τ_f. We also maintain a priority threshold $\tilde{f}$ of the active flows that are not maintained in $\mathcal{C}$, i.e., $\tilde{f} \notin \mathcal{C}$. An outline of our randomized algorithm is shown in Fig.3.24.

For each interval, we use a independent hash function, denoted by $h_\gamma(\cdot)$, to compute the priority for each flow. Given a flow ID f, the hash function $h_\gamma(\cdot)$ maps f into a real number between 0 and 1. We always keep the smallest priority in the priority counter p_f for each candidate flow in $\mathcal{C}$. In this way, a flow with a longer duration can also have a smaller priority.

When a new packet (f, t) arrives, we first convert t into an integer τ between 0 and R as before. We scan the whole candidate list at the beginning of each time interval such that the terminated flow can be removed as soon as it is terminated. Also, we reset $\tilde{p}$ to 1 and, generate

```
Initialize $\mathcal{C} \leftarrow \emptyset$ and $\tau_s \leftarrow 0$.

**for**(each coming packet $(f, t)$)
    $\tau \leftarrow \lfloor \frac{t}{\Delta} \rfloor \bmod (R+1)$;
    /* Remove out-of-date information */
    **if**($\tau \neq \tau_s$)
        **foreach**($f' \in \mathcal{C}$ and $\tau_{f'} \neq \tau_s$)
            $\mathcal{C} \leftarrow \mathcal{C} \setminus \{f'\}$;
        **end**
        $\tilde{p} \leftarrow 1$;
        $\tau_s \leftarrow \tau$;
        Generate $h_\gamma(\cdot)$;
    **end**
    /* Update counters for the current packet $(f, t)$ */
    **if**($f \in \mathcal{C}$)
        **if** ($\tau_f = (\tau - 1) \bmod (R+1)$)
            $c_f \leftarrow c_f + 1$;
            $p_f \leftarrow \min\{p_f, h_\gamma(f)\}$;
            $\tau_f \leftarrow \tau$;
        **end**
    **else if** ($h_\gamma(f) < \tilde{p}$)
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{f\}$;
        $c_f \leftarrow 1$;
        $p_f \leftarrow h_\gamma(f)$;
        $\tau_f \leftarrow \tau$;
    **end**
    /* Prune short-lived flows if $\mathcal{C}$ is full */
    **while**($|\mathcal{C}| = m + 1$)
        $\tilde{p} \leftarrow \tilde{p}/2$;
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{f'' \,|\, p_{f''} \geq \tilde{p}\}$;
    **end**
    /* Report the flow $f$ if $c_f > d$ */
    **if**($f \in \mathcal{C}$ and $c_f > d$)
        Report the packet $(f, t)$;
    **end**

**end**
```

Figure 3.24 Randomized Algorithm

a new independent hash function $h_\gamma(\cdot)$ for the flow priority in this interval. The hash function $h_\gamma(\cdot)$ maps a flow ID f into a number $h_\gamma(f) \in (0,1)$.

If a flow f is maintained in $\mathcal{C}$, we can use the wrap-around counter τ_f to update c_f without any errors. Same as the deterministic algorithm, there are only two possible cases.

- If $\tau_f = \tau$, we have updated this counter in this interval and should skip this packet.
- If $\tau_f = (\tau - 1) \bmod (R+1)$, the counter was updated in the previous interval and has not been updated in the current interval. We should increase its duration for the new packet by 1 and update its wraparound counter correspondingly. Also, we compute the priority of this flow with $h_\gamma(f)$. If $h_\gamma(f) < p_f$, we update its priority to $h_\gamma(f)$.

$$c_f = c_f + 1 \tag{3.74}$$

$$p_f = \min\{p_f, h_\gamma(f)\} \tag{3.75}$$

$$\tau_f = \tau. \tag{3.76}$$

If $f \notin \mathcal{C}$, we check the priority threshold $\tilde{p}$ of the flows $f \notin \mathcal{C}$. If $h_\gamma(f) < \tilde{p}$, this flow is inserted into $\mathcal{C}$ as a new flow,

$$c_f = 1 \tag{3.77}$$

$$p_f = h_\gamma(f) \tag{3.78}$$

$$\tau_f = \tau. \tag{3.79}$$

Otherwise, we skip this flow.

If there are too many flows in $\mathcal{C}$, we reduce the priority threshold $\tilde{p}$ by half, and remove the flows with a larger priority than it. In this way, we can adaptively find a proper sampling probability for our deterministic algorithm.

At last, if $c_f > d$, we report f as a LDF.

3.2.4.2 Theoretical Analysis

We can prove that the probability that an LDF can be detected approach to 1 if its duration approaches to infinite.

Theorem 6. *If the size of the candidate list is larger than the number of LDFs, our randomized algorithm can guarantee that*

$$\lim_{d_f \to \infty} P(f \textit{ is detected}) = 1. \tag{3.80}$$

Proof. Because the hash function $h_\gamma(\cdot)$ is generated randomly for each interval, the priority $h_\gamma(f)$ for a given LDF can be viewed as independent uniform distributed random variables. In the candidate list, we use p_f to maintain the minimum priorities across its active intervals. If the duration of an LDF approaches to infinite, we can know that

$$\lim_{d_f \to \infty} p_f = \lim_{\gamma \to \infty} \min\{h_\gamma(f)\} = 0. \tag{3.81}$$

Because the size of the candidate list is larger than the number of active LDFs, the LDF with $p_f \simeq 0$ is maintained in the candidate list. After the LDF is inserted into the candidate list, its duration counter c_f also increases to infinite. Therefore, this LDF is detected in our randomized algorithm with a probability 1. □

When the number of active flow is very large, the deterministic algorithm cannot provide any bound for the LDF detection. In this case, the sampling method is usually used in the network traffic monitoring to reduce the resource requirement. However, a uniform packet sampling method can reduce the probability that a LDF can be detected exponentially. Our randomized algorithm only requires that the size of the candidate list is larger than the number of the active LDFs, which is much less than the number of active flows. A flow with a longer duration can be maintained in the candidate list with a probability higher than other flows. In other words, our randomized algorithm prefer LDFs to short-lived flows, which is the property that the sampling method do not have.

3.2.5 Implementation Considerations

We provide a theoretical view of our algorithms in the above two sections, i.e., we can detect the LDFs by using a candidate list. However, we have not provided an efficient implementation to maintain the candidate list in our algorithms. Because there are a large number of packets that come in each timeout interval in a high-speed network, we must process each packet

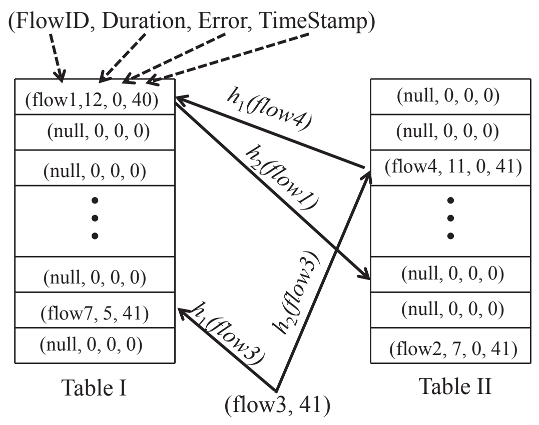


Figure 3.25 Hash Table with Cuckoo Hashing

efficiently. We provide a detailed implementation of our algorithms in this section. Because both algorithms share the same framework, we discuss their performance together. There are two major operations in the candidate list. Firstly, we need to look up the entry for the flow when a new packet comes. Secondly, we must prune short-lived flows from the candidate list if there are too many candidate flows. We provide a detailed description of these two major operations in the remaining parts.

3.2.5.1 Lookup

We implement the candidate list by using a hash table with Cuckoo hashing [59] in order to find the entry for each flow in constant time. The main challenge to maintain the flow information in a hash table is that we must handle the hash collisions in a proper way. In Chen's algorithm, there is no need to handle the hash collisions. Instead, they apply the same idea as the Bloom Filter, which uses multiple hash functions to reduce the errors caused by the hash collisions. Due to the hash collisions, Chen's algorithm can only provide a week bound for the flow duration estimation, i.e. the error is small with a high probability. In the previous section, we have shown that our deterministic algorithm can provide the optimal bound for the flow duration estimation with the same space as Chen's algorithm, i.e., the error can always be bounded by the tail of the flow durations $D^{res(\ell)}$ with $O(1/\varepsilon)$ space.

So far, the most efficient approach to handle hash collisions should be Cuckoo hashing [59]. As shown in Fig.3.25, the Cuckoo hashing requires two hash tables, each of which is associated with an independent hash function, denoted by $h_1(\cdot)$ and $h_2(\cdot)$. Each flow is maintained in either Table I or Table II, but never both of them. When a packet (f, t) comes, we only need to check two positions, i.e. $h_1(f)$ in Table I and $h_2(f)$ in Table II, to determine whether this flow is maintained in the candidate list. Therefore, we can find the entry of the candidate flows in a constant time.

If the flow is not maintained in either hash tables, we decide whether we should insert it into them based on $\tilde{c}$ and $\tilde{\tau}$ in our deterministic algorithm. If we want to insert the new flow into the candidate list, the Cuckoo hashing uses a pushing method to insert f into the hash tables. We first try to insert the new flow f into one of the two tables. If both positions are non-empty, we choose one of the tables randomly, e.g., Table I. We insert the new flow f into Table I and push out a candidate flow f' at its hash position. Next, we try to insert f' into the other table, i.e., Table II. If the hash position for f' in Table II is empty, we put f' in this table and stop. Otherwise, we insert f' into Table II and push out another candidate flow f'' in Table II. We try to insert f'' into Table I similarly. If the pushing process does not stop until a maximum number of loops, we rehash all flows in both tables.

The advantage of the Cuckoo hashing is that the running time to find a flow is very small. The performance of the Cuckoo hashing is close to the optimal bound for the hash tables if the size of the hash table is large enough. If we guarantee that the size of the hash table is at least twice of the size of the candidate list, the running time to insert a new flow is $O(1)$ in average [59].

3.2.5.2 Prune

When there are more than m flows in the candidate list, we try to remove short-lived flows. The process to remove short-lived flows in the deterministic and randomized algorithms are different to each other.

In the deterministic algorithm, we scan the hash table to find the minimum duration if the candidate list is too large. During this scan process, we can maintain the positions of the flows with the current minimum duration in a linked list. At the end of the scan process, we can remove all these flows from the candidate list. Because there are at most one scan processes in each interval, the running time to prune short-lived flows is $O(1/\epsilon)$.

In the randomized algorithm, we reduce the priority threshold by half if there are two many flows in $\mathcal{C}$. Given the number of active flows a_γ, we can find a proper threshold $\tilde{p}$ with at most $\log a_\gamma$ scan processes. The running time to prune short-lived flows in the randomized algorithm is $O(\frac{\log a_\gamma}{\epsilon})$ in each interval.

In order to process each packet in constant time, we can use another thread to prune expired or short-lived flows from the candidate list. In this way, there is no loop in the procedure to process each packet. For simplicity, we do not show the details to use multiple threads to prune expired or short-lived flows in Fig.3.23 or Fig.3.24.

3.2.5.3 Running Time

Based on the above analysis, we can get the running time of our deterministic and randomized algorithms in the following theorem.

Theorem 7. *Our deterministic algorithm can process each packet in $O(1)$ running time. The candidate list can be maintained in $O(1/\epsilon)$ running time for each interval.*

Proof. When a new packet comes, we only need to check two positions in the hash tables to determine whether it is maintained in the candidate list $\mathcal{C}$. If so, we can update its information in $O(1)$ running time. If not, we can determine whether we should insert it into $\mathcal{C}$ by checking $\tilde{c}$ and $\tilde{\tau}$, which also only takes $O(1)$ running time. In order to insert a new flow into $\mathcal{C}$, there is only a small probability that we need to move more than $\log m$ flows in the hash table. If the size of the hash table is twice of the number of the candidate flows in $\mathcal{C}$, the insertion only takes $O(1)$ running time in average. Therefore, our deterministic algorithm can process each packet in $O(1)$ running time.

At the beginning of each interval, we scan the candidate list to remove expired flows, which takes $O(m)$ running time. If the candidate list is full, we remove the flow with the minimum duration from $\mathcal{C}$ and set $\tilde{\tau}$ to the current time stamp. In this way, the process to prune short-lived flows runs at most once in each interval, which also takes $O(m)$ running time. Therefore, we can maintain the candidate list in $O(m)$ running time for each interval. Give $m = 1/\epsilon$, the

Type	Size	Length	Bots	Flows
Storm	4.6G	24 hours	13	9,666,871
Nugache	1.5G	24 hours	82	3,007,636

Table 3.1 Botnet Flows

running time of our deterministic algorithm in each interval is at most $O(1/\epsilon)$. $\square$

Theorem 8. *Our randomized algorithm can process each packet in $O(1)$ running time. The candidate list can be maintained in $O(\frac{\log \bar{a}}{\epsilon})$ running time for each interval, where $\bar{a}$ denotes the average number of active flows in each interval.*

Proof. The procedure to process each packet in the randomized algorithm is similar to the deterministic algorithm. It is easy to show that the running time to process each packet is also $O(1)$.

The procedure to remove expired flows takes $O(m)$ running time in each interval. Because we do not know the number of active flows in each interval in advance, we adjust the sampling rate in our randomized algorithm adaptively. The sampling rate is determined by the priority threshold in our randomized algorithm. We reduce the priority threshold by half each time to prune short-lived flows from the candidate list. Let $\bar{a}$ denote the average number of active flows in each interval. The threshold is reduced by at most $\log \bar{a}$ times in an interval in average. Therefore, the running time to maintain the candidate list can be bounded by $O(m \log \bar{a})$. Give $m = 1/\epsilon$, the running time of our randomized algorithm in each interval is at most $O(\frac{\log \bar{a}}{\epsilon})$. $\square$

In fact, we estimate the priority threshold based its value in the previous interval. In this section, we do not make any assumptions about the network traffic, which guarantee that our algorithm can be widely used in various situations. In practice, we can find a proper threshold with a few of the reductions, if the network traffic do not have large changes.

3.2.6 Evaluation

We evaluate our algorithms by using a traffic trace from the honeypots, which contains the traffic from two P2P botnets, i.e. Nugache and Storm. The IP addresses of the hosts/honeypots that was running as a bot, were identified. We can use these IP addresses to find all the flows

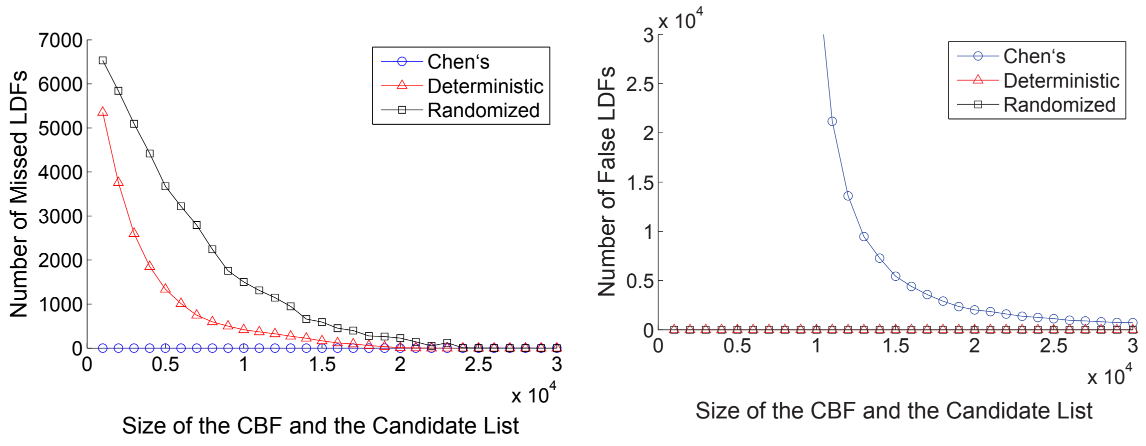


Figure 3.26 False Positive/Negative Error

related to one of the botnets. The properties of the botnet flows are summarized in Table 3.1. The number of active flows in each interval can have very large fluctuations in our traffic trace. The average number of flows in one minute is more than 10^4, and the total number of flows in one day is more than 10^7. We choose the timeout interval as 1 minutes, and the thresholds of the LDFs as 30 minutes. There are 7280 flows with a duration longer than or equal to 30 minutes.

In this section, we compare the performance of our algorithms to Chen's algorithm for the LDF detection. We evaluate the performance of our algorithm and Chen's algorithm, in terms of the false positive/negative error, the running time, and the duration estimation error. We implement three algorithms in C++ on a Linux (Fedora 13) machine with 3.2 GHz Pentium dual-core processor and 2GB RAM. Chen's algorithm uses $K = 3$ hash functions and two CBFs, each of which has the same size as the candidate list in our algorithms. All algorithms are implemented in a single thread. At last, we also evaluate the effects of the packet sampling to Chen's algorithm and our algorithms.

3.2.6.1 Detection Error

The false positive/negative errors of three algorithms are shown in Fig.3.26. The false positive error is evaluated by the number of false LDFs, and the false negative error is evaluated by the number of missed LDFs. Because Chen's algorithm always overestimates the duration

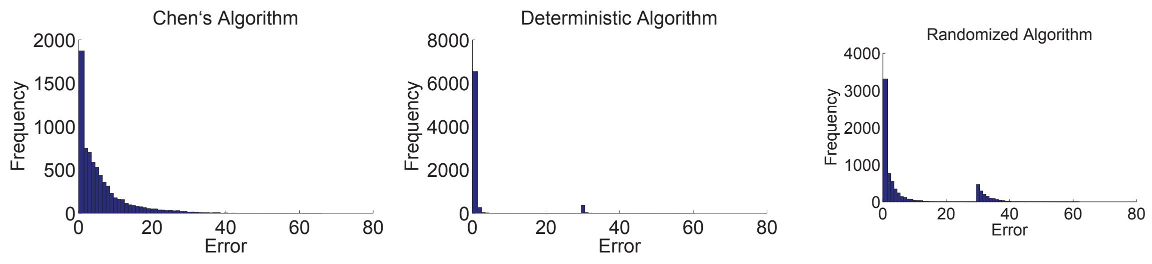


Figure 3.27 Distribution of Flow Duration Estimation Errors with $m = 10000$

for each flow, the number of false LDFs can increase very quickly. There is no LDF missed by Chen's algorithm, and thus all LDFs can be detected. Our algorithms solve the LDF detection problem in a different way. There is no false positive error in our algorithms. But some LDFs may be missed by our algorithms. Based on the evaluation, even if the size of the candidate list is very small, there are still some LDFs detected by both our algorithms.

3.2.6.2 Running Time

All algorithms are implemented by using a single thread, and we measure the total running time in each algorithm. We find that the running time of each algorithm is close to each other and does not change a lot for different sizes of the CBF or the candidate list. Our algorithms have a constant running time to precess each packet, which make them efficient enough to detect LDFs in high-speed networks.

3.2.6.3 Duration Estimation

In the duration estimation problem, we want to estimate the duration for any given flow f. All three algorithms can provide an estimator for the flow duration. In Chen's algorithm, the duration of any given flow f can be estimated as

$$\hat{d}_f^{(1)} = B_i(f) = \min_{k=1,\ldots,K} \{B_i[h_{ik}(f)]\} \text{ for } i = 1, 2. \tag{3.82}$$

We can use one of the CBF to estimate the duration for a given flow. In our deterministic algorithm, we can use c_f and e_f to estimate the duration of any candidate flow f in $\mathcal{C}$,

$$\hat{d}_f^{(2)} = c_f - e_f \text{ for } f \in \mathcal{C}. \tag{3.83}$$

If a flow f is not maintained in $\mathcal{C}$, we estimate its duration as 0. In our randomized algorithm, we can use a similar method. If the flow f is maintained in $\mathcal{C}$, we estimate its duration based on c_f,

$$\hat{d}_f^{(3)} = c_f \text{ for } f \in \mathcal{C}. \tag{3.84}$$

Otherwise, we also estimate its duration as 0.

The error for the flow duration estimation can be evaluated by the difference between the true duration d_f and its estimated duration $\hat{d}_f^{(i)}$ from three algorithms,

$$e_f^{(i)} = |d_f - \hat{d}_f^{(i)}|, \tag{3.85}$$

where $i = 1, 2, 3$.

We show the distribution of the flow duration estimation errors of the LDFs with $m = 10000$ in Fig.3.27. In Fig.3.27, the size of the candidate list is close to the average number of active flows. There are some LDFs missed in our deterministic and randomized algorithms. These missed LDFs have a duration close to the threshold $d = 30$ minutes, as shown in Fig.3.27. If we only consider the detected LDFs, we can see that the errors for their duration estimation are very small. When we increase the size of the candidate list, all LDFs can be detected and there is no error in their duration estimation. Although all LDFs can be detected in Chen's algorithm, there are still some errors in the flow duration estimation, even if the size of the CBF is very large.

3.2.6.4 Sampling Effects

We compare the effects of two uniform sampling methods on the LDF detection algorithms in this section, i.e. packet sampling and flow sampling. In a uniform packet sampling method, each packet is sampled with the same probability independently. In such method, the flows with large traffic volumes are sampled with a higher probability than others. Because the LDFs do not necessarily have high traffic volumes, many LDFs may be missed. Furthermore, the probability that a LDF can be detected decreases exponentially as the packet sampling rate decreases. Due to the biases in the packet sampling method, some advanced sampling methods [73; 81; 85] have been proposed for the network monitoring and measurement. In a

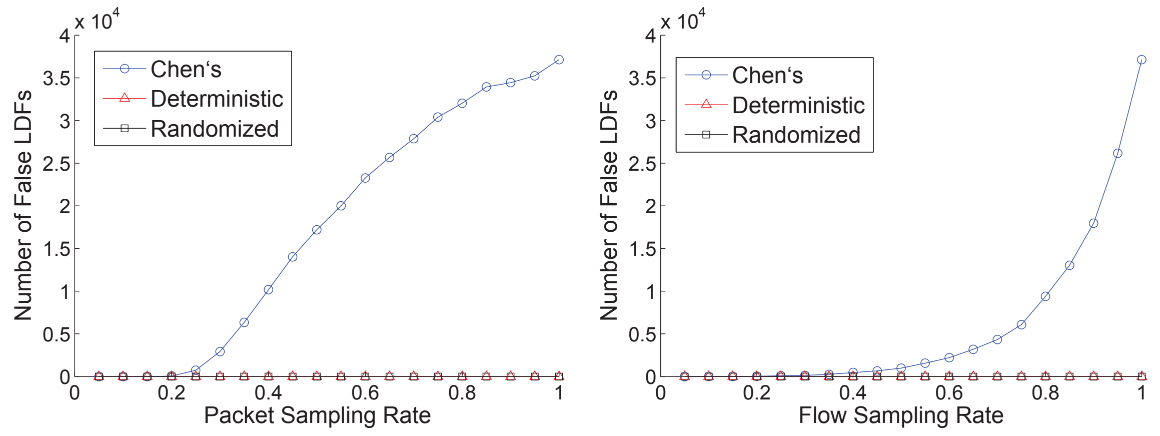


Figure 3.28 False Positives with Packet/Flow Sampling ($m = 10000$)

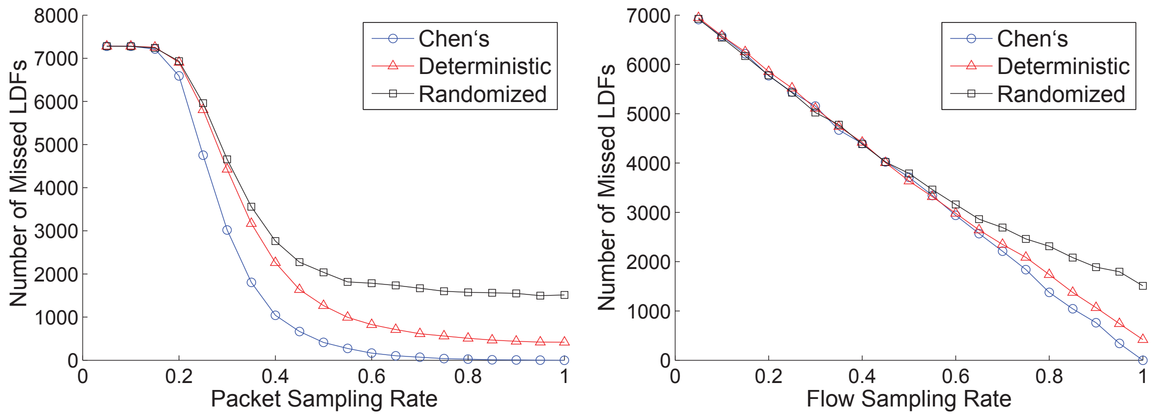


Figure 3.29 False Negatives with Packet/Flow Sampling ($m = 10000$)

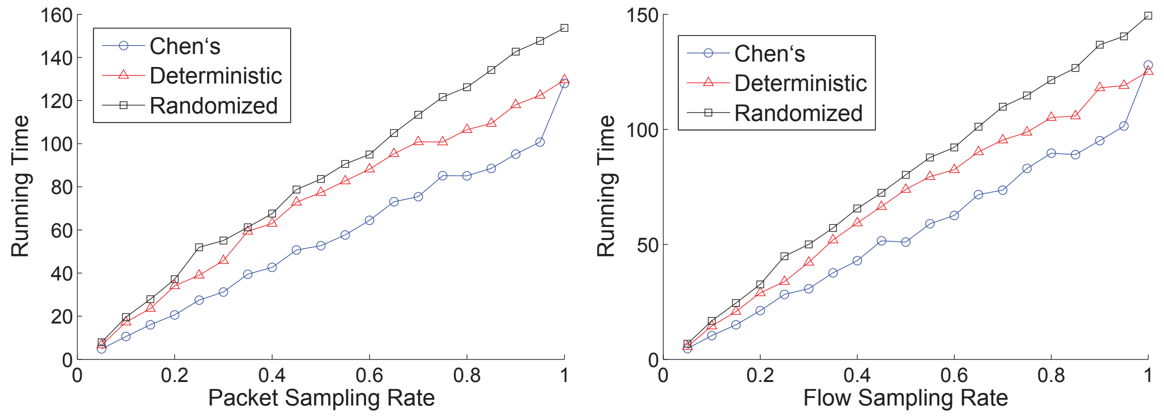


Figure 3.30 Running Time with Packet/Flow Sampling ($m = 10000$)

uniform flow sampling method, we focus on flows rather than individual packets. We use a hash function $\hbar(\cdot)$ to map a flow f to a priority, i.e., a real number between 0 and 1. Given a sampling rate ρ, if $\hbar(f) \leq \rho$, we keep a sample of this packet. Otherwise, we skip this packet. There are two advantages in the flow sampling method. The number of active flows can be reduced effectively to $\rho\bar{a}$. And the probability that a LDF can be detected is independent from its duration and traffic volume.

The false positive/negative errors are shown in Fig.3.28 and Fig.3.29. The running time is shown in Fig.3.30. The false negative error in our algorithms is very close to Chen's algorithm. We can see that there is a tradeoff between the false positive and the false negative in Chen's algorithm. If the sampling rate become larger, we can detect more LDFs but also have more false positive errors. The flow sampling always has fewer false positives than the packet sampling, and its false negatives decrease linearly as the sampling rate increases. Our algorithm can guarantee almost the same false negative error as Chen's algorithm, but do not allow any false positive errors.

3.2.7 Conclusion

In this section, we provide two data streaming algorithms for tracking LDFs in high-speed networks, which have only false negative errors. Our deterministic algorithm has been proved to be space-optimal for the flow duration estimation. Because the number of flows in each interval is often larger than the available space, our algorithms are more practical for the LDF detection than existing algorithms, which may report many short-lived flows as LDFs. In the future, we improve the performance of our algorithms when packet sampling is used in the network traffic measurement.

CHAPTER 4. REVERSIBLE SKETCHES

4.1 A Fast Sketch for Heavy-change Detection over High-Speed Network Traffic

There have been serious security problems that are hard to resolve, for example, botnets, polymorphic worm/virus, DDoS, etc. To address them, we need to monitor the traffic dynamics in a network-wide view, and more importantly, be able to detect attacks, failures, and traffic anomalies in a timely manner. Due to the rapid increase in link speeds and traffic volumes, it is often unfeasible to monitor every individual host or flow in the backbone network. Instead, we are forced to aggregate the packets into a small number of groups and apply the anomaly detection approaches on the aggregated flow for each group. Although the flow aggregation enables ISPs to detect traffic anomalies in a timely manner, it cannot preserve some critical information like IP addresses, port numbers, etc. Due to such missing information, it becomes very difficult (or often infeasible) for ISPs to identify the root causes of the traffic anomalies, and further resolve the network problems efficiently. In this section, we propose an efficient data structure, namely the *fast* sketch, which can both aggregate packets into a small number of aggregated flows, and further enable ISPs to identify the causes of detected anomalies. Using the fast sketch, the number of aggregated flows can achieve the lower bound of the heavy-change detection, and the running time to either process each packet or identify anomalous keys is sub-linear, which enable a real-time anomaly detection. Our work can significantly improve the efficiency and the reliability of the traffic anomaly detection.

	Sketch Size	Update Time	Communication Cost	Query Time
[31]	$O(\frac{1}{\epsilon}\log n)$	$O(\log n)$	$O(\frac{1}{\epsilon})$	$O(\frac{1}{\epsilon}\log n)$
[66]	$O(\frac{1}{\epsilon})$	$O(1)$	$O(\frac{1}{\epsilon})$	$O(n)$
[83]	$O\left(n^{\frac{1}{\log\log n}}\log\log n\right)$	$O\left(\frac{\log n}{\log\log n}\right)$	$O\left(n^{\frac{1}{\log\log n}}\log\log n\right)$	$O\left(\frac{1}{\epsilon}n^{\frac{1}{\log\log n}}\log\log n\right)$
[47]	$O(n^{1/2})$	$O(1)$	$O(n^{1/2})$	$O(\frac{1}{\epsilon}^2)$
Our	$O\left(\frac{1}{\epsilon}\log \epsilon n\right)$	$O\left(\log \epsilon n\right)$	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{1}{\epsilon}\log \epsilon n\right)$

Table 4.1 A comparison between our fast sketch and existing approaches

4.1.1 Introduction

The Internet has become an essential part of the daily life for billions of users worldwide. People are using and relying on a large variety of services built on the top of the Internet, such as web browsing, online banking, shopping, entertainment, VoIP, Video on demand, auction, social networks, etc. However, there have been serious security problems and network failures that are hard to resolve, for example, botnets, polymorphic worm/virus spreading, DDoS, etc. To solve them, many Internet Service Providers (ISPs) have chosen to use a distributed architecture for the network monitoring and detect traffic anomalies in a timely manner. Otherwise, the failure of doing so may cause catastrophic damages or unwanted results with impacts affecting online business, public safety, homeland security, personal privacy, the economy and the society at large.

In a distributed framework for the network measurement and monitoring, local monitors collect packet samples from routers and other network devices, and transfer their measurements to the Network Operation Centers (NOCs). And NOCs are responsible for mining characteristics of interest from collected measurements, and identifying the problems and the roots thereof. Due to the rapid increase in link speeds and traffic volumes, local monitors have to aggregate individual packets into a small number of aggregated flows, and only transfer their summaries, also referred as sketches, to the NOCs. There have been many approaches proposed for the traffic anomaly detection using the flow aggregation, e.g., signal processing [10], statistical analysis [15; 50; 51; 62], and so forth [26; 61].

An important approach for the anomaly detection is the *change detection* in the network traffic. We construct a statistical model for each aggregated flow based on the sketches, and detect the traffic anomaly as a significant deviation from this model. If flows or hosts change their traffic significantly in a short interval, there will be unexpected changes detected as traffic anomalies in the aggregated flows. For example, the equipment failures or the link congestions can cause sudden decreases in the traffic volume of the aggregated flows. And the DoS attacks and the worm spreading can introduce large increases. Some structured patterns can also be observed in the network traffic. The TCP SYN flooding attack can increase the traffic toward a server in a short interval, and then the traffic will decrease significantly. This is because the server maintains a large number of half-opened connections but there are no packet coming in these connections. The unexpected changes in the network traffic are important signs for the network management and the intrusion detection.

The traffic anomalies in the aggregated flows can be defined in either spatial or temporal domains [95]. In the spatial domain, we can compare the traffic volumes of each aggregated flows with each other. If there is an aggregated flow with the changes different from other flows significantly, we will raise an alarm for the traffic anomaly. Similarly, the traffic anomalies can also be detected in the temporal domain by analyzing the time series of each aggregated flows. Furthermore, some advanced attacks in the network may show a structured pattern in the temporal-spatial planes. We state our method for the the traffic anomaly detection in the spatial domain, which can be extended to the spatial domain or the temporal-spatial planes.

The performance of the traffic anomaly detection is highly depended on the accuracy of the summaries/sketches with respect to small space and running time. One of the major weaknesses of existing anomaly detection approaches is that it is often easy to trigger an alarm for the traffic anomaly but difficult to identify the roots of the anomaly alarm due to the space and time constraints. They can usually identify anomalous aggregated flows, but cannot find any flow keys in the packets responsible for the traffic anomaly. A flow can often be defined by five keys in the packet header, i.e., Source IP (32 bits), Destination IP (32 bits), Source Port (16 bits), Destination Port (16 bits), and Protocol (8 bits). Thus, the key space for the traffic anomaly detection can be as large as 2^{104}. If ISPs want to recover the desired anomalous keys, NOCs will need to collect sampled packets or other information from local monitors, which often requires a high communication cost and takes a long time to diagnose and resolve the network problem.

There have been various approaches proposed to help ISPs to diagnose anomalous aggregated flows responsible for network attacks or traffic anomalies, e.g., *Combinatorial Group Testing* (CGT) [31], *Random Projection* [66], *Modular Hashing* [83], *Chinese Reminder Theory* [47], and so on. Previous work mainly focus on the running time to process each packet at local monitors, and often ignore the running time to identify anomalous keys or the communication costs between local monitors and NOCs. In this section, we propose the *fast* sketch for the network-wide traffic anomaly detection. Our sketch can achieve nearly real-time traffic monitoring, and minimize the communication costs between local monitors and NOCs.

4.1.1.1 Our Contribution

We provide a detailed description to use our sketch for one of the most well-studied anomaly detection problem: the heavy-change detection. A comparison of our fast sketch and other approaches is shown in Table 4.1, where n is the size of the keys and ϵ is the error bound for the anomaly detection. The space is measured by the number of aggregated flows. The update running time is used to process each packet at local monitors, and the query running time is used to identify anomalous keys at NOCs. Our sketch implement the quotient technique to minimize the running time and the space. Our main contributions are summarized as following.

- We propose a novel sketch to aggregate flow information in the backbone network, which is designed to detect network-wide traffic anomalies.
- Local monitors share a small number of hash functions to compute the sketches, and the running time for the sketch computation is near-optimal.
- NOCs can identify the flow keys responsible for the traffic anomalies within a sub-linear running time. And local monitors only need to transfer the packets with the detected keys to the NOCs rather than all packet samples.
- The number of aggregated flows in our sketch achieves the lower bound of the heavy-change detection problem, which can optimize the communication cost between the local monitors and the NOCs.

- Our sketch can be easily implemented with existing traffic anomaly detection approaches, and improve their efficiency and reliability.

4.1.2 Problem Definition

There is a stream of packets across each local monitor. Each packet belongs to a specific flow f identified by the keys in the packet header,

$$f = \{SrcIP, SrcPort, DstIP, DstPort, Protocol\}. \tag{4.1}$$

The traffic volume of each flow f is measured by the total size of the packets in a specific time interval, e.g. 5 minutes. We divide packets into 5-minute intervals and try to find any heavy changes in the flow volumes between two consecutive intervals. Let $\mathbf{v}_t$ denote a vector of the traffic volumes during the t-th interval, where each element $\mathbf{v}_t[f]$ denote the traffic volume of the flow f during the t-th interval. And $V_t = \sum_{f=1}^{n} \mathbf{v}_t[f]$ denotes the total traffic volume during the t-th interval, where n is the size of the flow key.

The NOCs want to detect any flows with a heavy change in their traffic volumes.

Definition 5. A flow f is considered as a heavy-change flow if

$$|\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| \geq \theta \Delta_t \tag{4.2}$$

where $0 < \theta < 1$ is a given threshold, and Δ_t is the total change in the traffic volume in the current interval,

$$\Delta_t = \|\mathbf{v}_t - \mathbf{v}_{t-1}\|_1 = \sum_i |v_t[i] - v_{t-1}[i]|. \tag{4.3}$$

This problem is closely related to the compressed sensing in the signal processing [ric].

There are various ways to define the change in the flow volumes. For example, we can use the relative change in the traffic volume,

$$\frac{|\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]|}{|\mathbf{v}_t[f]|} > \theta. \tag{4.4}$$

Or we can define the change by using a time series method like the exponential smoothing

$$\mathbf{u}_1[f] = \mathbf{v}_0[f] \tag{4.5}$$

$$\mathbf{u}_t[f] = \alpha \mathbf{v}_{t-1}[f] + (1-\alpha)\mathbf{u}_{t-1}[f] \tag{4.6}$$

where $\mathbf{u}_t[f]$ is the prediction of the current traffic volume of the flow f. And the heavy-change flow can be detected by

$$|\mathbf{v}_t[f] - \mathbf{u}_t[f]| > \theta \|\mathbf{v}_t - \mathbf{u}_t\|_1. \tag{4.7}$$

Also, the traffic anomalies can be detected by other traffic features, i.e., cardinality, entropy, etc.

We can also define the traffic anomalies in the temporal domain.

Definition 6. A time interval τ is considered as heavy-change if

$$|V_\tau - V_{\tau-1}| \geq \theta \sum_{j=t-w}^{t} |V_j - V_{j-1}| \tag{4.8}$$

where w is the size of a given time window.

And the traffic anomalies can be defined in the spatial-temporal plane.

Definition 7. A pair (f, τ) is considered as heavy-change if

$$|\mathbf{v}_\tau[f] - \mathbf{v}_{\tau-1}[f]| \geq \theta \sum_{j=t-w}^{t} \|\mathbf{v}_j - \mathbf{v}_{j-1}\|_1. \tag{4.9}$$

In this section, we mainly focus on a basic traffic feature and a simple definition of the traffic anomalies to explain our fast sketch for easy understanding. Given $\theta = 1/k$, there will be at most k flows that can be identified as anomalous flows with a heavy change larger than the given threshold in Definition 1. In practice, it is very difficult to detect all the heavy-change flows exactly with limited space and running time. Usually, we are interested in the following approximate algorithm.

Definition 8. An (ϵ, δ)-approximate algorithm for the heavy-change flow detection can guarantee that:

- A flow f with $|v_f(t) - v_f(t-1)| \geq (\theta + \epsilon)\Delta V(t)$ will be reported as heavy-change with a probability at least $1 - \delta$.

- A flow f' with $|v_f(t) - v_f(t-1)| \leq (\theta - \epsilon)\Delta V(t)$ will be reported as heavy-change with a probability at most δ.

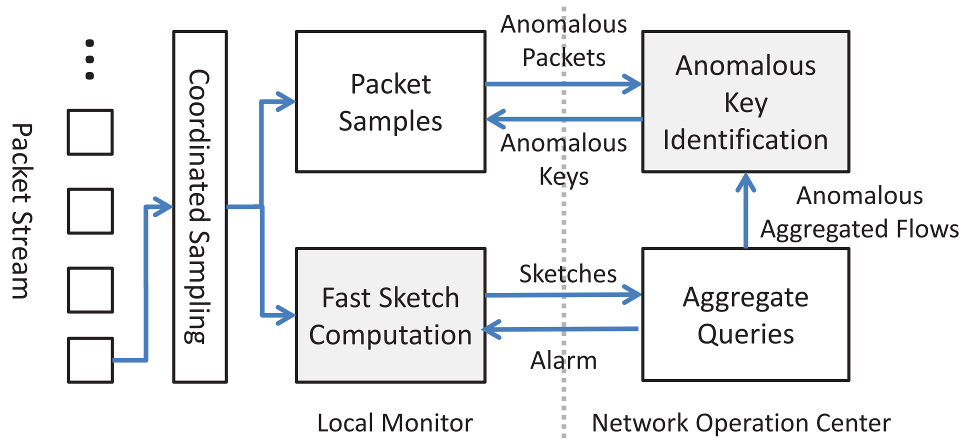


Figure 4.1 A framework for the distributed heavy-change detection

The error bound ϵ is chosen as $\epsilon < \theta$, and δ is a small probability to bound the false positive/negative errors. The space and running time of the approximate algorithm will be determined by the size of the flow key n, the error ϵ and the probability δ.

4.1.3 Our Sketch

In this section, we provide the design of our fast sketch for the heavy-change detection in a distributed monitoring system as shown in Fig.4.1. Our method for the traffic anomaly detection can be summarized as follows.

1. At each local monitor, we implement a coordinated sampling method [85] to collect packet samples from routers. A coordinated sampling method enables the NOCs to get a network-wide flow information without sampling the same packet multiple times at different monitors.

2. Local monitor keep the packet samples in the disk, and do some computation locally to mine useful information. Each local monitor only transfers a summary of their measurements in each interval to the NOCs, periodically. In this way, ISPs can save the packets at each local monitor, and reduce the communication costs as much as possible.

3. At the NOCs, we apply the traffic anomaly detection method with these summaries in several intervals to detect any attacks or network problems. For the heavy-change detection, the NOCs only need the first column of the sketches from local monitors.

4. If there is an alarm, the NOCs will request the whole sketches in an on-demand fashion. NOCs are able to use the detected anomalies to further identify the flow keys responsible to the network problems.

5. After the key recovery, NOCs can fetch only anomalous packets from the local monitors by sending anomalous keys, and solve the network problems efficiently by diagnosing these anomalous packets.

Our fast sketch aims to minimize the cost of collecting, transporting, storing, and computing for the traffic anomaly detection in a distributed network monitoring system. In the following parts, we first propose the sketch computation at the local monitors, and then give the traffic anomaly detection and the identification procedure of the anomalous keys at the NOCs.

4.1.4 Sketch Computation at Local Monitors

Our fast sketch uses a $\ell \times (1+\log(n/\ell))$ two-dimensional array of counters to maintain the flow information, denoted by $C_t[a,b]$, where $a = 0,\ldots,\ell-1$ and $b = 0,\ldots,\log(n/\ell)$. The number of rows in the sketch is chosen only based on the error bound, i.e., $\ell = \frac{4}{\epsilon}\log\frac{4}{\delta}$. And the number of columns is chosen based on both the size of the flow keys n and the number of rows ℓ.

Besides these counters, we also maintain a set of $\log\frac{4}{\delta}$ universal hash function $h_j(\cdot)$ to map each flow f into a row, and an independent hash function $p(\cdot)$ that map each flow f to ± 1. The hash function $h_j(\cdot)$ is designed in the way that we can easily identify the keys in each flow hashed to this row, and $p(\cdot)$ is used to provide an unbiased estimation of the traffic volume change.

The technique to construct the hash functions $h_j(\cdot)$ in our fast sketch is the *quotient*, which has been used to reduce the space requirement of the hash table [28], and also has many other applications [76]. Specifically, our hash functions are chosen as follows.

1. We first construct a set of universal hash functions $h'_j(\cdot)$ that map an integer $x \in \{0,\ldots,n/\ell\}$ to a row $\{0,\ldots,\ell-1\}$,

$$h'_j(x) = ((\alpha_j x + \beta_j) \bmod P) \bmod \ell, \tag{4.10}$$

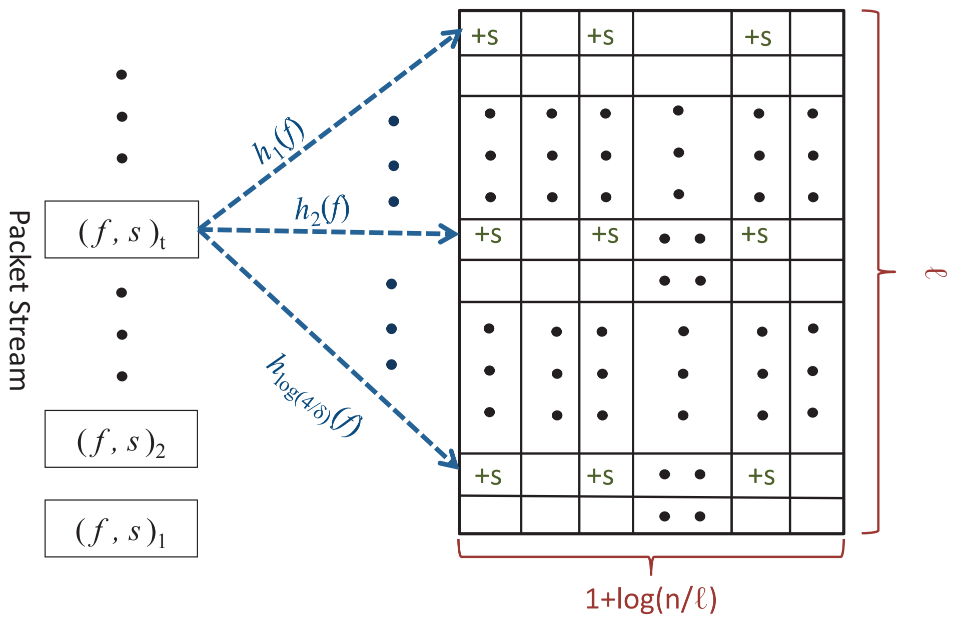


Figure 4.2 An example to update our sketch for a packet

where P is a prime larger than ℓ, and α_j, β_j are randomly and independently integers from $\{1, \dots, P-1\}$.

2. Next, a pair of the quotient function $q_j(\cdot)$ and the hash function $h_j(\cdot)$ is chosen based on each $h'_j(\cdot)$ as

$$q_j(f) = \lfloor f/\ell \rfloor, \tag{4.11}$$

$$h_j(f) = (f \bmod \ell) \oplus h'_j(\lfloor f/\ell \rfloor) \tag{4.12}$$

where $\oplus$ denotes the *logical exclusive or* of the bits of two integers.

All the counters are initialized with 0 at the beginning of each interval. When a packet (f, s) comes in the t-th interval, we map its flow f into $\log \frac{1}{\delta}$ rows in our fast sketch, as shown in Fig.4.2. We will add or subtract s to some selected counters in this row. Let $p(\cdot)$ be an independent hash function mapping f into $+1$ or -1 with an equally probability 1/2. We will add s if $p(f) = +1$, and otherwise will subtract s if $p(f) = -1$. In this way, each counter maintains a signed sum of the packet sizes. It is easy to show that each counter $C_t[a, b]$ can provide an unbiased estimation for the traffic volume, which can be found in the theoretical analysis section.

```
for($j \leftarrow 1, \ldots, \log \frac{4}{\delta}$)
        $C_t[h_j(f), 0] \leftarrow C_t[h_j(f), 0] + s \cdot p(f)$;
        $x \leftarrow q_j(f)$;
        for( $b \leftarrow 1, \ldots, \log(n/\ell)$)
            if(the $b^{th}$ bit in $x$ is 1)
                $C_t[h_j(f), b] \leftarrow C_t[h_j(f), b] + s \cdot p(f)$;
            end
        end
end
```

Figure 4.3 Update procedure

```
function Test($a$, $b$)
        if($|C^*_t[a, b] - C^*_{t-1}[a, b]| \geq \frac{\theta}{2}\Delta_t$)
            return true;
        else
            return false;
        end
end
```

Figure 4.4 Traffic anomaly detection in an aggregated flow

For the counter selection, we will always select the first counter in each row,

$$C_t[h_j(f), 0] = C_t[h_j(f), 0] + s \cdot p(f). \tag{4.13}$$

Next, we find the quotient of the flow f,

$$x = q_j(f) = \lfloor f/\ell \rfloor, \tag{4.14}$$

and update each counter $C_t[h_j(f), b]$ if the b^{th} bit in x is 1,

$$C_t[h_j(f), b] = C_t[h(f), b] + s \cdot p(f). \tag{4.15}$$

A description of the update procedure is shown in Fig.4.3.

4.1.4.1 Traffic Anomaly Detection and Attribution at NOCs

The NOCs collect the first columns of the fast sketches from local monitors in each interval, and add them together to get a network-wide view of the flow information due to the linear property of the fast sketches. Let $C_t^*[a,b]$ denote the array of counters by adding all local sketches in the t-th interval together,

$$C_t^*[a,b] = \sum_i C_t^i[a,b], \tag{4.16}$$

where i is the index of the local monitors. Besides the sketches, we also need to know the total traffic changes Δ_t for the traffic anomaly detection, which can be estimated using the Manhattan sketches [74]. Other traffic anomaly detection approaches may require other information to detect an aggregated flow as anomalous. We assume that there is such a test function available for us to determine whether an aggregated flow is anomalous or not. As an example, the test function for the heavy change detection is shown in Fig.4.4.

For the anomaly detection, the NOC will check the first counter in each row. If $Test(a,0)$ is true, we determine there is a heavy-change flow and try to identify its keys. Otherwise, we skip this row.

If there is a heavy-change flow in the row a, the NOC will fetch the remaining counters in this row from the local monitors. In order to identify the keys in the heavy-change flow f, we check the following counters one by one, which corresponds to each bit in the quotient. If $Test(a,b)$ is true, we set the bit in the quotient x to 1, and otherwise we set it to 0. By using this rule, we can find the quotient x in this row.

But we do not know which hash function should be used in the inverse function to recover the frequent item. Thus, we try each hash function one by one,

$$f = \varphi_j(x,a) \text{ for } j = 1,\dots,\log(4/\delta), \tag{4.17}$$

where the inverse function is chosen as

$$\varphi_j(x,y) = x\ell + y \oplus h_j'(x). \tag{4.18}$$

The correctness of our quotient technique can be easily verified,

$$\begin{aligned}\varphi_j(x,y) &= \lfloor f/\ell \rfloor \cdot \ell + (f \bmod \ell) \oplus h_j'(\lfloor f/\ell \rfloor) \oplus h_j'(\lfloor f/\ell \rfloor) \\ &= \lfloor f/\ell \rfloor \cdot \ell + (f \bmod \ell) \\ &= f. \end{aligned} \tag{4.19}$$

```
for($a \leftarrow 0, \ldots, \ell - 1$)
      if(Test($a$, 0))
        $x \leftarrow 0$;
        $r \leftarrow 1$;
        for($b \leftarrow 1, \ldots, \log(n/\ell)$)
          if(Test($a$, $b$))
            $x \leftarrow x + r$;
          end
          $r \leftarrow r \times 2$;
        end
        $y = a$;
        for($j \leftarrow 1, \ldots, \log(4/\delta)$)
          $f = \varphi_j(x, y)$;
          $\gamma \leftarrow 0$
          for($k \leftarrow 1, \ldots, \log(1/\delta)$)
            if(Test($h_k(f)$, 0))
              $\gamma \leftarrow \gamma + 1$;
            end
          end
          if($\gamma \geq \frac{1}{2}\log(1/\delta)$)
            Report $f$ as anomalous;
          end
        end
      end
end
```

Figure 4.5 Identification procedure

At last, we remove false positives by using a verification algorithm as the same as the Count-Min sketch [30]. We check the test result of the first counter in each row that a flow f is hashed into. If at least a half of them are true, we will report f as anomalous. Otherwise, we try to use the next hash function to recover the flow.

A description of the identification procedure is shown in Fig.4.5. After they identify the anomalous flow keys, the NOCs may also require the sampled packets with the detected keys from the local monitors.

4.1.5 Theoretical Analysis

We first introduce some notations in our theoretical analysis. Let $\mathcal{F}_{a,b}$ denote the set of flows that are hashed into the a-th row and have 1 at the b-th bit in their quotients. In other words, if a flow f is hashed into the a-th row, we consider that this flow is aggregated into $\mathcal{F}_{a,0}$. And if the b-th bit in its quotient is 1, we also consider that it is aggregated to $\mathcal{F}_{a,b}$ for $b > 0$.

The counter $C_t[a, b]$ maintains a signed sum of the traffic volumes of all flows aggregated to $\mathcal{F}_{a,b}$ during each interval at a local monitor,

$$C_t^i[a, b] = \sum_{f \in \mathcal{F}_{a,b}} p(f)\mathbf{v}_t[f], \tag{4.20}$$

where i denotes the index of the local monitor.

Due to the linear property of the sketch, the NOC can get the total signed sum for each aggregated flow $\mathcal{F}_{a,b}$,

$$C_t^*[a, b] = \sum_i C_t^i[a, b] = \sum_{f \in \mathcal{F}_{a,b}} p(f)\mathbf{v}_t[f]. \tag{4.21}$$

Let $V_t[a]$ denote the total traffic volumes of the aggregated flow at the a-th row,

$$V_t[a] = \sum_{f \in \mathcal{F}_{a,0}} \mathbf{v}_t[f]. \tag{4.22}$$

Similarly, the traffic volume change in the aggregated flow at the a-th row is denoted by

$$\Delta_t[a] = \sum_{f \in \mathcal{F}_{a,0}} |\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]|. \tag{4.23}$$

And we can always bound the change $C_t^*[a, b] - C_{t-1}^*[a, b]$ by $\Delta_t[a]$.

Lemma 7. For any a and b, we have

$$|C_t^*[a, b] - C_{t-1}^*[a, b]| \leq \Delta_t[a]. \tag{4.24}$$

Proof.

$$\begin{aligned} |C_t^*[a, b] - C_{t-1}^*[a, b]| &= |\sum_{f \in \mathcal{F}_{a,b}} p(f)(\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f])| \\ &\leq \sum_{f \in \mathcal{F}_{a,b}} |p(f)| \cdot |\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| \\ &= \sum_{f \in \mathcal{F}_{a,b}} |\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| \\ &= \Delta_t[a]. \end{aligned} \tag{4.25}$$

□

Next, we show that $p(f')C_t^*[a,b]$ is an unbiased estimation of the traffic volume of any flow $f' \in \mathcal{F}_{a,b}$.

Lemma 8. For any flow $f' \in \mathcal{F}_{a,b}$, we have

$$E\left(p(f')C_t^*[a,b]\right) = \mathbf{v}_t[f]. \tag{4.26}$$

Proof. We divide $C_t^*[a,b]$ into two parts,

$$C_t^*[a,b] = p(f')\mathbf{v}_t[f'] + \sum_{f\in\mathcal{F}_{a,b}\setminus\{f'\}} p(f)\mathbf{v}_t[f] \tag{4.27}$$

Because $p(\cdot)$ maps each flow into $+1$ or -1 with an equally probability $1/2$, we have

$$\begin{aligned} & E\left(p(f')C_t^*[a,b]\right) \\ = \ & E\left(p^2(f')\mathbf{v}_t[f'] + \sum_{f\in\mathcal{F}_{a,b}\setminus\{f'\}} p(f')p_a(f)\mathbf{v}_t[f]\right) \\ = \ & E(p^2(f'))\mathbf{v}_t[f'] + \sum_{f\in\mathcal{F}_{a,b}\setminus\{f'\}} E(p(f')p(f))\mathbf{v}_t[f] \\ = \ & 1\cdot\mathbf{v}_t[f'] + \sum_{f\in\mathcal{F}_{a,b}\setminus\{f^*\}} 0\cdot\mathbf{v}_t[f] \\ = \ & \mathbf{v}_t[f']. \end{aligned} \tag{4.28}$$

□

By choosing a proper size of our fast sketch, we can show that the traffic volume change $\Delta_t[a]$ in the aggregated flow at the a-th row is very small.

Lemma 9. By choosing $\ell = \frac{4}{\epsilon}\log\frac{4}{\delta}$, the change of the traffic volume $\Delta_t[a]$ in the aggregated flow at the a-th row is bounded by $\frac{\epsilon}{2}\Delta_t$ with a probability at least $1/2$.

Proof. Because each $h_j(\cdot)$ is an independent and universal hash function, a flow will be hashed by $\log(4/\delta)$ into rows with the equally probability. Thus, the expected change of the traffic volume in each aggregated flow $\mathcal{F}_{a,0}$ should be bounded by

$$E(\Delta_t[a]) \le \frac{\Delta_t \log(4/\delta)}{\ell} = \frac{\epsilon\Delta_t}{4}. \tag{4.29}$$

Based on Markov's inequality, we have

$$\begin{aligned} Prob\left(\Delta_t[a] \geq \frac{\epsilon}{2}\Delta_t\right) &\leq \frac{E(\Delta_t[a])}{\frac{\epsilon}{2}\Delta_t} \\ &\leq \frac{\frac{\epsilon\Delta_t}{4}}{\frac{\epsilon}{2}\Delta_t} \\ &= \frac{1}{2}. \end{aligned} \tag{4.30}$$

□

We can prove the correctness of our fast sketch in the following theorem.

Theorem 9. Given $\ell = \frac{4}{\epsilon}\log\frac{4}{\delta}$, our fast sketch can guarantee that

- Any heavy-change flow f^h with $|\mathbf{v}_t[f^h] - \mathbf{v}_{t-1}[f^h]| \geq (\theta+\epsilon)\Delta_t$ will be reported as anomalous with a probability at least $1-\delta$.
- Any normal flow f^n with $|\mathbf{v}_t[f^n] - \mathbf{v}_{t-1}[f^n]| < (\theta-\epsilon)\Delta_t$ will be reported as anomalous with a probability at most δ.

Proof. Heavy-change Flow f^h:

If there is a heavy-change flow f^h hashed into the a-th row with $|\mathbf{v}_t[f^h] - \mathbf{v}_{t-1}[f^h]| \geq (\theta+\epsilon)\Delta_t$, the change in the first counter can be divided into two parts.

$$\begin{aligned} & |C_t^*[a,0] - C_{t-1}^*[a,0]| \\ = \ & |p(f^h)(\mathbf{v}_t[f^h] - \mathbf{v}_{t-1}[f^h]) \\ & \quad + \sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} p(f)(\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f])| \\ \geq \ & |\mathbf{v}_t[f^h] - \mathbf{v}_{t-1}[f^h]| \\ & \quad -|\sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} p(f)(\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f])| \\ \geq \ & |\mathbf{v}_t[f^h] - \mathbf{v}_{t-1}[f^h]| \\ & \quad - \sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} |\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| \end{aligned} \tag{4.31}$$

We can have the following result similar to Lemma 3

$$E(|\sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} \mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]|) < \frac{\epsilon\Delta_t}{4} \tag{4.32}$$

and

$$Prob\left(|\sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} \mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| > \frac{\epsilon}{2}\Delta_t \right) < \frac{1}{2}. \tag{4.33}$$

And, if $|\sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} \mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| < \frac{\epsilon}{2}\Delta_t$, we must have

$$|C_t^*[a,0] - C_{t-1}^*[a,0]| > \frac{\theta}{2}\Delta_t \tag{4.34}$$

Thus, the test procedure will return true with a probability at least 1/2 in the a-th row.

Because we always have

$$\sum_{f\in\mathcal{F}_{a,b}\backslash\{f^h\}} |\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| \leq \sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} |\mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| \tag{4.35}$$

we can always recovery the quotient of f^h if $|\sum_{f\in\mathcal{F}_{a,0}\backslash\{f^h\}} \mathbf{v}_t[f] - \mathbf{v}_{t-1}[f]| < \frac{\epsilon}{2}\Delta_t$. If we find the quotient, we try all hash functions to further recover this flow key. Therefore, the heavy-change flow f^h can be reported as anomalous at the a-th row with a probability at least 1/2. Because each heavy-change flow is hashed into $\log\frac{4}{\delta}$ rows, the recovery procedure can fail with a probability less than $\delta/2$.

Next, we use the CM sketch to remove the false positives. Based on the analysis of the CM sketch, a heavy-change flow can pass the verification with a probability at least $1-\delta/2$. By considering both the recovery and the verification, a heavy-change flow can be reported as anomalous with a probability at least $(1-\delta/2)^2 > 1-\delta$.

Normal Flow f^n:

If there are only normal flows hashed into the a-th row, the test procedure will return true with a probability at most 1/2 based on Lemma 3. By using the CM sketch, a normal flow will be reported as anomalous with a probability at most δ.

□

The space and the running time of our fast sketch is provided in the following theorem.

Theorem 10. The number of counters can be bounded by $O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\log\frac{\epsilon n}{\log(1/\delta)}\right)$. The running time for update can be bounded by $O\left(\log\frac{1}{\delta}\log\frac{\epsilon n}{\log(1/\delta)}\right)$, and the running time for recovery can be bounded by $O\left(\frac{1}{\epsilon}\log^3\frac{1}{\delta}\log\frac{\epsilon n}{\log(1/\delta)}\right)$.

Proof. There are $\frac{4}{\epsilon}\log(4/\delta)$ rows and $1+\log\frac{\epsilon n}{4\log(4/\delta)}$ counters in each row in our fast sketch. Thus, the space requirement is bounded by $O\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\log\frac{\epsilon n}{\log(1/\delta)}\right)$.

To process each packet, we need to update $1+\log\frac{\epsilon n}{4\log(4/\delta)}$ counters in $\log\frac{4}{\delta}$ rows in our fast sketch. Thus, the running time for updating is bounded by $O\left(\log\frac{1}{\delta}\log\frac{\epsilon n}{\log(1/\delta)}\right)$.

In the identification procedure, we need to check the first counter in each row to determine whether there is a heavy-change flow, which requires $\frac{4}{\epsilon}\log\frac{4}{\delta}$ running time. Because there are at most $\frac{1}{\epsilon}\log(4/\delta)$ rows which can contain heavy-change flows, the running time to find the quotient requires at most $\frac{1}{\epsilon}\log(4/\delta)\log\frac{\epsilon n}{4\log(4/\delta)}$. In order to recover the flow keys, we need to try each hash function once for each quotient, and check the counters in the first counter in each row which it is hashed into. We can use $\log^2\frac{4}{\delta}$ running time to find the heavy-change flow. Thus, it will take $\frac{1}{\epsilon}\log^3(4/\delta)\log\frac{\epsilon n}{4\log(4/\delta)}$ running time to find all heavy-change flows. In sum, the running time for querying is $O\left(\frac{1}{\epsilon}\log^3\frac{1}{\delta}\log\frac{\epsilon n}{\log(1/\delta)}\right)$. □

4.1.6 Conclusion

We provide a fast sketch for the aggregate queries in this section, which can recover the roots of the traffic anomalies with the minimum number of aggregated flows. Our sketch can be used by various traffic anomaly detection approaches. If the traffic anomaly detection is based on other traffic features rather than the traffic volumes, the local monitors only need to replace each counter by another data structure for the estimation of other traffic features. And NOCs only need to use another test function for the traffic anomaly detection. The number of aggregated flows, and the running time of our fast sketch are still be near-optimal for the traffic information aggregation.

In this section, we provide a detailed theoretical analysis of the implementation of our fast sketch for the heavy-change detection in the network traffic. In the future, we will evaluate our fast sketch with different anomaly detection approaches by using real network traffic traces.

4.2 Identifying High-Cardinality Hosts (Super Spreaders) from Network-wide Traffic Measurements

Host cardinality is defined as the number of distinct peers that a host communicates with in the network. There have been several algorithms proposed to monitor network traffic and identify high-cardinality hosts at a centralized network operation center (NOC). Due to massive amounts of data and limitations on transforming and processing them at the NOC, it is desirable to design *mergeable* and *reversible* data structures summarizing traffic measurements in a distributed network monitoring system. A mergeable data structure summarizes traffic measurements at each local monitor, and these summaries from different monitors can be merged at the NOC, while preserving the error guarantee without increasing space. A reversible data structure can report interested (high-cardinality) hosts efficiently using compressed information without querying every single host in the network. In this section, we propose a new data streaming algorithm to identify high-cardinality hosts over the network-wide traffic measurements. Our algorithm introduces a new mergeable and reversible data structure for the distributed network monitoring system, which is designed by Noisy Group Testing. We have theoretically analyzed our algorithm and evaluated it against both synthetic and real-world data sets.

4.2.1 Introduction

Internet Service Providers (ISPs) collect traffic measurements for various purposes like customer accounting and traffic engineering [42], which are also used for traffic anomaly detection, cyber-attack attribution, network forensic analysis, etc. An important traffic feature of interest is the host cardinality [21; 47], defined as the number of distinct peers that a host communicates with. High-cardinality hosts, known as super-spreaders [91], are often the signs of many security problems, e.g., (distributed) denial-of-service attacks, spam emails, worm spreading, botnet takeover, etc. For example, a compromised host doing fast scanning for worm propagation often makes an unusually high number of connections in a short time. There is a significant increase in the numbers of destination IP addresses during the witty worm propagation. This figure was drawn from the trace data including the packets sent from infected hosts by the witty worm, which was collected at the UCSD network telescope [wit]. When a host is infected, it randomly generates destination IP addresses and tries to infect the hosts at these addresses. Thus, high cardinalities are important signs for the malware propagation in the network. Many previous works have verified the effectiveness of the cardinality as a primary feature for the network security [21; 47; 91; 97].

We study the detection problem of high-cardinality hosts with the following goals:

- Firstly, the detection of high-cardinality hosts requires a network-wide traffic view. The attack traffic may enter the network from multiple routers. If we only monitor the cardinality at every single router, the attackers would be missed at any of them. Therefore, we have to merge the traffic measurements from multiple routers, and get a network-wide view of the host cardinalities.

- Secondly, the packets from the same connection must be removed for the cardinality computation. When we merge the traffic measurements from multiple routers, we cannot simply add the cardinality of a host at each router together to calculate its total cardinality. Because each connection may travel multiple routers from the source to the destination, we need to design a method to only count the distinct connections.

- Thirdly, due to the large size of the traffic measurements, ISPs are only able to collect some summaries of the traffic measurements from local routers. We have to design a mergeable data structure, referred as a sketch, to reduce the communication costs.

- Lastly, we cannot compute the cardinality for every single host in order to identify the high-cardinality ones. Due to the large size of the IP addresses, we want to only estimate the cardinality for the high-cardinality hosts with limited space and running time.

The above challenges have only partially addressed in previous work, and there have been no algorithm that can solve all these challenges, to the best of our knowledge. In this section, we propose a new data streaming algorithm to compute a mergeable and reversible sketch, which can be used to identify high-cardinality hosts from network-wide traffic measurements.

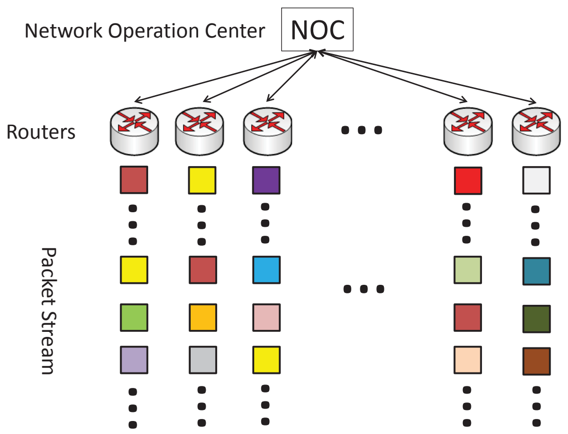


Figure 4.6 Network-wide Traffic Monitoring

Our data structure summarizing traffic measurement is designed based on the noise group testing [84; 24], which can identify high-cardinality hosts efficiently in a distributed network monitoring system. Our main idea is that we consider the identification of the high-cardinality hosts as a channel-coding problem, which also provides a new theoretical analysis method for this problem. Our work aims to provide a new scheme for the distributed network monitoring, and is much more efficient than the state-of-art solution.

4.2.2 Problem Definition

In this section, we focus on the identification of high-cardinality hosts from the network-wide traffic measurements. Assume there are k routers in the network, each of which monitors a stream of packets as shown in Fig.4.6. At the i-th router for $i = 1, \ldots, k$, there is a packet stream, denoted by

$$(s_{i1}, d_{i1}), (s_{i2}, d_{i2}), \ldots, (s_{it}, d_{it}), \ldots \tag{4.36}$$

where t denotes the current time, and $s_{it}, d_{it} \in \mathcal{U}$ denote the source and the destination in a packet, respectively. Let $\mathcal{U}$ denote the set of the source/destination addresses in the network.

Let $\mathcal{A}_{it}$ denote the set of packets observed at the i-th router in the current measurement window with a length τ,

$$\mathcal{A}_{it} = \{(s_{ij}, d_{ij}) \,|\, j \in [t - \tau, t]\}. \tag{4.37}$$

The set of the sources to a host $x \in \mathcal{U}$ is denoted by

$$\mathcal{S}_{it}^x = \{s \,|\, (s,x) \in \mathcal{A}_{it}\} \tag{4.38}$$

and the set of the destinations from this host x is denoted by

$$\mathcal{D}_{it}^x = \{d \,|\, (x,d) \in \mathcal{A}_{it}\}. \tag{4.39}$$

We define the source/destination cardinality of a host x as the number of distinct hosts in the union of the sets $\mathcal{S}_{it}^x/\mathcal{D}_{it}^x$,

$$S_t^x = |\cup_{i=1}^k \mathcal{S}_{it}^x| \tag{4.40}$$

$$D_t^x = |\cup_{i=1}^k \mathcal{D}_{it}^x| \tag{4.41}$$

where S_t^x denote the source cardinality of the host x, and D_t^x denote its destination cardinality.

The high-cardinality hosts are the ones with a large source/destination cardinality that exceeds a given threshold.

Definition 9. Given a threshold θ, a host $x \in \mathcal{U}$ is identified as a high-cardinality host if

$$S_t^x > \theta \text{ or } D_t^x > \theta. \tag{4.42}$$

Due to the resource limitation, we can only detect the high-cardinality hosts with some approximation errors. Let F_t denote the total number of distinct connections,

$$F_t = |\cup_{i=1}^k \mathcal{A}_{it}|. \tag{4.43}$$

It is easy to verify that F_t equals to the sum of the source/destination cardinalities,

$$F_t = \sum_{x \in \mathcal{U}} S_t^x = \sum_{x \in \mathcal{U}} D_t^x \tag{4.44}$$

An (ϵ, δ)-approximation algorithm can provide the following error guarantee.

Definition 10. An (ϵ, δ)-approximation algorithm can report any host $x \in \mathcal{U}$ such that

$$S_t^x > \theta + \epsilon F_t \text{ or } D_t^x > \theta + \epsilon F_t \tag{4.45}$$

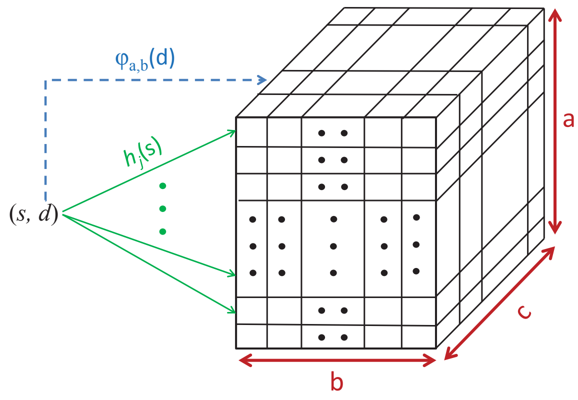


Figure 4.7 Our Sketch

as a high-cardinality host with a probability at least $1 - \delta$, and will report a host $x' \in \mathcal{U}$ such that

$$S_t^{x'} < \theta - \epsilon F_t \text{ or } D_t^{x'} < \theta - \epsilon F_t \tag{4.46}$$

as a high-cardinality host with a probability at most δ.

Given limited memory, engineers can determine the parameters δ and ϵ to balance the detection error and the missing probability. An optimal selection of δ and ϵ is determined by the distribution of the host cardinalities in the network, which is unknown during the traffic monitoring. Usually, we choose ϵ small enough to guarantee that $\theta > \epsilon F_t$.

4.2.3 Our Algorithm

We describe a new data streaming algorithm for the host identification with a high destination cardinality, which is well-known as the super-spreader problem. The problem for a high source cardinality can be solved with the same algorithm by exchanging s and d in our solution. We first introduce the data structure, i.e. the sketch, used to summarize traffic measurements. And then, we describe the update algorithm at each local router, and the merge algorithm at the NOC. At last, we provide the query algorithm for the high-cardinality host identification, which is also the main contribution in this work.

4.2.3.1 Sketch

As shown in Fig.4.7, our sketch summarizing traffic measurements consists of a three-dimensional counters, denoted by $C_{it}[a, b, c]$, where $0 \le a \le \ell - 1$, $0 \le b \le \eta$, and $0 \le c \le \omega$. Let $m = |\mathcal{U}|$ denote the number of the IP addresses in the Internet, which equals to 2^{32} in IPv4, and ℓ denote the number of layers in our sketch. A layer can be denoted by $C_{it}[a, *, *]$, and the number of layers is chosen as

$$\ell = O(\frac{1}{\epsilon} \log \frac{1}{\delta}). \tag{4.47}$$

In this way, we can guarantee that the error of the cardinality estimation due to other hosts can be bounded by ϵ with a high probability $1 - \delta$.

At each layer, there are $\eta + 1$ vectors, denoted by $C_{it}[a, b, *]$. Each host will be mapped multiple vectors in a layer, that is similar as the reversible sketches. All the hosts mapped into the (a, b) vector create the (a, b)-th group. We use Kane's optimal algorithm [58] to estimate the cardinality of the destinations in the (a, b)-th group. Each vector $C_{it}[a, b, *]$ is corresponding to a group in the group testing method. However, the probability that the cardinality estimation algorithm can bound the error by ϵ is very small, even if there is only one host in this group. Let p denote the probability that the cardinality estimation algorithm succeeds in bounding the error by ϵ. The cardinality estimation algorithm can only guarantee that $p \ge 2/3$, using $O(\frac{1}{\epsilon^2} + \log m)$ bits [58]. Therefore, we must work on a noisy version of the group testing problem, in which, the test result for each group can be wrong with some probabilities.

In our algorithm, we use an error-correcting code to map each host into a group, and identify high-cardinality hosts considering the errors from the cardinality estimation algorithm. Considering the test result from each group as a random variable, the entropy of a random variable with Bernoulli distribution p is denoted by

$$H(p) = -p \log p - (1 - p) \log(1 - p). \tag{4.48}$$

We can show that if we choose

$$\eta = O\left(\frac{1}{1 - H(p)} \log \frac{m}{\ell}\right) \tag{4.49}$$

as the codeword length, we can identify high-cardinality hosts in the noisy group testing problem.

Each vector is an implementation of Kane's optimal algorithm for the cardinality estimation in [58]. Each counter maintains the least significant bit of a nonnegative hash value of the destination. The details about the cardinality estimation are beyond the scope of this section. We assume that the Kane's optimal algorithm in [58] can report the cardinality in each group with the error bound ϵ and the success probability p. The size of each vector in our data structure can be chosen as

$$\omega = O(\frac{1}{\epsilon^2}). \tag{4.50}$$

Besides this three dimensional data structure, we also maintain a independent data structure to estimate the total number of flows F_t in the network. We can use Kane's optimal algorithm for this purpose, by using $O(\frac{1}{\epsilon^2})$ space.

4.2.3.2 Update

When a packet $(s_{it'}, d_{it'})$ with $t - \tau < t' < t$ comes at the i-th router, we will update our sketch by using a similar procedure in [67]. We first use a set of κ hash functions to find a set of layers for the source $s_{it'}$,

$$h_j(s_{it'}) = (s_{it'} \bmod \ell) \oplus h'_j(\lfloor s_{it'}/\ell \rfloor) \tag{4.51}$$

where $h'_j(\cdot)$ is an universal hash function that maps $s_{it'}/\ell$ into $[0, \ell - 1]$, and $j = 1, \ldots, \kappa$. The value of ℓ should be chosen as a power of 2 in order to guarantee that $h_j(s_{it'})$ is in the range $[0, \ell - 1]$.

For each layer that $s_{it'}$ is hashed into, we find the quotient of the source $s_{it'}$ as

$$q_j(s_{it'}) = \lfloor s_{it'}/\ell \rfloor. \tag{4.52}$$

We will always add the destination $d_{it'}$ into the first vector in each layer as

$$\begin{aligned} & C_{it}[h_j(s_{it'}), 0, \varphi_{h_j(s_{it'}),0}(d_{it'})] \\ & = max\{C_{it}[h_j(s_{it'}), 0, \varphi_{h_j(s_{it'}),0}(d_{it'})], lsb(\psi(d_{it'}))\}, \end{aligned} \tag{4.53}$$

where $\psi(\cdot)$ is an universal hash function that maps $d_{it'}$ into $[0, m-1]$, and $lst(\cdot)$ denotes the least significant bit. The first vector will be used to filter false positives in the query procedure. We only show a simplified version of the updating algorithm for the cardinality estimation. A detailed updating algorithm for each vector $C_t[a, b, *]$ can be found in [58].

For the following vectors, we will use an error-correcting code to encode its quotient $q_j(s_{it'})$ into a codeword $W(q_j(s_{it'}))$. And then, we add the destination $d_{it'}$ into the vectors only if the b-th bit is 1 in its codeword $W(q_j(s_{it'}))$,

$$\begin{aligned}&C_{it}[h_j(s_{it'}), b, \varphi_{h_j(s_{it'}),b}(d_{it'})]\\&= max\{C_{it}[h_j(s_{it'}), b, \varphi_{h_j(s_{it'}),b}(d_{it'})], lsb(\psi(d_{it'}))\}.\end{aligned} \tag{4.54}$$

In this way, we can identify the high-cardinality hosts in the noisy case by considering the errors in the cardinality estimation algorithm. There are many ways to choose the error-correcting code in our algorithm. A simple way is the repetition code. We simply run the cardinality estimation algorithm several times for each bit in the quotient $q_j(s)$ and use the majority to decode $q_j(s)$. A similar idea was used in [47], where a large Bloom filter is used to bound the error in the cardinality estimation. In our evaluation, we show that we can choose an efficient error-correcting code to reduce the space and the running time, significantly.

At each local router, we also add $(s_{i'}, d_{it'})$ to the data structure for the estimation of the number of flows, i.e., F_t.

4.2.3.3 Merge

At the end of each measurement interval, the routers will send their sketches $C_{it}[*, *, *]$ to the NOC. And the NOC will merge all sketches together to get a network-wide view of the network traffic. Because each counter maintains the least significant bit of a nonnegative hash value of the destination, we need to calculate the same value for the network-wide measurements. Thus, we use the MAX operation to merge all counters together in these sketches. Let $C_t[*, *, *]$ denote the merged sketch at the NOC. We set each counter $C_t[a, b, c]$ to the maximum value

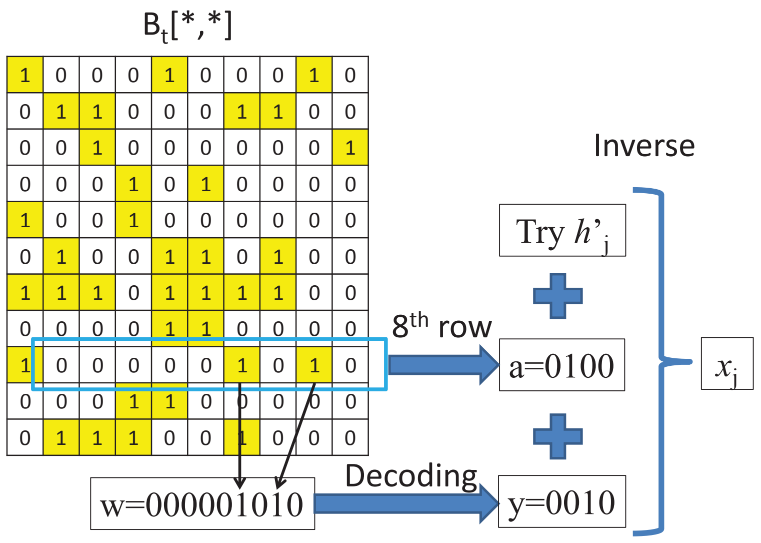


Figure 4.8 High-cardinality Host Recovery

among all counters $C_{it}[a, b, c]$ at the same position,

$$C_t[a, b, c] = \max_i \{C_{it}[a, b, c]\}. \tag{4.55}$$

The data structure for the estimation of the number of flows can be merged in the same way.

4.2.4 Query

Given a threshold θ, our goal is to identify all the hosts x with $D_t^x > \theta$ but allow a small error ϵF_t, which have $D_t^x > \theta + \epsilon F_t$. We divide the whole procedure into four basic steps.

1. *Threshold Computation:* We need to estimate the number of the flows F_t, and add the term $\frac{\epsilon}{2} F_t$ to the threshold in order to bound the false positive errors. We use the merged data structure for the F_t estimation, and run Kane's algorithm to estimate the number of flows in this interval. The estimated number of flows is denoted by $\hat{F}_t$.

2. *Cardinality Test:* We scan each vector $C_t[a, b, *]$ one-by-one, and test whether a high-cardinality host is mapped into this vector. We create an $\ell \times \eta$ binary matrix $B_t[*, *]$ to record the scanning results. For each vector $C_t[a, b, *]$, we use Kane's algorithm to estimate the number of destinations in the (a, b)-th group. Let $\hat{F}_t[a, b]$ denote the estimated cardinality from $C_t[a, b, *]$. If

$$\hat{F}_t[a, b] > \theta + \frac{\epsilon}{2} \hat{F}_t, \tag{4.56}$$

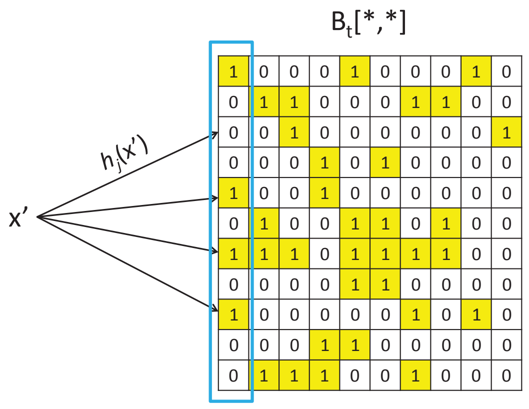


Figure 4.9 False Positive Filter

we mark the bit at $B_t[a,b]$ to 1, and otherwise 0. If there is a high-cardinality host x mapped into this layer $C_t[a,*,*]$, there should be some groups with a high cardinality. In other words, if the b-th bit in the codeword $W(q_j(x))$ of the high-cardinality host x is 1, all of its destinations should be added into this group according to our update algorithm. Therefore, we should get $B_t[a,b]$ as 1 with a probability at least p.

3. *High-cardinality Host Recovery:* Next, we try to recover the high-cardinality host x from the binary matrix $B_t[*,*]$, if there exists some. The bits from $B_t[a,1]$ to $B_t[a,\eta]$ should consist of a binary code that is close to the codeword $W(q_j(x))$ of the high-cardinality host x. Due to the randomness of Kane's algorithm, there are some errors in $B_t[a,1\cdots\eta]$. We apply a decoding algorithm of the error-correcting code, and recover an original message, denoted by y. By choosing a large enough codeword, we can guarantee that the message y should be the same as the quotient $q_j(x)$ with a high probability. Given $y = q_j(x)$, we still do not know which hash function is used to map x into this layer. Thus, we try each hash function one by one, and add all possible high-cardinality candidates to a candidate set $\mathcal{X}$. For each hash function $h_j(\cdot)$, we can recover a candidate host x_j from the message y as

$$x_j = \varphi_j(a,b,y) = y \times \ell + a \oplus h'_j(y). \tag{4.57}$$

If we have $y = q_j(x)$ for the high-cardinality host x, we will have $a = h_j(x)$ and

$$\begin{aligned} x_j &= q_j(x) \cdot \ell + h_j(x) \oplus h'_j(x/\ell) \\ &= (x/\ell) \cdot \ell + (x \bmod \ell) \oplus h'_j(x/\ell) \oplus h'_j(x/\ell) \\ &= (x/\ell) \cdot \ell + (x \bmod \ell) \\ &= x. \end{aligned} \tag{4.58}$$

Therefore, the high-cardinality host x must be added into the set $\mathcal{X}$, if the decoding algorithm returns the correct message y. We add all high-cardinality candidates at each layer into the set $\mathcal{X}$. An example is shown in Fig.4.8.

4. *False Positive Filter:* At the last step, we try to remove false positives in the set $\mathcal{X}$ by using the first column in $B_t[*, *]$. For each candidate $x \in \mathcal{X}$, we use the same set of hash functions $h_j(\cdot)$ to map them into the bits in the first column $B_t[*, 0]$, as shown in Fig.4.9. If x is a high-cardinality host, there should be more than half of the bits in its hash positions that are 1s with a high probability. Therefore, if the number of 1 bits is less than half of the number of hash functions, we will remove x from $\mathcal{X}$. Our algorithm will report the final result $\mathcal{X}$ as the set of the high-cardinality hosts.

Our algorithm follows a general procedure in the group testing problem, which has been widely used in the coding theory. In the following theoretical analysis section, we will analyze our algorithm as a channel-coding problem, which provides the proof for the error bound. In addition, we prove that the running time to identify high-cardinality hosts in our algorithm is sub-linear, which guarantees that our algorithm is practical for the high-speed network monitoring.

4.2.5 Theoretical Analysis

In the previous section, we provide a description of our data structure and the algorithms to update it and further identify the high-cardinality hosts from it. There are many parameters in our data structure, which have not been determined exactly. In this section, we provide a detailed theoretical analysis to help network engineers to choose proper parameters based on their requirements. The main contribution of our work is that we can identify high-cardinality hosts *efficiently* in a distributed network monitoring system. Here, the efficiency means that both the space and the running time can be bounded by a polynomial function of the bit length of the host identification, i.e. $poly(\log m)$.

4.2.5.1 Cardinality Estimation

A fundamental component in our algorithm is Kane's optimal algorithm for the cardinality estimation. Based on their work, we have the following result.

Lemma 10. There is an algorithm for $(1 \pm \epsilon)$-approximating the cardinality in a data stream using space $O(\frac{1}{\epsilon^2}+\log m)$ bits, with 2/3 success probability, and with $O(1)$ update and reporting times. [58]

An important requirement for the cardinality estimation is the mergeable property. We briefly describe the main techniques in Kane's algorithm and then show that their data structure is mergeable. The first technique is the balls-and-bins model. We have α different balls and throw them into β bins. Then, the number of non-empty bins γ is a random variable that has the expectation $E[\gamma] = \beta(1 - (1 - \frac{1}{\beta})^{\alpha})$. When α's value is less than β, the variance of γ can be bounded and the observation value of γ is highly concentrated around $E[\gamma]$ with a high probability. This is also the idea to use Bloom filter [17] to estimate the cardinality in [47]. A Bloom filter is a bit vector, and each bit indicates whether this bin is hit by some balls. Then, γ is the number of 1s in the Bloom filter. It is well known that Bloom filter is mergeable. Supposing we have two Bloom filters from two sets of balls, we can use OR operation to merge two Bloom filters into one, and use the number of 1's in the merged Bloom filter to estimate the number of distinct balls in the union of two sets.

The second technique is sampling. Since it requires that the number of distinct balls thrown into the bins is smaller than the number of bins in order to bound the estimation error, we need to determine a proper sampling rate to reduce the number of balls thrown into the bins. We can use a pair-wise independent hash function to map the same balls into the same hash value. We replace each bit in the Bloom filter by a counter to record the least significant bit, denoted by lsb, in the hash values of balls thrown into each bin. If the lst in a bin is r, this bin will be hit with a sampling rate at least $\frac{1}{2^r}$. After throwing all balls, we can determine a rough estimation of the number of distinct balls, and then choose a proper sampling rate $\frac{1}{2^{r'}}$ to compute γ that is the number of bins with the lst at least r'. If we have two Count Bloom filters recording the lsts from two sets of balls, we can use the MAX operation to merge them together. Each counter records the maximum of the lst of the balls thrown into each bin. In this way, we can estimate the number of distinct balls as the same as the Bloom filter.

Based on these observations, we have the following result.

Lemma 11. There is a mergeable data structure for $(1 \pm \epsilon)$-approximating the cardinality in multiple data streams using space $O(\frac{1}{\epsilon^2} + \log m)$ bits, with $2/3$ success probability, and with $O(1)$ update and $O(\frac{1}{\epsilon^2})$ reporting times.

Proof. The correctness of the mergeable data structure can be proved by the above analysis about the balls-and-bins model and the sampling technique. The space and the update time can be got from Kane's result. Because we need to merge different data structures to report the cardinality for multiple data streams, the running time for merging and reporting is $O(\frac{1}{\epsilon^2})$, which is bounded by the number of counters. □

4.2.5.2 High-cardinality Host Recovery

In this part, we analyze the running time to identify the high-cardinality hosts from the bit matrix $B_t[*, *]$. Firstly, we need to determine the number of hash functions and the number of layers in $C_t[*, *, *]$.

Lemma 12. Given the number of hash functions $\kappa = \log(\frac{6}{\delta})$ and the number of layers $\ell = \frac{4}{\epsilon}\log(\frac{6}{\delta})$, there is a layer a_x for any high-cardinality host x with $D_t^x > \theta + \epsilon F_t$, such that only x is hashed into this layer and no other such hosts are hashed into this layer with a probability at least $1 - \frac{\delta}{3}$.

Proof. Each host is hashed by $\log \frac{3}{\delta}$ hash functions into $\frac{4}{\epsilon} \log \frac{3}{\delta}$ layers. For a given item x and the j^{th} hash function, the sum of the destination cardinalities of the other hosts hashed to the same layer $h_j(x)$ can be bounded by

$$\frac{\epsilon F_t \log(6/\delta) - D_t^x}{4\log(6/\delta)} < \frac{\epsilon F_t}{4}. \tag{4.59}$$

Based on the Markov inequity, the probability is less that 1/2 that the sum of the cardinalities of the other hosts hashed into the layer $h_a(x)$ is larger than $\frac{\epsilon}{2}F_t$. Therefore, the $h_j(x)$-th layer contains only one high-cardinality host x with a probability at least 1/2. Considering all $\log(\frac{3}{\delta})$ hash functions, there is a layer a_x such that only one high-cardinality host x is hashed into this layer and no other such hosts are hashed into this layer with a probability at least $1 - \frac{\delta}{3}$. □

Next, we consider the layer a_x for each high-cardinality host x. We show that we can add x into the candidate set $\mathcal{X}$ at the layer a_x with a high probability.

Lemma 13. If there is only a high-cardinality host x hashed into the a_x-th layer, our query algorithm will add x into $\mathcal{X}$ with a probability at least $1 - \frac{\delta}{3}$, given the length of the codeword $\eta > \frac{1}{1-H(2/3)}\log(\frac{m}{\ell})$.

Proof. For the high-cardinality x, we suppose it is hashed by the j-th hash function into the a_x-th layer. We will encode its quotient $q_j(x)$ into a codeword $W(q_j(x))$ with a bit length η. At the NOC, we will run the cardinality estimation algorithm and get the bit matrix $B_t[*,*]$. If the b-th bit in $W(q_j(x))$ is 1, we will get 1 at $B_t[a_x, b]$ with a probability at least $p = 2/3$. Otherwise, if its b-th bit is 0, we will get 0 with a probability at least $p = 2/3$. Therefore, we can treat $B_t[a_x, b]$ as the received symbol through a channel as shown in Fig.4.10. Based on the information theory, we can find a way to encode $q_j(x)$ and recover it from $B_t[a_x, *]$ with a high probability. Due to the progress in the list-decoding method [49], we can encode $q_j(x)$ with a rate close to $1 - H(2/3)$ that is the capacity of the channel in Fig.4.10, and recover $q_j(x)$ in $O(poly(\log(\frac{m}{\ell})))$ running time with a high probability.

Once we find the quotient $q_j(x)$ of the high-cardinality host x, we can try all hash functions and further recover this host x. Because each host is hashed into $\log\frac{6}{\delta}$ layers, the recovery procedure can fail with a probability less than $\delta/3$. □

The size of the candidate hosts $|\mathcal{X}|$ can be bounded by the following lemma.

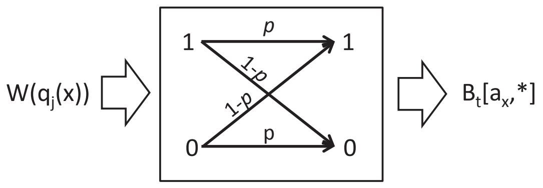


Figure 4.10 A Channel Model for Cardinality Estimation

Lemma 14. The number of candidate hosts in our algorithm is bounded by $\frac{1}{\epsilon}\log^2(\frac{3}{\delta})$ with a probability at least $1-\delta$.

Proof. There are at most $\frac{1}{\epsilon}$ hosts that can have a cardinality more than ϵF_t. Each high-cardinality host is hashed into $\log(\frac{3}{\delta})$ layers, and there are at most $\log(\frac{3}{\delta})$ candidate hosts returned in each such layer. If there is no high-cardinality host hashed into a layer a, the decoding algorithm will fail with a high probability. In this case, there will be no candidate returned in this layer. Therefore, the number of candidate hosts can be bounded by $\frac{1}{\epsilon}\log^2(\frac{3}{\delta})$. □

4.2.5.3 False Positive Filter

After the high-cardinality host recovery, there are many false positives in the candidate set $\mathcal{X}$. We use a similar method as the Count-Min sketch [30] to remove false positives.

Lemma 15. For a high-cardinality host $x \in \mathcal{X}$ with $D_t^x > \theta + \epsilon F_t$, our algorithm will report it as high cardinality with a probability at least $1-\delta/3$, and for a low-cardinality host $x' \in \mathcal{X}$ with $D_t^{x'} < \theta - \epsilon F_t$, our algorithm will report it as high cardinality with a probability at most $\delta/3$.

Proof. With the same analysis in the proof of Lemma 3, the bit $B_t[a,0]$ is 1 with a probability at least 1/2, if there is a high-cardinality host hashed into the a-th layer. Therefore, the probability that the number of 1s in the hash positions of a high-cardinality host is more than $\frac{1}{2}\log(\frac{6}{\delta})$ is at least $1-\delta/3$. Thus, a high-cardinality host x will pass the filter with a probability at least $1-\delta/3$.

Similarly, the bit $B_t[a,0]$ is 1 with a probability at most 1/2, if there is no high-cardinality host hashed into this layer. Thus, a low-cardinality host x' can pass the filter with a probability at most $\delta/3$. $\square$

4.2.5.4 Final Results

Based on the above lemmas, we can provide the final results about our algorithm. We first prove the correctness of our algorithm.

Theorem 11. A high-cardinality host x with $D_t^x > \theta + \epsilon F_t$ can be reported with a probability at least $1-\delta$, and a low-cardinality host x' with $D_t^{x'} < \theta - \epsilon F_t$ will be reported with a probability at most δ.

Proof. There are three events that can cause x missed in the final result.

1. There is no layer such that only x is hashed into this layer and no other high-cardinality hosts.
2. The decoding process of the quotient of x fails in recovering its quotient.
3. The last step falsely remove x from the candidate set $\mathcal{X}$.

Based on the above analysis, each event can only happen with a probability less than $\delta/3$. By the union bound, our algorithm will report a high-cardinality host x with a probability at least $1-\delta$.

For a low-cardinality host, the false positive filter will remove it from $\mathcal{X}$ with a probability at least $1-\delta/3$. Therefore, our algorithm will falsely report a low-cardinality host x' with a probability at most δ. $\square$

Next, we analyze the space and the running time in our algorithm.

Theorem 12. Our data structure only requires

$$O(\frac{1}{\epsilon}(\frac{1}{\epsilon} + \log m)\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)})) \tag{4.60}$$

bits.

Proof. There are $O(\frac{1}{\epsilon})\log(\frac{1}{\delta}))$ layers in our sketch. In each layer, there are $O(\log(\frac{\epsilon m)}{\log(1/\delta)})$ vectors. And each vector only requires $O(\frac{1}{\epsilon} + \log m)$ bits. By multiplying them tougher, we prove our result. Other parts in our algorithm will require a space much less than the sketch $C_{it}[*,*,*]$. □

Because the cardinality estimation can be done in $O(1)$ running time for both update and query, the performance of our algorithm is mainly determined by the identification of high-cardinality hosts at the NOC.

Theorem 13. The update running time at each monitor is

$$O(\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)})). \tag{4.61}$$

The merge running time and the query running time at the NOC are

$$O(\frac{1}{\epsilon^3}\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)})) \tag{4.62}$$

and

$$O(\frac{1}{\epsilon}\log(\frac{1}{\delta})\log^{O(1)}(\frac{\epsilon m}{\log(1/\delta)})+\frac{1}{\epsilon}\log^3(\frac{1}{\delta})) \tag{4.63}$$

respectively.

Proof. At each monitor, we use $O(\log(\frac{1}{\delta}))$ hash functions to map each host to layers. For each layer, we need to update at most $O(\log(\frac{\epsilon m}{\log(1/\delta)}))$ vectors. But each vector only requires $O(1)$ running time. Therefore, the update running time at a local monitor is $O(\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)}))$.

In each vector, there are $O(\frac{1}{\epsilon^2})$ counters. And there are total $O(\frac{1}{\epsilon}\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)}))$ vectors in our sketch. Therefore, the running time for merging sketches in our algorithm is $O(\frac{1}{\epsilon^3}\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)}))$.

To compute the binary matrix $B_t[*,*]$, we only need $O(\frac{1}{\epsilon}\log(\frac{1}{\delta})\log(\frac{\epsilon m}{\log(1/\delta)}))$ running time. To recover a high-cardinality host in each layer, we need $O(\log^{O(1)}(\frac{\epsilon m}{\log(1/\delta)}))$ running time for the list-decoding algorithm [49], and $O(\log(\frac{1}{\delta}))$ running time to map quotient back to the host identification. Thus, we need $O(\frac{1}{\epsilon}\log(\frac{1}{\delta})(\log^{O(1)}(\frac{\epsilon m}{\log(1/\delta)})+\log(\frac{1}{\delta})))$ running time to get the candidate set $\mathcal{X}$.

Because there are at most $O(\frac{1}{\epsilon}\log^2(\frac{1}{\delta}))$ candidates in $\mathcal{X}$, the algorithm to filter false positives will require $O(\frac{1}{\epsilon}\log^3(\frac{1}{\delta}))$ running time. Therefore, the running time for the query can be bounded by $O(\frac{1}{\epsilon}\log(\frac{1}{\delta})\log^{O(1)}(\frac{\epsilon m}{\log(1/\delta)})+\frac{1}{\epsilon}\log^3(\frac{1}{\delta}))$. □

Based on the above results, we can identify high-cardinality hosts with a running time bounded by a polynomial function of the bit length of the host identification, i.e. $O(poly(\log m))$. And the space requirement of our algorithm is close to the lower bound of the compress sensing [35].

There is a close relation between our reversible data structure and the compressed sensing problem [20; 36; ric]. For a signal $\mathbf{f}$, its compressed measurement equals to $\mathbf{Mf}$, where $\mathbf{M}$ is a carefully chosen $l \times m$ matrix with $l \ll m$. The vector $\mathbf{Mf}$ is called as the measurement vector, and can be used recover $\mathbf{f}$ beyond the information bound. The high-cardinality host identification introduces a new challenge for the compressed sensing. Besides the noises in the original signal $\mathbf{f}$, there are also noises in the measurement vector $\mathbf{Mf}$. In other words, even if there is no noise in $\mathbf{f}$, we still have some errors in the measurement vector $\mathbf{Mf}$. This is because there is no efficient data structure that can be used to estimate the cardinality without the approximation or the randomness. We show that the traditional methods using the error-correcting code can be used to recover a sparse signal under the noises in the measurement vector $\mathbf{Mf}$.

4.2.6 Evaluation

We use false positive rate and false negative rate to evaluate our algorithm. False positives are those source IP addresses which are recovered by our algorithm but not actually super-spreaders. False negatives are those source IP addresses which are real super-spreaders but not recovered by our algorithm.

4.2.6.1 Synthetic Data Evaluation

Each synthetic data set contains hundreds of thousands of lines and each line contains a source and a destination. We want to recover those sources with large number of distinct destinations (high connection degrees). Figure 4.11 shows a part of our experimental results. For this data set, we randomly select 100 sources as superspreaders and 1000 sources as non-superspreaders. We set the universe as 2^{16} both for the sources and destinations. For the superspreaders, we randomly select a connection degree which is larger than half of the universe

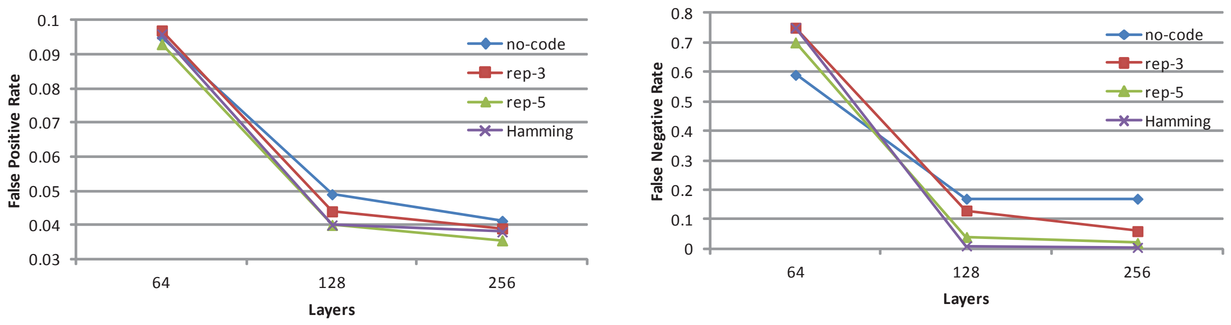


Figure 4.11 Performance naive algorithm v.s. repetition code

for each of them. And for the non-superspreaders, we randomly assign a connection degree which is under 50 for each. All the destinations for each source are generated randomly. And for each data set, we run our algorithm 50 times and calculate the average evaluation result. Some parameters are $\epsilon = 0.01$, hash function number $k = 3$. We denote the three versions as no-code, rep-3 and Hamming. No-code means for each bit of the quotient, we only keep one instance of cardinality estimator for it; rep-3 means that for each bit of quotient, we keep 3 instances; Hamming means that we encode the quotient with Hamming code and for each bit of the code word we keep an instance of the cardinality estimator.

In figure 4.11, we can see the difference when using same layer number but different layer structures. The line with legend $= 1$ is the evaluation results for the naive algorithm where in each layer for each bit of the quotient we only use one instance of the optimal cardinality estimation algorithm. And the lines with legends $= 3, 5$ are the results of using repetition code where 3 or 5 instances are used for each bit of the quotient. We also tested the accuracy when using different number of layers. We can see from the figure that using repetition code can reduce both the false positive rate and the false negative rate. And the more layers we use, the higher accuracy we can get.

We also compare the accuracies of three different versions of our algorithm under the same space condition. Figure 4.12 shows the evaluation results when we use same space for each of the three versions. The x-axis shows the space scale we use: 9 means we use totally 2^9 instances of the cardinality estimator for the whole algorithm and so on. We can see from figure 4.12 that the Hamming version is performing better than the no-code version since both

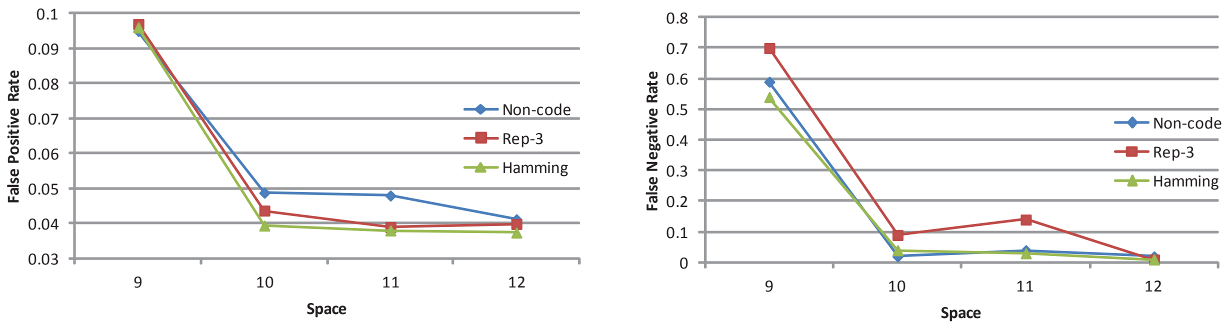


Figure 4.12 Accuracy of three different algorithm versions under same space condition

Table 4.2 Superspreader detection in witty worm trace

ϵ	FalsePositive	FalseNegative
0.01	0.165	0.55
0.02	0.34	0.85
0.03	0.67	0.93

the false positive rate and the false negative rate of the previous one is lower than the later one. The rep-3 version's performance is between the above two basically because under same space condition, it has to use much smaller layer number than the other two, which significant reduces its accuracy.

4.2.6.2 Real Trace Evaluation

We also tested our algorithm of version rep-3 on the real trace data from [wit]. We divided the trace into time slots of 5 minutes and try to detect superspreader on each slot. Table 4.2 shows the results under different ϵ value. We can see that the higher ϵ is, the more accuracy we can get.

4.2.7 Conclusion

In sum, we provide a mergeable and reversible data structure for the high-cardinality host detection. Our solution is a general solution that can be easily extended to other traffic features or different application domains. We also provide a detailed theoretical analysis of our data structure, which can help engineers to determine the parameters in practice. The high-cardinality host identification also introduces a new challenge for the compressed sensing problem, which will attract more research interests in the future. We hope our work can improve the practice of the traffic monitoring and the anomaly detection in the Internet.

CHAPTER 5. LOW-RANK MATRIX APPROXIMATION

5.1 Sketch-based Network-wide Coordinated Traffic Anomaly Detection

Internet has become an essential part of the daily life for billions of users worldwide, who are using a large variety of network services and applications everyday. However, there have been serious security problems and network failures that are hard to resolve, for example, botnet attacks, polymorphic worm/virus spreading, DDoS, and flash crowds. To address many of these problems, we need to have a network-wide view of the traffic dynamics, and more importantly, be able to detect traffic anomalies in a timely manner. Spatial analysis methods have been proved to be effective in detecting network-wide traffic anomalies that are not detectable at a single monitor. To our knowledge, Principle Component Analysis (PCA) is the best-known spatial detection method for the coordinated low-profile traffic anomalies. However, existing PCA-based solutions have scalability problems in that they require linear running time and space to analyze the traffic measurements in a centralized Network Operation Center (NOC), which makes it often infeasible to be deployed for monitoring large-scale high-speed networks. We propose a sketch-based streaming PCA algorithm for the network-wide traffic anomaly detection in a distributed fashion. Our algorithm only requires logarithmic running time and space at both local monitors and Network Operation Centers (NOCs), and can detect both high-profile and coordinated low-profile traffic anomalies with bounded errors.

5.1.1 Introduction

Internet has become an essential part of the daily life for billions of users worldwide. People are using and relying on a large variety of services built on the top of the Internet, such as web browsing, online banking, shopping, entertainment, VoIP, Video on demand, auction, social networks, etc. However, everyday we are still reading news stories about major security breaches, new polymorphic worm/virus spreading, identity theft, botnet activity, DDoS or phishing emails. To address many of these problems (e.g. DDoS, botnet, worm/virus, etc.), we need to have a network-wide view of the traffic dynamics, and more importantly, be able to detect traffic anomalies in a timely manner. Otherwise, the failure of doing so may cause catastrophic damages or unwanted results with impacts affecting online business, public safety, homeland security, personal privacy, the economy and the society at large.

Traffic anomalies can occur due to a variety of problems. Firstly, security threats like DDoS, worms, and botnets, can generate extremely large-volume anomalous traffic. Secondly, unusual events can cause traffic anomalies, like equipment failures, vendor implementation errors, and software bugs. Thirdly, abnormal user behaviors can change the traffic patterns, for example, flash crowds, non-malicious large file transfers, etc. In the early days, traffic anomalies often involve unusual large-volume traffic, i.e. high-profile traffic, which are mainly caused by traditional DoS, worm, or flash crowds. In recent years, new threats like botnets introduce low-profile but in a coordinated manner, which only generate a small amount of traffic but follow specific coordinated traffic patterns. Besides these, there are also some traffic anomalies that are low-profile and non-coordinated, e.g. Black mails and spam voice IP calls.

For the purpose of addressing problems like intrusion detection, fault detection and recovery, and QoS provision, many ISPs have chosen to use a distributed architecture for the network monitoring. In this framework, local monitors collect data from routers and other network devices, perform some processing at or close to the data sources, and transfer their data to the NOCs. Then, NOCs are responsible for mining characteristics of interest from collected data, and identifying the problems and the roots thereof. Many of such measurements from these systems are also the data sources for the traffic anomaly detection. Monitoring and detecting network-wide traffic anomalies have been and are still challenging for the following reasons. Firstly, the Internet traffic exhibits huge fluctuations and long range dependence, which makes traffic anomalies often be hidden by large volumes of normal traffic. Secondly, traffic anomalies show an extreme diversity and new varieties of traffic anomalies are emerging everyday. Thirdly, ISPs want to detect traffic anomalies when they are still at a low-profile volume in order to reduce the damage as much and early as possible. Last but not the least, there are many systems where data, computing, and other resources are distributed and cannot be transported to a center for various reasons, e.g. low bandwidth, security, privacy, and load balancing issues.

Due to the above challenges, the spatial analysis method like PCA [62] has been introduced for the traffic anomaly detection and verified to be effective for the coordinated low-profile traffic anomalies. But there are several challenges for applying PCA in practice. Currently, there is no well-known method to determine the parameters in the PCA-based detection methods. Furthermore, large-volume traffic anomalies and the stealthy poisoning attacks can contaminate normal traffic patterns. Last, PCA requires a singular value decomposition (SVD) of a $n \times m$ matrix with $n \gg m$, where n is the length of the measurements and m is the number of links or aggregated flows. The computation complexity of SVD is $O(nm^2)$ and the space requirement is $O(nm)$, which would become a bottleneck to perform PCA in the high-speed network.

In this paper, we focus on the last challenge, i.e. the performance of the PCA-based detection method, and propose a novel sketch-based streaming algorithm, which can significantly reduce the computation and storage overhead. Because large-volume traffic anomalies can contaminate the normal traffic patterns, NOCs should use as many data as possible to train the traffic anomaly detector. By updating the traffic anomaly detector frequently, NOCs can detect the stealthy poisoning attacks with a high probability. Therefore, efficient algorithms for the PCA computation are very useful for the NOCs to detect network-wide traffic anomalies. Our main contributions are summarized as follows:

1. Our algorithm is efficient in both space and running time, which can achieve $O(w \log n)$ running time and $O(w \log^2 n)$ space at local monitors, and $O(m^2 \log n)$ running time and $O(m \log n)$ space at the NOCs, where w is the number of aggregated flows at local monitors and m is the total number of flows at the NOC.

2. We also provide theoretical guarantees on the performance of our algorithm.

3. Our algorithm is flexible for ISPs to balance the computation and the storage in a distributed measurement and monitoring system.

4. Our algorithm can detect both high-profile and coordinated low-profile traffic anomalies as an outlier in the regular traffic patterns like the PCA-based methods.

5. Experimental results show that our algorithm can use very small sketches to detect traffic anomalies for all possible parameters. And the computation overhead is much less than the existing method.

5.1.2 Problem Definition

5.1.2.1 Background

Internet is a global system of interconnected computer networks, which can provide data interchanging by using the standardized Internet Protocol Suite (TCP/IP). The computer networks are organized into several autonomous systems (AS), each of which is independently operated by an Internet Service Provider (ISP). The success of the Internet mainly owes to the *end-to-end* principle, which results in a simple network infrastructure. All the data transported by the Internet are divided into IP packets, and each packet is forwarded hop-by-hop by routers. There are a source address and a destination address in each packet's header, which are used by routers to determine the forwarding path from the source to the destination. The communication between two computers is controlled by a transmission protocol like TCP, which creates an individual *end-to-end* flow.

Due to the exponential increase in terms of the number of users and applications, it has become not feasible to maintain statistics for each individual *end-to-end* flow if not impossible. Thus, ISPs often aggregate *end-to-end* flows at different levels, such as origin autonomous systems, ingress links, applications, etc. For example, ISPs can use the origin-destination (OD) flow, defined as all packets that enter the network at one origin router and exits at another destination router.

5.1.2.2 Principle Component Analysis

Lakihina et al. [62] applied PCA on the aggregated flows to build a statistical model for the normal traffic, and further detected traffic anomalies. All traffic measurements from m

aggregated flows within the sliding window of the length n are organized into a $n \times m$ matrix $\mathbf{X}$. Let x_{ij} denote the traffic measurement of the j-th flow at the i-th time interval, which can be the traffic volume, the entropy of IP addresses, the frequency of the byte values in the payload, and so forth. The matrix $\mathbf{X}$ is adjusted into a matrix $\mathbf{Y}$ with zero column mean,

$$y_{ij} = x_{ij} - \bar{x}_{tj} \tag{5.1}$$

where $\bar{x}_{tj} = \sum_{i=t-n+1}^{t} x_{ij}/n$ and t is the current interval.

PCA is applied on $\mathbf{Y}$ and treats each row as a point in an m-dimensional space and each column as a variable. PCA performs a coordinate rotation that aligns the transformed axes with the directions that make the projections of the row vectors on each axis get as large variance as possible. Principal components are the unit vectors along these axes. The first principal component of the matrix $\mathbf{Y}$, denoted by $\mathbf{v}_1$, can be found as,

$$\mathbf{v}_1 = \arg \max_{\|\mathbf{x}\|=1} \|\mathbf{Y}\mathbf{x}\| \tag{5.2}$$

where *arg max* stands for the vector $\mathbf{x} = (x_1, \dots, x_m)^T$ that satisfies $\|\mathbf{x}\| = 1$ and makes the function $\|\mathbf{Y}\mathbf{x}\|$ get the maximum value. Here, $\|\mathbf{x}\|$ stands for the Euclidean norm. With the first $r-1$ principal components, i.e. $\mathbf{v}_1, \dots, \mathbf{v}_{r-1}$, the r-th principal component $\mathbf{v}_r$ can be found by subtracting the first $r-1$ principal components from $\mathbf{Y}$,

$$\mathbf{v}_r = \arg \max_{\|\mathbf{x}\|=1} \|(\mathbf{Y} - \sum_{j=1}^{r-1} \mathbf{Y}\mathbf{v}_j\mathbf{v}_j^T)\mathbf{x}\|. \tag{5.3}$$

A pair of vectors $\mathbf{v} \in \mathcal{R}^m$ and $\mathbf{u} \in \mathcal{R}^n$ are singular vectors of the matrix $\mathbf{Y}$, if $\mathbf{Y}\mathbf{v} = \eta\mathbf{u}$ and $\mathbf{u}^T\mathbf{Y} = \eta\mathbf{v}^T$, where η is the corresponding singular value. The principle components, i.e. $\mathbf{v}_1, \dots, \mathbf{v}_m$, are one of the pairs of singular vectors. Usually, the corresponding singular values of each principle component are ordered, i.e. $\eta_1 \geq \eta_2 \geq \dots \geq \eta_m \geq 0$. The matrix $\mathbf{Y}$ can be decomposed by the singular value decomposition (SVD),

$$\mathbf{Y} = \sum_{j=1}^{m} \eta_j \mathbf{u}_j \mathbf{v}_j^T. \tag{5.4}$$

5.1.2.3 Traffic Anomaly Detection

Because the first r principle components with $r \ll m$ can capture the main patterns of the normal traffic, a measurement vector $\mathbf{y}_{i*}$ should reside in the subspace of $\mathbf{v}_1, \dots, \mathbf{v}_r$, and the last $m-r$ principle components are assumed to contain only random fluctuations. Therefore, $\mathbf{y}_{i*} = (y_{i1}, \ldots, y_{im})^T$ can be decomposed into normal and abnormal subspaces,

$$\mathbf{y}_{i*} = \mathbf{y}_{i,normal} + \mathbf{y}_{i,anomaly} \tag{5.5}$$

where $\mathbf{y}_{i,normal} = \mathbf{P}\mathbf{P}^T\mathbf{y}_{i*}$ and $\mathbf{y}_{i,anomaly} = (\mathbf{I} - \mathbf{P}\mathbf{P}^T)\mathbf{y}_{i*}$ with $\mathbf{P} = [\,\mathbf{v}_1, \ldots, \mathbf{v}_r\,]$. The distance of a measurement vector $\mathbf{y}_{i*}$ from the normal pattern can be computed as,

$$d_Y(\mathbf{y}_{i*}) = \|\mathbf{y}_{i,anomaly}\| = \sqrt{\sum_{j=r+1}^{m} (\mathbf{v}_j^T \mathbf{y}_{i*})^2}. \tag{5.6}$$

The distance $d_Y(\mathbf{y}_{i*})$ equals to the squared prediction error (SPE) [54]. The observed traffic is considered malicious if

$$d_Y(\mathbf{y}_{i*}) > Q_\varrho, \tag{5.7}$$

where Q_ϱ denotes the threshold that is computed by the *Q-statistic* developed by Jackson and Mudholkar [54].

$$Q_\varrho^2 = \phi_1 \left[\frac{c_\varrho \sqrt{2\phi_2 h_0^2}}{\phi_1} + 1 + \frac{\phi_2 h_0 (h_0 - 1)}{\phi_1^2} \right]^{1/h_0}, \tag{5.8}$$

where

$$c_\varrho = 1 - \varrho, \tag{5.9}$$

$$h_0 = 1 - \frac{2\phi_1\phi_3}{3\phi_2^2}, \tag{5.10}$$

$$\phi_k = \sum_{j=r+1}^{m} \sigma_j^{2k} \text{ (for } k = 1, 2, 3), \tag{5.11}$$

and σ_j is the standard deviation of the projection of the measurements on the j-th principal component, which can be estimated as

$$\sigma_j = \frac{1}{\sqrt{n-1}} \|\mathbf{Y}\mathbf{v}_j\| = \frac{\eta_j}{\sqrt{n-1}}. \tag{5.12}$$

5.1.2.4 Objective of this research

Our algorithm aims at computing PCA in a distributed monitoring system, and further detecting traffic anomalies. Firstly, the computation should be very efficient at both the local monitors and the NOCs. Secondly, the algorithm can be implemented in a distributed environment, which requires careful considerations about the storage and communication overhead.

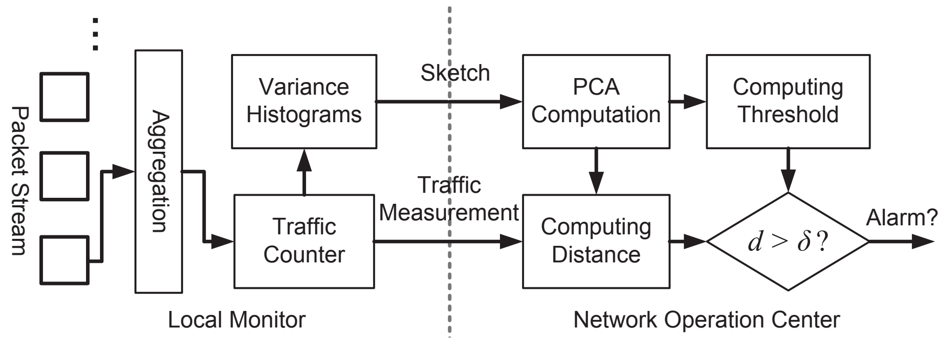


Figure 5.1 System Model of Sketch-based Algorithm

Last but not the least, it should support continuous updating to adjust the traffic anomaly detector due to the evolution of the traffic.

5.1.3 Sketch-based Algorithm

The system model of our sketch-based algorithm is shown in Fig. 5.1, which utilizes the random projection to reduce the computation complexity at the NOCs, and the variance estimation [93] to reduce the space requirement at the local monitors. There are five modules, each of which is described in the following subsection.

5.1.3.1 Traffic Counter

ISP implements an aggregation method and reports a pair $(FlowID, Size)$ to the volume counter, where $Size$ denotes the packet size and $FlowID$ is the index of the aggregated flow. The traffic counter maintains a bucket for each flow. A bucket U_j stores the traffic volume of the j-th flow at the current time interval. When a pair $(FlowID, Size)$ with $FlowID = j$ comes at the current time interval, the corresponding bucket U_j will be increased by $Size$. When a time interval ends, it just reports the traffic volume to the Variance Histogram and the NOC. Next, the value in the bucket is set to zero for the next interval. The Traffic Counter can also implement some streaming algorithms to compute other statistics over the aggregated flows, e.g., the number of packets, the entropy of the ports, etc.

5.1.3.2 Variance Histograms

For the traffic anomaly detection, we only interested in the recent measurements within a sliding window of the length n. When a new measurement become available, we will remove the oldest one for the traffic anomaly detection. Because the length of the sliding window can be very large, it is inefficient to maintain all recent n measurements in the memory. We design a data structure, i.e. Variance Histogram, to approximate the sequence of the measurements in the sliding window. Especially, we are interested in the variance and their distance of the recent measurements among multiple aggregated flows.

The traffic measurement x_{ij} at each time interval is treated as a data element. Given a sequence of data elements in the sliding window,

$$\{x_{(t-n+1)j}, \dots, x_{tj}\}, \tag{5.13}$$

their variance is defined as

$$V_{tj} = \sum_{i=t-n+1}^{t} (x_{ij} - \bar{x}_{tj})^2 \tag{5.14}$$

where $\bar{x}_{tj}$ is the mean of data elements,

$$\bar{x}_{tj} = \frac{1}{n} \sum_{i=t-n+1}^{t} x_{ij}. \tag{5.15}$$

A Variance Histogram contains a list of buckets for each flow, denoted by B_{pj} for $p = 1, \dots, N$, which can approximate the variance of the data elements in the sliding window. The sequence of data elements in the sliding window are divided into buckets, where each bucket contains a subsequence of the data elements,

$$B_{pj} = \{x_{(s-n_{pj}+1)j}, \dots, x_{(s-1)j}, x_{sj}\}. \tag{5.16}$$

The following statistics information for the data elements in each bucket B_{pj} are maintained in the Variance Histogram:

- the time stamp of the oldest data element τ_{pj}:

$$\tau_{pj} = \min\{i | x_{ij} \in B_{pj}\} \tag{5.17}$$

- the number of data elements n_{pj}:

$$n_{pj} = |B_{pj}| \tag{5.18}$$

- the mean of data elements μ_{pj}:

$$\mu_{pj} = \frac{1}{n_{pj}} \sum_{x_{ij} \in B_{pj}} x_{ij} \tag{5.19}$$

- the variance of data elements V_{pj}:

$$V_{pj} = \sum_{x_{ij} \in B_{pj}} (x_{ij} - \mu_{pj})^2 \tag{5.20}$$

Besides the above statistics, we also maintain two arrays of counters for the traffic anomaly detection:

- Z_{pkj}: the sum of $x_{ij} r_{ik}$ for all x_{ij} and $k = 1, \ldots, \ell$;
- R_{pkj}: the sum of corresponding random numbers r_{ik} for $k = 1, \ldots, \ell$;

where r_{ik} are random numbers from the standard normal distribution or other specific stable distributions [90].

Our algorithm starts with an empty list of buckets and updates the list of buckets with three steps, as shown in Fig.5.2. Firstly, when a new data element x_{tj} comes, the current time stamp is updated to t. We check the oldest bucket B_{Nj} and delete it if it is expired, where N denotes the number of buckets in the list. Secondly, the new element constitutes a new bucket B_{1j} and each old bucket B_{pj} becomes $B_{(p+1)j}$ for $p = 1, \ldots, N$. Last, we check whether there are qualified pairs of buckets that can be merged. Let $B_A = B_{(p+1)j} \cup B_{(p+2)j}$ and $B_B = \cup_{q=1}^{p} B_{qj}$. We merge two adjacent buckets $B_{(p+1)j}$ and $B_{(p+2)j}$ if and only if they satisfy the following merging rules.

- Rule 1: $V_{A \cup B} - V_B \leq \frac{\varepsilon}{5} V_B$.
- Rule 2: $n_A \leq \frac{\varepsilon}{10} n_B$.
- Rule 2: $n_A + n_B \leq n/2$.

```
**Step1:** *Check the time stamp of the last bucket* $B_{Nj}$
    **if**($\tau_{Nj} \leq t - n$)
        delete $B_{Nj}$;
    **end**

**Step2:** *Create a new bucket* $B_{1j}$
    $\tau_{1j} = t$; $n_{1j} = 1$;
    $\mu_{1j} = x_{tj}$; $V_{1j}=0$;
    **for**($k = 1, \ldots, l$)
        $Z_{1kj} = x_{tj} r_{tk}$;
        $R_{1kj} = r_{tk}$;
    **end**

**Step3:** *Traverse the bucket list to merge buckets*
    $p = 1$; $B_B = B_{1j}$;
    **while**($B_{(p+2)j}$ exists)
        $B_A = B_{(p+1)j} \cup B_{(p+2)j}$;
        **if**($n_A + n_B > n/2$)
            return;
        **else if**($n_A \leq \frac{\varepsilon}{10} n_B$ and $V_{A \cup B} - V_B \leq \frac{\varepsilon}{5} V_B$)
            delete $B_{(p+2)j}$;
            $B_{(p+1)j} = B_A$;
        **else**
            $p = p + 1$;
            $B_B = B_B \cup B_{pj}$;
        **end**
    **end**
```

Figure 5.2 Algorithm for Updating Variance Histogram

When two adjacent buckets B_{pj} and B_{qj} merge into a new bucket $B_{(p\cup q)j}$, the merged bucket's time stamp is set to be the time stamp of the older one, and the merged bucket's information can be calculated as following,

$$\begin{aligned}
n_{(p\cup q)j} &= n_{pj} + n_{qj}, && (5.21)\\
\mu_{(p\cup q)j} &= \frac{n_{pj}\mu_{pj} + n_{qj}\mu_{qj}}{n_{pj} + n_{qj}}, && (5.22)\\
V_{(p\cup q)j} &= V_{pj} + V_{qj} + \frac{n_{pj}n_{qj}}{n_{pj} + n_{qj}}(\mu_{pj} - \mu_{qj})^2, && (5.23)\\
Z_{(p\cup q)kj} &= Z_{pkj} + Z_{qkj} \text{ for } k = 1, \dots, \ell, && (5.24)\\
R_{(p\cup q)kj} &= R_{pkj} + R_{qkj} \text{ for } k = 1, \dots, \ell. && (5.25)
\end{aligned}$$

Let $B_{all,j} = \cup_{p=1}^{N} B_{pj}$ denote the bucket by merging all buckets together, and $\hat{V} = V_{all,j}$ be the estimated variance. We get the following result [93].

Lemma 16. Variance Histogram maintains an ε-approximate variance,

$$(1-\varepsilon)V \le \hat{V} \le V, \tag{5.26}$$

with $O(\frac{1}{\varepsilon}\log n)$ space and $O(1)$ running time.

At each local monitor, we implement a Variance Histogram for each flow and n pseudo random number generators shared by all flows among local monitors. The architecture for the sketch computation at a local monitor is shown in Fig. 5.3. The volume counter only uses a bucket to maintain the traffic volume at the current time interval t for each flow. When a time interval ends, the volume counter reports the traffic volume x_{tj} to the j^{th} Variance Histogram. Then the Variance Histogram updates its buckets as shown in Fig. 5.2. At each time interval, we can compute an approximation of the sketch as,

$$\hat{z}_{kj} = \frac{1}{\sqrt{\ell}}(Z_{all,kj} - n_{all,j}\mu_{all,j}R_{all,kj}), \tag{5.27}$$

where $n_{all,j}$, $\mu_{all,j}$, $Z_{all,kj}$, and $R_{all,kj}$ are the elements in $B_{all,j} = \cup_{p=1}^{N} B_{pj}$.

5.1.3.3 PCA Computation

The PCA computation is done in a lazy mode. If there is no anomaly, the NOC will use the previous PCA result and don't require the current sketches from the local monitors. Otherwise,

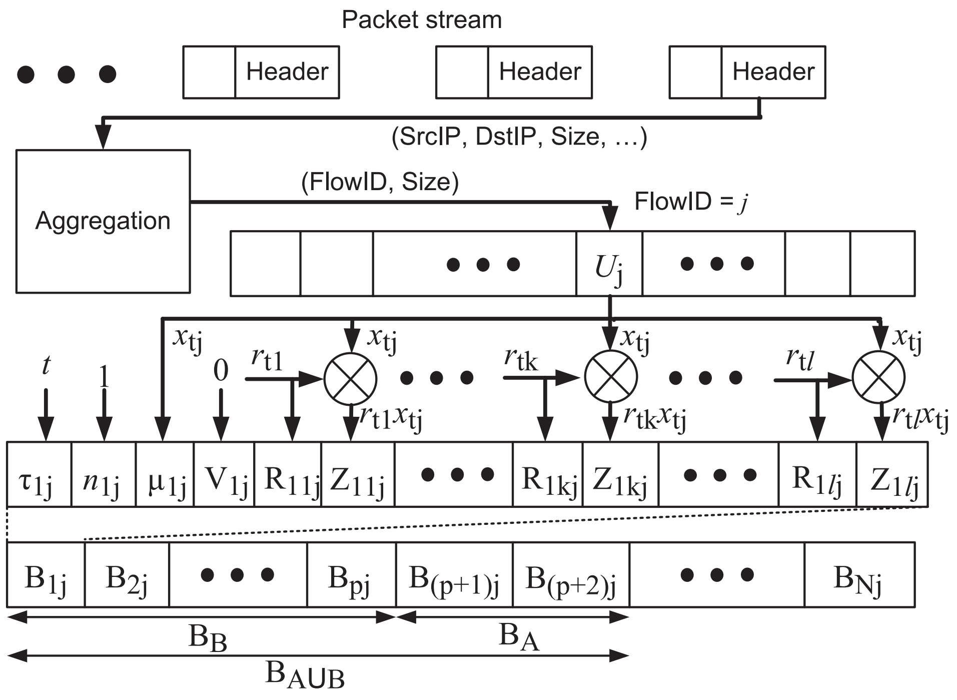

Packet stream
Header
Header
Header
(SrcIP, DstIP, Size, ...)
Aggregation
(FlowID, Size)
FlowID = j
U_j
x_{tj}
t
1
0
r_{t1}
r_{tk}
r_{tl}
$r_{t1}x_{tj}$
$r_{tk}x_{tj}$
$r_{tl}x_{tj}$
τ_{1j} n_{1j} μ_{1j} V_{1j} R_{11j} Z_{11j} R_{1kj} Z_{1kj} R_{1lj} Z_{1lj}
B_{1j} B_{2j} B_{pj} $B_{(p+1)j}$ $B_{(p+2)j}$ B_{Nj}
B_B
B_A
B_{AUB}

Figure 5.3 Sketch computation with Variance Histogram

the NOC gets all sketches

$$\{\hat{z}_{kj} : k = 1, \dots, \ell,\ j = 1, \dots, m\} \tag{5.28}$$

and the means $\mu_{all,j}$ from local monitors, and organizes them into a $\ell \times m$ matrix $\hat{\mathbf{Z}}$. PCA is applied on the matrix $\hat{\mathbf{Z}}$ and its SVD is computed,

$$\hat{\mathbf{Z}} = \sum_j \hat{\lambda}_j \hat{\mathbf{b}}_j \hat{\mathbf{a}}_j^T. \tag{5.29}$$

5.1.3.4 Anomaly Distance

Given the principle components, i.e. $\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_m$, we choose the last few principle components to compute the anomaly distance of the traffic vector $\mathbf{y}_{i*} = \mathbf{x}_i - (\mu_{all,1}, \dots, \mu_{all,m})^T$,

$$d_{\hat{Z}}(\mathbf{y}_{i*}) = \sqrt{\sum_{j=r+1}^{m} (\hat{\mathbf{a}}_j^T \mathbf{y}_{i*})^2} \tag{5.30}$$

where r is an integer less than m.

Before computing the anomaly distance, we need to determine the size of the normal subspace, denoted by r. There are several techniques which can be used to determine the size of the normal space, such as $k\sigma$-heuristic, Cattell's Scree Test, and so forth. Here, we give a brief introduction about the 3σ-heuristic that was used in [62]. The projection of the matrix $\hat{\mathbf{Z}}$ on the j-th principle component, i.e. $\hat{\mathbf{Z}}\hat{\mathbf{a}}_j$, is examined one by one. When a projection is found that the value of an element in $\hat{\mathbf{Z}}\hat{\mathbf{a}}_j$ exceeds $3\sigma_j$ from the mean, where σ_j denotes the standard deviation, this and all remaining principle components are selected to compute the anomaly distance.

5.1.3.5 Threshold Computation

The threshold computation is based on the fault detection in multivariate process control [54]. Because $\hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_m$ are m orthonormal vectors, we get

$$\|\mathbf{y}_{i*}\| = \sqrt{\sum_{j=1}^{m} (\hat{\mathbf{a}}_j^T \mathbf{y}_{i*})^2}. \tag{5.31}$$

Let $\hat{\mathbf{Q}} = [\,\hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_r\,]$, and then we have

$$d_{\hat{Z}}(\mathbf{y}_{i*}) = \sqrt{\mathbf{y}_{i*}^T \mathbf{y}_{i*} - \sum_{j=1}^{r} (\hat{\mathbf{a}}_j^T \mathbf{y}_{i*})^2} = \|(\mathbf{I} - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^T)\mathbf{y}_{i*}\|. \tag{5.32}$$

We can compute the threshold δ_ϱ based on the *Q-statistic*.

$$\delta_\varrho^2 = \varphi_1 \left[\frac{c_\varrho \sqrt{2\varphi_2 h_1^2}}{\varphi_1} + 1 + \frac{\varphi_2 h_1 (h_1 - 1)}{\varphi_1^2} \right]^{1/h_1} \tag{5.33}$$

where $c_\varrho = 1 - \varrho$,

$$h_1 = 1 - \frac{2\varphi_1\varphi_3}{3\varphi_2^2}, \quad \varphi_k = \frac{1}{(n-1)^k} \sum_{j=r+1}^{m} \hat{\lambda}_j^{2k} \quad (k = 1, 2, 3). \tag{5.34}$$

The NOC first computes the anomaly distance according to (5.32). If $d_{\hat{Z}}(\mathbf{y}_{i*}) < \delta_\varrho$, there won't be any traffic anomalies and the NOC will do nothing. If $d_{\hat{Z}}(\mathbf{y}_{i*}) > \delta_\varrho$, there could be two reasons. One is that the sketches at the NOC are out of date. The NOC will first get current sketches from local monitors, and recompute the anomaly distance and the threshold. The other is that there is an anomaly in the traffic. After the re-computation based on the current sketches, if the NOC still gets $d_{\hat{Z}}(\mathbf{y}_{i*}) > \delta_\varrho$, it will identify $\mathbf{y}_{i*}$ as a traffic anomaly. Otherwise, the NOC won't report any anomalies and will only update the PCA. An example

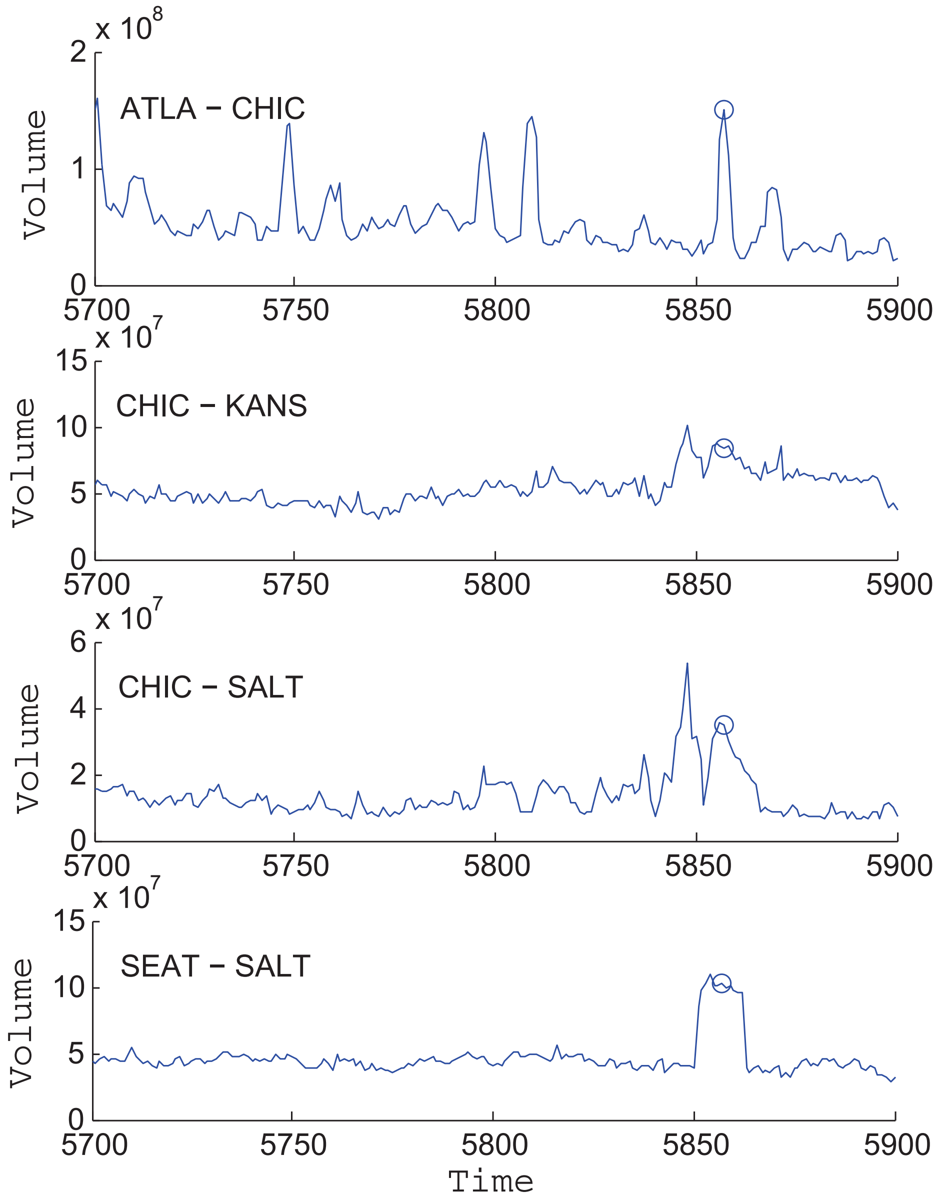


Figure 5.4 An Example of Coordinated Traffic Anomalies in the Internet2 Network

of the traffic anomaly detected in the Internet2 Network is shown in Fig.5.4. When there are coordinated changes in multiple flows, our PCA-based method can detect such changes as a network-wide traffic anomaly.

5.1.4 Theoretical Analysis

Our sketch-based algorithm detects traffic anomalies based on the same principle as Lakhina's method [62], which aims at detecting low-profile coordinated traffic anomalies. If there is an increase on several flows at the same time, we can detect that the anomaly distance will exceed the threshold, as shown in Fig. 5.4. In fact, our algorithm is an approximation algorithm for Lakhina's method, which can improve the computation complexity. We first analyze the computation complexity and then prove the error bounds in our sketch-based algorithm.

5.1.4.1 Performance Analysis

Because the variance estimation algorithm only needs $O(\frac{1}{\varepsilon}\log n)$ space and $O(1)$ running time [93], we have the following theorem.

Theorem 14. The sketch-based method requires $O(w\log n)$ running time and $(w\log^2 n)$ space at the local monitor. The computation complexity is $O(m^2\log n)$ and the space requirement is $O(m\log n)$ at the NOC.

Proof. According to the algorithm in [93], we need only $O(1)$ running time to update the variance buckets. But we also need to update the sketches Z_{pkj} and the random numbers R_{pkj}. Therefore, the sketch-based method needs $O(\ell)$ running time to update the variance histograms. A bucket needs $O(\ell)$ space to storage the statistics information and we have at most $O(\log n)$ buckets for each flow. In general, the local monitor needs $O(w\log^2 n)$ space and $O(w\log n)$ running time for $\ell = O(\log n)$.

At the NOC, the computation complexity of the SVD on a $\ell \times m$ matrix is $O(m^2\ell)$, which means that the computation complexity is at most $O(m^2\log n)$. In order to save the matrix $\hat{\mathbf{Z}}$, NOC needs $O(m\ell) = O(m\log n)$ memory space. At each time step, NOC uses the traffic vector $\mathbf{y}_{i*}$ and pre-computed principle components to detect traffic anomalies, which only requires

$O(m^2)$ running time. In general, the sketch-based method requires $O(m^2 \log n)$ running time and $O(m \log n)$ space at the NOC. $\square$

If the local monitors only have limited computation resources or bandwidth, we can maintain the VH and compute the sketches at the NOC side. In this way, the local monitors only need to implement the Volume Counter which only requires $O(1)$ running time to process each packet. The NOC needs $O(m \log n)$ running time and $O(m \log^2 n)$ space in this case.

5.1.4.2 Error Bounds

In this subsection, we explain why our algorithm can compute principal components based on the sketches and further detect traffic anomalies like Lakhina's method. Before the proof of our algorithm, we first provide some properties of the random projection of the traffic matrix $\mathbf{Y}$. Next, we show the PCA can be approximated with the sketches. Last, we show the anomaly distance can be bounded.

Let $\mathbf{R}$ be an $n \times \ell$ random matrix, which consists of the random number r_{ik} from the standard normal distribution. We define the random projection of $\mathbf{Y}$ as a matrix $\mathbf{Z}$,

$$\mathbf{z}_j = \frac{1}{\sqrt{\ell}}\mathbf{R}^T\mathbf{y}_j \quad \text{and} \quad \mathbf{Z} = \frac{1}{\sqrt{\ell}}\mathbf{R}^T\mathbf{Y}, \tag{5.35}$$

where $\mathbf{y}_j$ and $\mathbf{z}_j$ are a column in $\mathbf{Y}$ and $\mathbf{Z}$, respectively. The vector $\mathbf{z}_j$ is also called the random projection of $\mathbf{y}_j$, which has the following properties.

Lemma 17. Let $\mathbf{R}$ be an $n \times \ell$ random matrix from the standard normal distribution and $\mathbf{z}_j = \frac{1}{\sqrt{\ell}}\mathbf{R}^T\mathbf{y}_j$. We have

- $\mathrm{E}(\|\mathbf{z}_j\|^2) = \|\mathbf{y}_j\|^2$
- $\mathrm{P}(|\|\mathbf{z}_j\|^2 - \|\mathbf{y}_j\|^2| \geq \varepsilon\|\mathbf{y}_j\|^2) < 2e^{-(\varepsilon^2-\varepsilon^3)\frac{\ell}{4}}$

where ε is an arbitrary positive constant.

Proof.

$$\begin{aligned}
E(\|\mathbf{z}_j\|^2) &= E(\sum_{k=1}^{\ell} z_{kj}^2) \\
&= E\left(\sum_{k=1}^{\ell}(\frac{1}{\sqrt{\ell}}\sum_{i=t-n+1}^{t} r_{ik}y_{ij})^2\right) \\
&= \frac{1}{\ell}\sum_{k=1}^{\ell}\left(\sum_{i=t-n+1}^{t} y_{ij}^2 E(r_{ik}^2)\right. \\
&\quad \left. + \sum_{i=t-n+1}^{t-1}\sum_{i'=i+1}^{t} 2y_{ij}y_{i'j}E(r_{ik})E(r_{i'k})\right) \\
&= \sum_{i=t-n+1}^{t} y_{ij}^2 \\
&= \|\mathbf{y}_j\|^2 \qquad (5.36)
\end{aligned}$$

Let $W_k = \frac{\sqrt{\ell}}{\|\mathbf{y}_j\|}z_{kj} = \frac{1}{\|\mathbf{y}_j\|}\sum_{i=t-n+1}^{t} r_{ik}y_{ij}$, which is a standard normal variable. We define

$$W = \frac{\ell}{\|\mathbf{y}_j\|^2}\|\mathbf{z}_j\|^2 = \sum_{k=1}^{\ell} W_k^2 \qquad (5.37)$$

It follows the chi-square distribution χ_ℓ^2. Therefore,

$$\begin{aligned}
P(\|\mathbf{z}_j\|^2 \geq (1+\varepsilon)\|\mathbf{y}_j\|^2) &= P(W \leq (1+\varepsilon)k) \\
&= P(e^{\theta W} \geq e^{(1+\varepsilon)\ell\theta}) \\
&\leq \frac{E(e^{\theta W})}{e^{(1+\varepsilon)\ell\theta}} \qquad (5.38)
\end{aligned}$$

using Markov's inequality.

$$\begin{aligned}
P(\|\mathbf{z}_j\|^2 \geq (1+\varepsilon)\|\mathbf{y}_j\|^2) &\leq \frac{\Pi_{k=1}^{\ell}E(e^{\theta W_k})}{e^{(1+\varepsilon)\ell\theta}} \\
&= \left(\frac{E(e^{\theta W_1^2})}{e^{(1+\varepsilon)\theta}}\right)^{\ell} \qquad (5.39)
\end{aligned}$$

Because W_1 follows the standard normal distribution,

$$E(e^{\theta W_1^2}) = \frac{1}{\sqrt{1-2\theta}}. \qquad (5.40)$$

The above equation holds for any $\theta < 1/2$. Thus we get,

$$P(W \geq (1+\varepsilon)\ell) \leq \left(\frac{e^{-2(1+\varepsilon)\theta}}{1-2\theta}\right)^{k/2} \qquad (5.41)$$

The optimal choice of θ is $\varepsilon/2(1+\varepsilon)$. So we get,

$$P(W \geq (1+\varepsilon)\ell) \leq \left((1+\varepsilon)e^{-\varepsilon}\right)^{k/2} < e^{-(\varepsilon^2-\varepsilon^3)\frac{\ell}{4}} \tag{5.42}$$

Similarly,

$$P(W \leq (1-\varepsilon)\ell) \leq \left((1+\varepsilon)e^{-\varepsilon}\right)^{k/2} < e^{-(\varepsilon^2-\varepsilon^3)\frac{\ell}{4}} \tag{5.43}$$

According to the above two equations, we get $\mathrm{P}(|\|\mathbf{z}_j\|^2 - \|\mathbf{y}_j\|^2| \geq \varepsilon\|\mathbf{y}_j\|^2) < 2e^{-(\varepsilon^2-\varepsilon^3)\frac{\ell}{4}}$ □

Besides the standard normal distribution, there are several probability distributions which have been proposed for the random projection. Alon [5] introduced the tug-of-war algorithm, where the random matrix $\mathbf{R}$ is generated from the probability distribution,

$$r_{ik} = \begin{cases} -1 & \text{with probability } 1/2 \\ +1 & \text{with probability } 1/2 \end{cases}. \tag{5.44}$$

Later, Achlioptas [2] gave a more efficient algorithm, i.e. the sparse random projection, in which $r_{ik} = -1$, 0, or 1 with a probability $\frac{1}{2s}$, $1-\frac{1}{s}$ and $\frac{1}{2s}$, respectively, where s is an integer. In the sparse random projection, only $1/s$ of the data need to be processed. Recently, the very sparse random projection has been recommended by Li [65], which uses $\mathbf{R}$ of entries in $\{-1,0,1\}$ with probability $\{\frac{1}{2\sqrt{n}}, 1-\frac{1}{\sqrt{n}}, \frac{1}{2\sqrt{n}}\}$. For the sparse random projection, we have the following properties,

Lemma 18. Let $\mathbf{R}$ be an $n \times \ell$ random matrix with entries in $\{-1,0,1\}$ with probabilities $\{1/2s, 1-1/s, 1/2s\}$ and $\mathbf{z}_j = \frac{1}{\sqrt{\ell}}\mathbf{R}^T\mathbf{y}_j$. For $\forall\varepsilon > 0$, we have

- $\mathrm{E}(\|\mathbf{z}_j\|^2) = \|\mathbf{y}_j\|^2$;
- $\mathrm{P}(|\|\mathbf{z}_j\|^2 - \|\mathbf{y}_j\|^2| \geq \varepsilon\|\mathbf{y}_j\|^2) < 2e^{-(\varepsilon^2/2-\varepsilon^3/3)\frac{\ell}{2}}$.

This lemma can be proved by using a similar method in Lemma 17. In the following part, we can use either the standard normal distribution or the sparse random projection, both of which give the same result.

The sketch $\hat{z}_{kj}$ is a sketch of a subsequence of the traffic volumes within the sliding window as shown in Fig. 5.5. We organize $\hat{z}_{kj}$ into an $\ell \times m$ matrix $\hat{\mathbf{Z}}$. Based on the properties of

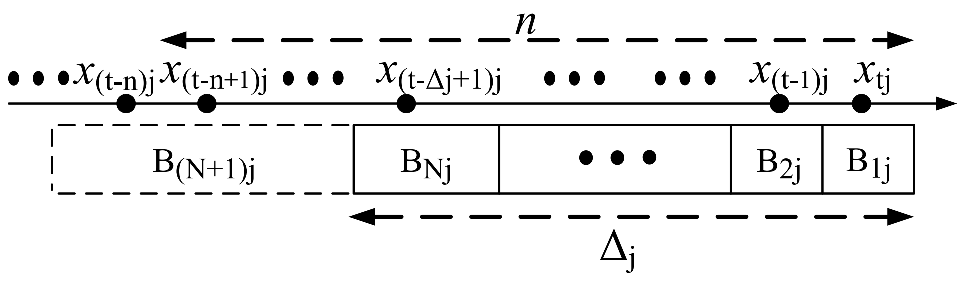


Figure 5.5 An illustration of sketch approximation

the random projection, we can prove that our sketch $\hat{\mathbf{z}}_j$ has similar properties to the random projection.

Lemma 19. Let r_{ik} for $k = 1, \ldots, \ell$, be generated from the probability distribution of the random projection, and $\hat{z}_{kj}$ be the sketch maintained by our algorithm. If $\ell > C\frac{\log n}{\varepsilon^2}$, we have

$$||\hat{\mathbf{z}}_j\|^2 - \|\mathbf{y}_j\|^2| \leq 2\varepsilon\|\mathbf{y}_j\|^2 \tag{5.45}$$

$$||\hat{\mathbf{Z}}\|_F^2 - \|\mathbf{Y}\|_F^2| \leq 2\varepsilon\|\mathbf{Y}\|_F^2 \tag{5.46}$$

with a probability at least $1 - 2e^{-\frac{C}{4}\log n}$, where $\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n x_{ij}^2}$ is the Frobenius norm of a matrix $\mathbf{X} \in \mathcal{R}^{n\times m}$.

Proof. The variance estimation algorithm maintains the variance $\hat{V}$ of a subsequence of the data elements in the sliding window of the size n, which has the following property [93],

$$(1-\varepsilon)V < \hat{V} < V. \tag{5.47}$$

Let $\hat{\mathbf{y}}_j$ denotes the subsequence maintained in the Variance Histogram for the j^{th} flow,

$$\hat{\mathbf{y}}_j = (\overbrace{0, \ldots, 0, \underbrace{y_{(t-\Delta_j+1)j}, \ldots, y_{tj}}_{\Delta_j}}^{n})^T. \tag{5.48}$$

According to the variance estimation algorithm, we have

$$\begin{aligned} \|\hat{\mathbf{y}}_j - \mathbf{y}_j\|^2 &= |\hat{V} - V| \\ &< \varepsilon V = \varepsilon\|\mathbf{y}_j\|^2. \end{aligned} \tag{5.49}$$

Based on the sketch computation Eq.(5.27), we have

$$\begin{aligned}\hat{z}_{kj} &= \frac{1}{\sqrt{\ell}}\left(\sum_{i=t-\Delta_j+1}^{t}(x_{ij}-\bar{x}_{tj})r_{ik}\right)\\ &= \frac{1}{\sqrt{\ell}}\mathbf{r}_k\hat{\mathbf{y}}_j. \end{aligned} \tag{5.50}$$

where $\mathbf{r}_k = (r_{(t-n+1)k}, \ldots, r_{tk})$. Therefore, we have

$$\hat{\mathbf{z}}_j - \mathbf{z}_j = \frac{1}{\sqrt{\ell}}\mathbf{R}(\hat{\mathbf{y}}_j - \mathbf{y}) \tag{5.51}$$

where $\hat{\mathbf{z}}_j$ and $\mathbf{z}_j$ are the j-th column in $\hat{\mathbf{Z}}$ and $\mathbf{Z}$, respectively.

We apply the properties of the random projection on the vector $\hat{\mathbf{z}}_j - \mathbf{z}_j$,

$$\|\hat{\mathbf{z}}_j - \mathbf{z}\|^2 \leq (1+\varepsilon)\|\hat{\mathbf{y}}_j - \mathbf{y}\|^2. \tag{5.52}$$

Based on Eq.(5.49) and Eq.(5.52), we have

$$\|\hat{\mathbf{z}}_j - \mathbf{z}_j\|^2 \leq \varepsilon\|\mathbf{y}_j\|^2. \tag{5.53}$$

Therefore, we have

$$\begin{aligned}\left|\|\hat{\mathbf{z}}_j\|^2 - \|\mathbf{y}_j\|^2\right| &\leq \|\hat{\mathbf{z}}_j - \mathbf{z}_j\|^2 + \|\mathbf{z}_j - \mathbf{y}_j\|^2\\ &\leq 2\varepsilon\|\mathbf{y}\|_j^2 \end{aligned} \tag{5.54}$$

with a probability $1 - 2e^{-\frac{C}{4}\log n}$.

Because $\|\hat{\mathbf{Z}}\|_F^2 = \sum_j \|\hat{\mathbf{z}}_j\|^2$ and $\|\mathbf{Y}\|_F^2 = \sum_j \|\mathbf{y}_j\|^2$, we have

$$\left|\|\hat{\mathbf{Z}}\|_F^2 - \|\mathbf{Y}\|_F^2\right| \leq 2\varepsilon\|\mathbf{Y}\|_F^2 \tag{5.55}$$

with a probability $1 - 2e^{-\frac{C}{4}\log n}$. □

Let $\hat{\mathbf{Z}} = \sum_j \hat{\lambda}_j \hat{\mathbf{b}}_j \hat{\mathbf{a}}_j^T$ and $\mathbf{Y} = \sum_j \eta_j \mathbf{u}_j \mathbf{v}_j^T$, we first prove that the singular values are approximately preserved.

Lemma 20. If $l > C\frac{\log n}{\varepsilon^2}$ for a large enough constant C and an arbitrary positive constant ε,

$$(1-2\varepsilon)\sum_{j=1}^{r}\eta_j^2 \leq \sum_{j=1}^{r}\hat{\lambda}_j^2 \leq (1+2\varepsilon)\sum_{j=1}^{r}\eta_j^2 \tag{5.56}$$

for $\forall r$ with the probability $1 - 2e^{-\frac{C}{4}\log n}$.

Proof. Because $\hat{\lambda}_1^2, \ldots, \hat{\lambda}_r^2$ are the first r largest eigenvalues of the matrix $\hat{\mathbf{Z}}^T\hat{\mathbf{Z}}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are an orthonormal set of vectors, we have

$$\begin{aligned}\sum_{j=1}^{r} \hat{\lambda}_j^2 &\geq \sum_{j=1}^{r} \mathbf{v}_j^T(\hat{\mathbf{Z}}^T\hat{\mathbf{Z}})\mathbf{v}_j \\ &\geq \sum_{j=1}^{r} \mathbf{v}_j^T\left((1-2\varepsilon)\mathbf{Y}^T\mathbf{Y}\right)\mathbf{v}_j \\ &= (1-2\varepsilon)\sum_{j=1}^{r}\eta_j^2. \end{aligned} \tag{5.57}$$

We can also know that

$$\begin{aligned}\hat{\lambda}_j^2 &= \hat{\mathbf{a}}_j^T\hat{\mathbf{Z}}^T\hat{\mathbf{Z}}\hat{\mathbf{a}}_j \\ &\leq \hat{\mathbf{a}}_j^T(1+2\varepsilon)\mathbf{Y}^T\mathbf{Y}\hat{\mathbf{a}}_j \\ &\leq (1+2\varepsilon)\|\mathbf{Y}\hat{\mathbf{a}}_j\|^2. \end{aligned} \tag{5.58}$$

Because $\eta_1^2, \ldots, \eta_r^2$ are the first r largest eigenvalues of the matrix $\mathbf{Y}^T\mathbf{Y}$, we can further get

$$\sum_{j=1}^{2}\hat{\lambda}_j^2 \leq (1+2\varepsilon)\sum_{j=1}^{r}\eta_j^2. \tag{5.59}$$

□

Using the above lemmas, we can bound the error of the covariance matrix in order to get a good approximation of the anomaly distance. According to the fact that principal components consists of an orthonormal set of vectors and $\|\mathbf{Y}\|_F^2 = \sum_{j=1}^{m}\eta_j^2$, we can prove the following lemma.

Lemma 21. Let $\mathbf{V} = \mathbf{Y}^T\mathbf{Y}$ and $\hat{\mathbf{A}} = \hat{\mathbf{Z}}^T\hat{\mathbf{Z}}$. If $l > C\frac{\log n}{\varepsilon^2}$ for a large enough constant C, then with the probability $1 - 2e^{-\frac{C}{4}\log n}$, we have

$$\|\mathbf{V} - \hat{\mathbf{A}}\|_F \leq 2\sqrt{\varepsilon}\|\mathbf{Y}\|_F^2. \tag{5.60}$$

Proof. Firstly, because $\hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_m$ are an orthonormal set of vectors,

$$\|\mathbf{V} - \hat{\mathbf{A}}\|_F^2 = \sum_{j=1}^{m}\|(\mathbf{V} - \hat{\mathbf{A}})\hat{\mathbf{a}}_j\|^2. \tag{5.61}$$

For each $j = 1, \dots, r$, we have

$$\|(\mathbf{V} - \hat{\mathbf{A}})\hat{\mathbf{a}}_j\|^2 \;=\; \|\mathbf{V}\hat{\mathbf{a}}_j\|^2 + \hat{\lambda}_j^4 - 2\hat{\lambda}_j^2\hat{\mathbf{a}}_j^T\mathbf{V}\hat{\mathbf{a}}_j. \tag{5.62}$$

Based on Eq.(5.58), we have

$$\begin{aligned}\hat{\mathbf{a}}_j\mathbf{V}\hat{\mathbf{a}}_j &= \|\mathbf{Y}\hat{\mathbf{a}}_j\|^2 \\ &\geq \frac{1}{1+2\varepsilon}\hat{\lambda}_j^2.\end{aligned} \tag{5.63}$$

Because $\hat{a}_1, \dots, \hat{a}_m$ are an orthonormal set of vectors, we have

$$\sum_j^m \|\mathbf{V}\hat{\mathbf{a}}_j\|^2 = \sum_j^m \eta_j^4. \tag{5.64}$$

Therefore,

$$\|\mathbf{V} - \hat{\mathbf{A}}\|_F^2 \;\leq\; \sum_{j=1}^m \left(\eta_j^4 + \hat{\lambda}_j^4 - \frac{2}{1+2\varepsilon}\hat{\lambda}_j^4\right). \tag{5.65}$$

Secondly, $\mathbf{v}_1, \dots, \mathbf{v}_m$ are an orthonormal set of vectors, and we have

$$\|\mathbf{V} - \hat{\mathbf{A}}\|_F^2 = \sum_j^m \|(\mathbf{V} - \hat{\mathbf{A}})\mathbf{v}_j\|^2. \tag{5.66}$$

For each $j = 1, \dots, m$, we have

$$\|(\mathbf{V} - \hat{\mathbf{A}})\mathbf{v}_j\|^2 \;=\; \eta_j^4 + \|\hat{\mathbf{A}}\mathbf{v}_j\|^2 - 2\eta_j^2\mathbf{v}_j^T\hat{\mathbf{A}}\mathbf{v}_j \tag{5.67}$$

We have

$$\begin{aligned}\mathbf{v}_j^T\hat{\mathbf{A}}\mathbf{v}_j &\geq \mathbf{v}_j^T(1-2\varepsilon)\mathbf{Y}^{\mathbf{T}}\mathbf{Y}\mathbf{v}_j \\ &= (1-2\varepsilon)\eta_j^2\end{aligned} \tag{5.68}$$

And we also have

$$\sum_j^m \|\hat{\mathbf{A}}\mathbf{v}_j\|^2 = \sum_j^m \hat{\lambda}_j^4. \tag{5.69}$$

Therefore, we have

$$\|\mathbf{V} - \hat{\mathbf{A}}\|_F^2 \;\leq\; \sum_{j=1}^m \left(\eta_j^4 + \hat{\lambda}_j^4 - 2(1-2\varepsilon)\eta_j^4\right). \tag{5.70}$$

Finally, based on (5.65) and (5.70), we get

$$\begin{aligned} \|\mathbf{V} - \hat{\mathbf{A}}\|_F^2 &\leq \sum_{j=1}^{m} 2\varepsilon \left(\eta_j^4 + \frac{1}{1+2\varepsilon} \hat{\lambda}_j^4 \right) \\ &\leq 2\varepsilon \sum_{j=1}^{m} \left(\hat{\lambda}_j^4 + \eta_j^4 \right). \end{aligned} \tag{5.71}$$

Based on Lemma 20 and $\|\mathbf{Y}\|_F^2 = \sum_{j=1}^m \eta_j^2$, we get the following result

$$\|\mathbf{V} - \hat{\mathbf{A}}\|_F^2 \leq 4\varepsilon \|\mathbf{Y}\|_F^4. \tag{5.72}$$

□

Next, we want to prove that $d_Y(\mathbf{y})$ can be approximated by $d_{\hat{Z}}(\mathbf{y})$. For this purpose, we first cite a theorem from the matrix perturbation theory.

The column space of a matrix $\mathbf{M}$ is the subspace spanned by the columns, which is denoted by

$$\mathcal{R}(\mathbf{M}) = \{\mathbf{M}\mathbf{x} : \mathbf{x} \in \mathcal{R}^m\}. \tag{5.73}$$

The set of all eigenvalues of the matrix $\mathbf{M}$ is denoted by

$$\mathcal{L}(\mathbf{M}) = \{\lambda : \mathbf{M}\mathbf{x} = \lambda\mathbf{x}, \exists \mathbf{x} \neq \mathbf{0}\}. \tag{5.74}$$

And Θ denotes the canonical angle between two subspaces,

$$\Theta(\mathcal{M}, \mathcal{N}) = \sin^{-1} \mathbf{\Sigma}, \tag{5.75}$$

where $\mathcal{M}$ and $\mathcal{N}$ are r-dimensional subspaces of $\mathcal{R}^m$. The columns of their orthogonal bases can be transformed by a unitary matrix to

$$\begin{pmatrix} \mathbf{I} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \text{and} \begin{pmatrix} \mathbf{\Gamma} \\ \mathbf{\Sigma} \\ \mathbf{0} \end{pmatrix} \tag{5.76}$$

if $2r \leq m$, or

$$\begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \text{and} \begin{pmatrix} \mathbf{\Gamma} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \\ \mathbf{\Sigma} & \mathbf{0} \end{pmatrix} \tag{5.77}$$

if $2r > m$.

Then we have the following property of two matrices [87].

Lemma 22. Matrix Perturbation: Let $\mathbf{M}$ have the spectral resolution

$$\begin{pmatrix} \mathbf{U}_1^T \\ \mathbf{U}_2^T \end{pmatrix} \mathbf{M} \left(\mathbf{U}_1 \, \mathbf{U}_2 \right) = \begin{pmatrix} \mathbf{L}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_2 \end{pmatrix} \tag{5.78}$$

where $(\mathbf{U}_1, \mathbf{U}_2)$ is unitary with $\mathbf{U}_1 \in \mathcal{R}^{n\times r}$. Let $\mathbf{B} \in \mathcal{R}^{n\times r}$ have orthonormal columns, and for any symmetric $\mathbf{H}$ of order r, let $\mathbf{E} = \mathbf{MB} - \mathbf{BH}$. If $\nu = \min |\mathcal{L}(\mathbf{L}_2) - \mathcal{L}(\mathbf{H})| > 0$, then we have

$$\| \sin \Theta[\mathcal{R}(\mathbf{U}_1), \mathcal{R}(\mathbf{B})]\|_F \le \frac{\|\mathbf{E}\|_F}{\nu}. \tag{5.79}$$

We apply the above lemma to the covariance matrices $\hat{\mathbf{A}}$ and $\mathbf{V}$, and then we can get an error bound for the anomaly distance.

Theorem 15. If $l > C\frac{\log n}{\varepsilon^2}$ for a large enough constant C, then

$$|d_{\hat{Z}}(\mathbf{y}) - d_Y(\mathbf{y})| \le \frac{2\sqrt{3\varepsilon}}{|\eta_{r+1}^2 - \eta_r^2|} \|\mathbf{Y}\|_F^2 \|\mathbf{y}\| \tag{5.80}$$

with the probability $1 - 2e^{-\frac{C}{4}\log n}$.

Proof. Let

$$\hat{\mathbf{Q}} = [\,\hat{\mathbf{a}}_1, \cdots, \hat{\mathbf{a}}_r\,], \tag{5.81}$$

$$\hat{\mathbf{Q}}_c = [\,\hat{\mathbf{a}}_{r+1}, \cdots, \hat{\mathbf{a}}_m\,], \tag{5.82}$$

$$\mathbf{P} = [\,\mathbf{v}_1, \cdots, \mathbf{v}_r\,], \tag{5.83}$$

$$\mathbf{P}_c = [\,\mathbf{v}_{r+1}, \cdots, \mathbf{v}_m\,]. \tag{5.84}$$

We have the following spectral resolutions,

$$\begin{pmatrix} \hat{\mathbf{Q}}^T \\ \hat{\mathbf{Q}}_c^T \end{pmatrix} \hat{\mathbf{A}} \left(\hat{\mathbf{Q}} \, \hat{\mathbf{Q}}_c \right) = \begin{pmatrix} \mathbf{\Lambda}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_2 \end{pmatrix}, \tag{5.85}$$

$$\begin{pmatrix} \mathbf{P}^T \\ \mathbf{P}_c^T \end{pmatrix} \mathbf{V} \left(\mathbf{P} \, \mathbf{P}_c \right) = \begin{pmatrix} \mathbf{M}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 \end{pmatrix}, \tag{5.86}$$

where

$$\begin{aligned}
\mathbf{\Lambda}_1 &= diag(\hat{\lambda}_1^2, \ldots, \hat{\lambda}_r^2), & (5.87)\\
\mathbf{\Lambda}_2 &= diag(\hat{\lambda}_{r+1}^2, \ldots, \hat{\lambda}_m^2), & (5.88)\\
\mathbf{M}_1 &= diag(\eta_1^2, \ldots, \eta_r^2), & (5.89)\\
\mathbf{M}_2 &= diag(\eta_{r+1}^2, \ldots, \eta_m^2) & (5.90)
\end{aligned}$$

and $diag(\cdot)$ denotes a diagonal matrix.

Let

$$\mathbf{E} = \mathbf{V}\hat{\mathbf{Q}} - \hat{\mathbf{Q}}\mathbf{\Lambda}_1. \tag{5.91}$$

Because $\hat{\mathbf{Q}}\mathbf{\Lambda}_1 = \hat{\mathbf{A}}\hat{\mathbf{Q}}$, we have

$$\mathbf{E} = \mathbf{V}\hat{\mathbf{Q}} - \hat{\mathbf{A}}\hat{\mathbf{Q}}. \tag{5.92}$$

According to Lemma 22, we have

$$\|\sin\Theta[\mathcal{R}(\mathbf{P}), \mathcal{R}(\hat{\mathbf{Q}})]\|_F \leq \frac{\|\mathbf{E}\|_F}{\nu} = \frac{\|\mathbf{V} - \hat{\mathbf{A}}\|_F}{\nu} \tag{5.93}$$

where $\nu = |\eta_{r+1}^2 - \lambda_r^2| \approx |\eta_{r+1}^2 - \eta_r^2|$.

The project matrices of $\mathcal{R}(\mathbf{P})$ and $\mathcal{R}(\hat{\mathbf{Q}})$ are $\mathbf{P}\mathbf{P}^T$ and $\hat{\mathbf{Q}}\hat{\mathbf{Q}}^T$, respectively. Then, according to Matrix Perturbation Theory, we have

$$\begin{aligned}
\|\mathbf{P}\mathbf{P}^T - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^T\|_F &= \sqrt{2}\|\sin\Theta[\mathcal{R}(\mathbf{P}), \mathcal{R}(\hat{\mathbf{Q}})]\|_F\\
&\leq \sqrt{2}\frac{\|\mathbf{V} - \hat{\mathbf{A}}\|_F}{\nu}. & (5.94)
\end{aligned}$$

Then we get

$$\begin{aligned}
|d_{\hat{Z}}(\mathbf{y}) - d_Y(\mathbf{y})| &= |\|(\mathbf{I} - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^T)\mathbf{y}\| - \|(\mathbf{I} - \mathbf{P}\mathbf{P}^T)\mathbf{y}\||\\
&\leq \|(\mathbf{I} - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^T)\mathbf{y} - (\mathbf{I} - \mathbf{P}\mathbf{P}^T)\mathbf{y}\|\\
&= \|(\mathbf{P}\mathbf{P}^T - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^T)\mathbf{y}\|\\
&\leq \|\mathbf{P}\mathbf{P}^T - \hat{\mathbf{Q}}\hat{\mathbf{Q}}^T\|_F\|\mathbf{y}\|\\
&\leq \sqrt{2}\frac{\|\mathbf{V} - \hat{\mathbf{A}}\|_F}{\hat{\nu}}\|\mathbf{y}\|\\
&\leq \frac{2\sqrt{2\varepsilon}}{|\eta_{r+1}^2 - \eta_r^2|}\|\mathbf{Y}\|_F^2\|\mathbf{y}\|. & (5.95)
\end{aligned}$$

□

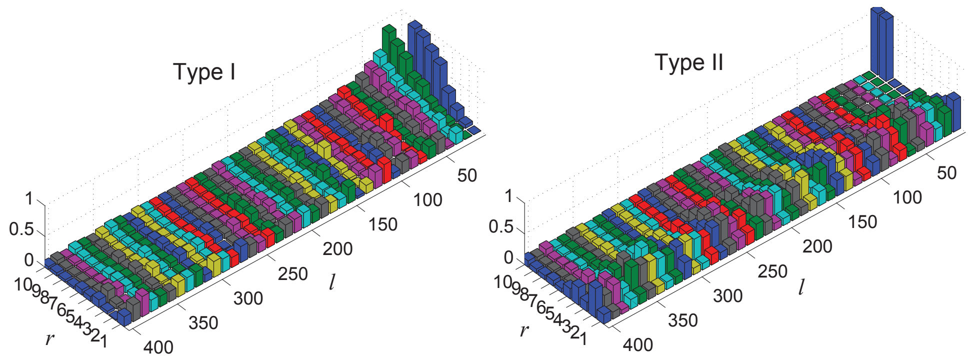


Figure 5.6 Type I and Type II errors with 5-minutes interval

Therefore, the anomaly distance can be approximated up to the multiplicative factor $(1 \pm \varepsilon^{1/2})$.

5.1.5 Experimental Evaluation

In this section, we evaluate our methods on the data from Abilene Observatory Data Collections. The data collection is running on nine routers after Feb 2008, i.e. ATLA, CHIC, HOUS, KANS, LOSA. NEWY, SALT, SEAT, and WASH. We use an one-month data collection between June 9th, 2008 and July 9th, 2008. The packets are aggregated into origin-destination (OD) flows based on both BGP and ISIS routing information. Because the traffic anomalies are unknown and hard to be determined only based on the Netflow traces, we first apply Lakhina's method to detect anomalies by using a fix size r for the normal subspace. And these detected anomalies are used as the 'real' anomalies to evaluate the detection accuracy of our algorithm. The length of the sliding window is two weeks. Currently, there is no good method to choose r, and thus we try possible values for the size r from 1 to 10. We set $\varepsilon = 0.01$ in the VH algorithm and $\varrho = 0.01$ for the threshold computation in the *Q-statistics*. We use our sketch-based

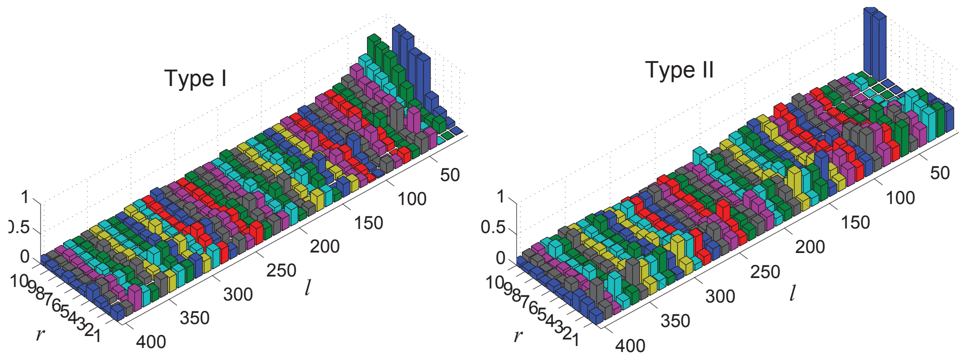


Figure 5.7 Type I and Type II errors with 1-minute interval

method as an approximation of the Lakhina's method and compute both Type I errors and Type II errors with different size l of the sketches, i.e. $l = 10, 20, \dots$

$$\begin{aligned} \text{Type I} &= \frac{\text{number of false anomalies}}{\text{total number of true normal observations}}, \\ \text{Type II} &= \frac{\text{number of false normal observations}}{\text{total number of true anomalies}}. \end{aligned}$$

We show Type I and Type II errors with five-minutes interval and one-minute interval in Fig.5.6 and Fig.5.7, respectively. When the size r is too small, we get large errors due to that the normal traffic cannot be retained in the normal subspace. If $r \geq 5$, 90% energy is retained in the normal subspace, i.e. $\|\mathbf{y}_{i,normal}\|^2/\|\mathbf{y}_i\|^2 \simeq 0.9$. In this case, both Type I and Type II errors decrease quickly at the beginning and then reach a nearly optimal value.

We check the eigenvalues of the measurement matrix $\mathbf{Y}$, and choose the size of the normal subspace as $r = 6$ which is proper for our data. We show the Type I and Type II errors with $r = 6$ and $l = 10, 20, \dots, 1000$ in Fig.5.8. If the size l of the sketch is more than 200, there is no remarkable decrease in both errors. The computation overhead at the NOC is determined by the PCA computation, which requires m^2n and m^2l for both Lakhina's method and our method respectively. We show the computation overhead at the NOC in Fig. 5.9. Our method

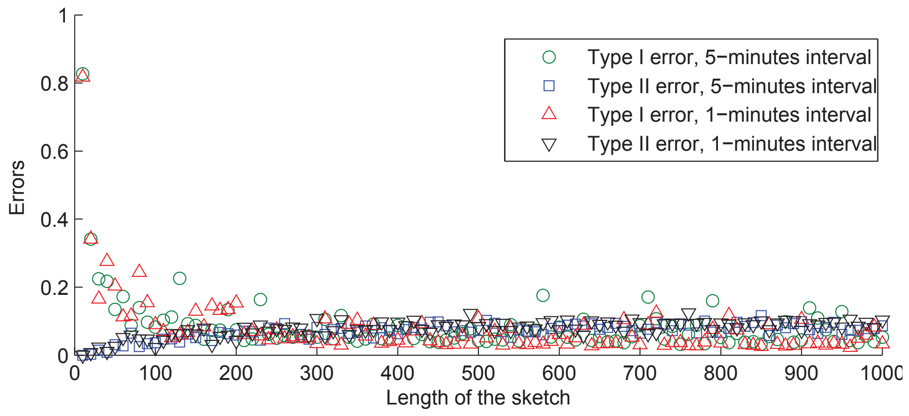


Figure 5.8 Type I and Type II errors with $r = 6$

requires much less computation than Lakhina's method.

If we want to further reduce the errors, we must set smaller values for ε and ϱ. We can bound the error of the estimated threshold and the distance in terms of η_j and $\mathbf{Y}$. Therefore, the accuracy of our algorithm depends on the properties of the covariance matrix $\mathbf{V}$ of the traffic measurements. If all the eigenvalues of $\mathbf{V}$ are close to each other, we first have $\nu = \eta_{r+1}^2 - \eta_r^2$ that can be very small. Therefore, we cannot bound the error of the distance. When η_j is close to each other, the value of the threshold δ_ϱ can also be far from Q_ϱ. In fact, the PCA-based detection method also has a high false alarm rate because the traffic measurements cannot reside in a low dimensional subspace.

5.1.6 Conclusion

In this paper, we study the network-wide traffic anomaly detection problem. Our algorithm achieves $O(w \log n)$ running time and $O(w \log^2 n)$ space at local monitors. The NOC could run PCA-based detection method with $O(m^2 \log n)$ running time and $O(m \log n)$ space. Our

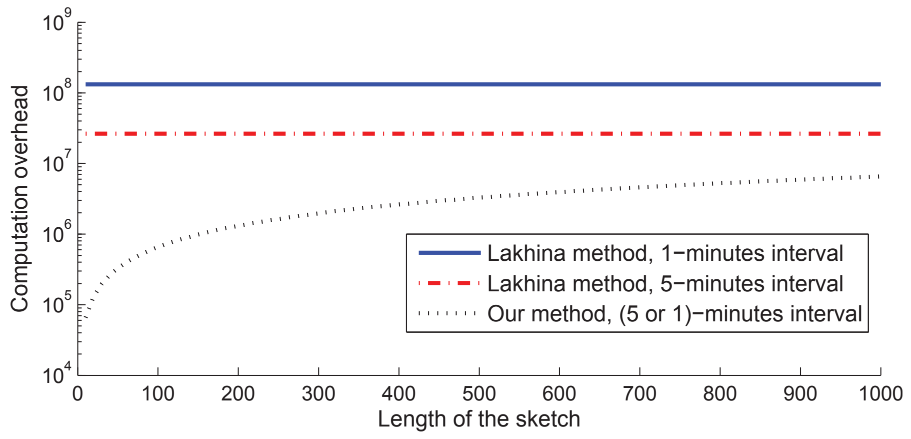


Figure 5.9 The computation overhead at the NOC in the logarithmic scale

algorithm also makes the ISPs be able to implement the detection method by paying careful consideration about the space requirement, the communication cost, and other resources over a distributed computing environment. In the future, we will apply our sketch-based method on various statistical anomaly detection methods, e.g. Markov models, Bayesian networks, etc.

5.2 Monitoring Traffic Activity Graphs with Low-rank Matrix Approximation

Recently, Traffic Activity Graphs (TAGs) have been proposed to understand, analyze, and model network-wide communication patterns. The topological properties of the TAGs have been shown to be very helpful for malware analysis, anomaly detection, and attack attribution. In a TAG, nodes represent hosts in the network and edges are observed flows that indicate certain communication relations or interactions of interest among the hosts. The challenge is to how to capture and analyze TAGs which are usually large, sparse and complex and often have overly-large space and computation requirements. In this section, we present a new sampling-based low-rank approximation method for monitoring TAGs. The resulted solution can reduce the computation complexity for the communication pattern analysis from $O(mn)$ to $O(m+n)$, where m and n denote the number of sources and destinations, respectively. The experimental results with real-world traffic traces show that our method outperforms existing solutions in terms of efficiency and the capability of processing and identifying unknown TAGs.

5.2.1 Introduction

Traffic analysis plays an essential part in network security and management, which covers a wide range of research topics such as measurements, classifications, anomaly detections, and so on. However, the remarkable growth in the development and deployment of new networking technologies and applications such as web-based social networking and peer-to-peer technology along with significant growth of security threats and sophisticated criminal activities make effective traffic analysis even more harder. To understand, analyze and model such complex and fast-evolving networked systems and the users' behavior on them, we need effective methodologies and appropriate (and new) traffic features to capture network-wide communication patterns and complex (often hidden or very subtle) structures or relationships. Recently, Traffic Activity Graphs (TAGs) [29; 52; 55; 56; 72] have been proposed for analyzing the interactions among the hosts in such complex Internet applications. In a TAG, nodes represent hosts in the network, each of which has a unique IP address, and edges are observed flows that represent certain

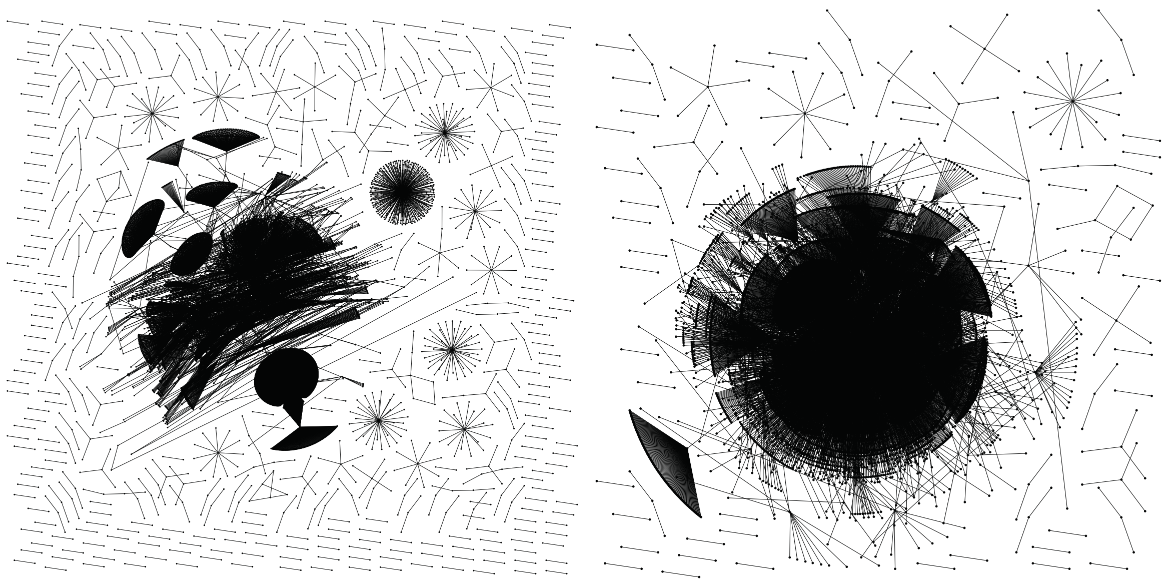


Figure 5.10 HTTP TAG (left) and Email TAG (right) in five minutes

communications or interactions of interest among the hosts. Two examples of the TAGs in the real traffic traces from the WIDE project (http://imdc.datcat.org/collection/1-05L9-X=WIDE-TRANSIT+150+Megabit+Ethernet+Trace+2008-03-18) are shown in Fig.5.10. The flows corresponding to an application of interest are identified by their service ports, i.e. HTTP port 80 and Email port 25.

The topologies of the TAGs from the same application in different contexts or enterprises often look very similar, but those TAGs from different applications usually vary significantly and show large diversity. For example, Email TAG shows a star topology while HTTP TAG often shows a bi-mesh structure, as shown in Fig.5.10. The topological properties of the TAGs can be helpful for a wide variety of applications, e.g. anomaly detection, malware identification, application classification, etc. Collins *et al.* [29] utilized the topology of the TAGs to detect worms and botnets. Furthermore, Shah and Zaman [86] showed that the source of the worms can be identified if the topology of the TAG for the worm spreading is given. Jin *et al.* [55] applied the orthogonal nonnegative matrix tri-factorization (tNMF) method [34] to discover the community structures in various application TAGs. Their method can be extended to identify, classify and understand unknown applications.

However, it is still very challenging to monitor TAGs in large-scale high-speed networks such as the Internet. Due to large volume of traffic, it is often infeasible to save or process all packets in order to obtain the exact TAGs. Thus, packet sampling is usually implemented to reduce the packet arrival rate, which can reduce space requirement. Nowadays, most routers implement the uniform sampling method, e.g., NetFlow. In this method, each packet is sampled randomly with the same probability (typically between 0.001 and 0.01). The high-volume flows are sampled with higher probability than the low-volume ones. Due to the sampling bias, the topological information of the TAGs can get lost a lot. cSamp [85] was proposed to provide a fine-grained flow level measurements, which chose flow sampling instead of traditional packet sampling to avoid the sampling bias against small flows. By sampling the traffic with cSamp, we can get an unbiased estimation of the TAGs, which is good. However, a drawback is that the sampled TAGs generated by cSamp become more sparse than the original ones, which makes it even more difficult to discover the communication patterns and community structures in the TAGs after sampling.

Another challenge for monitoring TAGs is the overly-large space and computation requirements. The TAGs in the real network traffic are usually large, sparse and complex. And they are continuously evolving due to the dynamics of network activities and user behaviorial changes. It is often very difficult to find any patterns or anomalies in such large dynamic graphs. The sampling methods like cSamp cannot solve this problem because the sampled TAGs are still large and complex, and even worse, they are more sparse. Therefore, a low-rank approximation is preferred to analyze and understand large graphs like in [37; 88; 89]. For example, the communication patterns can be found by the tNMF method, which approximates the TAGs by three nonnegative low-rank matrices. However, existing algorithms such as the tNMF method still require high computation overhead, which makes it hard to be used for monitoring TAGs in real-world networks. Another drawback is that they do not consider or intend to take advantage of the ways of packet being sampled. That is, they often split the data collection step from the analysis step.

In this section, we consider integrating the sampling problem and the low-rank approximation problem in one framework. Thereby, we can take advantage of the properties of the sampled packets to improve the performance of the low-rank approximation tNMF method.

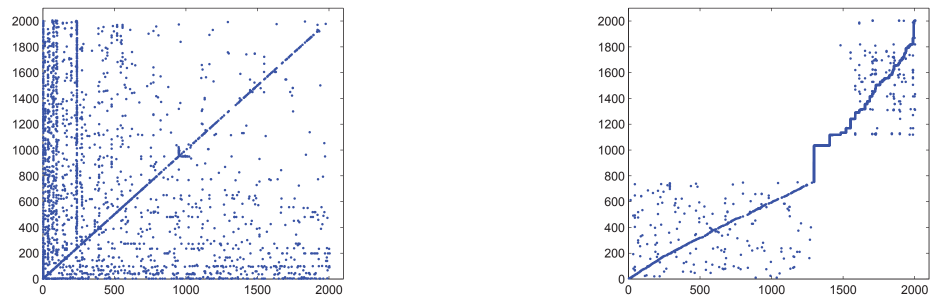
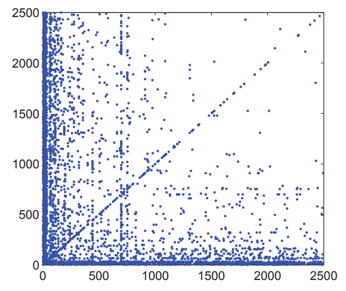


Figure 5.11 HTTP matrix before permutation (left) and after permutation (right)

We follow the cSamp method, and propose a new sampling method to provide a high *host* coverage, which requires that all the packets sent from or to a random selected set of hosts are sampled. We show that a high *host* coverage can provide compact information to discover the network-wide communication patterns and compute the low-rank approximation of the TAGs than a high flow coverage. Our method provides an efficient algorithm for the tNMF method, which reduce the computation complexity for each updating step from $O(mn)$ to $O(m+n)$ where m and n denote the number of sources and destinations in the TAGs. The experimental results with real-world traffic traces shows that our method outperforms existing solutions in terms of efficiency and the capability of processing and identifying unknown TAGs. Another advantage is to use our method to detect the TAGs of malware spreading (e.g., worm and botnets) which are often unknown during the time period packets being sampled, but we are able to find and monitor their communication pattern and discover the source once they are detected by some honeypots or intrusion detection systems.

5.2.2 Problem Definition

A TAG represents the network-wide communication patterns among the hosts, which is also referred as Traffic Dispersion Graph [52]. A TAG is a graphic representation of the social interactions among the hosts. In the Internet, a host sends a sequence of packets to communicate with another host, and the set of packets creates a flow from a source to a destination.



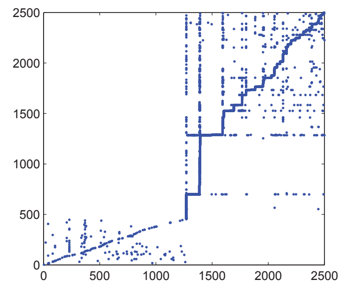


Figure 5.12 Email matrix before permutation (left) and after permutation (right)

If the communication is finished, the corresponding flow will be terminated. A host can create multiple flows to communicate with different hosts simultaneously. In a TAG, a node represents a distinct host, and a directed edge indicates that there is at least one flow from a source to a destination.

A TAG is defined as a bipartite graph

$$\mathcal{G} = \{\mathcal{V}_s \times \mathcal{V}_d, \mathcal{E}\} \tag{5.96}$$

where $\mathcal{V}_s$ ($\mathcal{V}_d$) is a set of nodes representing the sources (destinations) in the network and $\mathcal{E}$ is a set of edges corresponding to the flows. A directed edge $(i, j) \in \mathcal{E}$ indicates that there exists at least one flow from the source $i \in \mathcal{V}_s$ to the destination $j \in \mathcal{V}_d$. An edge filter can be implemented to extract edges of interest. For example, the edges can be chosen based on transport protocols (TCP, UDP, etc.), port numbers (e.g. HTTP port 80, Email port 25), or the size of the payload. Given a set of service ports $\mathcal{P}$ associated with an application of interest, all flows using a prot in $\mathcal{P}$ will be added into the edge set $\mathcal{E}$.

Given a TAG $\mathcal{G} = \{\mathcal{V}_s \times \mathcal{V}_d, \mathcal{E}\}$, we can define its adjacency matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$ as $a_{ij} = 1$ if $(i, j) \in \mathcal{E}$ and $a_{ij} = 0$ if $(i, j) \notin \mathcal{E}$, where a_{ij} is the element at the intersection of the i-th row and the j-th column. In the matrix $\mathbf{A}$, the i-th row $\mathbf{a}_{i*}$ corresponds to the i-th source and the j-th column $\mathbf{a}_{*j}$ corresponds to the j-th destination. If we permute rows and columns such that the hosts in the same community will be near to each other, we can see block structures, as shown in Fig.5.11 and Fig.5.12. Each dense block in the matrix $\mathbf{A}$ corresponds to a community in the

TAG. The problem of extracting the community structures from the TAG can be formulated as a co-clustering problem of the adjacency matrix $\mathbf{A}$, where rows and columns are simultaneously clustered into k groups and l groups, respectively. Each community is corresponding to a group of rows or columns that are similar to each other.

The community structures in the TAGs can be found by the matrix decomposition method, i.e. the tNMF method, which approximately decomposes a matrix $\mathbf{A}$ into three low-rank nonnegative matrices,

$$\mathbf{A} \simeq \mathbf{FSG} \tag{5.97}$$

where $\mathbf{F} \in \mathbf{R}_+^{m\times k}$, $\mathbf{S} \in \mathbf{R}_+^{k\times l}$, and $\mathbf{G} \in \mathbf{R}_+^{l\times n}$. For the objective of the approximation, we optimize

$$\begin{aligned} \min_{\mathbf{F}\geq 0,\mathbf{S}\geq 0,\mathbf{G}\geq 0} \quad & J(\mathbf{F},\mathbf{S},\mathbf{G}) = \|\mathbf{A}-\mathbf{FSG}\|_F, \\ s.t. \quad & \mathbf{F}^T\mathbf{F} = \mathbf{I}, \quad \mathbf{G}\mathbf{G}^T = \mathbf{I}, \end{aligned} \tag{5.98}$$

where $\|\cdot\|_F$ is the Frobenius norm, $\mathbf{I}$ is an identity matrix defined as a diagonal matrix where diagonal elements are equal to 1, and $k, l \ll \min(m,n)$. $\mathbf{F}$, $\mathbf{S}$, and $\mathbf{G}$ are first initialized as random positive dense matrices. Then, they are updated according to three rules,

$$f_{ip} \leftarrow f_{ip}\sqrt{\frac{(\mathbf{A}\mathbf{G}^T\mathbf{S}^T)_{ip}}{(\mathbf{F}\mathbf{F}^T\mathbf{A}\mathbf{G}^T\mathbf{S}^T)_{ip}}} \tag{5.99}$$

$$g_{qj} \leftarrow g_{qj}\sqrt{\frac{(\mathbf{S}^T\mathbf{F}^T\mathbf{A})_{qj}}{(\mathbf{S}^T\mathbf{F}^T\mathbf{A}\mathbf{G}^T\mathbf{G})_{qj}}} \tag{5.100}$$

$$s_{pq} \leftarrow s_{pq}\sqrt{\frac{(\mathbf{F}^T\mathbf{A}\mathbf{G}^T)_{pq}}{(\mathbf{F}^T\mathbf{FSG}\mathbf{G}^T)_{pq}}} \tag{5.101}$$

until the relative square error (RSE) is less than a pre-defined threshold,

$$RSE = \frac{\|\mathbf{A}-\mathbf{FSG}\|_F^2}{\|\mathbf{A}\|_F^2}. \tag{5.102}$$

The correctness and convergency of the updating algorithm can be found in [34].

5.2.2.1 Goal of This Work

The main challenge for monitoring TAGs is the large size of the graphs. Fig.5.13 shows the size of the HTTP TAGs in the WIDE traces. The dimension of the matrix $\mathbf{A}$ increases up to 10^4

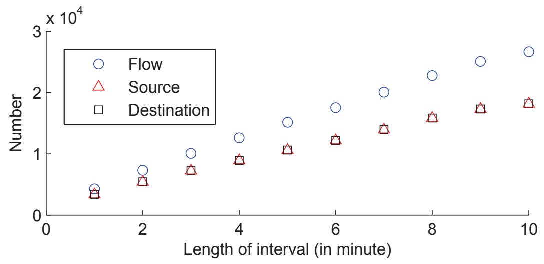


Figure 5.13 The Size of HTTP TAGs

for an interval less than 5 minutes. Because the computation cost of the updating step in the tNMF method is $O(mn)$, it is very difficult to discover and analyze the communication patterns in large TAGs. Usually, we can only work on an approximation of the original TAGs. A low-rank approximation of the matrix $\mathbf{A}$ is useful for many applications including the communication pattern analysis. Because the matrices $\mathbf{F}$, $\mathbf{S}$ and $\mathbf{G}$ in the tNMF method are also low-rank, a low-rank approximation of $\mathbf{A}$ can provide enough information to find the communication patterns in the TAGs.

In this section, we want to find a low-rank matrix $\hat{\mathbf{A}}$ such that the error $\|\hat{\mathbf{A}} - \mathbf{A}\|_F$ is small.

Problem 1. Given a set of sampled packets, we want to find a low-rank approximation of the adjacency matrix $\mathbf{A}$ of a TAG, such that

$$\begin{aligned} \text{minimize} \quad & E\left(\|\hat{\mathbf{A}} - \mathbf{A}\|_F^2\right), \\ \text{subject to} \quad & rank(\hat{\mathbf{A}}) \leq k \end{aligned} \tag{5.103}$$

where $E(\cdot)$ denotes the expectation, $\|\mathbf{A}\|_F^2 = \sum_{i,j} a_{ij}^2$ is the Frobenius norm, and k is the maximum possible rank for $\hat{\mathbf{A}}$.

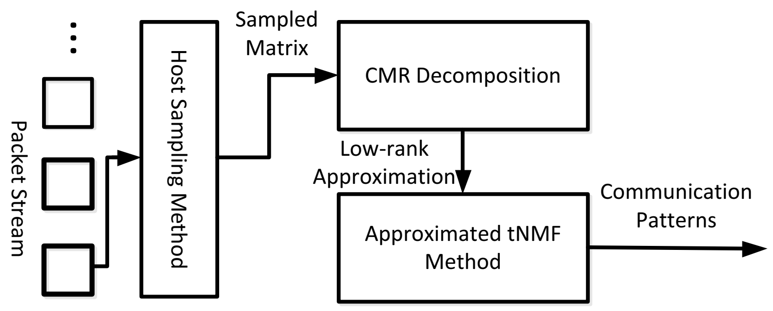


Figure 5.14 The Framework of Our Method

5.2.3 Our Method

We first propose our sampling method that can provide a high *host* coverage and uniformly sampled rows and columns from the matrix $\mathbf{A}$. Next, we compute the approximation matrix $\hat{\mathbf{A}}$ based on the CMR decomposition, where $\mathbf{M}$ is computed based on the sampled rows and columns. Last, we discuss the approximated tNMF method with the low-rank matrix $\hat{\mathbf{A}}_k$. The framework of our method is shown in Fig.5.14.

5.2.3.1 Host Sampling

Although the sampling methods have been widely studied in the traffic analysis and the data stream computation, most of previous methods are designed to sample individual items (e.g. packets), and are not used to sample a subset of items that is defined by a specific property (e.g. the same source address). cSAMP uses the hash-based sampling methods to sample all the packets in the same flow which are identified by the flow ID (srcIP, dstIP, srcPort, dstPort, protocol). We also use the hash-based method to sample packets from or to the same host which are identified by the IP addresses.

Let ρ denote the desired sampling rate for each flow, $0 < \rho < 1$. We use two independent hash functions, denoted by $h_s(\cdot)$ and $h_d(\cdot)$, for the source/destination IP, respectively. The hash function maps the key, i.e. srcIP and dstIP, into a value between 0 and 1. When a new packets comes, we first compute its hash values. If $h_s(srcIP) < \varrho$ or $h_d(dstIP) < \varrho$ with $\varrho = 1 - \sqrt{1-\rho}$, we will sample this packet. Otherwise, we skip this packet. The probability that a packet with the flow ID (srcIP, dstIP, srcPort, dstPort, protocol) is sampled will equal to,

$$\begin{aligned}
& P\left((srcIP, dstIP, srcPort, dstPort, protocol)\right. \\
& \qquad \left.\text{is sampled}\right) \\
= & \; P\left(h_s(srcIP) < \varrho\right) + P\left(h_d(dstIP) < \varrho\right) \\
& \; -P\left(h_s(srcIP) < \varrho \text{ and } h_d(dstIP) < \varrho\right) \\
= & \; (1-\sqrt{1-\rho}) + (1-\sqrt{1-\rho}) - (1-\sqrt{1-\rho})^2 \\
= & \; \rho. \qquad (5.104)
\end{aligned}$$

Therefore, each flow is sampled with a probability ρ. Our sampling method, referred as hSamp, provides a uniform sample set of the flows as the same as cSamp. The sampled packets can be used for other applications, e.g. traffic engineering, anomaly detection, etc.

Because the flows in the same row of $\mathbf{A}$ will have the same srcIP, all the packets from a random set of hosts are sampled. Thus, we can get a random set of rows in $\mathbf{A}$. Similarly, all the packets to a random set of hosts are sampled, and we can also get a random set of columns in $\mathbf{A}$. This property makes the hSamp different from the cSamp. Each flow in the cSamp is sampled independently. However, the flows connected to the same host will be sampled together in the hSamp. In other words, the hSamp remains the correlations of the flows which are important to discover the communities in the TAGs. If the hosts in a community are sampled, the flows within the community will be sampled with a higher probability than the other flows. Otherwise, the hSamp will miss most of the flows in a community. Approximately speaking, the hSamp tends to randomly sample communities in the TAGs rather than individual flows.

5.2.3.2 CMR Decomposition

Matrix decompositions have a wide range of applications. SVD/PCA is perhaps the most well-known method. For any matrix $\mathbf{A} \in \mathcal{R}^{m\times n}$, let $\mathbf{a}_{i*}$ denote the i-th row, and $\mathbf{a}_{*j}$ denote the j-th column. There exist orthogonal matrices $\mathbf{U} \in \mathcal{R}^{m\times k}$ and $\mathbf{V} \in \mathcal{R}^{n\times k}$ such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \qquad (5.105)$$

where $\mathbf{\Sigma} \in \mathcal{R}^{k \times k}$ is a diagonal matrix with nonnegative real numbers $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k > 0$ on the diagonal, and $\mathbf{u}_i \in \mathcal{R}^m$ and $\mathbf{v}_i \in \mathcal{R}^n$ are the i-th pair of singular vectors, i.e. $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$. The optimal rank-$r$ approximation of $\mathbf{A}$ with $r \leq k$ can be computed as

$$\mathbf{A}_r = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T = \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T, \tag{5.106}$$

where $\mathbf{\Sigma}_r$ is a diagonal matrix with the first r largest singular vales, $\mathbf{\Sigma}_r = diag\{\sigma_1, \ldots, \sigma_r, 0, \ldots, 0\}$. The matrix $\mathbf{A}_r$ minimize the Frobenius norm of the errors,

$$\|\mathbf{A} - \mathbf{A}_r\|_F = \min_{rank(\mathbf{X}) \leq r} \|\mathbf{A} - \mathbf{X}\|_F \tag{5.107}$$

The computation of the SVD is very high for a large matrix $\mathbf{A}$, which requires $O(mn^2)$.

An efficient method for the matrix decomposition is the CUR decomposition [37],

$$\underbrace{\begin{pmatrix} \\ \\ \quad\quad \mathbf{A} \quad\quad \\ \\ \\ \end{pmatrix}}_{m \times n} \simeq \underbrace{\begin{pmatrix} \\ \\ \ \mathbf{C} \ \\ \\ \\ \end{pmatrix}}_{m \times \alpha} \underbrace{\begin{pmatrix} \\ \ \mathbf{U} \ \\ \\ \end{pmatrix}}_{\alpha \times \beta} \underbrace{\begin{pmatrix} \\ \quad\quad \mathbf{R} \quad\quad \\ \\ \end{pmatrix}}_{\beta \times n} \tag{5.108}$$

where $\mathbf{C}$ is a set of columns in $\mathbf{A}$ and $\mathbf{R}$ is a set of rows in $\mathbf{A}$. The matrix $\mathbf{U}$ is a dense matrix, which is used to reconstruct the matrix like $\mathbf{\Sigma}$ in the SVD decomposition. Previous researches have found that that singular vectors can be preserved in the sampled rows and columns.

Our method is based on the above two methods. We propose the CMR decomposition by using uniform sampled rows and columns in the matrix $\mathbf{A}$. Given the packets from the host-based sampling method, we can construct the row/column matrices, i.e., $\mathbf{C}$ and $\mathbf{R}$, for the TAG of interest. For convenience, we assume $\mathbf{C}, \mathbf{R} \in \mathcal{R}^{m \times n}$.

If the j-th column in $\mathbf{A}$ is sampled, let $\mathbf{c}_{*j} = \varrho^{-1}\mathbf{a}_{*j}$. Otherwise, $\mathbf{c}_{*j} = \mathbf{0}$. The relation between $\mathbf{C}$ and $\mathbf{A}$ can be written as

$$\mathbf{C} = \mathbf{A}\mathbf{Q} \tag{5.109}$$

where $\mathbf{Q} \in \mathcal{R}^{n\times n}$ is a diagonal sampling matrix with

$$\begin{aligned} q_{ii} &= \begin{cases} \varrho^{-1} & \text{with a probability } \varrho \\ 0 & \text{otherwise} \end{cases} \\ q_{ij} &= 0 \text{ for } \forall i \neq j. \end{aligned} \tag{5.110}$$

Apparently, we have

$$E(\mathbf{Q}) = \mathbf{I}. \tag{5.111}$$

Similarly, the relation between $\mathbf{R}$ and $\mathbf{A}$ can be written as

$$\mathbf{R} = \mathbf{PA} \tag{5.112}$$

where $\mathbf{P} \in \mathcal{R}^{m\times m}$ is a diagonal sampling matrix with the same probability distribution as $\mathbf{P}$.

The matrix $\mathbf{M}$ can be computed as

$$\mathbf{M} = (\mathbf{PAQ})^{-}, \tag{5.113}$$

where the matrix $\mathbf{PAQ}$ includes the common elements in both $\mathbf{C}$ and $\mathbf{R}$, which are from the flow with $h_s(srcIP) < \varrho$ and $h_d(dstIP) < \varrho$. Here, $\mathbf{X}^{-}$ denotes the Moore-Penrose pseudo-inverse of the matrix $\mathbf{X}$,

$$\mathbf{X}^{-} = \mathbf{V}_X \mathbf{\Sigma}_X^{-1} \mathbf{U}_X^T \tag{5.114}$$

where $\mathbf{U}_X \mathbf{\Sigma} \mathbf{V}^T$ is the SVD of $\mathbf{X}$.

Last, we compute the low-rank approximation of $\mathbf{A}$ as

$$\hat{\mathbf{A}}_k = \mathbf{C}\mathbf{M}_k\mathbf{R}. \tag{5.115}$$

where $\mathbf{M}_k$ is the optimal rank-k approximation of $\mathbf{M}$.

5.2.3.3 Approximated tNMF

We find the network-wide communication patterns by solving the following problem.

$$\begin{aligned} \min_{\mathbf{F}\geq 0, \mathbf{S}\geq 0, \mathbf{G}\geq 0} \quad & J(\mathbf{F}, \mathbf{S}, \mathbf{G}) = \|\hat{\mathbf{A}}_k - \mathbf{FSG}\|_F^2, \\ s.t. \quad & \mathbf{F}^T\mathbf{F} = \mathbf{I}, \quad \mathbf{G}\mathbf{G}^T = \mathbf{I}. \end{aligned} \tag{5.116}$$

In other words, we try to find the tNMF matrices of the matrix $\mathbf{A}$ which only approximates a matrix $\hat{\mathbf{A}}_k$ that is close to $\mathbf{A}$ but with at most rank k. Thus we cannot track all communication patterns, but only track the “core” communication patterns.

Let $\mathbf{M}_k = \mathbf{U}\mathbf{\Sigma}_k^{-1}\mathbf{V}^T$ be the SVD of $\mathbf{M}_k$, and the matrix $\hat{\mathbf{A}}_k$ can be replaced by its low-rank approximation,

$$\hat{\mathbf{A}}_k = \mathbf{C}\mathbf{U}\mathbf{\Sigma}_k^{-1}\mathbf{V}^T\mathbf{R} = \mathbf{C}_k\mathbf{\Sigma}_k^{-1}\mathbf{R}_k^T \tag{5.117}$$

where $\mathbf{C}_k = \mathbf{C}\mathbf{U}$ and $\mathbf{R}_k = \mathbf{R}^T\mathbf{V}$. The updating process can be done as following

$$f_{ip} \leftarrow f_{ip}\sqrt{\frac{(\mathbf{C}_k\mathbf{\Sigma}_k^{-1}\mathbf{V}_k^T\mathbf{G}^T\mathbf{S}^T)_{ip}}{(\mathbf{F}\mathbf{F}^T\mathbf{C}_k\mathbf{\Sigma}_k^{-1}\mathbf{R}_k^T\mathbf{G}^T\mathbf{S}^T)_{ip}}} \tag{5.118}$$

$$g_{qj} \leftarrow g_{qj}\sqrt{\frac{(\mathbf{S}^T\mathbf{F}^T\mathbf{C}_k\mathbf{\Sigma}_k^{-1}\mathbf{R}_k^T)_{qj}}{(\mathbf{S}^T\mathbf{F}^T\mathbf{C}_k\mathbf{\Sigma}_k^{-1}\mathbf{R}_k^T\mathbf{G}^T\mathbf{G})_{qj}}} \tag{5.119}$$

$$s_{pq} \leftarrow s_{pq}\sqrt{\frac{(\mathbf{F}^T\mathbf{C}_k\mathbf{\Sigma}_k^{-1}\mathbf{R}_k^T\mathbf{G}^T)_{pq}}{(\mathbf{F}^T\mathbf{F}\mathbf{S}\mathbf{G}\mathbf{G}^T)_{pq}}}, \tag{5.120}$$

By doing this, the computation complexity of the updating process can be reduced from $O(mn)$ to $O(m+n)$ with $k \ll m, n$.

The matrix multiplications must be done in a smart way in order to utilize the low-rank property to reduce the computation overhead. In order to update $\mathbf{F}$, we need to compute $\mathbf{A}\mathbf{G}^T\mathbf{S}^T$ and $\mathbf{F}\mathbf{F}^T\mathbf{A}\mathbf{G}^T\mathbf{S}^T$, which requires at least $O(mn)$ products. With the low-rank approximation, we can update $\mathbf{F}$ as

$$\hat{\mathbf{A}}_k\mathbf{G}^T\mathbf{S}^T = \mathbf{C}_k(\mathbf{\Sigma}_k^{-1}(\mathbf{R}_k\mathbf{G}^T)\mathbf{S}^T) \tag{5.121}$$

$$\mathbf{F}\mathbf{F}^T\hat{\mathbf{A}}_k\mathbf{G}^T\mathbf{S}^T = (\mathbf{F}(\mathbf{F}^T\mathbf{C}_k))(\mathbf{\Sigma}_k^{-1}(\mathbf{R}_k\mathbf{G}^T)\mathbf{S}^T) \tag{5.122}$$

both of which require $O(m+n)$ products. Similarly, $\mathbf{S}^T\mathbf{F}^T\hat{\mathbf{A}}$ and $\mathbf{S}^T\mathbf{F}^T\hat{\mathbf{A}}\mathbf{G}^T\mathbf{G}$ can also be computed in $O(m+n)$ as

$$\mathbf{S}^T\mathbf{F}^T\hat{\mathbf{A}}_k = \mathbf{S}^T(\mathbf{F}^T\mathbf{C}_k)\mathbf{\Sigma}_k^{-1}\mathbf{R}_k \tag{5.123}$$

$$\mathbf{S}^T\mathbf{F}^T\hat{\mathbf{A}}_k\mathbf{G}^T\mathbf{G} = \mathbf{S}^T(\mathbf{F}^T\mathbf{C}_k)\mathbf{\Sigma}_k^{-1}(\mathbf{R}_k\mathbf{G}^T)\mathbf{G}. \tag{5.124}$$

And $\mathbf{F}^T\hat{\mathbf{A}}\mathbf{G}^T$ and $\mathbf{F}^T\mathbf{F}\mathbf{S}\mathbf{G}\mathbf{G}^T$ are computed as

$$\mathbf{F}^T\hat{\mathbf{A}}\mathbf{G}^T = (\mathbf{F}^T\mathbf{C}_k)\mathbf{\Sigma}_k^{-1}(\mathbf{R}_k\mathbf{G}^T) \tag{5.125}$$

$$\mathbf{F}^T\mathbf{F}\mathbf{S}\mathbf{G}\mathbf{G}^T = (\mathbf{F}^T\mathbf{F})\mathbf{S}(\mathbf{G}\mathbf{G}^T) \tag{5.126}$$

which requires $O(m+n)$.

5.2.4 Theoretical Analysis

Given a sampling rate ϱ, we show that the approximation matrix $\hat{\mathbf{A}}$ is close to the original matrix $\mathbf{A}$ with a high probability. First, the product of any two matrices $\mathbf{A}$ and $\mathbf{B}$ can be approximated by random sampled rows and columns. In this section, we only provide the main results and skip their proofs due to the page limitation.

Lemma 23. Given two matrices $\mathbf{A} \in \mathcal{R}^{m\times n}$ and $\mathbf{B} \in \mathcal{R}^{n\times t}$, we have

$$E(\mathbf{ADB}) = \mathbf{AB} \tag{5.127}$$

$$P(\|\mathbf{ADB}-\mathbf{AB}\|_F^2 \geq \epsilon\|\mathbf{A}\|_F^2\|\mathbf{B}\|_F^2) \leq \frac{\varrho^{-1}-1}{\epsilon n} \tag{5.128}$$

where $\epsilon > 0$ and $\mathbf{D} \in \mathcal{R}^{n\times n}$ is a sampling matrix that has $d_{ii} = \varrho^{-1}$ with a probability ϱ on the diagonal.

Proof. First, we have

$$E(\mathbf{ADB}) = \mathbf{A}E(\mathbf{D})\mathbf{B} = \mathbf{A}(\mathbf{I})\mathbf{B} = \mathbf{AB}. \tag{5.129}$$

Next, let $\mathbf{Z} = \mathbf{AB}$ and $\hat{Z} = \mathbf{ADB}$.

$$z_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \tag{5.130}$$

$$\hat{z}_{ij} = \sum_{k=1}^{n} d_{kk}a_{ik}b_{kj} \tag{5.131}$$

Because d_{kk} is independent random variable, we have

$$\begin{aligned} E\left((z_{ij}-\hat{z}_{ij})^2\right) &= E\left((\sum_{k=1}^{n}(d_{kk}-1)a_{ik}b_{kj})^2\right) \\ &= \sum_{k=1}^{n} E\left((d_{kk}-1)^2\right)a_{ik}^2 b_{kj}^2 \\ &= \sum_{k=1}^{n}(\varrho^{-1}-1)a_{ik}^2 b_{kj}^2 \\ &\geq \frac{\varrho^{-1}-1}{n}(\sum_k a_{ik}^2)(\sum_k b_{kj}^2) \\ &= \frac{\varrho^{-1}-1}{n}\|\mathbf{a}_{i*}\|^2\|\mathbf{b}_{*j}\|^2 \end{aligned} \tag{5.132}$$

Therefore,

$$\begin{aligned} E\left(\|\mathbf{ADB}-\mathbf{AB}\|_F^2\right) &\geq \frac{\varrho^{-1}-1}{n}\sum_{i,j}\|\mathbf{a}_{i*}\|^2\|\mathbf{b}_{*j}\|^2 \\ &= \frac{\varrho^{-1}-1}{n}\|\mathbf{A}\|_F^2\|\mathbf{B}\|_F^2 \end{aligned} \quad (5.133)$$

Based on Markov's inequality, we have

$$P(\|\mathbf{ADB}-\mathbf{AB}\|_F^2 \geq \epsilon\|\mathbf{A}\|_F^2\|\mathbf{B}\|_F^2) \leq \frac{\varrho^{-1}-1}{\epsilon n} \quad (5.134)$$

□

Next, we solve the regression problem based on sampled rows, which will be used to provide the error bound for our CMR decomposition.

Lemma 24. Given two matrices $\mathbf{A} \in \mathcal{R}^{m\times n}$ and $\mathbf{B} \in \mathcal{R}^{m\times n}$, let $\tilde{\mathbf{X}}_{opt}$ be the solution to

$$\min_{\mathbf{X}} \|\mathbf{D}(\mathbf{AX}-\mathbf{B})\|_F, \quad (5.135)$$

and $\mathbf{X}_{opt}$ be the solution to

$$\min_{\mathbf{X}} \|\mathbf{AX}-\mathbf{B}\|_F, \quad (5.136)$$

where $\mathbf{D} \in \mathcal{R}^{m\times m}$ is a sampling matrix that has $d_{ii} = \varrho^{-1}$ with a probability ϱ on the diagonal. We have

$$P\left(\|\mathbf{A}\tilde{\mathbf{X}}_{opt}-\mathbf{B}\|_F^2 \geq (1+\epsilon)\|\mathbf{AX}_{opt}-\mathbf{B}\|_F^2\right) \leq \frac{2(\varrho^{-1}-1)}{\epsilon m}. \quad (5.137)$$

Proof. Based on the normal equation of (5.136), we have

$$\mathbf{A}^T(\mathbf{AX}_{opt}-\mathbf{B}) = 0. \quad (5.138)$$

Therefore, we have

$$\|\mathbf{A}\tilde{\mathbf{X}}_{opt}-\mathbf{B}\|_F^2 = \|\mathbf{AX}_{opt}-\mathbf{B}\|_F^2 + \|\mathbf{A}(\tilde{\mathbf{X}}_{opt}-\mathbf{X}_{opt})\|_F^2 \quad (5.139)$$

Let $\mathbf{A} = \mathbf{U\Sigma V}^T$ denote the SVD of $\mathbf{A}$ and $k = rank(\mathbf{A})$. Because $\mathbf{U}$ is in the column space of $\mathbf{A}$, we have

$$\mathbf{U}^T(\mathbf{AX}_{opt}-\mathbf{B}) = 0. \quad (5.140)$$

Based on the normal equation of (5.135), we have

$$\mathbf{A}^T\mathbf{D}(\mathbf{A}\tilde{\mathbf{X}}_{opt} - \mathbf{B}) = 0. \tag{5.141}$$

$$\mathbf{U}^T\mathbf{D}(\mathbf{A}\tilde{\mathbf{X}}_{opt} - \mathbf{B}) = 0. \tag{5.142}$$

Thus, we have

$$\begin{aligned}\mathbf{U}^T\mathbf{D}\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt}) &= \mathbf{U}^T\mathbf{D}(\mathbf{A}\tilde{\mathbf{X}}_{opt} - \mathbf{B}) \\ &\quad + \mathbf{U}^T\mathbf{D}(\mathbf{B} - \mathbf{A}\mathbf{X}_{opt}) \\ &= \mathbf{U}^T\mathbf{D}(\mathbf{B} - \mathbf{A}\mathbf{X}_{opt})\end{aligned} \tag{5.143}$$

Based on Lemma 1 and (5.140), we have

$$\begin{aligned}&P\left(\|\mathbf{U}^T\mathbf{D}(\mathbf{A}\mathbf{X}_{opt} - \mathbf{B})\|_F^2 \geq \epsilon\|\mathbf{A}\mathbf{X}_{opt} - \mathbf{B}\|_F^2\right) \\ &\quad \leq \frac{(\varrho^{-1} - 1)\|\mathbf{U}\|_F^2}{\epsilon m} \\ &\quad \leq \frac{(\varrho^{-1} - 1)k}{\epsilon m}\end{aligned} \tag{5.144}$$

Therefore, we have

$$\begin{aligned}&P\left(\|\mathbf{U}^T\mathbf{D}\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2 \geq \epsilon\|\mathbf{A}\mathbf{X}_{opt} - \mathbf{B}\|_F^2\right) \\ &\qquad \leq \frac{(\varrho^{-1} - 1)k}{\epsilon m}.\end{aligned} \tag{5.145}$$

Based on Lemma 1, we have

$$\begin{aligned}&P\Big(\|\mathbf{U}^T\mathbf{D}\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt}) - \mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2 \\ &\qquad \geq \epsilon\|\mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2\Big) \leq \frac{(\varrho^{-1} - 1)}{\epsilon m}.\end{aligned} \tag{5.146}$$

Thus, we have

$$\begin{aligned}&\|\mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2 \\ &\quad \leq \|\mathbf{U}^T\mathbf{D}\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt}) - \mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2 \\ &\qquad + \|\mathbf{U}^T\mathbf{D}\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2 \\ &\quad \leq \epsilon_1\|\mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt} - \mathbf{X}_{opt})\|_F^2 + \epsilon_2\|\mathbf{A}\mathbf{X}_{opt} - \mathbf{B}\|_F^2\end{aligned} \tag{5.147}$$

with a probability at least

$$\left(1-\frac{(\varrho^{-1}-1)}{\epsilon_1 m}\right)\left(1-\frac{(\varrho^{-1}-1)k}{\epsilon_2 m}\right). \tag{5.148}$$

Therefore, we have

$$\begin{aligned} P\left(\|\mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt}-\mathbf{X}_{opt})\|_F^2 \geq \frac{\epsilon_2}{1-\epsilon_1}\|\mathbf{A}\mathbf{X}_{opt}-\mathbf{B}\|_F^2\right) \\ \leq \frac{(\varrho^{-1}-1)}{\epsilon_1 m}+\frac{(\varrho^{-1}-1)k}{\epsilon_2 m}. \end{aligned} \tag{5.149}$$

Because

$$\begin{aligned} \|\mathbf{U}^T\mathbf{A}(\tilde{\mathbf{X}}_{opt}-\mathbf{X}_{opt})\|_F^2 &= \|\mathbf{U}^T\|_F^2\|\mathbf{A}(\tilde{\mathbf{X}}_{opt}-\mathbf{X}_{opt})\|_F^2 \\ &= k\|\mathbf{A}(\tilde{\mathbf{X}}_{opt}-\mathbf{X}_{opt})\|_F^2 \end{aligned} \tag{5.150}$$

By choosing $\epsilon = 2\epsilon_2$ and $\epsilon_1 = 0.5$, we have

$$\begin{aligned} P\left(\|\mathbf{A}(\tilde{\mathbf{X}}_{opt}-\mathbf{X}_{opt})\|_F^2 \geq \epsilon\|\mathbf{A}\mathbf{X}_{opt}-\mathbf{B}\|_F^2\right) \\ \leq \frac{2(\varrho^{-1}-1)}{mk}+\frac{\varrho^{-1}-1}{\epsilon m}. \end{aligned} \tag{5.151}$$

Thus, we get

$$P\left(\|\mathbf{A}\tilde{\mathbf{X}}_{opt}-\mathbf{B}\|_F^2 \geq (1+\epsilon)\|\mathbf{A}\mathbf{X}_{opt}-\mathbf{B}\|_F^2\right) \leq \frac{2(\varrho^{-1}-1)}{\epsilon m}. \tag{5.152}$$

□

Last, we show that $\hat{\mathbf{A}}$ is close to $\mathbf{A}$ with a high probability.

Theorem 16. Let $\mathbf{C}$ be the matrix of the random sampled rows and $\mathbf{R}$ be the matrix of random sampled columns, from the matrix $\mathbf{A} \in \mathcal{R}^{m\times n}$. And $\mathbf{M}$ is the pseudo-inverse of the matrix of the common elements in both $\mathbf{C}$ and $\mathbf{R}$. The approximation matrix is computed as $\hat{\mathbf{A}} = \mathbf{CMR}$. We have

$$\|\hat{\mathbf{A}}-\mathbf{A}\|_F^2 \leq \epsilon(1+\epsilon)\|\mathbf{A}\|_F^2 \tag{5.153}$$

with a probability at least $1-\frac{2(\varrho^{-1}-1)}{\epsilon m}-\frac{\varrho^{-1}-1}{\epsilon n}$.

Proof. Let $\mathbf{X}_{opt}$ be the solution to

$$\min_{\mathbf{X}}\|\mathbf{PAQX}-\mathbf{PA}\|. \tag{5.154}$$

Thus, we have

$$\mathbf{X}_{opt} = (\mathbf{PAQ})^{-}\mathbf{PA}. \tag{5.155}$$

Based on Lemma 2, we have

$$\|\mathbf{AQX}_{opt} - \mathbf{A}\|_F^2 \leq (1+\epsilon)\|\mathbf{AQY}_{opt} - \mathbf{A}\| \tag{5.156}$$

with a probability at least $1 - \frac{2(\varrho^{-1}-1)}{\epsilon m}$, where $\mathbf{Y}_{opt}$ is the solution to

$$\min_{\mathbf{Y}} \|\mathbf{AQY} - \mathbf{A}\|. \tag{5.157}$$

By using Lemma 1, we have

$$\begin{aligned} \|\mathbf{AQY}_{opt} - \mathbf{A}\|_F^2 &= \min_{\mathbf{Y}} \|\mathbf{AQY} - \mathbf{A}\|_F^2 \\ &\leq \|\mathbf{AQ} - \mathbf{A}\|_F^2 \leq \epsilon\|\mathbf{A}\|, \end{aligned} \tag{5.158}$$

with a probability at least $1 - \frac{\varrho^{-1}-1}{\epsilon n}$.

Therefore, we have

$$\begin{aligned} \|\hat{\mathbf{A}} - \mathbf{A}\|_F^2 &= \|\mathbf{AQX}_{opt} - \mathbf{A}\|_F^2 \\ &\leq (1+\epsilon)\|\mathbf{AQY}_{opt} - \mathbf{A}\|_F^2 \\ &\leq \epsilon(1+\epsilon)\|\mathbf{A}\|_F^2 \end{aligned} \tag{5.159}$$

with a probability at least $1 - \frac{2(\varrho^{-1}-1)}{\epsilon m} - \frac{\varrho^{-1}-1}{\epsilon n}$. □

Thus $\hat{\mathbf{A}} = \mathbf{CMR}$ is a good approximation of the matrix $\mathbf{A}$, and we use $\hat{\mathbf{A}}_k$ as the low-rank approximation of $\mathbf{A}$.

5.2.5 Evaluation

In this section, we evaluate our method with real network traffic traces. The WIDE traces were collected on Mar 18, 2008, which consisted of packets on a 150 Megabit Ethernet external link between WIDE backbone and its upstream. We use them to evaluate our method with the TAGs for known applications, e.g. HTTP, Email, etc. The other is a malware trace, which contains two P2P botnets, i.e., Nugache and Storm. The IP addresses of the hosts/honeypots which was running as a bot, were identified. We apply our method to extra the communication patterns of the malwares, which can used by the intrusion detection system.

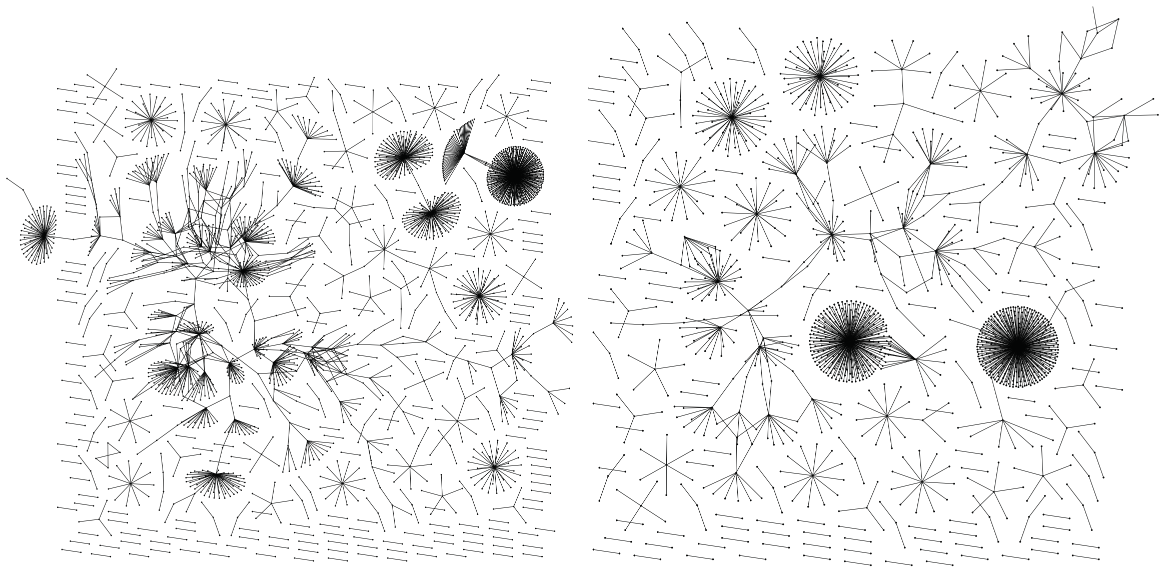


Figure 5.15 HTTP TAGs in five minutes with cSAMP (left) and hSAMP (right)

5.2.5.1 HTTP TAGs

We first use the HTTP TAGs (Port 80) to evaluate our method. The WIDE traces are sampled by hSAMP and cSAMP, respectively. Fig.5.15 shows the HTTP TAGs after cSAMP with $\rho = 0.1$ and hSAMP with $\rho = 0.1$, in a 5-minutes interval. We choose a larger sampling rate $\rho = 0.1$ than the typical value in practice in order to compare two methods. In this way, the original TAGs are small enough to be visualized, and the sampled TAGs are dense enough to remain useful information.

We can see that the community structure is clearer in the sampled TAGs by hSAMP than cSAMP, and the nodes have higher degrees in the sampled TAGs by hSAMP than cSAMP. In the cSAMP, each flow is sampled independently. The correlation among the hosts will be missed due to the low sampling rate, and thus the communication patterns in the TAGs will also become ambiguous. However, the hSAMP keeps all the connections from or to a random selected set of hosts. The correlations among these hosts are fully preserved, and the correlations among the other hosts are interpreted by the matrix decomposition method. Because the hosts connect to the hosts in the same community with a higher probability than the other hosts, the hSAMP samples the flows within the communities with a higher probability than the flows among the communities. Thus, the communication patterns can be preserved more visible by using hSAMP

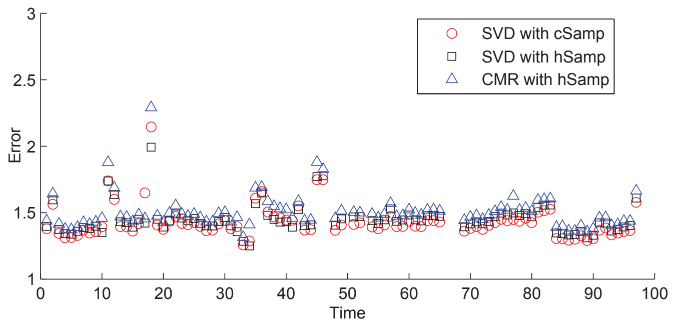


Figure 5.16 Approximation Errors with $k = 10$ in HTTP TAGs

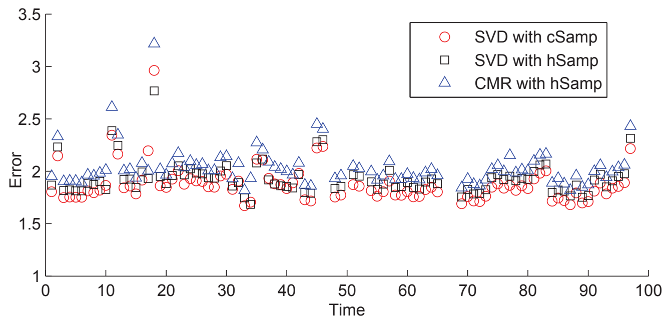


Figure 5.17 Approximation Errors with $k = 30$ in HTTP TAGs

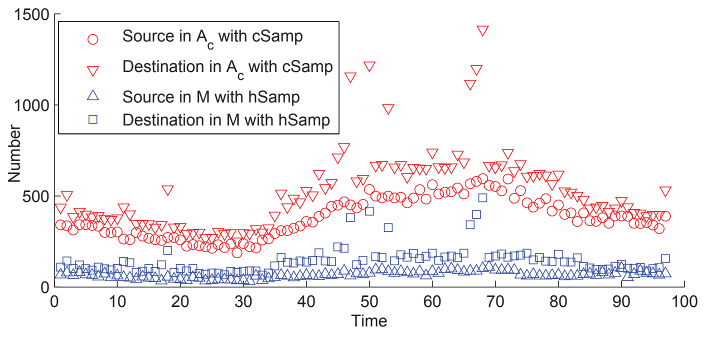


Figure 5.18 Number of Sources and Destinations in HTTP TAGs

than cSamp.

We compute the low-rank approximation by the SVD and the CMR based on the sampled packets. The results are compared by the approximation error to the optimal rank-k approximation of the original matrix $\mathbf{A}$,

$$e = \frac{\|\hat{\mathbf{A}}_k - \mathbf{A}\|_F^2}{\|\mathbf{A}_k - \mathbf{A}\|_F^2}. \tag{5.160}$$

The TAGs are constructed by using the active flows in one minute. The the size of the TAGs in one minute is small enough for us to compute the SVD of the original matrix $\mathbf{A}$. And the TAGs are constructed for different time.

We compute a low-rank approximation of the HTTP TAGs with the rank $k = 10$ and $k = 30$, as shown in Fig.5.16 and Fig.5.17. The errors with the CMR and the SVD are very close. We can see that the hSamp can remain the same information as the cSamp. The CMR decomposition can find a good approximation of the original matrix $\mathbf{A}$ by using the property of the host sampling method. Furthermore, the computation cost for the SVD with cSamp is much higher than the computation cost for the CMR decomposition, because the size of the matrix $\mathbf{M}$ is much smaller than the sampled matrix $\mathbf{A}_c$ in the cSamp, as shown in Fig.5.18.

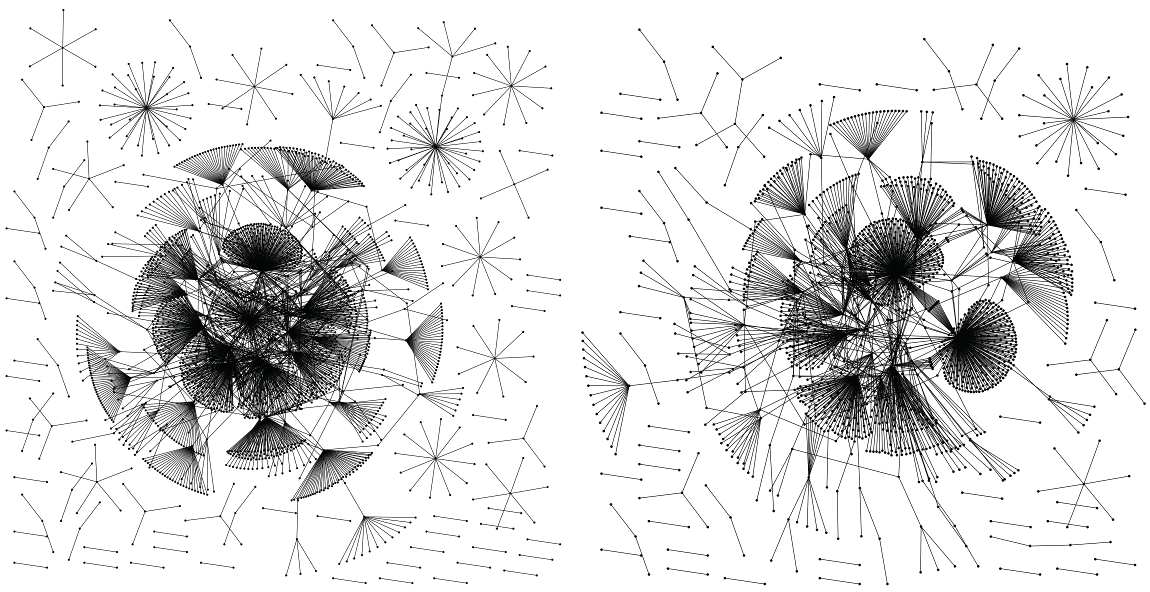


Figure 5.19 Email TAGs in five minutes with cSamp (left) and hSamp (right)

5.2.5.2 Email TAGs

We do a similar analysis on the Email TAGs (Port 25). We show the Email TAGs in a 5-minutes interval with both sampling methods in Fig.5.19. Both TAGs look similar, but the cSamp cause more disconnected subgraphs in the TAGs than the hSamp. Fig.5.20 and Fig.5.21 show the errors in the Email TAGs in an one-minute interval with the rank $k = 10$ and $k = 30$, respectively. Although the topologies of the Email TAGs are quite different for the HTTP TAGs, the errors between the CMR and the SVD are still very close. Therefore, we can see that the CMR decomposition can work on various TAGs.

We also find some persistent communication patterns in the Email TAGs. If a community appears in the TAGs with a high frequency, it will be very useful to reveal some common user behavior patterns. We apply our method on the Email TAGs in the WIDE trace. There are 4 hosts which appeared in the same communities with a high frequency in the WIDE traces. All of these 4 hosts are listed in the Policy Block List of the spam email. By checking the short payload in the packets sent or received by these hosts, we find that most packets are related to the email address scanning on Yahoo Mail.

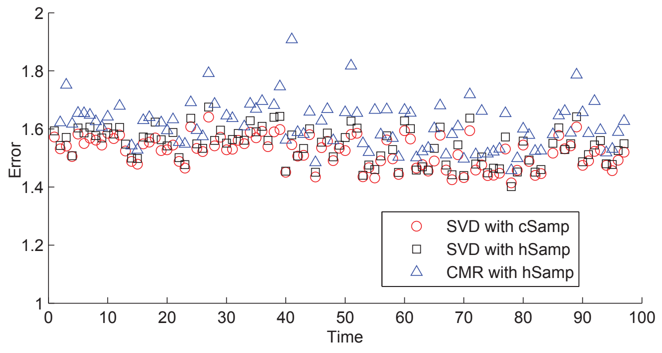


Figure 5.20 Approximation Errors with $k = 10$ in Email TAGs

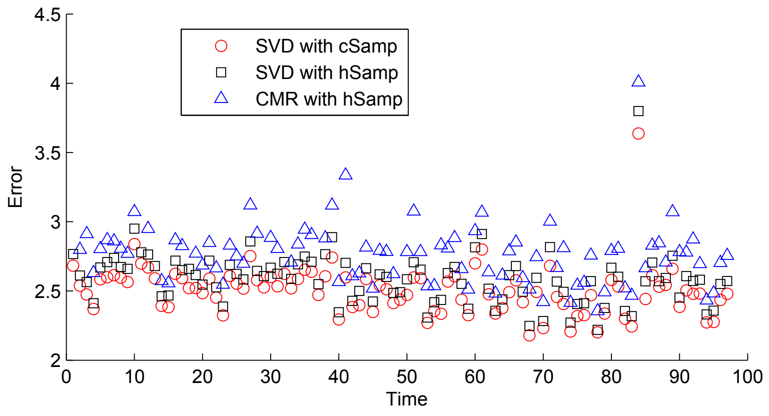


Figure 5.21 Approximation Errors with $k = 30$ in Email TAGs

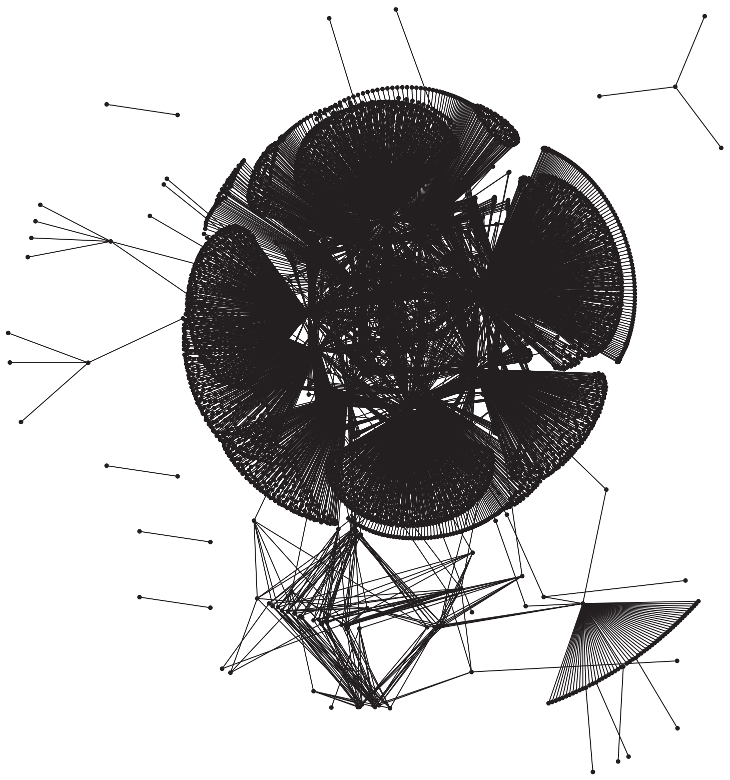


Figure 5.22 Malware TAGs in one minute

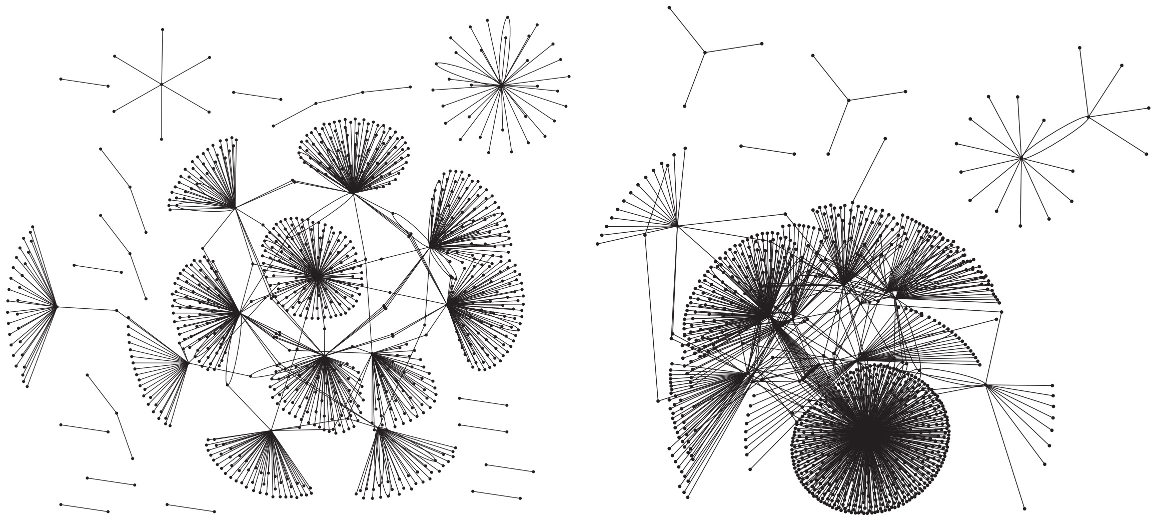


Figure 5.23 Malware TAGs in one minute with cSamp (left) and hSamp (right)

5.2.6 Malware TAGs

In the section, we analyze the TAGs from a traffic trace including botnet activities. Because botnets usually use dynamic ports to maintain connections in order to bypass the intrusion detection system, we consider all flows in the traffic trace. The original TAG in an one-minute interval is shown in Fig.5.22. Because each bot maintain connections with each other to exchange commands and updates, there is a clear community structure in their TAGs. We show the TAGs in a one-minutes interval with both sampling methods in Fig.5.23. We can see that the community of the botnet is preserved in hSamp, which is hard to be identified in cSamp. The community of the bots with cSamp become too sparse such that they look like normal users in the HTTP TAGs. However, in the TAGs with hSamp, we can still find a well-connected community in the TAGs, which are not present in HTTP or Email TAGs.

5.2.7 Conclusion

We propose a new framework to combine the packet sampling and the low-rank approximation together for monitoring communication patterns and community structures in the TAGs. Our method provides an efficient algorithm for the low-rank approximation tNMF method. In the future, we will design efficient algorithms to discover abnormal patterns in the TAGs, which can help us to detect traffic anomalies and network attacks.

CHAPTER 6. SUMMARY

6.1 Conclusion

In this dissertation, we design data streaming algorithms for large-scale cyber-attack detection. We consider a fundamental technique to process packets in a wire speed: hashing. We rely on hash functions to design data streaming algorithms with other techniques, e.g., time stamps, coding theory, group testing, low-rank matrix approximation, etc. Our proposed techniques can serve as fundamental components for the network traffic monitoring in general. Our main contributions are as follows:

- *Click Frauds*: We propose a near-optimal data structure to support approximate membership query over sliding windows, which can be used to detect click frauds. The space can be bounded by $O(n(\log 1/\delta + \log n))$ and the running time is bounded by $O(1)$ for both update and query.

- *Botnet C&C Communications*: We propose to use long duration flows (LDFs) to track the Command-and-Control flows in the botnets, and evaluate its performance with a real botnet trace. Our algorithms for tracking LDFs has been proved to be space-optimal for the flow duration estimation, and are more practical for the LDF detection than existing algorithms, which may report many short-lived flows as LDFs.

- *Super spreaders*: We design a mergeable and reversible sketch to detect hosts with a high cardinality in a distributed monitoring system. We can identify super spreaders with a small number of aggregated measurements.

- *PCA-based Traffic Anomalies*: The computation of PCA is $O(nm^2)$ and the space is $O(nm)$, which becomes a bottleneck for PCA-based anomaly detection. Our algorithm achieves $O(w \log n)$ running time and $O(w \log^2 n)$ space at local monitors. The NOC could run PCA-based detection method with $O(m^2 \log n)$ running time and $O(m \log n)$ space. Our algorithm also makes the ISPs be able to implement the detection method by paying careful consideration about the space requirement, the communication cost, and other resources over a distributed computing environment.

- *Traffic Activity Graphs*: We propose a new framework to combine the packet sampling and the low-rank approximation together for monitoring communication patterns and community structures in the Traffic Activity Graphs. Our method provides an efficient algorithm for the low-rank approximation tNMF method, which can be further used to identify traffic anomalies.

6.2 Future Work

Data streaming algorithms have a wide applications in the big data analysis. In this dissertation, we design several efficient algorithms for the cyber-attack detection in high-speed network traffic. The optimal algorithms for most of problems in this chapter are still unknown. There are also a lot of problems which still lack efficient algorithms, e.g. entropy estimation, matrix rank estimation, etc. We provide a brief overview of some open problems for the cyber-attack detection, which are not covered in this dissertation.

6.2.1 Entropy and Distributions

Entropy has been used as an important statistics for the traffic anomaly detection. Due to the complexity of the entropy computation, it is very challenging to estimate the entropy over data streams. Given a data stream $\{x_1, \ldots, x_i, \ldots\}$, the empirical probability distribution, denoted by p_x for $x \in \mathcal{U}$, is defined as $p_x = f_x/N$ where f_x is the frequency of the item x and N is the number of items, i.e. the size of the sliding window. The empirical entropy is defined as

$$H(p) = \sum_{x \in \mathcal{U}} -p_x \log p_x. \tag{6.1}$$

Lall et. al. [64] first studied the entropy estimation over the network traffic and shown that neither approximation nor randomization along would allow an efficient algorithm. Chakrabarti [23] et. al. provided a near-optimal algorithm for the entropy estimation over the entire history with $O(\frac{1}{\epsilon^2} \log \frac{R}{\delta})$ space. Their method can be extended to handle the sliding window model by using the bucket structure. The core idea in their algorithm is to draw a random sample according to a specific distribution from the data stream. There were also some works which aimed to test the properties of the distribution of the data stream [11]. If there are multiple data streams in a distributed monitoring system, we are interested to compute the distance between the distributions of multiple data streams. It is still unknown how to inference the distance between the distributions of multiple data streams [53; 16].

6.2.2 Linear Algebra Problems

Clarkson and Woodruff [27] introduced some linear algebra problems for the data stream computation. They considered the turnstile model of the data stream computation [71], in which each item is an update to entries of a matrix $\mathbf{A}$,

$$(i_0, j_0, x_0), \ldots, (i_t, j_t, x_t), \ldots, \tag{6.2}$$

where (i_t, j_t, x_t) denote an update to add x_t to the entry (i_t, j_t) in $\mathbf{A}$. They studied four problems, i.e. Matrix Product, Linear Regression, Rank-k Approximation, and Rank Decision, including their upper and lower bounds.

1. Matrix Product: the algorithm finds a matrix $\mathbf{C}$ with

$$\|\mathbf{A}^T\mathbf{B} - \mathbf{C}\| \leq \epsilon \|\mathbf{A}\| \|\mathbf{B}\|. \tag{6.3}$$

2. Linear Regression: Given a vector $\mathbf{b} \in \mathbb{R}^d$, the algorithm finds a vector $\mathbf{x} \in \mathbb{R}^d$ such that

$$\|\mathbf{A}^T\mathbf{x} - \mathbf{b}\| \leq (1+\epsilon) \min_{\mathbf{x}' \in \mathbb{R}^d} \|\mathbf{A}\mathbf{x}' - \mathbf{b}\|. \tag{6.4}$$

3. Rank-k Approximation: the algorithm finds a matrix $\hat{\mathbf{A}}_k$ of rank at most k such that

$$\|\mathbf{A} - \hat{\mathbf{A}}_k\| \leq (1+\epsilon) \|\mathbf{A} - \mathbf{A}_k\| \tag{6.5}$$

where $\mathbf{A}_k$ is the best rank-k approximation to $\mathbf{A}$.

4. Rank Decision: Given an integer k and a matrix $\mathbf{A}$, the algorithm outputs 1 if and only if the rank of $\mathbf{A}$ is at least k.

The lower bound for these problems was derived based on the communication complexity. They also provided a sketch for these problem by using the sign matrix, i.e. each entry is ± 1. So far, there has been no work to study linear algebra problems in the sliding window model or the distributed monitoring system.

BIBLIOGRAPHY

[ric] Rice dsp group. compressed sensing resource: http://dsp.rice.edu/cs.

[wit] The caida dataset on the witty worm - march 19-24, 2004, colleen shannon and david moore. support for the witty worm dataset and the ucsd network telescope are provided by cisco systems, limelight networks, the us department of homeland security, the national science foundation, and caida, darpa, digital, envoy, and caida members. http://www.caida.org/passive/witty/.

[1] Milton Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,*. Dover Publications, Incorporated, 1974. ISBN 0486612724.

[2] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[3] Charu C. Aggarwal. *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387287590.

[4] Charu C. Aggarwal. *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387287590.

[5] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. *PODS '99*, pages 10–20, 1999.

[6] Martin F. Arlitt and Carey L. Williamson. Web server workload characterization: the search for invariants. In *SIGMETRICS '96*, pages 126–137, 1996.

[7] Nikolas Askitis. Fast and compact hash tables for integer keys. In *ACSC '09*, pages 113–122, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.

[8] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS '02*, pages 1–16, New York, NY, USA, 2002. ACM. ISBN 1-58113-507-6.

[9] Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting Distinct Elements in a Data Stream. In *6th International Workshop on Randomization and Approximation Techniques in Computer Science*, Cambridge, Massachusetts, September 2002.

[10] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. *IMW '02*, pages 71–82, 2002.

[11] Tugkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In *STOC '04*, pages 381–390, 2004. ISBN 1-58113-852-0.

[12] Radu Berinde, Graham Cormode, Piotr Indyk, and Martin J. Strauss. Space-optimal heavy hitters with strong error bounds. In *PODS '09*, pages 157–166, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-553-6.

[13] Daniel K. Blandford and Guy E. Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Trans. Algorithms*, 4:17:1–17:25, 2008.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[15] D. Brauckhoff, K. Salamatian, and M. May. Applying pca for traffic anomaly detection: Problems and solutions. In *INFOCOM 2009, IEEE*, pages 2866–2870, 2009.

[16] Vladimir Braverman and Rafail Ostrovsky. Measuring independence of datasets. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 271–280, 2010. ISBN 978-1-4503-0050-6.

[17] Andrei Broder, Michael Mitzenmacher, and Andrei Broder I Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[18] Andrei Z. Broder, Marc Najork, and Janet L. Wiener. Efficient url caching for world wide web crawling. In *WWW '03*, pages 679–689, New York, NY, USA, 2003. ACM.

[19] Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[20] Emmanuel J. Candes, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

[21] Jin Cao, Yu Jin, Aiyou Chen, Tian Bu, and Zhi-Li Zhang. Identifying high cardinality internet hosts. In *INFOCOM '09, IEEE*, 2009.

[22] Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *STOC '78*, pages 59–65, 1978.

[23] Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for computing the entropy of a stream. In *SODA '07*, pages 328–335, 2007. ISBN 978-0-898716-24-5.

[24] Chun Lam Chan, Pak Hou Che, S. Jaggi, and V. Saligrama. Non-adaptive probabilistic group testing with noisy measurements: Near-optimal bounds with efficient algorithms. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 1832 –1839, sept. 2011.

[25] Aiyou Chen, Yu Jin, Jin Cao, and Li Li. Tracking long duration flows in network traffic. In *INFOCOM '10, IEEE*, 2010.

[26] P. Chhabra, C. Scott, E.D. Kolaczyk, and M. Crovella. Distributed spatial anomaly detection. *INFOCOM '08*, pages 1705–1713, 2008.

[27] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *STOC*, pages 205–214, 2009. ISBN 978-1-60558-506-2.

[28] J. G. Clerry. Compact hash tables using bidirectional linear probing. *IEEE Trans. Comput.*, 33:828–834, September 1984.

[29] M. Patrick Collins and Michael K. Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *RAID*, pages 276–295, Berlin, Heidelberg, 2007. Springer-Verlag.

[30] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. ISSN 0196-6774.

[31] Graham Cormode and S. Muthukrishnan. What's new: finding significant differences in network data streams. *IEEE/ACM Trans. Netw.*, 13(6):1219–1232, 2005. ISSN 1063-6692.

[32] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows: (extended abstract). *SODA '02*, pages 635–644, 2002.

[33] Fan Deng and Davood Rafiei. Approximately detecting duplicates for streaming data using stable bloom filters. In *SIGMOD '06*, pages 25–36, 2006. ISBN 1-59593-434-0.

[34] Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *KDD*, pages 126–135, 2006. ISBN 1-59593-339-5.

[35] Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. In *SODA*, pages 1190–1197, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-98-6.

[36] D.L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4): 1289–1306, 2006. ISSN 0018-9448.

[37] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices iii: Computing a compressed approximate matrix decomposition. *SIAM J. Comput.*, 36(1):184–206, 2006. ISSN 0097-5397.

[38] D.-Z. Du and F. K. Hwang. *Combinatorial Group Testing and its Applications*. Singapore: World Scientific, 1993.

[39] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, Budapest, Hungary, September 2003.

[40] Sourav Dutta, Souvik Bhattacherjee, and Ankur Narang. Towards intelligent compression in streams: A biased reservoir sampling based bloom filter approach. In *arXiv:1111.0753. IBM TECHREPORT RI11015*, Nov 2011.

[41] David Eppstein and Michael Goodrich. Space-efficient straggler identification in round-trip data streams via newtons identities and invertible bloom filters. In *Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 637–648. Springer Berlin / Heidelberg, 2007.

[42] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3): 270–313, 2003. ISSN 0734-2071.

[43] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[44] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the Analysis of a Near-optimal Cardinality Estimation Algorithm. In *The 2007 Conference on Analysis of Algorithms*, Juan des Pins, France, June 2007.

[45] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005. ISSN 0163-5808.

[46] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2008. ISBN 9780131873254.

[47] Xiaohong Guan, Pinghui Wang, and Tao Qin. A new data streaming method for locating hosts with large connection degree. In *GLOBECOM '09*, pages 1 –6, nov. 2009.

[48] Sudipto Guha, D. Gunopulos, and Nick Koudas. Correlating synchronous and asynchronous data streams. In *KDD '03*, pages 529–534, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0.

[49] Venkatesan Guruswami. Algorithmic results in list decoding. *Found. Trends Theor. Comput. Sci.*, 2(2):107–195, January 2007. ISSN 1551-305X.

[50] Ling Huang, Xuan Long Nguyen, M. Garofalakis, J.M. Hellerstein, M.I. Jordan, A.D. Joseph, and N. Taft. Communication-efficient online detection of network-wide anomalies. *INFOCOM '07*, pages 134–142, 2007.

[51] Yiyi Huang, Nick Feamster, Anukool Lakhina, and Jim (Jun) Xu. Diagnosing network disruptions with network-wide analysis. *SIGMETRICS '07*, pages 61–72, 2007.

[52] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *IMC*, pages 315–320, 2007. ISBN 978-1-59593-908-1.

[53] Piotr Indyk and Andrew McGregor. Declaring independence via the sketching of sketches. In *SODA '08*, pages 737–745, 2008.

[54] J E Jackson and G S Mudholkar. Control procedures for residuals associated with principal component analysis. *Thechnometrics*, pages 341–349, 1979.

[55] Yu Jin, Esam Sharafuddin, and Zhi-Li Zhang. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In *SIGMETRICS*, pages 49–60, 2009.

[56] Wolfgang John and Sven Tafvelin. Heuristics to classify internet backbone traffic based on connection patterns. In *ICOIN*, pages 1–5, 2008.

[57] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz-4-sale: surviving organized ddos attacks that mimic flash crowds. In *NSDI'05*, pages 287–300, Berkeley, CA, USA, 2005. USENIX Association.

[58] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS '10*, pages 41–52, New York, NY, USA, 2010. ACM.

[59] A. Kirsch and M. Mitzenmacher. The power of one move: Hashing schemes for hardware. In *INFOCOM '08*, pages 106 –110, april 2008.

[60] Jeff Kline, Sangnam Nam, Paul Barford, David Plonka, and Amos Ron. Traffic anomaly detection at fine time scales with bayes nets. *ICIMP '08*, pages 37–46, 2008.

[61] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *IMC '03*, pages 234–247, New York, NY, USA, 2003. ACM. ISBN 1-58113-773-7.

[62] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *SIGCOMM Comput. Commun. Rev.*, 34(4):219–230, 2004.

[63] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM '05*, pages 217–228, 2005.

[64] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. Data streaming algorithms for estimating entropy of network traffic. In *SIGMETRICS '06/Performance '06*, pages 145–156, 2006. ISBN 1-59593-319-0.

[65] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. *KDD' 06*, pages 287–296, 2006.

[66] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and identification of network anomalies using sketch subspaces. *IMC '06*, pages 147–152, 2006.

[67] Yang Liu, Wenji Chen, and Yong Guan. A fast sketch for aggregate queries over high-speed network traffic. In *INFOCOM '12, IEEE*, 2012.

[68] B.A. Mah. An empirical model of http network traffic. In *INFOCOM '97*, volume 2, pages 592 –600 vol.2, apr 1997.

[69] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Duplicate detection in click streams. In *WWW '05*, pages 12–21, 2005. ISBN 1-59593-046-9.

[70] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131475738.

[71] S. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, 2005. ISSN 1551-305X.

[72] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *Security '10*, 2010.

[73] Gunwoo Nam, Pushkar Patankar, Seung-Hwan Lim, Bikash Sharma, George Kesidis, and Chita R. Das. Clock-like flow replacement schemes for resilient flow monitoring. In *ICDCS '09*, pages 129–136, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3659-0.

[74] Jelani Nelson and David P. Woodruff. Fast manhattan sketches in data streams. In *PODS '10*, pages 99–110, 2010. ISBN 978-1-4503-0033-9.

[75] Nam H. Nguyen, Thong T. Do, and Trac D. Tran. A fast and efficient algorithm for low-rank approximation of a matrix. In *STOC*, pages 215–224, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-506-2. doi: http://doi.acm.org/10.1145/1536414.1536446.

[76] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, 31:353–363, February 2002. ISSN 0097-5397.

[77] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51:122–144, May 2004. ISSN 0196-6774.

[78] Spiros Papadimitriou and Philip Yu. Optimal multi-scale patterns in time series streams. In *SIGMOD '06*, pages 647–658, New York, NY, USA, 2006. ACM. ISBN 1-59593-434-0.

[79] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB '05*, pages 697–708. VLDB Endowment, 2005. ISBN 1-59593-154-6.

[80] Ely Porat. An optimal bloom filter replacement based on matrix solving. In *CSR '09*, pages 263–273, 2009. ISBN 978-3-642-03350-6.

[81] Anirudh Ramachandran, Srinivasan Seetharaman, Nick Feamster, and Vijay Vazirani. Fast monitoring of traffic subpopulations. In *IMC '08*, pages 257–270, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-334-1.

[82] O. Rottenstreich, Y. Kanizo, and I. Keslassy. The variable-increment counting bloom filter. In *INFOCOM, 2012 Proceedings IEEE*, pages 1880 –1888, march 2012.

[83] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A. Dinda, Ming-Yang Kao, and Gokhan Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Trans. Netw.*, 15(5):1059–1072, 2007. ISSN 1063-6692.

[84] D. Sejdinovic and O. Johnson. Note on noisy group testing: Asymptotic bounds and belief propagation reconstruction. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 998 –1003, 29 2010-oct. 1 2010.

[85] Vyas Sekar, Michael K. Reiter, Walter Willinger, Hui Zhang, Ramana Rao Kompella, and David G. Andersen. Csamp: a system for network-wide flow monitoring. In *NSDI'08*, pages 233–246, Berkeley, CA, USA, 2008. USENIX Association.

[86] Devavrat Shah and Tauhid Zaman. Detecting sources of computer viruses in networks: Theory and experiment. In *SIGMETRICS*, 2010.

[87] G.W. Stewart and Ji guang Sun. *Matrix perturbation theory.* Academic Press, Boston, 1990.

[88] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Sparse graph mining with compact matrix decomposition. *Stat. Anal. Data Min.*, 1(1):6–22, 2008. ISSN 1932-1864.

[89] Hanghang Tong, Spiros Papadimitriou, Jimeng Sun, Philip S. Yu, and Christos Faloutsos. Colibri: fast mining of large static and dynamic graphs. In *KDD*, pages 686–694, 2008. ISBN 978-1-60558-193-4.

[90] Santosh S. Vempala. *The Random Projection Method*. American Mathematical Society, Rhode Island, 2004.

[91] Shobha Venkataraman, Dawn Song, Phillip B. Gibbons, and Avrim Blum. New streaming algorithms for fast detection of superspreaders. In *NDSS'05*, pages 149–166, 2005.

[92] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *WOSN '09*, pages 37–42, 2009. ISBN 978-1-60558-445-4.

[93] Linfeng Zhang and Yong Guan. Variance estimation over sliding windows. *PODS '07*, pages 225–232, 2007.

[94] Linfeng Zhang and Yong Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *ICDCS '08*, pages 77–84, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3172-4.

[95] Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network anomography. In *IMC*, pages 30–30, Berkeley, CA, USA, 2005. USENIX Association.

[96] Yin Zhang, Matthew Roughan, Walter Willinger, and Lili Qiu. Spatio-temporal compressive sensing and internet traffic matrices. *SIGCOMM Comput. Commun. Rev.*, 39 (4):267–278, 2009. ISSN 0146-4833.

[97] Qi Zhao, Abhishek Kumar, and Jun Xu. Joint data streaming and sampling techniques for detection of super sources and destinations. In *IMC '05*, pages 7–7, Berkeley, CA, USA, 2005. USENIX Association.

[98] Marcin Zukowski, Sándor Héman, and Peter Boncz. Architecture-conscious hashing. In *DaMoN '06*, New York, NY, USA, 2006. ACM.