

Towards Ideal Network Traffic Measurement: A Statistical Algorithmic Approach

A Thesis
Presented to
The Academic Faculty

by

Qi Zhao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2007

Copyright © 2007 by Qi Zhao

Towards Ideal Network Traffic Measurement: A Statistical Algorithmic Approach

Approved by:

Prof. Jun (Jim) Xu
College of Computing, Advisor

Prof. Mostafa H. Ammar
College of Computing

Prof. Nick Feamster
College of Computing

Prof. Xiaoli Ma
School of Electrical and Computer Engineering,
External Member

Prof. Ellen W. Zegura
College of Computing

Date Approved: July 12th, 2007

To my parents,

Mrs. Yuanzhong Qi, Mr. Xihan Zhao

and my wife Huijuan

ACKNOWLEDGEMENTS

This thesis would not have been completed without the invaluable support, advise and guidance I received from various people in the past a few years. First, I would like to thank my advisor, Prof. Jun (Jim) Xu, for his guidance, support, and encouragement. He has been a great teacher, and provided the opportunities and encouragement crucial for the beginning of anything significant. Much of what lies in the following pages can be credited to his patient advise and kind guidance. Thank him for providing me continuous Graduate Research Assistant (GRA) support through his NSF grants NETS-NBD 0519745 and NSF CAREER Award ANI 0238315 and invaluable help on my job hunting.

Many thanks are due to Prof. Xiaoli Ma for her excellent advise guidance in our collaboration and valuable feedback as the external member on my thesis committee. I thank Prof. Mostafa Ammar, Prof. Ellen Zegura and Prof. Nick Feamster for being great teachers and mentors and for their interest in my graduate work and ever valuable comments. Thanks are also due to Prof. Mostafa, Prof. Zegura and Prof. Feamster for being on my thesis committee.

The collaborations with AT&T research labs, IBM T.J.Watson Research Center and University of Rochester were enjoyable experience of my Ph.D studies. I thank my coauthors, Dr. Zihui Ge (AT&T), Dr. Zhen Liu (IBM), Prof. Mitsunori Ogiwara (CS, Univ. Rochester), Dr. Haixun Wang (IBM), Dr. Jia Wang (AT&T), for providing opportunity to learn and broaden my horizons and support throughout these years.

Next, I would like to thank Shanita Williams and Dani Denton. They were always ready with a cheerful smile and knowledgeable advise on all matters administrative, and I thank them and other members of the administrative staff at the College of Computing for their help.

My peers in the college, especially friends from the networking group deserve a special note of gratitude. Some were collaborators on research projects while others provided company at lunch and on various hiking and camping trips. Spirited debates in the reading group meetings and discussions in the corridors and at the lunch table were the greatest gift of graduate school. For all this I thank

Abhishek, Minho, Meng, Qi, Wenrui, Yong, Donghua, Jun, Yang, Yiyi, Chuck, Amogh, Ravi, Manish, Hyewon, Ruomei, Srini and other folks from the NTG.

I am deeply indebted to my dear wife Huijuan for her love and understanding through my graduate years. She is always behind me gave her unconditional support. Finally, to my parents who brought me into this world and started all these. From half the way across the globe, they have provided a bedrock of support. It would have been absolutely impossible to get to this point without their patient love and care, support and encouragement throughout my life. This thesis is dedicated to them.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
LIST OF TABLES	xi
LIST OF FIGURES	xii
SUMMARY	xiv
I INTRODUCTION	1
1.1 Contributions	4
1.1.1 Network Data Inference with Multiple Data Sources	4
1.1.2 Network Data Streaming	5
1.1.3 Hardware Enhancement for High-speed Massive Data Analysis	8
1.2 Dissertation Organization	9
II BACKGROUND	11
2.1 Network Tomography	11
2.1.1 Traffic matrix estimation	13
2.2 Sampling for Passive Internet Measurement	15
2.3 Network Data Streaming	17
2.3.1 Data streaming in database and theoretical computer science	17
2.3.2 Network applications of data streaming	18
2.4 Statistics Counter Architecture	19
III TRAFFIC MATRIX ESTIMATION WITH MULTIPLE INFORMATION SOURCES	22
3.1 Introduction	22
3.2 Problem Statement	25
3.2.1 Terminologies	25
3.2.2 Imperfect data sources	26
3.2.3 Traffic matrix estimation	28
3.3 Methodology	28
3.3.1 Modeling measurement noises	28
3.3.2 Estimating traffic matrix	32

3.4	Evaluation	35
3.4.1	Data gathering methodology	35
3.4.2	Experimental results	36
3.5	Existing Challenges in Practice	39
3.5.1	Partial NetFlow coverage	39
3.5.2	Removal of dirty data	41
3.5.3	Handling routing changes	44
3.6	Evaluation of Enhancements	45
3.6.1	Incomplete NetFlow data	46
3.6.2	Dirty data	49
3.6.3	Routing changes	52
3.7	Combining Time Series Information	53
3.8	Conclusion	58
IV	MEASUREMENT OF TRAFFIC AND FLOW MATRICES	60
4.1	Introduction	60
4.2	System Overview and Performance Requirements	62
4.3	The Bitmap Based Algorithm	63
4.3.1	Online streaming module	64
4.3.2	Data analysis module	65
4.3.3	Extension for operational issues	67
4.4	Sampling	69
4.4.1	Sampling versus squeezing	70
4.4.2	Consistent sampling	73
4.5	The Counter-array Based Scheme	73
4.5.1	Online streaming module	74
4.5.2	Data analysis module	75
4.6	Evaluation	78
4.6.1	NLANR trace-driven experiments	78
4.6.2	Experiments based on ISP traffic matrices	83
4.7	Conclusion	84

V	DETECTION OF SUPER SOURCES AND DESTINATIONS	85
5.1	Introduction	85
5.2	Related Work	87
5.3	The Simple Scheme	89
5.3.1	Limitations of traditional hash-based flow sampling	89
5.3.2	Our scheme	91
5.3.3	Complexity analysis	94
5.3.4	Accuracy analysis	95
5.4	The Advanced Scheme	95
5.4.1	Online streaming module	96
5.4.2	Estimation module	97
5.4.3	Accuracy analysis	99
5.4.4	Identity sampling module	101
5.5	Estimating Outstanding Fan-outs	101
5.5.1	Online streaming module	102
5.5.2	Estimation module	102
5.6	Evaluation	103
5.6.1	Traffic Traces and Flow definitions	104
5.6.2	Accuracy of the simple scheme	104
5.6.3	Accuracy of the advanced scheme	105
5.6.4	Accuracy of the extension to estimate outstanding fan-outs	110
5.7	Conclusion	111
VI	FINDING GLOBAL ICEBERGS OVER DISTRIBUTED DATA SETS	112
6.1	Introduction	112
6.1.1	Problem statement	113
6.1.2	Our solutions and contributions	114
6.2	Related Work	116
6.3	The Sampling-based Scheme	118
6.3.1	Sampling process and the estimator	119
6.3.2	The worst-case MSE and the near-optimal sampling strategy	120
6.3.3	Tight tail bounds through a new Large deviation theorem	122

6.3.4	Numerical results	123
6.4	Counting-sketch-based Scheme	124
6.4.1	0/1 case	125
6.4.2	The general case	128
6.5	Evaluation	134
6.5.1	Data sets	134
6.5.2	Experiment setup and results	135
6.6	Conclusion	137
VII	DESIGN OF A NOVEL STATISTICS COUNTER ARCHITECTURE WITH OPTI-	
	MAL SPACE AND TIME EFFICIENCY	139
7.1	Introduction	139
7.1.1	Motivation	139
7.1.2	Hybrid SRAM/DRAM counter architectures	141
7.1.3	Our approach and contributions	142
7.2	Our Scheme	145
7.3	Analysis	148
7.3.1	Notations and summary of results	149
7.3.2	Bounding the probability of $D_{s,t}$ using simplified Chernoff bound	152
7.3.3	Using second moment information to obtain new bound of $\Pr[D_{s,t}]$. . .	154
7.4	Evaluation	160
7.4.1	Numerical examples of the tail bounds	161
7.4.2	Simulation using real-world Internet traffic	163
7.5	Conclusion	165
VIII	CONTRIBUTION AND FUTURE WORK	166
8.1	Conclusion	166
8.2	Future Work	169
8.2.1	Network Data Inference with Multiple Data Sources	169
8.2.2	Network Data Streaming	170
8.2.3	Statistics Counter Architecture	170
APPENDIX A	—	172
REFERENCES	191

VITA	199
-----------------------	------------

LIST OF TABLES

1	Data sets used in our experiments.	135
2	Communication cost of our scheme.	136
3	Probability of overflow when $N = 10^6$, $n = 10^{12}$, $\mu = 1/12$ and $l = 4$ bits	161
4	Probability of overflow when $N = 10^6$, $n = 10^{12}$, $\mu = 1/30$ and $l = 5$ bits	161
5	Cost-benefit comparison for different schemes for the reference system with a million 64-bit counters, $\mu = 1/30$, and $K = 500$ slots.	162
6	Statistics of buffer size with single counter increment	164
7	Statistics of buffer size with 4 counter increments	164

LIST OF FIGURES

1	Traffic rate over 5-minute intervals (both x and y axis are in logscale)	32
2	The weighted cumulative distribution of relative errors of estimated traffic matrices elements for our schemes and tomogravity method.	36
3	MRE as a function of different levels of noise in NetFlow and SNMP measurements.	37
4	MRE under different values of λ (x axis is in logscale)	47
5	The fraction of traffic vs. the magnitude of the relative error.	48
6	Impact of partial deployment of NetFlow on traffic matrix estimation.	49
7	The stability of the NetFlow deployment decision.	50
8	Impact of dirty data on traffic matrix estimation.	51
9	CDFs of the relative errors of the estimated TM elements with and without routing changes during a measurement interval	54
10	MRE of AR(P) model	56
11	Comparison between this scheme and prior solutions	58
12	System model	62
13	Example of timeline	69
14	Measured average error from Monte-Carlo simulation vs. the value from Equation 22 ($X = 1\text{M}$ packets, $T = 10\text{M}$ packets, and $b = 1$ Mbits).	72
15	Original vs. estimated traffic matrix elements. Both axes are on logscale	80
16	The RMSRE for various threshold T	81
17	The average error of the bitmap scheme and the sampling based scheme.	81
18	The RMSRE for various threshold T for flow matrix.	82
19	Cumulative distribution of traffic with certain average error.	83
20	Traditional flow sampling vs. filtering after sampling	91
21	System model of the advanced scheme	96
22	Actual vs. estimated fan-outs of sources by the simple scheme given the flow sampling rate 1/4. Notice both axes are on logscale.	103
23	Original vs. estimated fan-out of sources for trace <i>IPKS+</i> by the simple scheme given the flow sampling rate 1/16. Notice both axes are on logscale.	106
24	Actual vs. estimated fan-out of sources for trace <i>IPKS+</i> with flow sampling rate 1/4. The aforementioned second definition of source and destination labels is used here. Note that both x-axis and y-axis are on logscale.	107

25	Actual vs. estimated fan-out of sources by the advanced scheme. Notice both axes are on logscale.	107
26	Actual vs. estimated fan-out of sources for trace <i>IPKS+</i> under the second flow definition by the advanced scheme. Notice both axes are on logscale.	108
27	Average relative error for various fan-out values in the trace <i>IPKS+</i>	108
28	Probability that the estimate \widehat{F}_S is within a factor of $(1 \pm \epsilon)$ of the actual fan-out F_s for various values of ϵ	109
29	Actual vs. estimated fan-out of sources by extension of the advanced scheme including deletions. Notice both axes are on logscale.	109
30	Original (x-axis) vs. estimated (y-axis) global frequency counts by the counting-sketch-based scheme. Notice both axes are on logscale.	136
31	Relative errors of our estimations.	137
32	Hybrid SRAM/DRAM counter architecture	141
33	Timing diagram for our algorithm	147
34	variance of $b(s, t)$	155
35	Expanded depiction of variance of $b(s, t)$ for three cases.	157
36	Case i by setting $N = 1,000,000$, $K = 600$, $l = 4$, $u = 1/12$. Notice that both x and y axis are logscale.	189
37	Case ii by setting $N = 1,000,000$, $K = 600$, $l = 4$, $u = 1/12$. Notice that both x and y axis are logscale.	190

SUMMARY

With the emergence of computer networks as one of the primary platforms of communication, and with their adoption for an increasingly broad range of applications, there is a growing need for high-quality network traffic measurements to better understand, characterize and engineer the network behaviors. Due to the inherent lack of fine-grained measurement capabilities in the original design of the Internet, it does not have enough data or information to compute or even approximate some traffic statistics such as traffic matrices and per-link delay. While it is possible to infer these statistics from indirect aggregate measurements that are widely supported by network measurement devices (e.g., routers), how to obtain the best possible inferences is often a challenging research problem. We name this as “too little data” problem after its root cause. Interestingly, while “too little data” is clearly a problem, “too much data” is not a blessing either. With the rapid increase of network link speeds, even to keep sampled summarized network traffic (for inferring various network statistics) at low sample rates results in too much data to be stored, processed, and transmitted over measurement devices. In summary high-quality measurements in today’s Internet is very challenging due to resource limitations and lack of built-in support, manifested as either “too little data” or “too much data”. The main contribution of this dissertation is to design new software and hardware technologies to address these challenges. We propose a novel *statistical algorithmic solution*, which consists of the following three complementary methodologies.

First, *network data inference* with multiple data sources are proposed to counter “too little data” problem. We find that in order to meet the ever-increasing demand on traffic monitoring large ISPs are deploying new traffic measurement capabilities onto their IP backbones. For example, in AT&T backbone, besides Simple Network Management Protocol (SNMP) which has run in the whole network for a long time, Cisco sampled NetFlow also started to be instrumented at ingress/egress edge routers of the network in recent years. Each of them generates a separate measurement data set

which provides a complementary yet orthogonal observation for the complete picture of the statistics of interest. Our methodology is to investigate these data sets altogether to offer better inference accuracy. Another important advantage of this methodology is to identify accidental errors occurring in measurement results (called “dirty data”). We find that although different measurement capabilities work independently they may collect some common information in the full information spectrum. This information redundancy could be compared and verified between multiple measurement results and then the errors could be identified and removed. Using this methodology we design a set of methods for robust traffic matrix estimation and detection of dirty data by correlating both link-level and path-level information.

Second, *network data streaming* has been recognized as a new approach to alleviate “too much data” problem in network research community. It is concerned with processing a long data stream (e.g., network traffic) in a single pass using a small working memory to answer a class of queries regarding the stream. This methodology usually requires some extra hardware such as SRAM chips to support high-speed network links (e.g., 40Gbps) but is able to provide more accurate results typically. In this dissertation we design data streaming algorithms for estimation of several important traffic statistics that have traditionally been considered hard to be measured at high-speed network links and routers. We also devise some novel methods to correlate data streaming and traditional sampling techniques for measuring network traffic, which link this network data streaming methodology with the aforementioned methodology which combines multiple data sources together. In addition, these works lead to some notable mathematical results and methods such as a new large deviation theorem that finds applications in various areas.

Enhancing storage and processing hardware in a measurement system is another way to alleviate the challenges. In this dissertation, we focus on a specific fundamental tool of measurement: *counting*. In particular, we explore the problem—*how to maintain a large counter array efficiently in high speed?*. We can see that many network measurement applications need maintain a large number of counters incremented by a high-speed massive data stream (e.g., network traffic) in order to record all kinds of information. As line rates get ever-increasing, each counter matched by a packet must be read, incremented and written in a tiny cycle (e.g., a few nanoseconds). Thus we have to use fast memory (SRAM) to hold these counters. However, a router potentially need support millions of

real-time counters and wide counters (e.g., 64 bits) are required in order to avoid overflow quickly. Therefore, placing all counters in SRAM requires potentially infeasible amounts of fast memory. We design a optimal hybrid SRAM/DRAM statistics counter architecture to counter this problem. We also derive a tight statistical bound for the performance of this architecture, which adopts the large deviation theorem developed in our research on network data streaming.

To summarize, we present some new practices and proposals to better alleviate the fundamental problems in measuring large-scale high-speed networks: “too little data” and “too much data” problems. The contribution is four fold: i) designing universal methodologies towards ideal network traffic measurements; ii) providing accurate estimations for several critical traffic statistics guided by the proposed methodologies; iii) offering multiple useful and extensible building blocks which can be used to construct a universal network measurement system in the future; iv) leading to some notable mathematical results such as a new large deviation theorem that finds applications in various areas.

CHAPTER I

INTRODUCTION

The Internet is one of the most complex and large-scale distributed systems in the world. It comprises hundreds of millions of end hosts and thousands of competing ISPs. Concerns about the Internet's capabilities to meet various ever-increasing demands on performance and functionality has led to a call to high-quality measurement of network traffic especially for Internet Service Providers (ISP). An ISP must use some network measurement infrastructures deployed in its network to obtain traffic reports for characterizing the state of the network, the demands of traffic, the consumption of network resources (e.g., bandwidth), and the performance experienced by traffic going through the network so that it can ensure that the network resources are matched adequately with demands and the anomaly is prevented,

The challenges for high-quality network traffic measurement stem from the lack of fine-grained measurement/analysis capabilities in the original design of the Internet. One of the main reasons for the success of today's Internet is that end hosts do not need visibility into the interior of the underlying network that connects them and transmits data between one and another. Rather, in the Internet protocol suite, the functionality required for data transmission is developed in layers, with each layer responsible for a different facet of the communications. For example, the transport layer provides a host with the appearance of a conduit through which data is transferred to another host while the lower layers deal with packet routing in the network and the actual data transmission over physical links. Thus there are scant mechanisms built into the standard Internet protocols that enable measurement of network interior from end hosts since what happens in the interior is the business of the lower layers. Also the best effort service model which dominates the global Internet reinforces the problem. This minimalist service model allows router to be *stateless*, that is, routers need not maintain any fine-grained state (e.g., per user or per flow state) about traffic so that no hard quality guarantees on a fine-grained level can be offered. Typically only aggregate performance metrics are ubiquitously reported by router interfaces such as loss and utilization statistics. Nowadays with the

emergence of a wide range of applications which provide the services beyond the original best effort model such as guaranteed services, there is a growing need for differentiated measurements. ISPs need to characterize fine-grained traffic demands – even down to the level of individual customers – to better match available resources and so insulate traffic from the underlying variability of network conditions. For examples, measurement of traffic volume from each network customer, perhaps differentiated by applications, supports usage accounting/ pricing and helps verify conformance to agreed network performance requirements; ISPs want to measure individual link performance to identify congestion and take corrective actions before performance agreements are violated.

The global Internet creates a complex picture that is particularly hard to interpret. The lack of built-in capabilities for fine-grained measurement usually leads to a fundamental problem in network measurement namely “too little data”. Sometimes ISPs do not have enough data to directly measure the traffic statistics of interest which are only components in aggregate measurements. For example, troubleshooting packet losses requires knowing network performance on every individual link, while in practice it may only be feasible to measure end-to-end performance between any hosts, that is, the composite performance along a path that consists of multiple links. A recent response to this “too little data” problem is *network tomography* which is first coined in [121] due to the similarity between network inference and medical tomography and then attracted intensive research effort. Instead of measuring the traffic statistics of interest directly, which is hard, this method infers individual components from collections of aggregate measurements which are readily available in a network. For example, when troubleshooting loss rates, correlating end-to-end performance measurements along intersecting network paths reveals the performance on the intersection of those paths. This method often results in a highly underconstrained linear inverse problem and so allows many solutions. Some side information is introduced to differ in how a single “best” solution is identified. It typically has the limited inference accuracy due to undesirable side information. We will survey this method in some more details in Section. 2.1.

The explosive growth of the Internet in size and speed causes another fundamental problem in network measurement: “too much data” problem. Network links operate at high speeds, and past trends predict that the speeds is still keeping increasing rapidly. Routers that operate at up to 40 Gbps are currently being deployed and allow enormous traffic passing through. For example,

AT&T IP backbone collects data from tens of thousands of network interfaces and a single high-speed network interface could in principle generate hundreds of gigabytes of flow statistics per day if fully utilized. Thus collection and inference of all pertinent network statistics impose an impracticable overhead from the following three aspects. First, processing and storage resources on today's measurement devices are comparatively scarce since they are already employed in the regular work such as routing and switching packets. Second, for many traffic measurement infrastructures (e.g., SNMP and Cisco NetFlow), measured data must be transmitted to collection stations for further storage and analysis. The transmission of measured data could consume significant amount of network bandwidth. Third, complicated and costly computation component is required for analysis and storage of the data. These three aspects motivate the necessity of data reduction on storage, computation and transmission. They also motivate the needs of enhancing storage and processing hardware to accommodate faster and more measured data on the other hand.

The most common method for data reduction is sampling which only captures a small fraction of the total traffic traversing the measurement point, and answers queries based on an inspection of the sample. The sampled traffic contains the complete information for every sampled packet. The statistics of the raw traffic can be inferred from that by “inverting” the sampling process, i.e., compensating for the effects of sampling. Sampling technique trades off the opposing goals of controlling estimation accuracy and sample volume. To make sampling operation affordable in terms of both processing and storage resources, the sampling rates are typically low in high-speed networks, implying large inaccuracies in the estimates based on sample. We will describe some classic sampling techniques in detail in Section. 2.2.

So far, we see two fundamental problems in contemporary network traffic measurement and analysis: “too little data” and “too much data” problems. We also briefly review some recent proposals to resolve these problems. However, these proposals either do not scale to high speeds or lack the desirable estimation accuracy and hereby cannot offer sufficiently desirable measurements results for today's Internet. In this dissertation, we present some new practices and proposals to better resolve the aforementioned problems in large-scale high-speed networks, which make a solid step towards building an ideal network traffic measurement system.

1.1 Contributions

The primary contribution of this dissertation lies in the proposal of new methodologies to resolve the “too little data” and “too much data” problems existing in today’s Internet traffic measurement systems and the adoptions of these methodologies to design data structures, algorithms and hardware enhancements for several important applications in a large ISP network. The first methodology we present is *network data inference with multiple data sources* to integrate accurate and rich side information into network tomography models for alleviating “too little data” problem. This methodology provides a general framework to model the inference problem by organizing multiple data sets available in a network. Some statistical signal processing techniques are used to derive the optimal solutions based on the models. The other proposed methodology is called *network data streaming*. This methodology is able to replace or work with traditional sampling technique to achieve data reduction for “too much data” problem. Several data streaming schemes are presented in this dissertation. These individual schemes share a common architectural theme which supports the monitoring of highly dynamic populations by summarizing the traffic streams into compact traffic sketches. Integrating this with sampling techniques will enable a broad range of application for accurate and comprehensive monitoring of high-speed network traffic. Besides these two novel methodologies, *hardware enhancement* is another way to resolve the challenges in network measurement. In this dissertation we focus on optimizing the existing hardware resources to support data structures and algorithms associated with fast and massive network traffic analysis.

1.1.1 Network Data Inference with Multiple Data Sources

The quality of side information is critical to the performance of the aforementioned network tomography method. The introduced side information in prior network tomography work is either arbitrary statistical assumptions or rough approximations, leading to large inaccuracy of the inference. So the key to improve the performance of network tomography is to obtain better side information. We find that in order to meet the ever-increasing demand on traffic monitoring large ISPs keep deploying new traffic measurement infrastructures on their IP backbones. For example, in AT&T backbone, besides Simple Network Management Protocol (SNMP) having run in the whole network for a long

time, Cisco sampled NetFlow also started to be instrumented at ingress/egress routers of the network in recent years. Every infrastructure generates a separate measurement data set. Each of these data sets provides a complementary yet orthogonal observation for the statistics of interest. Our idea is to integrate all these data sets together to produce better statistical models and corresponding side information. Besides improving the inference accuracy this methodology has another important advantage: identifying accidental errors occurring in measurement results (called “dirty data”). We find that although different measurement infrastructures work independently they may collect some common information in the full information spectrum. This information redundancy could be compared and verified between different measurement results and then the errors could be identified and removed.

The key challenge of this methodology is how to organize all the things together to achieve the best possible results on statistics inference and dirty data removal. Our weapon is signal processing technique. The specific problem we attack is *traffic matrix estimation* in a tier-1 ISP IP backbone. A traffic matrix quantifies aggregate traffic volume between any origin/destination (OD) pairs in a network, which is essential for efficient network provisioning and traffic engineering. Prior work has tried to infer traffic matrices from two different measurement infrastructures (SNMP and Cisco NetFlow), but never from both (see Section. 2.1.1). In our work, we use some statistical signal processing techniques to correlate the data obtained from both measurement infrastructures, which by exploiting the statistically orthogonal nature of noises in independent measurements, achieve better accuracy than obtainable from a single data source and removes dirty data. Furthermore we systematically explore the possibility to combine another source of information – the implicit temporal correlations of OD flow – to further improve the inference performance. Our practice provides valuable insights on the methodology and effectiveness of combining multiple data sources in a network for the best possible inference. .

1.1.2 Network Data Streaming

As high-speed network links result in “too much data” to be processed in real-time or stored, network measurement devices have to rely on some method to achieve data reduction. Speed and volume reduction are two different ways to implement data reduction. Sampling is the well-known

method to implement speed reduction by randomly or pseudo-randomly selecting representative packets from the original traffic. However, the low sampling rates required to winnow the huge population of interest down to a manageable size, imply large inaccuracies in the estimates based on sampling. The complete information spectrum about the sampled data is captured in this process, part of which is irrelevant to the statistics of interest and hereby useless to the subsequent inference. In this dissertation we propose an alternative approach to do reduction for large amounts of data: network data streaming¹. It implements volume reduction by processing each and every packet in the original traffic but only extracting the most relevant information in each packet. Since this methodology does not reduce the traffic arrival rate to the system and high link speeds impose very stringent limits on worst-case computational complexity of processing each packet, we have to use fast memory (e.g., static random access memory (SRAM)) to accommodate the resulting data sketch. Fast memory (e.g., SRAM) is much more expensive than slow memory (e.g., dynamic random access memory (DRAM)) so that only a small amount of fast memory can be instrumented in a measurement device (e.g., a line card). Thus the resulting data sketches have to be made small enough to fit in fast memory. How to implement a data structure which is small and informative enough to record relevant information is the key challenge to design network data streaming schemes for measuring high-speed network traffic.

In this dissertation, we propose several data streaming schemes to estimate some important traffic statistics in the area of network traffic measurement. The contribution is four fold. First, these schemes allow us to obtain accurate estimations on these particular problems. Second, they provide useful extensible building blocks to construct a universal network traffic measurement system in the future. Third, we devise some novel methods to correlate data streaming and traditional sampling techniques for measuring network traffic, which links this data streaming methodology with the aforementioned first methodology which combines multiple data sources together. This implies that if the new data streaming measurement infrastructure is deployed in the Internet it can work with other infrastructures very well. Fourth, these works lead to some notable mathematical results and methods such as a new large deviation theorem that finds applications in various areas.

¹As a note of clarification, the term *data streaming* here has no connection with the transmission of multimedia data known as media (audio and video) streaming [99].

Estimating traffic and flow matrices Using data streaming algorithms we again attack the aforementioned problem: traffic matrix estimation. We propose a novel data streaming algorithm that can process traffic stream at very high speed (e.g., 40 Gbps) and produce traffic sketches that are orders of magnitude smaller than the original traffic stream to achieve data reduction. By correlating the sketches collected at any OD flow, the volume of traffic flowing between the origin and destination can be accurately determined. Compared with our previous scheme which combines the existing SNMP and Cisco sampled NetFlow data, this scheme need implement a brand new measurement infrastructure and instrument some hardware extension such as SRAM chips to hold the small traffic sketch for high-speed processing. But the new scheme has much simpler mathematical structure and is able to provide better estimation accuracy than all the prior approaches given the same space complexity. In addition, we propose a similar algorithm to estimate *flow matrix*, a finer-grained characterization than traffic matrix, which is concerned with not only the total traffic between an OD pair (traffic matrix), but also how it splits into flows of various sizes.

Detection of super sources and destinations The second problem is detection of sources or destinations that have communicated with a large number of distinct destinations or sources during a short time interval. This problem is a critical building block for many network intrusion systems to detect port scans, worm propagation, and DDoS attacks, and estimate the spreading rates of Internet worms, etc.. This problem is also important for finding “hot spots” in peer-to-peer and content distribution networks to balance the workload and improve the system performance. Directly working on the raw traffic clearly suffers the “too much data” problem in a high-speed network. We proposes two novel data streaming algorithms and the corresponding data structures to resolve the problem. Both algorithms are based on a new design insight: sampling and data streaming are often suitable for capturing different and complementary regions of the information spectrum and a close collaboration between them is a desirable way to recover the complete information. We devise two concrete methods to correlate sampling and data streaming in this work, which could be applied into other applications.

Finding global icebergs over distributed data sets The last work on network data streaming in this dissertation is finding global icebergs, i.e., the items whose frequencies of occurrence are above a certain threshold, across all the nodes in a network. It has a broad range of applications from network measurement, through system monitoring, to biosurveillance. Clearly, shipping all the data sets to a central server for finding global icebergs produces “too much data” for the transmission in the network and processing on the server. In this work, we design two data streaming algorithms running at each participating node and generating very compact data sketches. These sketches are shipped to a central server periodically and the central server correlates the collected sketches to find global icebergs. Thus the communication cost and the processing load on the server are significantly reduced. The proposed schemes again adopt one of our proposed methods on the collaboration of sampling and data streaming. In this work, we also derive a new large deviation theorem to rigorously bound the performance of the proposed schemes, which finds a number of applications in various areas.

1.1.3 Hardware Enhancement for High-speed Massive Data Analysis

Enhancing storage and processing hardware in a measurement system is another way to alleviate the challenges. In this dissertation, we focus on a specific fundamental tool of measurement: *counting*. In particular, we explore the problem: how to maintain a large counter array efficiently in high speed?. We can see that many network measurement applications need maintain a large number of counters incremented by a high-speed massive data stream (e.g., network traffic) in order to record various information. For example, network device vendors have introduced filter-based accounting, where customers can count traffic that matches a rule specifying a predicate on packet header values. Similarly, Cisco provides NetFlow-based accounting, where packets can be logged for later analysis, and 5-tuple flow labels can be aggregated and counted on the router. Cisco also provides Express Forwarding commands, which allow per-prefix counters. This problem also arises in a variety of network data streaming algorithms (e.g., [71, 73, 133, 75]), where a large array of counters is used to track various network statistics and hereby implement various counting sketches respectively. As line rates get ever-increasing, each counter matched by a packet must be read, incremented and written in very short time (e.g., a few nanoseconds). So we have to use fast memory to hold

these counters. However, a router potentially need support millions of real-time counters and wide counter (e.g., 64 bits) is required in order to avoid overflow quickly. Therefore, placing all counters in SRAM requires potentially infeasible amounts of fast memory. In this dissertation we propose a optimal hybrid SRAM/DRAM statistics counter architecture to counter the problem in an SRAM-efficient way. We also derive a tight statistical bound for the performance of this architecture, which adopts the large deviation theorem proposed in the work of finding global icebergs.

1.2 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 provides background information. It consists of four parts corresponding to four major approaches for resolving the challenges imposed by ever growing Internet: network tomography, sampling, network data streaming and hardware enhancement. Section. 2.1 summarizes the brief history of network tomography and the major problems attacked in this area. In particular, we survey the problem of traffic matrix estimation in more details in Section. 2.1.1 since it is studied in both data streaming and data inference and reflects the natural connection of these methodologies. We survey different categories of sampling technique applied in network measurement in Section. 2.2. In Section. 2.3 we discuss data streaming algorithms in the database, theoretical computer science and networking communities. And Section. 2.4 mainly describes two recent proposals to build efficient statistics counter architecture for high-speed massive data analysis.

In Chapter 3, we present our contributions using network data inference with multiple data sources. We combine sampled NetFlow data with SNMP link counts together to estimate traffic matrices in a tier-1 ISP backbone. We also use these two information sources to attack the open problem of dirty data identification and removal. The possibility of further combining time series model of traffic matrices for improving estimation is carefully studied and evaluated as well.

Chapter 4, Chapter 5 and Chapter 6 present our main discoveries on network data streaming: (1) measurement of traffic and flow matrices directly; (2) detection of sources/destinations which communicate with a large number of distinct destinations/sources during a short time interval; (3) finding the global frequent items over distributed data sets. We use both theoretical analysis and simulation or experimental results to characterize and evaluate our solutions and compare them to

existing solutions that provide similar services.

We design a new statistics counter architecture with optimal space and time efficiency to support counter increments triggered by massive high-speed traffic in Chapter 7.

Finally, Chapter 8 concludes the dissertation with a summary of contributions and pointers to future work.

CHAPTER II

BACKGROUND

Over the past decade, a large number of research efforts are devoted to enhancing the network measurement capabilities. Despite of their different methodologies and goals, their essence is always to resolve either “too little data” problem or “too much data” problem which are the fundamental challenges imposed by the evolving Internet. In this chapter, we survey some representative methodologies in the area. The remainder of this chapter is organized as follows. Section 2.1 discuss the basics of network tomography and its taxonomy. In particular, we present a classic problem in network tomography – traffic matrix estimation – which we study extensively in this dissertation, and survey the previous approaches on this problem. Section 2.2 presents several sampling techniques that are employed in network measurement area. Some of them are adapted and improved in our work. In Section 2.3, we discuss a few network data streaming algorithms and corresponding data structures in literature, associated with the specific problems related to this dissertation. We also summarize the recent proposals for high-performance statistics counter architecture for supporting high-speed massive data analysis in Section 2.4.

2.1 Network Tomography

Several network measurement problems suffering “too little data” problem bear a strong resemblance to *inverse problems* in statistical signal processing in which the interested statistics of a complex system are not observable. Usually we cannot measure these traffic statistics of interest directly without the special cooperation from internal network. However, some other useful measurements can be conducted according to passive logging traffic or active network probing. These measurements can obtain the information indirectly relating to the statistics of interest. Subsequently, some advanced inference techniques can be used to extract the hidden information of interest. Y. Vardi was one of the first to rigorously study this sort of problems and coined the term *network tomography* [121] due to the similarity between network inference and medical tomography.

Two categories of network tomography problems have been addressed in the recent literature. One is estimation of link-level statistics such as loss rate and delay per link based on path-level (end-to-end) traffic measurements [3, 18, 104, 65, 48, 29, 114, 134, 101, 119, 120, 41, 93]. The path-level traffic measurements typically consist of counts of packets/bytes transmitted and received between end hosts by passively monitoring and packet traversal delays by actively probing the network paths. All the measured data are inherently random due to perturbations and measurement noise. The other category is estimation of path-level traffic statistics such as the aggregate traffic volume between an ingress–egress node pair (The combination of all these pairs is called *traffic matrix* which will be discussed in detail in Section. 2.1.1.) based on link-level measurements [121, 118, 19, 20, 90, 117, 85, 80, 131, 130]. Here the obtainable link-level measurements typically consist of counts of packet/bytes that pass through links in the network by SNMP which include random measurement noises.

The inherent randomness in both link-level and path-level measurements motivates the adoption of statistical methodologies for large-scale network inference. Many network tomography problems can be roughly approximated by the (not necessarily Gaussian) linear model

$$\mathbf{Y}_t = \mathbf{A}\mathbf{X}_t + \varepsilon \quad (1)$$

where \mathbf{Y}_t is a vector of measurements (e.g., link counts or end-to-end delays) recorded at a given time t at a number of different measurement points. \mathbf{X}_t is a vector of traffic statistics of interest (e.g., traffic matrix, mean delay or logarithms of packet transmission probabilities over a link) and ε is a noise vector. \mathbf{A} is the *routing matrix* which has different meanings given different statistics of interest. For example, if \mathbf{X}_t denotes the traffic matrix, \mathbf{A} describes how the OD flows traverse the network, say,

$$\mathbf{A}_{i,j} = \begin{cases} F_{i,j} & \text{if the traffic of OD flow } j \text{ traverses link } i, \\ 0 & \text{otherwise.} \end{cases}$$

Here $F_{i,j}$ is the fraction of traffic from OD flow i that traverses link j . In some cases, the vector \mathbf{X}_t is a random vector with an underlying parameterized distribution $f(\mathbf{X}_t|\theta_t)$, and it is the parameters

θ_t that interest us. The network tomography typically resolves the problems of using the observation \mathbf{Y}_t to estimate θ_t , \mathbf{X}_t or \mathbf{A} .

What makes the large-scale network inference problem 1 apart from other network inference problems is the potentially very large dimension of \mathbf{A} (e.g., hundreds of rows and tens of thousands of columns in a tier-1 ISP backbone). Generally \mathbf{A} is not full rank. Therefore the associated high-dimensional problem of estimating \mathbf{X}_t is a ill-posed (underconstrained) linear inverse problem which has a very extensive experience from fields as diverse as seismology, astronomy and medical imaging [92, 14, 35, 95]. All the solutions lead to the conclusion that some sort of side information must be brought in, producing a result which may be good or bad depending on the quality of this information. In most of the large-scale Internet inference problems studied to date, the components of the noise vector ε are assumed to be approximately independent Gaussian, Poisson, binomial or multinomial distributed. When the noise is Gaussian distributed with covariance independent of $\mathbf{A}\mathbf{X}_t$, methods such as recursive linear least squares can be implemented using conjugate gradient, GaussSeidel and other iterative equation solvers. When the noise is modeled as Poisson, binomial or multinomial distributed, more sophisticated statistical methods, such as reweighted nonlinear least squares, maximum likelihood via expectationmaximization (EM) and maximum a posteriori via Markov chain Monte Carlo (MCMC) algorithms, become necessary.

Generally, network tomography approach cannot generate highly desirable result due to the imperfect side information brought in and the inherent underconstrained nature of the resulting ill-posed inverse problem. In our research of network data inference, our strategy for improvement is to use both path-level and link-level measurements available in the network altogether to construct the better side information for the inference.

2.1.1 Traffic matrix estimation

In this section we especially survey the previous work on a well known network tomography problem, i.e., traffic matrix estimation in a large ISP network, which is intensively studied in this dissertation. This problem attracts the significant effort of researchers since the late 1990s [121, 118, 19, 20, 90, 117, 131, 130, 85, 80, 79, 63, 116, 58, 94]. It first appeared as a network tomography problem in [121]. The side information brought in to the ill-posed linear inverse problem (1) is

traffic models for OD flows. [121, 118] introduced a Poisson model assuming independent identically Poisson distribution for the values of OD flows. And based on LAN network data, Cao et al. [19] revised the Poisson assumption to propose a Gaussian model coupled with an assumption of power-law relationship between the mean and variance of an OD flow. These methods used simple statistical models for OD flows (e.g., Poisson, Gaussian) that contain neither spatial nor temporal correlations in the OD flow model. A comparative study of these methods [85] revealed that these methods were highly dependent upon an initial prior estimate of the traffic matrix.

Some other techniques are devised to include more side information coming from additional sources such as routing policy and link classification in a ISP network. This extra measurement data was used to calibrate the OD flow model. The calibrated model is then incorporated in some type of estimation procedure. The OD flow model used in [130, 131] is that of a gravity model that captures the fraction of traffic destined to an egress point as a portion of the total arriving traffic at an ingress point of the network. This gravity model further evolves to *generalized gravity model* by considering link classification (e.g., access links v.s. peer links) and routing policies. The methods proposed in [90, 117] tackle the problem via another approach. Their key idea is to explicitly change the link weights in routing configuration, thereby changing the routing paths and moving OD flow onto different paths. By doing this enough times, they increase the rank of the corresponding routing matrix. The inter-router SNMP link loads are then collected from each of the link weight used. The side information here can be thought as the additional SNMP data from altered routing configurations.

With recent advances in traffic monitoring techniques, some other techniques such as [58, 94] are also designed to directly measure and estimate traffic matrices, which is out of the scope of network tomography. Feldmann et al. [58], proposed a measurement method that combines flow-level measurements (i.e., NetFlow data) at all ingress links with reachability information about all egress links. In [94], the authors examined the feasibility of the centralized solution in [58] by outlining the computation, communication and storage overheads, for traffic matrices at different granularity levels and concluded that it does incur prohibitive costs. They proposed a new scheme that is distributed and relies only on a limited use of flow measurement data. Our proposed scheme essentially combine the flow-level measurement information with the link-level aggregate measurements used

in network tomography to produce the better results.

2.2 Sampling for Passive Internet Measurement

Network tomography is a way to live with “too little data”. To resolve “too much data” problem, sampling is a natural way to achieve data reduction for capturing network traffic at line rate. It randomly or pseudorandomly selects representative packets and then forms the estimation of statistics of the unsampled traffic. The framework of sampling for passive network measurement is being standardized by IETF [42] Router vendors also provide sampling based monitors [88, 110] that sample packets periodically, as an approximation of uniform packet sampling, and aggregate the samples into transport layer flows. The flow records are exported to a collection station that can then estimate various statistics of the monitored traffic from the sampled data in a network-wide view.

In the following we briefly discuss some sampling techniques which have been employed for network traffic measurement. We first present the class of uniform sampling including systematic and random sampling. Systematic sampling (i.e., periodical sampling) is a statistical method involving the selection of every k^{th} element from a sampling frame, where k is an integer representing the sampling interval. Its implementation is quite straightforward: set a counter to the sampling interval, decrement on each packet arrival, select a packet on reaching zero, then reset the counter and repeat; see [68] for specification of a commercial implementation and it is used in Cisco sampled NetFlow [88] to collect packets for forming flow records. However, it is vulnerable if the chosen sampling interval hides a pattern in the population since any pattern would threaten randomness. Potential sources of patterns (periodicity) are timers in protocols and periodically scheduled applications. Another drawback is that it is to some extent predictable and hence open to deliberate manipulation or evasion.

The potential problems of systematic sampling can be avoided by suitable use of random sampling with some implementation overhead. In random sampling the sampling interval between successive samples are independent random variables with a common distribution (systematic sampling is a degenerate case where the random variable takes a constant value.) to make it avoid the above problems. Choosing the intervals to be geometrically distributed (for count-based sampling)

or exponentially distributed (for time-based sampling) avoid predictability. A simple implementation of random sampling is to generate, immediately following the last sample, the length of the interval until the next sample. However, when the interval distribution is unbounded (e.g., exponential or geometric distribution) some generated intervals would not fit in storage unless a cutoff is applied. The special case of geometric random sampling with mean inter-sample count m , which can avoid this problem, is making the sampling decision for each object with probability $\frac{1}{m}$.

Nonuniform sampling is to sample objects with probabilities that can be a function of their characteristics. This design can be used to boost the chance of sampling objects that are rare but important by setting the proper sampling function. A particularly attractive form of nonuniform probability sampling is size-dependent sampling. In this sampling we choose sampling probability related to the size of the particular object to achieve specific purposes such as reducing estimation variance. Duffield et al. [45] proposed one size-dependent sampling technique based on flow bytes, namely *smart sampling*. Conceptually, smart sampling selects a flow of x bytes with probability $\min(1, x/z)$, where z is a predefined threshold. In other words, flows of size greater than the threshold z are always collected, while smaller flows are selected with a probability proportional to their sizes. This strategy achieves smaller estimation error of the total traffic volume passing through than the uniform sampling since random omission of an elephant flow can cause a large fluctuation in the perceived volume of traffic. Our works also design some similar size-dependent sampling strategy to collect object identities for query in Chapter 5 and 6.

When we say “select something with probability p ”, it typically implies a common implementation using pseudorandom number generator in some library. Another implementation which is often used in network measurement area is hashing. Hash-based sampling offers both a convenient way to emulate random sampling and a powerful way to consistently select subset of objects whose contents share a common property. The basic idea is as follows. We use a hash function h with input the object content set C . Given some subset $S \subset h(C)$, called the selection range, a packet with content c is selected if $h(c) \in S$ and recorded in a data structure such as a hash table. Suppose we wish to sample the packets belonging to a random subset of flows passing through a router, we can use the flow label of the packet as the input to h . The hash-based sampling ensures that all packets belong to a flow are either selected or not selected together. This hash-based flow sampling technique is

very useful for inference of flow level statistics [122]. We enhance this technique in Chapter 5 to significantly reduce the overhead of recording the sampled subset. Another example of hash-based sampling is trajectory sampling [44] where all routers in a network hash-sample packets using an identical hash function and selection range. The input of the hash is restricted to those packet fields that are invariant from hop to hop, e.g., time-to-live field is excluded since it is decremented per hop. Thus a given packet is sampled either at all points on its routing path along the network or at none. Applications of trajectory sampling include estimation of traffic intensities along the network paths, detection of routing loops and network attack path tracking. We use a variation of trajectory sampling to balance the estimation performance and workload in Chapter 4.

2.3 Network Data Streaming

As we introduced in Chapter 1, data streaming is an alternative data reduction technique which uses a small of working memory to process a large stream of data in order to answer queries regarding the data stream. In the past two decades, there has been a substantial amount of research on data streaming in the database and theoretical computer science that have developed various techniques as well as complexity bounds for problems in this field. Some results and the their extensions have appeared in networking community recently, In this section, we fist briefly surface some literature of data streaming in database and theoretical computer science areas and then discuss some recent proposals of data streaming in network measurement area.

2.3.1 Data streaming in database and theoretical computer science

Muthukrishnan [87] provides an excellent survey of data streaming and its theoretical roots. Babcock et al. [13] also present a survey in database area and introduce their own attempts to build a data stream management system with SQL-like query interface. Most data streaming algorithms in the area focus on analyzing a single or a limited number of streams. Applications include: (i) approximating certain functions over the stream such as quantiles [8, 11, 12, 38, 83, 84], the k_{th} frequency moment [6, 31, 56, 108], etc.(ii) performing relational operations such as join [5, 37, 59, 60, 64, 98] and (iii) data mining and knowledge discovery such as clustering and histogram maintenance [4, 23, 61, 62, 123, 128], change detection [25, 26, 33], etc. Some of the change detection

techniques have been applied (after some adaptations) to network monitoring applications [72, 109].

Distributed data streaming has been studied in [12, 27, 55] in the theoretical computer science and database contexts. System issues for large-scale distributed stream processing such as load management, high availability, and federated operation were investigated in [27]. In [55], Feigenbaum and Kannan proposed to ship “synopses” of the raw data from physically separated network elements to a central server for processing. They use a space-efficient one-pass algorithm to compute the L_1 difference between two data streams. In [12], Babcock and Olston proposed techniques to answer top- k queries over the union of distributed streams. The algorithm maintains local top- k values for each stream, and compensate the local skew with factors so that they are close to the global top- k . This reduces the amount of update that needs to be sent to the central station.

2.3.2 Network applications of data streaming

Techniques from data streaming have been appeared in various proposed solutions for a number of problems in network measurement. Estan and Varghese [49] proposed algorithms to identify traffic heavy hitters (“elephants”) in a high-speed link using a small amount of fast memory. Furthermore the schemes to online identify 1-dimensional and 2-dimensional hierarchical traffic heavy hitters were proposed in [132]. While these works successfully address the problems by monitoring just a few large flows, a range of applications that would be served better with approximate monitoring of all flows. Kumar et al. [76] develop a data streaming algorithm for estimating the volume of per flow traffic traversing a high-speed link, which is considered more challenging than only tracking large flows. The solution proposes a novel data structure called Space Code Bloom Filter (SCBF) which is an approximate representation of a *multiset*; each element in this multiset is a flow and its multiplicity is the number of packets in the flow. The multiplicity of an element in the multiset represented by SCBF can be estimated through either Maximum Likelihood Estimation (MLE) or Mean Value Estimation (MVE). [73] further devises data streaming mechanism to estimate the distribution of flow sizes going through a high-speed network link. The whole mechanism consists of two components: the online streaming component using an array of counters to process each and every incoming packet and the offline module which uses techniques from Bayesian statistics to design an estimation algorithm based on Expectation Maximization (EM), to infer the most likely flow size

distribution that would results in the observed counter values after collisions in hashing. They also extended this work to estimate the flow size distribution for any arbitrary traffic subpopulations [75] by integrating a NetFlow-like packet sampler.

Besides heavy hitter detection, heavy change detection which is to find flows whose size change significantly from one period to another is another popular primitive for massive network traffic analysis. Krishnamurthy et al. [72] proposed a K-ary sketch in conjunction with various time series forecasting models to perform change detection for a high-speed network link. The follow-up work [109] even proposed efficient reversible hashing scheme which can be used in K-ary sketch, to infer the key (i.e., flow label) of culprit flows from sketches with negligible extra memory and small extra memory access for recording streaming data. Cormode et al. [33] provides another solution based on a structure of combinatorial group testing which gives a flexible framework for detecting any kind difference given a suitable test definition. This structure can be used to find absolute, relative and variation differences, between traffic in different time period, interfaces or routers.

Estan et al. [51] design a mechanism to count the total number of flows. The streaming data structure used in their solution is quite simple – an array of bits. A hash function is computed over the flow label of each arriving packet to generate an index into the bit array and the bit at the corresponding location is set to 1. The estimation phase models the insertion process as an instance of the Coupon Collector’s problem [86] to derive a simple estimator for the total number of flows as a function of the size of the bit array and the number of bits set to 1. Our works in Chapter 4 and Chapter 5 adapt and extend this estimator for other network measurement problems.

2.4 Statistics Counter Architecture

The problem of how to efficiently maintain a large number (say millions) of statistics counters that need to be updated at very high speed has received considerable research attention recently [111, 102]. This problem arises in a variety of router management and network data streaming algorithms (e.g., [71, 73, 133, 75]), where a large array of counters is used to track various network statistics and to implement data sketches. We also present a concrete application which requires this facility in Chapter 4.

While fitting these counters entirely in SRAM meets the update speed requirement, a large

amount of SRAM may be needed with a typical counter size of 32 or 64 bits, and hence the high cost. To save the cost, a hybrid SRAM/DRAM counter architecture is proposed in the seminal work of [111] as a more SRAM-efficient way of maintaining a large counter architecture. It is a hybrid architecture in which DRAM is used to store the full statistics counters but a small amount of SRAM is used to enable counter increments at line speed. The baseline idea of this architecture is to store some lower order bits of each counter in SRAM, and the full-size counter in DRAM. The increments are made only to these SRAM counters, and when the value of a SRAM counter becomes close to overflow, it will be scheduled to be *flushed* to the corresponding DRAM counter. The “flush” operation here is defined as adding the value of the SRAM counter to the corresponding DRAM counter and resetting the SRAM counter to 0. The key research challenge in this architecture is the design of a counter management algorithm (CMA) that flushes the right set of SRAM counters to DRAM at the right time.

Shah et al. [111] propose a CMA called *LCF* (Largest Counter First). *LCF* picks the counter with the largest value to be updated to DRAM. Intuitively, this strategy is optimal because it always updates to DRAM the SRAM counter that is closest to overflowing. However, due to the necessity of finding the largest value among a large number of counters *LCF* is hard to implement at a high speed. One obvious scheme that requires no additional space or hardware complexity is to examine each and every the counter value. A more efficient scheme would presumably maintain some kind of index data structure that maintains some ordering on the counter values. For example, Bhagwan et al. [15] describe an implementation of a pipelined heap structure that can determine the largest value at a fairly high speed. However, maintaining a heap in hardware incurs high implementation complexity and a large amount of SRAM, which is about twice the size needed to store the SRAM counters.

A more efficient CMA, called *LR(b)* (Largest Recent with threshold b), is introduced in [102]. *LR(b)* avoids the expensive operation of maintaining a priority queue, by only keeping track of counters that are larger than a threshold b using a bitmap. A tree structure is imposed on the bitmap (resulting in a hierarchical bitmap) to allow for a fast retrieval of the “next counter to be flushed”. This retrieval operation has complexity $O(\log N)$, where N is the number of counters in the array, but using a large base such as 8 makes the complexity essentially a small constant. The

hardware control logic in $LR(b)$ is simpler than LCF , and uses much less SRAM. In Chapter 7, we propose a much more efficient CMA which halves the total SRAM usage of $LR(b)$ with extremely simple control logic and data structure. In Chapter 7, we will show a more efficient CMA algorithm.

CHAPTER III

TRAFFIC MATRIX ESTIMATION WITH MULTIPLE INFORMATION SOURCES

3.1 Introduction

A traffic matrix quantifies the volume of traffic between origin/destination (OD) pairs in a network. Obtaining accurate traffic matrix is essential in a number of network management tasks in operational IP networks such as capacity planning and traffic engineering, etc. Realtime traffic matrix also enables online diagnosis and mitigation of network events such as network device failures, routing changes, and traffic congestion. For example, when a link fails, the network operators need to determine whether such an event will cause congestion based on the current traffic matrix and routing configurations, and re-optimize OSPF link weights to alleviate the traffic congestion. Also, without an accurate traffic matrix, network operators are not able to diagnose the severity of network events and evaluate the effectiveness of possible solutions.

As described in Section 2.1.1, the existing approaches for estimating traffic matrices in general either infer it from aggregate link-level measurements, say SNMP link counts or path-level measurements, say sampled NetFlow records (but not from both). Neither approach is perfect, since SNMP link counts are sparse compared to the number of OD flows to be inferred, and NetFlow data is very noisy due to the low sampling rate (and can be sparse too). We can do better since neither makes the best out of the data we already have. In this chapter, we present a new approach to combine these two types of information for better estimation accuracy, which reflects our proposed methodology for the “too little data” problem – combining multiple information sources together. Concretely we propose and answer the following question —

When both SNMP link loads and sampled NetFlow records are available, can we combine them to obtain more accurate estimation of traffic matrices than those obtained from any of the data sources? and how?

This problem is practically important and the solution is useful since both SNMP and NetFlow are now widely supported by vendors and deployed in most of the operational IP networks. Sometimes the path-level measurements such as NetFlow are not available at all the ingress nodes of a large network due to the high instrumentation cost. We also extend our solution to accommodate this scenario. While a few prior work such as [85, 131] briefly mentioned this problem, this work is the first to offer a comprehensive solution which fully takes advantage of using multiple readily available data sources to achieve better estimation accuracy¹.

More importantly, our work leads to some new insight that SNMP link counts and sampled NetFlow records can serve as “error correction codes” to each other. This insight helps us to solve another challenging open problem in traffic measurement —

How to deal with dirty data (i.e., measurement errors in SNMP and NetFlow due to hardware, software or transmission problems)?

We design techniques that, by comparing notes between the above two information sources, identify and remove dirty data. This capability of removing dirty data not only improves the accuracy of traffic matrix estimation, but may also benefit a number of other applications that depend on these data.

The previous research on traffic matrix estimation has mostly focused on a single time snapshot. It is well believed that an OD flow evolves over time and has some temporal correlations. But this problem has never been carefully studied when sampled NetFlow is instrumented at the network edge completely or partially. In this work we make the first solid step to settle this challenging problem —

Is there any temporal correlation existing in traffic matrices evolving over time? If yes, how to combine this information with other information together to further improve the estimation optimally?

We carefully explore this problem and find that the time series information only provides the

¹The work [116] studies the similar problem but uses different approach to take advantage of the flow measurement data. It uses 24 hour NetFlow data to estimate the parameters in their model.

marginal gain on estimation in our design. We are investigating some other possibilities for further improvement.

In this work, we make five important contributions. First, we provide a comprehensive formulation and design an algorithm for estimating traffic matrix during a given time interval by using both SNMP link loads and sampled NetFlow data. The algorithm simply uses the traffic matrix estimated solely based on NetFlow data as *a prior* and further calibrates it using the SNMP link loads in a well-designed weighted manner. We find that under the existing configuration of sampled NetFlow the prior from it is pretty accurate already and our algorithm by combining SNMP link loads improves the estimation accuracy slightly (5% improvement of the weighted errors), specially when the measurement noise of SNMP link loads is high.

Second, we enhance the above algorithm to handle the case that the NetFlow data is not complete due to partial deployment or data loss in transit. In this case the prior is generated by combining the traffic matrix elements directly estimated by the existing NetFlow data and those produced by the generalized gravity model [130]. One of the contributions we make here is to discover the probability model in the gravity model using the Equivalent Ghost Observation (EGO) method. Then it will be further calibrated using the SNMP link loads after carefully setting up the relative weight between NetFlow data and the generalized gravity model. The experimental results in Section 3.6 show that only with a small portion of NetFlow data the estimation accuracy can be significantly improved. We also study the problem on where to turn on the NetFlow to collect flow measurement data in order to achieve the best performance on estimating traffic matrices given a fixed percentage of deployment of NetFlow.

Third, we propose novel algorithms to identify and remove dirty data in the traffic measurement. This will not only help in traffic matrix estimation but also in a number of other important network management tasks such as anomaly detection. We assume that, in practice, there are only a small number of OD pairs/links which produce dirty data at a given time and cause inconsistency in the measurement data. We identify dirty data by finding the simplest explanation of the inconsistency.

Fourth, we also develop the algorithm to estimate traffic matrices upon topology and routing changes such as link failure, hot potato routing, and BGP session reset events. This is very helpful in evaluating the impact of such network events and mitigating undesirable events. The routing

changes can promptly be reported by monitoring OSPF and BGP routing updates [112, 126]. Then we can obtain the corresponding NetFlow data before and after the routing change respectively. Using them as the *a priori*, the traffic matrices can be estimated much more accurately than that omitting the routing change.

Finally, we develop a linear predictive model for capturing the temporal correlations existing in OD flows evolving over time. This model can be used to predict the future traffic matrices using the history data. We find that the resulting prediction has the reasonably good accuracy. We also extend our previous methodology to integrate this information. Unfortunately the new solution only produces marginal improvement on estimation.

The rest of the chapter is organized as follows. Section 3.2 formally defines the traffic matrix estimation problem. We describe our base model for traffic matrix estimation based on both SNMP link load data and sampled NetFlow data in Section 3.3 and evaluate the proposed methodology using empirical data collected from a tier-1 ISP network in Section 3.4. Section 3.5 discusses a number of factors that affect traffic matrix estimation and proposes enhanced methods and Section 3.6 evaluates the proposed enhancements. In Section 3.7, we explore the possibilities to build a time series model to capture temporal correlation among traffic matrices and use it to further improve estimation accuracy. We conclude the work in Section 3.8.

3.2 Problem Statement

In this section, we state precisely our problem of estimating traffic matrix from imperfect (noisy and possibly “dirty”) SNMP link counts and NetFlow measurement data. After a brief introduction of terminologies, we pinpoint the source of noise and dirty data in both types of data, and formulate the problem we would like to solve in this work.

3.2.1 Terminologies

The topology of an IP network can be viewed as a set of routers and links. The routers and links that are internal to the network are *backbone* routers and links, while others are *edge* routers and links. The routers inside the network run Interior Gateway Protocol (IGP) to learn how to reach each other.

The two most common IGP are OSPF and IS-IS, which compute shortest paths based on configurable link weights. The edge routers at the periphery of the network learn how to reach external destinations through Border Gateway Protocol (BGP). Both IGP and BGP together determine how traffic flows through the network.

Traffic matrices are often computed at interface, router, or PoP level. We use the term “node” to denote the entity at which level the traffic matrices are computed. Given edge nodes i and j , the traffic between i and j is defined as the total traffic volume that enters the network at node i and exits the network at node j . We refer to node i as the *ingress node* and node j as the *egress node*. The pair of ingress node i and egress node j are referred to as an *Origin-Destination (OD) pair* of the traffic flow. We refer to the aggregated traffic of an OD pair as an *OD flow*. The *traffic matrix* is thus the OD flows of all possible ingress and egress OD pairs. Instead of matrix form in Chapter 4, we represent the traffic matrix in a vector form in this work, where each element corresponds to the traffic volume of one OD flow. We illustrate our schemes and experiments at router level, while they can also be applied to interfaces and PoP levels. In the rest of this work, the terms “node” and “router” are equivalent.

3.2.2 Imperfect data sources

The following measurement capabilities are deployed in most of commercial IP networks. Unfortunately, none of them alone is sufficient for providing direct and highly accurate measurement of traffic matrix. In addition, data collection is distributed among multiple network entities that are not fully synchronized, which results in noise in the data. To make matters worse, due to factors such as hardware and software errors, the quantities reported by SNMP and NetFlow measurements can deviate significantly from the actual quantity, which we regard as *dirty data* (distinguished from noise). The imperfectness of data, classified roughly into the following three categories, poses significant challenges to our goal of estimating traffic matrix accurately.

Link load measurements: The link load measurements are readily available via Simple Network Management Protocol (SNMP), which periodically polls statistics (e.g., byte counts) of each link in an IP network. The data are coarse-grained and the commonly adopted sampling interval is 5 minutes in operational IP networks. These link counts contain some noise since the measurement

station cannot complete the management information base (MIB) polling for thousands of network interfaces on hundreds of routers all at the same time at the beginning of the 5-minute intervals, making the actual polling intervals shifted as well as being longer or shorter than 5 minutes. We will show that this noise can be modeled as Gaussian random variables. In addition, the link counts can be lost during transit because SNMP uses UDP as the transport protocol, and may be incorrect due to hardware problem or software bugs. Such link counts are referred to as dirty data.

Flow level measurement: The traffic flow statistics are measured at each ingress node via NetFlow [88]. A *flow* is defined as a unidirectional sequence of packets between a particular source and destination IP address pair. For each flow, NetFlow maintains a record in router memory containing a number of fields including source and destination IP addresses, source and destination BGP routing prefixes, source and destination ASes, source and destination port numbers, protocol, type of service, flow starting and finishing timestamps, number of bytes and number of packets transmitted. This flow level information would be sufficient to provide direct traffic matrix measurement if complete NetFlow data were collected for the entire network. However, due to high cost of deploying and enabling flow level measurement via NetFlow, sampling is a common technique to reduce the overhead of detailed flow level measurement. The flow statistics are computed after applying sampling at both packet level and flow level. Since the sampling rates are often low, inference from the NetFlow data (through scaling) may be noisy. Also, NetFlow is often only partially deployed because products from some vendors do not support NetFlow in a way consistent to our needs (i.e., different from Cisco NetFlow specification) and some do not support it at all. Similar to SNMP data, NetFlow data may also be lost in transit, resulting in dirty data.

Topology and routing measurement: The network topology can be computed based on the configuration data of each router in an IP network. Both intra-domain (e.g., OSPF) and inter-domain (i.e., BGP) routing information are available via deployed monitors (e.g., [112]). Realtime access to these data allows us to compute the forwarding table at a given time within each router and identify topology and routing changes that affect traffic matrix.

3.2.3 Traffic matrix estimation

We formulate our problem of estimating traffic matrix from both SNMP link counts and NetFlow records as follows. Assume there are n OD flows and m links in an IP network. Let $\mathbf{X} = (x_1, x_2, \dots, x_n)^T$ denote the real OD flows to be estimated and let $\mathbf{B} = (b_1, b_2, \dots, b_m)^T$ denote the link loads when routing traffic demand \mathbf{X} over the network. \mathbf{B} and \mathbf{X} are related by a routing matrix \mathbf{A} :

$$\mathbf{B} = \mathbf{A}\mathbf{X}$$

where \mathbf{A} is an $m \times n$ matrix whose element on the j -th row and the i -th column, a_{ji} , indicates the fraction of traffic from flow i being routed through link j .

Let $\hat{\mathbf{X}} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n)^T$ be the estimated OD flow traffic matrix obtained from sampled NetFlow data (after compensating for the sampling), where \hat{x}_i is the estimator of x_i . Let $\hat{\mathbf{B}} = (\hat{b}_1, \hat{b}_2, \dots, \hat{b}_m)^T$ be the corresponding SNMP link load measurement adjusted by the length of polling intervals. Ideally, we would like to have

$$\hat{\mathbf{B}} = \mathbf{A}\hat{\mathbf{X}}$$

in which case we will simply use this $\hat{\mathbf{X}}$ as our estimate. However, in practice, this is rarely true due to aforementioned noises and dirty data in both SNMP and NetFlow measurements. The question we are going to answer is: “What is the best estimate of \mathbf{X} based on imperfect measurements of $\hat{\mathbf{X}}$ and $\hat{\mathbf{B}}$ from NetFlow and SNMP counts respectively?”

3.3 Methodology

3.3.1 Modeling measurement noises

As we discussed in Section 3.2, both NetFlow and SNMP data can be inaccurate due to the sampling and polling processes used in the measurement. We refer to such measurement inaccuracies as *measurement noises*. In this section, we study the NetFlow sampling process in flow measurement and the SNMP polling process in link load measurement and present our models that precisely capture the measurement noise incurred in these processes. In particular, we define

$$\mathbf{X} = \hat{\mathbf{X}} + \epsilon^{\mathbf{X}}$$

$$\mathbf{B} = \hat{\mathbf{B}} + \epsilon^{\mathbf{B}}$$

where $\varepsilon^{\mathbf{X}}$ and $\varepsilon^{\mathbf{B}}$ are the measurement noises of NetFlow data and SNMP link loads respectively. We will show that accurately modeling these measurement noises enables us to derive good estimate of the traffic matrices. In addition, it is also helpful in distinguishing the dirty data (Section 3.5.2) from the measurement noises.

3.3.1.1 Noise $\varepsilon^{\mathbf{X}}$ in flow measurement

NetFlow typically uses packet sampling to reduce the processing load and storage space. We model this packet sampling process as Bernoulli trials.

Let $\mathbf{F} = (f_1, f_2, \dots, f_n)^T$ be the observed byte counts of OD flows from NetFlow data (before compensation for sampling) and $\mathbf{R} = (r_1, r_2, \dots, r_n)^T$ be the sampling rates of OD flows. We can compute $\hat{\mathbf{X}}$, which is an unbiased estimator of \mathbf{X} , as follows:

$$\hat{\mathbf{X}} = (\hat{x}_1, \hat{x}_2, \dots, \hat{x}_n) = \left(\frac{f_1}{r_1}, \frac{f_2}{r_2}, \dots, \frac{f_n}{r_n} \right)^T$$

The Mean Square Error (MSE) of estimator \hat{x}_i , $1 \leq i \leq n$, is

$$\text{MSE}(\hat{x}_i) = \frac{1 - r_i}{r_i} \sum_{k=1}^{w_i} \sum_{j=1}^{m_k^{(i)}} s_{i,k,j}^2 \quad (2)$$

where w_i is the number of flows in the i -th OD flow, $m_k^{(i)}$ is the number of packets in the k -th flow of the i -th OD flow, and $s_{i,k,j}$ is the size of the j -th packet in the k -th flow of the i -th OD flow.

The expression for MSE in (2) is a function of the size of each packet in the OD flow, which is undesirably expensive to compute in practice. Therefore, we adapt the following approximate (and upper bound) MSE derived by Duffield et al in [45]:

$$\text{MSE}(\hat{x}_i) \approx \frac{(1 - r_i) \hat{x}_i s_{max}}{r_i}$$

where s_{max} is the largest packet size. In this work, we use $s_{max} = 1,500$ bytes.

However, for a large operational ISP, storing and transmitting the data collected by the packet-sampled NetFlow are often still prohibitive due to its large volume. To make the data size manageable, [45] proposed a new IP flow measurement infrastructure which performs an additional flow-level sampling process, namely *smart sampling*, on the data collected by packet-sampled NetFlow. Conceptually, smart sampling selects a flow of x bytes with probability $\min(1, x/z)$, where

z is a predefined threshold. In other words, flows of size greater than the threshold z are always collected, while smaller flows are selected with a probability proportional to their sizes.

As given in [45], the combined sample process (with both packet-level NetFlow sampling and flow-level smart sampling) has the following properties:

$$\hat{x}_i = \sum_{k=1}^{w'_i} \max(z, \frac{c_{i,k}}{r_i}) \quad (3)$$

$$\text{MSE}(\hat{x}_i) \approx \hat{x}_i \left(\frac{(1-r_i)s_{max}}{r_i} + z \right) \quad (4)$$

where w'_i is the number of flows in the i -th OD flow after smart sampling, and $c_{i,k}$ is the observed size (by NetFlow) of the k -th flow of the i -th OD flow.

Finally, we approximate the measurement noise introduced by NetFlow sampling and smart sampling (if applicable) as Gaussian noises:

$$\varepsilon_i^X \sim N(0, \sigma_i^2)$$

where $\sigma_i^2 = \text{MSE}(\hat{x}_i)$.

A careful observation on the above will find that this model is not rigorous under one condition: when all flows of an OD flow have been missed from the sampling, the estimated traffic volume of the OD flow becomes zero according to the unbiased estimator, and so does the MSE of this estimate. In this case, the corresponding ε_i^X is not well defined. We now address this problem.

What we are interested in is the conditional distribution of the OD flow size given it is not sampled by either NetFlow or smart sampling. Consider the simple case where all packets for the OD flow have equal size, s . Let L_0 , L_1 and L_2 denote the size (in number of packets) of an OD flow originally, after NetFlow packet sampling, and after smart sampling respectively. We can derive the conditional probability of $\Pr[L_0 = l | L_2 = 0]$ below. The derivation of (5) can be found in Appendix A.1.

$$\Pr[L_0 = l | L_2 = 0] = \sum_{j=0}^{\min\{l, r_i z/s\}} \binom{l}{j} r_i^{j+1} (1-r_i)^{l-j} \frac{2 - \frac{2j}{r_i z/s}}{r_i z/s + 1} \quad (5)$$

From (5), we can compute the mean square error (MSE) when the observed OD flow is zero as

$$\text{MSE}_0 = sE[L_0^2 | L_2 = 0]$$

and our definition of σ_i^2 becomes

$$\sigma_i^2 = \begin{cases} \text{MSE}(\hat{x}_i) & \text{if we observe OD flow } i \text{ in the sampled data} \\ \text{MSE}_0 & \text{otherwise} \end{cases}$$

3.3.1.2 Noise ε^B in link load measurement

The primary source of errors in SNMP link load measurement is due to the disalignment of polling intervals. Consider a target measurement for byte counts, b_i , over a link i , during interval $[t, t + l]$. The actual measurement, derived from two consecutive SNMP pollings for link i , however is on interval $[t + \Delta_1, (t + \Delta_1) + (l + \Delta_2)]$. We denote the result from this measurement as m_i . Note that the magnitude of Δ_1 and Δ_2 are typically much smaller than l (i.e., $|\Delta_1|, |\Delta_2| \ll l$): Δ_1 and Δ_2 is typically in the order of several tens of seconds and l is in the order of several minutes. If we assume that the traffic rate over link i in a short period of time (e.g., $[t - l, t + 2l]$) can be described as a Wiener process with parameter θ_i^2

$$b_i(t) - b_i(s) \sim N(0, \theta_i^2(t - s))$$

then

$$\hat{b}_i = \frac{m_i l}{l + \Delta_2}$$

is an unbiased estimator for b_i . The mean square error is approximately

$$\text{MSE}(\hat{b}_i) \approx |\Delta_1| l \theta_i^2$$

To quantify θ_i^2 , we measure the difference in the average traffic rate of two consecutive polling intervals. Figure 1 shows the scatter plot of the difference in the average traffic rate of two consecutive 5-minutes intervals versus the traffic rate in the first 5-minute interval for a few thousands links in a large tier-1 ISP backbone network. We observe the trend of a linear relationship between the difference, which is proportional to θ_i^2 , and the average traffic rate, which is $\frac{m_i}{l + \Delta_2} = \frac{\hat{b}_i}{l}$. Therefore, we can further approximate the mean square error as

$$\text{MSE}(\hat{b}_i) \approx |\Delta_1| \lambda \hat{b}_i$$

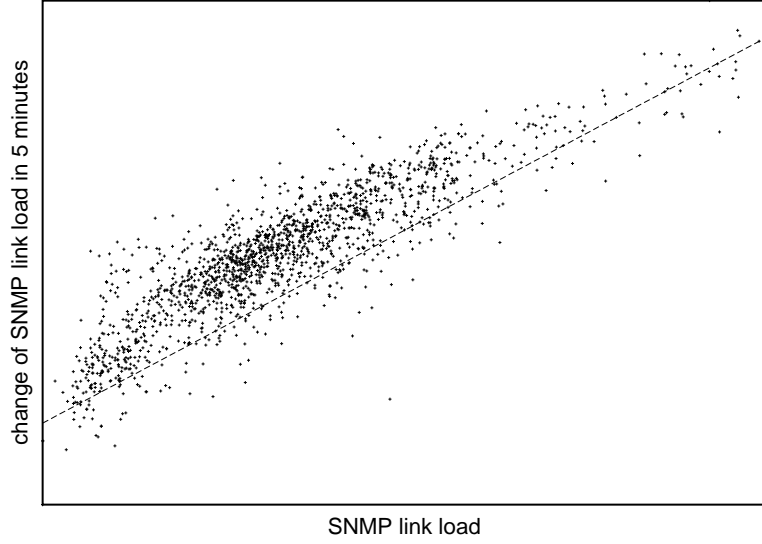


Figure 1: Traffic rate over 5-minute intervals (both x and y axis are in logscale)

where λ is a constant that can be derived by fitting the scatter plot. For the completeness of the model, we also define a small constant as the MSE in case a link load measurement is zero. However, we do not encounter this situation in the data we explored.

Similarly to ε^X , we then model the measurement noises of link load introduced in SNMP polling as Gaussian random variables:

$$\varepsilon_i^B \sim N(0, \mu_i^2)$$

where $\mu_i^2 = \text{MSE}(\hat{b}_i)$.

3.3.2 Estimating traffic matrix

Let us now revisit our problem formulation by combining the above model we derived.

$$\hat{\mathbf{X}} = \mathbf{X} + \varepsilon^{\mathbf{X}} \tag{6}$$

$$\hat{\mathbf{B}} = \mathbf{A}\mathbf{X} + \varepsilon^{\mathbf{B}} \tag{7}$$

where both $\varepsilon^{\mathbf{X}}$ and $\varepsilon^{\mathbf{B}}$ are zero-mean Gaussian random variables. Put into a matrix-vector form, our system can hence be described as

$$\mathbf{Y} = \mathbf{H}\mathbf{X} + \mathbf{N} \tag{8}$$

where $\mathbf{H} = (\mathbf{I}; \mathbf{A})$ is an $(m + n) \times n$ matrix which vertically concatenates an identity matrix \mathbf{I} and the routing matrix \mathbf{A} , \mathbf{Y} is the concatenated vector of $\hat{\mathbf{X}}$ and $\hat{\mathbf{B}}$ and \mathbf{N} is the concatenated vector of $\varepsilon^{\mathbf{X}}$ and $\varepsilon^{\mathbf{B}}$. What we are looking for is a good estimator of the traffic matrix \mathbf{X} from the observable \mathbf{Y} .

Our system in (8) fits well in the framework of the well-known Gauss–Markov theorem [70], which states that in a linear model in which the errors are uncorrelated and have expectation zero, the best linear unbiased estimators (BLUE) of the coefficients are the least-squares (LS) estimators. That is, among all unbiased estimators for \mathbf{X} , the one that minimizes the normalized residual errors (defined below), yields the smallest variance. Note that for Gauss–Markov theorem to hold, the errors need not to be normally distributed. We model the SNMP and NetFlow measurement noises as Gaussian only for the purpose of defining dirty data and distinguishing them from measurement noises, as we will discuss in Section 3.5.2.

The weighted LS estimator of \mathbf{X} in (8) is found by the pseudo-inverse solution of the normalized equivalent of (8) in which the errors are homoscedastic:

$$\dot{\mathbf{X}} = (\mathbf{H}^T \mathbf{K}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{K}^{-1} \mathbf{Y} \quad (9)$$

Here \mathbf{K} is the covariance matrix of \mathbf{N} ($\mathbf{K} = \mathbb{E}[\mathbf{N}\mathbf{N}^T]$), which is a diagonal matrix as we assume all measurement errors are uncorrelated. To relate back to our models in Section 3.3.1.1 and 3.3.1.2, we denote $\boldsymbol{\Sigma}^2 = (\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2)^T$ and $\boldsymbol{\Gamma}^2 = (\mu_1^2, \mu_2^2, \dots, \mu_m^2)^T$. The above LS estimator $\dot{\mathbf{X}}$ solves the quadratic optimization problem that aims at minimizing the total weighted squared-error in the observations, i.e.,

$$\text{minimize} \quad \left\| \frac{\mathbf{X} - \hat{\mathbf{X}}}{\boldsymbol{\Sigma}} \right\|_2 + \left\| \frac{\mathbf{A}\mathbf{X} - \hat{\mathbf{B}}}{\boldsymbol{\Gamma}} \right\|_2 \quad (10)$$

Note that the division in (10) is an element-by-element division where the numerator and denominator are vectors of same length. To compute $\dot{\mathbf{X}}$, we use the the Singular-Value Decomposition (SVD) routine in Matlab to solve for the pseudo-inverse, and similar to [130], we adopt Iterative Proportional Fitting (IPF) to avoid negative values of the traffic matrix, which are without any physical meaning.

The size of \mathbf{H} could be very large in a large network, which makes the computational complexity (9) high. For example, in our experiment with a large tier-1 ISP backbone network, it can

take as much as tens of minutes to obtain a solution on a 900 MHz processor. This may hurt the applicability of the above method to some applications such as online diagnosis and failure detection, which require faster response time. To satisfy the requirement of these applications we design the following technique which reduces the computational complexity significantly while retaining a high accuracy of the derived traffic matrix. The idea is straightforward. We first sort the OD flows by their sizes in $\hat{\mathbf{X}}$. Then we divide them into two sets by comparing them to a threshold value T (e.g., 0.01% of the total volume). Let \mathbf{X}_L be the subvector of \mathbf{X} in which the corresponding OD flow has $\hat{x}_i \geq T$, and let \mathbf{X}_S be the subvector of the remaining \mathbf{X} such that $\hat{x}_i < T$. To speed up the computation, we hence focus only on obtaining a good estimate of \mathbf{X}_L while treating \mathbf{X}_S as known, which take values equal to their corresponding \hat{x}_i . Our problem in (6) and (7) becomes

$$\begin{aligned}\hat{\mathbf{X}}_L &= \mathbf{X}_L + \varepsilon^{\mathbf{X}_L} \\ \hat{\mathbf{B}} - \mathbf{A}_S \hat{\mathbf{X}}_S &= \mathbf{A}_L \mathbf{X}_L + \varepsilon^{\mathbf{B}}\end{aligned}$$

where \mathbf{A}_L and \mathbf{A}_S are the submatrices (columns) of the routing matrix \mathbf{A} that corresponds to \mathbf{X}_L and \mathbf{X}_S respectively. We apply the same solution technique in solving the above reduced system.

Here the threshold T should determine the desirable tradeoff between the computational complexity and the estimation accuracy. Fortunately, the OD flows in operational networks are often highly skewed [16]: a small number of OD pairs have very large traffic volume, while the majority of OD pairs have substantially low traffic between them. This is a very favorable property for our scheme. In our experiments, we can reduce the running time of the traffic matrix computation (for the same backbone network and on the same processor as above) to a few seconds by setting an appropriate threshold, meanwhile not comprising the overall accuracy by much (shown in Section 3.4).

We should note that the prior estimate of NetFlow measurement may significantly underestimate the volume of an OD flow due to unexpected errors. In this case, it is possible that an OD flow in \mathbf{X}_L be mistakenly placed in \mathbf{X}_S , which contaminates the rest of the computation. We rely on dirty data detection (Section 3.5.2) to correct such problems.

3.4 *Evaluation*

3.4.1 **Data gathering methodology**

We evaluate our techniques based on real network measurement data gathered from a large tier-1 ISP backbone network which consists of tens of Point of Presence (PoPs), hundreds of routers, and thousands of links, and carries over one petabyte of data traffic per day. Ideally, we would like to use both the real physical and logical (routing) network topology, and the true traffic matrix and link load information in our experiments. The former is readily available through the methods introduced in [112]. The latter, however, cannot be easily measured from the network, and is in fact the objective of this work.

To construct a traffic matrix that is as close to reality as possible, we use the data collected from our deployed IP flow measurement collection infrastructure [45] which applies the sampled NetFlow with sampling rate $1/500$ and the smart sampling with threshold 20MB. The data were collected over one month period from 8/15/2005 to 9/18/2005. In fact, our measurement infrastructure has very good coverage on the periphery of the network – the aggregated traffic volume computed from the collected data accounts for over 90% of the total volume observed by SNMP. We aggregate flows into a set of hourly traffic matrices [58]. Due to sampling (sampled NetFlow + smart sampling), some of the elements in the traffic matrix are zero (i.e., the traffic between the corresponding OD pair during the hour is completely missed by sampling). We fill in each zero-valued element a small number drawn randomly according to the model described in Section 3.3.1.1, unless the traffic matrix element is prohibited by routing (e.g., the traffic from one peering link to another peering link), in which case we keep its value as zero. We then simulate OSPF routing to derive a routing matrix \mathbf{A} , and project the above traffic matrices on \mathbf{A} to derive the corresponding link load information. In this way, we obtain a set of “actual” traffic matrices, an accurate routing matrix, and a set of “actual” link load data that are all consistent with each other.

Once the “actual” traffic matrices and the link load data are available, we can simulate the measurement noises and obtain a set of NetFlow and SNMP measurement data. We introduce noises on an element-by-element basis, following the models described in Sections 3.3.1.1 and 3.3.1.2. Such “contaminated” data best capture the quality of the network measurement data in reality, and thus will be used as input data in evaluating our algorithms.

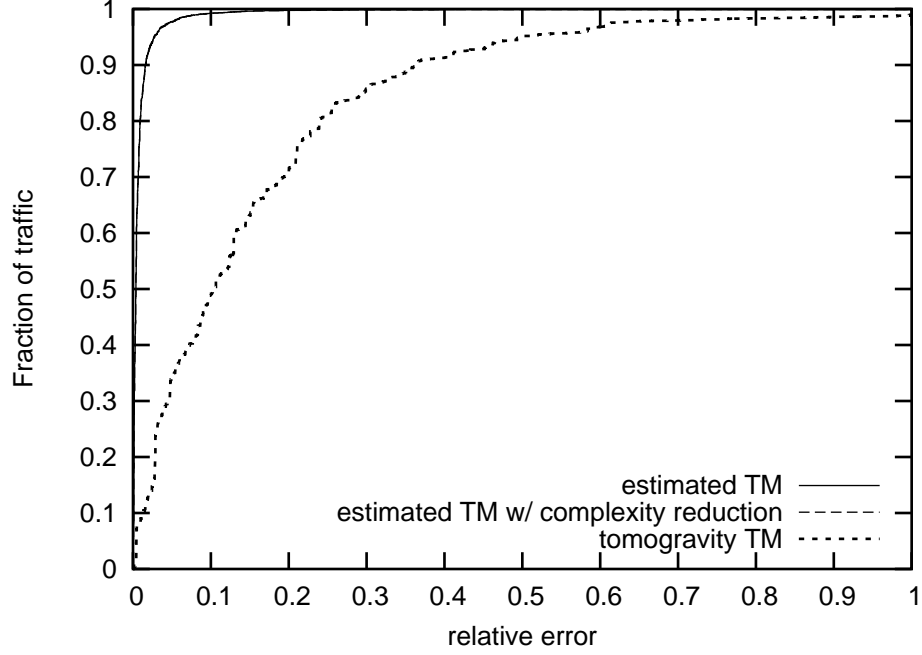


Figure 2: The weighted cumulative distribution of relative errors of estimated traffic matrices elements for our schemes and tomogravity method.

3.4.2 Experimental results

We first compare our approach in Section 3.3.2 and the tomogravity method introduced by Zhang et al. [130]. Figure 2 shows the weighted cumulative distribution (CDF) of relative errors of the estimated traffic matrix elements for these two methods, where the weight is the traffic volume of the OD flow. We observe a much better performance for our scheme: more than 90% of the traffic has negligible relative errors and almost all the traffic (i.e., 100%) has error less than 10% while the corresponding fractions for the tomogravity method are 20% and 50%, respectively. It demonstrates the advantage by making use of the NetFlow data in traffic estimation. Figure 2 also shows the cumulative distribution of relative errors when the complexity reduction scheme is in place. We set the threshold T to be 0.0005 times the volume of the the largest OD flow reported by NetFlow. The result is encouraging. The curve for complexity reduction scheme has no discernible difference compared to that without complexity reduction. This is because majority of OD flows are relatively small. However, the running time of traffic matrix computation is shortened from tens of minutes to several seconds. This suggests that we can focus only on the large flows in order

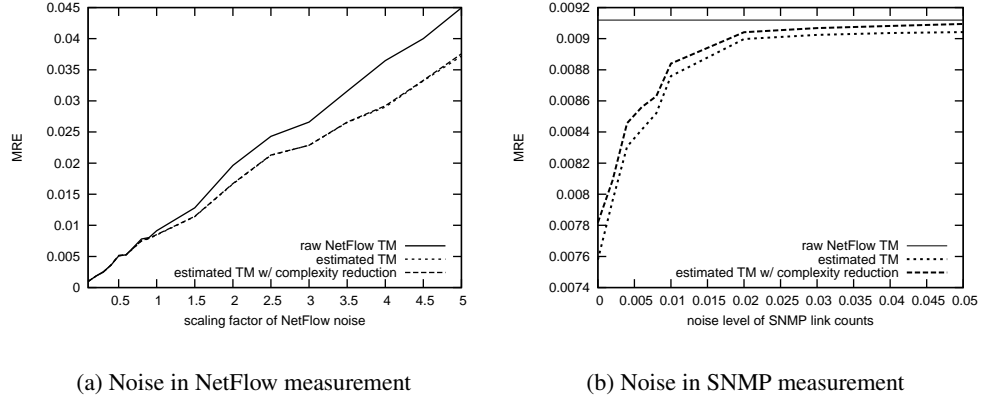


Figure 3: MRE as a function of different levels of noise in NetFlow and SNMP measurements.

to tradeoff for fast computation while not compromising the overall accuracy of the derived traffic matrices much.

Figure 2 provides us a view on the error distribution of all traffic matrix elements. To measure the overall accuracy of the derived traffic matrices, we adopt the Mean Relative Error (MRE) metric, which is introduced in [63] as follows:

$$MRE = \frac{1}{N_T} \sum_{i: x_i > T} \left| \frac{\hat{x}_i - x_i}{x_i} \right|$$

where N_T is the number of matrix elements that are greater than a threshold value T , i.e., $N_T = |\{x_i | x_i > T, i = 1, 2, \dots, N\}|$. Consistent with [63], we choose the value of T so that the OD flows under consideration carries approximately 90% of the total traffic.

We now evaluate the impact of different levels of noise in NetFlow and SNMP measurements on the traffic matrix inference. In particular, we tune the noise level according to the model presented in Section 3.3. Figure 3(a) compares MREs of the estimated traffic matrix using our scheme (with and without complexity reduction) with that of the traffic matrix directly obtained from the raw NetFlow data (i.e., the prior of our scheme) under different levels of noise in NetFlow measurement. Here the standard deviation of the noise of SNMP link loads is set to 0.01 times of the actual link loads which is a typical value in operational networks. We vary the scaling factor of NetFlow noise. Scaling factor 0 corresponds to perfect NetFlow measurement with no sampling conducted and scaling factor 1 corresponds to the existing measurement setup in the tier-1 ISP network from which our

data are collected, i.e., sampled NetFlow with $1/500$ sampling rate and smart sampling with 20MB threshold. A scaling factor of k corresponds to having a standard deviation k times of that from the existing setup, as the result of changing sampling rate or smart sampling threshold. We observe that MREs of both approaches increase as the scaling factor increases. Our scheme only improves to a very limited extend the estimation accuracy from the raw NetFlow data when the scaling factors are small. This can be explained as follows: NetFlow, although suffering from sampling errors, still provides good estimation of the traffic matrix when noise is small (i.e., sampling rate is high). Using a set of SNMP measurements, especially with poor quality, as part of the linear constraints does not provide a force that is strong enough to correct the sampling errors in NetFlow measurement, or on the other contrary, it may even make the result a little worse. However, when the accuracy from NetFlow degrades (scaling factor becomes large), the noisy SNMP measurement would guide the estimation closer to the actual values. This is manifested by the more pronounced improvement when the scaling factor becomes large (e.g., when the sampling rate becomes low). Note that we also compute the traffic matrices using tomography under the same settings. The MRE is 0.18 (not shown in Figure 3(a)), which is significantly higher than our scheme.

Next we study the impact of noise of SNMP measurement on the traffic matrix computation given a fixed noise level of NetFlow measurement (sampled NetFlow with $1/500$ sampling rate and smart sampling with 20MB threshold). Figure 3(b) shows MREs of the traffic matrix under different levels of noise in SNMP link load measurement. A noise level at α implies the standard deviation of the introduced Gaussian noise is α times of the true link load. We observe that the improvement from combining SNMP information to NetFlow measurement diminishes gradually as the noise in SNMP measurement increases. This is manifested by the reducing gap between the two curves, demonstrating the reduced power of recalibrating the NetFlow estimate by the noisy SNMP link loads. On the other hand, we find that once the SNMP measurement is not extremely distorted (e.g., $\alpha \leq 0.05$), our scheme always has an observable improvement in comparison to raw NetFlow traffic matrix (the horizontal line in Figure 3(b)). This is reassuring, since it suggests that our scheme by combining multiple data sources can be safely applied and it will not underperform methods that only use single data source (e.g., sampled NetFlow).

We also plot in Figure 3 the result when we apply our complexity reduction technique. Similar

to Figure 2, the results indicate that our proposed complexity reduction technique introduces little inaccuracy in traffic matrix computation.

Our evaluation thus far has found that our scheme improves, but to a very limited extent, the accuracy of the raw NetFlow traffic matrix. This is due to the ideal setting of the measurement environment under evaluation. Next, we discuss more practical problems presented in operational networks, where the advantage of using multiple data sources becomes evident.

3.5 Existing Challenges in Practice

Our inference techniques in Section 3.3 are based on the assumption that our observation of the SNMP data and NetFlow data are ideal. There are three aspects of this assumption. First, the SNMP data is considered perfect in the sense that there is no measurement error other than the small Gaussian noise. Second, the NetFlow data is considered perfect in the sense that all ingress points have NetFlow enabled and the gathering and accounting of NetFlow records are lossless and error-free (other than the sampling error). Third, there is a subtle assumption that network routing does not change during a measurement interval, since otherwise the routing matrix in (10) cannot be treated as a constant matrix. In practice, however, none of these three aspects will hold. In the following sections we will show how to enhance our previous scheme to handle the situation when the above assumption does not hold.

3.5.1 Partial NetFlow coverage

As discussed before, instrumenting a reliable IP flow measurement collection infrastructure (e.g., NetFlow) across the whole network is a difficult task for a large ISP. According to our experience it would take long time to obtain reasonably good coverage over the network periphery due to various operational issues (e.g., vendor implementation problems). Even when the measurement infrastructure is present, ensuring reliable data feeding and timely processing is nontrivial. It is well expected to have NetFlow data missing for some OD flows. Therefore the model we setup in Section 3.3.1.1 cannot always be completely populated. In this section we describe our solution in resolving this challenge.

The basic idea is to fill in the traffic matrix elements, which are not covered by NetFlow, with

our best estimate. So what is the next best thing when direct flow measurement is unavailable? The answer is the generalized gravity model. As shown in previous work [130, 131], generalized gravity model provides a reasonably good prior estimate of traffic demand. A simple gravity model works as follows. Let the total traffic volume going to an ingress router i be $T_{i,*}$. Then the traffic matrix element² $T_{i,j}$ is believed to be proportional to $T_{i,*} \cdot T_{*,j}$. This belief uniquely determines a traffic matrix vector $\mathbf{T}^{(g)}$, where “(g)” stands for gravity. However, $\mathbf{T}^{(g)}$ may not be consistent with the link count observations. Therefore, the tomogravity solution is to find a \mathbf{T} that is closest to $\mathbf{T}^{(g)}$ in a weighted fashion (i.e., “minimizing $\|(\mathbf{T} - \mathbf{T}^{(g)})/\mathbf{W}\|_2$ ” where \mathbf{W} is a weight vector³) among all traffic matrix vectors that are consistent with the link count observations (i.e., “subject to $\|\mathbf{AX} - \mathbf{B}\|_2$ being minimized”). It is determined empirically that setting weights w_i proportional to the square root of the estimation of x_i ($T_i^{(g)}$ in this case) results in the best estimation accuracy. The generalized gravity model enhances the simple gravity model by explicitly considering some routing policies, such as no transit for peers and hot-potato routing. It distinguishes edge links that connect to a customer (referred to as access links) and that connect to a peer (referred to as peering links), and then applies gravity assumption separately on traffic among the access links and between the access links and peering links. These enhancements enable the generalized gravity model to achieve good accuracy in the derived traffic matrices (around 30% relative error in a similar sized network [130]).

It is however very hard to blend the gravity model (simple as well as generalized) with the model derived from the NetFlow observations because the the gravity model is vaguely specified as “the probability model under which the above optimization procedure will produce a good estimator”. The implicit probability model in this gravity model has never been made explicit in [130] and the later works. One of the contributions we made here is to make it explicit using our Equivalent Ghost Observation (EGO) method. Let x'_1, x'_2, \dots, x'_n be the terms of the aforementioned $\mathbf{T}^{(g)}$ (in the matrix form) written into the vector form.⁴ We can prove, again using the Gauss–Markov theorem,

²Different from previous column vector notation, here $T_{i,j}$ denotes the volume of the traffic from ingress point i to egress point j .

³Here the division over W is an element-by-element division where the numerator and denominator are vectors of same length.

⁴We have also explored the scheme in which x'_i is determined by a “conditional” generalized gravity model, i.e., removing the traffic observed by NetFlow before applying gravity model. However, we find little performance difference using this variation in our experiments.

that if we take out all the beliefs of the gravity model (i.e., $T_{i,j} \propto T_{i,*} \cdot T_{*,j}$) and replace it with “ghost observations” x'_i where the error $x_i - x'_i$ is Gaussian with distribution $N(0, v_i^2)$ where v_i is proportional to the square root of x'_i , then the LS estimator of $(x_1, x_2, \dots, x_n)^T$ is exactly the result of the optimization specified by the gravity model (with the beliefs). We refer to these nonexistent observations x'_i , $i = 1, 2, \dots, n$, as EGO, as they are statistically equivalent to the beliefs in the gravity model.

Now blending our model with the gravity model becomes straightforward. For x_i terms that we have real (but noisy) observation from NetFlow, we use the probability model introduced in Sec. 3.1.1, that is, $x_i = \hat{x}_i + \varepsilon_i^x$ and $\varepsilon_i^x \sim N(0, \sigma_i^2)$. For x_i terms that we do not have real observation, we use the aforementioned EGO x'_i , but model the estimation error as having distribution $N(0, \lambda \sigma_i^2)$. Then applying the Gauss–Markov theorem to this blended model results in an estimator that is found to be fairly accurate. Note this estimator is no longer LS and BLUE since the gravity model (equivalently the EGO’s) is only a belief that is not backed up by actual observations. Here λ is a normalization factor introduced to capture the relative credibility of EGO in comparison to the NetFlow observations. We will study the impact of different choices of λ in Section 3.6.1, where we find the overall result insensitive to the choice of λ in a fairly large “good range”.

3.5.2 Removal of dirty data

The SNMP and NetFlow measurements, as the outcome of large scale complex measurement systems, inevitably suffer from a so-called *dirty data* problem. Unlike the data inaccuracy (Gaussian noise) discussed in Section 3.3, dirty data are results of unexpected hardware, software or transmission problems that cannot be modeled as measurement noises. For example, a reset of a network interface card during a collection interval may mess up the SNMP counters, thus producing completely bogus measurement result. More frequently than we would want, dirty data has caused many problems in network management including false alarms in anomaly detection, inaccurate traffic report and billing statement, and erroneous outcome from network analysis tools. In this section, we describe our algorithms to removing dirty measurement data by taking advantage of the multiple data sources that are available.

Since we define dirty data as measurement errors that cannot be captured by our noise model, it

is natural to devise an iterative dirty data removal process as follows:

- (i) let $\tilde{\mathbf{X}}$ be the result of solving (10); compute $\tilde{\mathbf{B}} = \mathbf{A}\tilde{\mathbf{X}}$;
- (ii) compute measurement noise $\tilde{\varepsilon}^{\mathbf{X}} = \tilde{\mathbf{X}} - \hat{\mathbf{X}}$ and $\tilde{\varepsilon}^{\mathbf{B}} = \tilde{\mathbf{B}} - \hat{\mathbf{B}}$; all $\tilde{\varepsilon}_i^{\mathbf{X}}/\sigma_i$ and $\tilde{\varepsilon}_j^{\mathbf{B}}/\mu_j$ should be Gaussian $N(0, 1)$ by our noise model, where $1 \leq i \leq n$ and $1 \leq j \leq m$.
- (iii) if the biggest element in $|\tilde{\varepsilon}_i^{\mathbf{X}}|/\sigma_i$ and $|\tilde{\varepsilon}_j^{\mathbf{B}}|/\mu_j$ is above a threshold (e.g., 3.09), we mark it as dirty and set $\hat{x}_i = \tilde{x}_i$ (or $\hat{b}_j = \tilde{b}_j$).
- (iv) repeats (i)-(iii) until no dirty data can be found.

The above approach, although intuitive, does not find good result. A possible reason to its poor performance is that the pseudo-inverse solution distributes the energy of dirty data to all possible explanations. In consequence, dirty data do not stand out.

Now we describe our approaches in identifying dirty data from contaminated measurement $\hat{\mathbf{X}}$ and $\hat{\mathbf{B}}$. Our methodology is to detect dirty data by applying Occam's Razor principle, which aims at finding a simplest explanation for the observed phenomena. The intuition is as follows. If an OD flow from NetFlow is dirty, it may cause inconsistency with all SNMP link measurement on the path the OD flow traverses. On the other hand, if an SNMP link load is dirty, it is inconsistent with the total traffic of all OD flows routed through the link. Since dirty data are rare, the simplest explanation of the inconsistency identifies the source of dirty data.

Let $\xi^{\mathbf{X}}$ and $\xi^{\mathbf{B}}$ be the dirty data component in the measurement. We have

$$\mathbf{X} = \hat{\mathbf{X}} + \varepsilon^{\mathbf{X}} + \xi^{\mathbf{X}}$$

$$\mathbf{B} = \hat{\mathbf{B}} + \varepsilon^{\mathbf{B}} + \xi^{\mathbf{B}}$$

Since a non-zero $|\xi_i^{\mathbf{X}}|$ should be much larger than σ_i (otherwise, it can be faithfully modeled by measurement noise), we expect such $|\xi_i^{\mathbf{X}}| \gg |\varepsilon_i^{\mathbf{X}}|$. Similarly, a non-zero $\xi_j^{\mathbf{B}}$ in SNMP data should have $|\xi_j^{\mathbf{B}}| \gg |\varepsilon_j^{\mathbf{B}}|$. We thus let dirty data component include both the measure noise and dirty data itself and distinguish them afterwards by comparing with Σ and Γ . Let ξ be the concatenated vector of $\varepsilon^{\mathbf{X}} + \xi^{\mathbf{X}}$ and $\varepsilon^{\mathbf{B}} + \xi^{\mathbf{B}}$. The problem becomes

$$\text{minimize } \|\xi\|_0 \quad \text{subject to } \mathbf{D} = \mathbf{C}\xi \quad (11)$$

where $||\xi||_0$ is the L_0 norm of vector ξ , $\mathbf{D} = \mathbf{A}\hat{\mathbf{X}} - \hat{\mathbf{B}}$ is the residual vector describing the inconsistency of NetFlow measurement and SNMP measurement, and $\mathbf{C} = (\mathbf{I}, -\mathbf{A})$ is an $m \times (m + n)$ matrix that horizontally concatenates an identity matrix \mathbf{I} and the negation of routing matrix \mathbf{A} . It can be easily verified that $\mathbf{D} = \mathbf{C}\xi$ is equivalent to $\mathbf{B} = \mathbf{A}\mathbf{X}$.

The L_0 norm is not convex and is notoriously difficult to minimize. Therefore in practice one needs to either approximate the L_0 norm with a convex function or use some heuristics, for example the greedy algorithms proposed in [78, 129]. Here we propose two schemes: one is a greedy heuristic algorithm for L_0 norm minimization and the other is doing L_1 norm minimization which is a common approach to approximate L_0 norm minimization.

Greedy algorithm

The algorithm starts with an empty set \mathbf{Z} of dirty data and then iteratively adds new dirty data to it. During each iteration, for each element $p \notin \mathbf{Z}$, the algorithm tests how much it can reduce the residual $\mathbf{D} - \mathbf{C}\xi$ by including p as a dirty data and chooses the element which can reduce the most L_2 norm of the residual, i.e., minimizing $||\mathbf{D} - \mathbf{C}\xi||_2$. The algorithm stops whenever either the residual energy falls below some tolerance factor or the number of dirty data exceeds some pre-defined threshold.

L_1 norm minimization

As shown in [40, 129], L_1 norm minimization results in the sparsest solution for many large under-determined linear systems and therefore is a common approach to approximate and convexify L_0 norm minimization. Here, we apply the same principle. That is,

$$\text{minimize } ||\xi||_1 \quad \text{subject to } \mathbf{D} = \mathbf{C}\xi$$

We can further transform the above into an unconstrained optimization problem by moving the constraints into the objective function in the form of a penalty term, i.e.,

$$\text{minimize } \theta ||\xi||_1 + ||\mathbf{D} - \mathbf{C}\xi||_1$$

where $\theta \in [0, 1]$ is a weight parameter that controls the degree to which the constraints $\mathbf{D} = \mathbf{C}\xi$ are satisfied. We find the optimization result not very sensitive to the choice of θ . Thus, we set it to 0.01 in the rest of the chapter.

We can cast the above problem into the following equivalent Linear Programming (LP) problem, for which solutions are available even when \mathbf{C} is very large, owing to modern interior point linear programming methods.

$$\begin{aligned}
& \text{minimize} && \theta \sum_i u_i + \sum_j v_j \\
& \text{subject to} && \mathbf{D} = \mathbf{C}\xi + \mathbf{d} \\
& && \mathbf{u} \geq \xi, \mathbf{u} \geq -\xi \\
& && \mathbf{v} \geq \mathbf{d}, \mathbf{v} \geq -\mathbf{d}
\end{aligned}$$

3.5.3 Handling routing changes

Changes to the routing tables can happen anytime and generally do not align with the beginning of SNMP measurement intervals. If a route change happens within a polling interval, the corresponding measurement would reflect the total traffic volume of both before and after the route change. This creates a problem for our model $\mathbf{A}\mathbf{X} = \mathbf{B}$, since the routing matrix \mathbf{A} is no longer constant. Moreover, change of internal routing structure may have impact on the ingress and egress point of traffic demand, resulting in changes in the traffic matrix to be estimated. For example, traffic destined to a peer may shift its egress point from one peering link to another due to hot-potato routing. To address this problem, we propose the following solution.

Assume the routing only changes once during that measurement epoch (the solution can be easily adapted to the case where there are more than one routing changes happening.). Let \mathbf{A}_1 and \mathbf{A}_2 be the routing matrix before and after the change. Let \mathbf{X}_1 and \mathbf{X}_2 denote the corresponding traffic matrices to be estimated. We have

$$\mathbf{A}_1\mathbf{X}_1 + \mathbf{A}_2\mathbf{X}_2 = \mathbf{B}$$

Since we can obtain the exact time of routing change from monitoring tools [112] and the flow records from NetFlow are timestamped, we can easily compute $\hat{\mathbf{X}}_1$ and $\hat{\mathbf{X}}_2$ if the flow measurement covers all OD flows. In the case where flow measurement is not complete, we have to rely on the generalized gravity model to populate the a priori estimate. However, since SNMP measurement is indivisible within a polling interval, we can only prorate the traffic volume for the duration when

\mathbf{A}_1 or \mathbf{A}_2 prevails and so do their MSE vectors. Obviously, this introduces additional risk of inaccuracies of the estimated traffic of the OD flow, which we can compensate by increasing the value of λ in Section 3.5.1.

Similar to (8) the problem here also can be described by the following linear system.

$$\mathbf{Y} = \mathbf{H}\mathbf{X} + \mathbf{N} \quad (12)$$

Here \mathbf{X} is the concatenated vector of \mathbf{X}_1 and \mathbf{X}_2 , \mathbf{N} is the concatenated vector of $\varepsilon^{\mathbf{X}_1}$, $\varepsilon^{\mathbf{X}_2}$ and $\varepsilon^{\mathbf{B}}$, $\mathbf{H} = (\mathbf{I}; (\mathbf{A}_1, \mathbf{A}_2))$ is an $(m + 2n) \times 2n$ matrix which vertically concatenates an identity matrix \mathbf{I} and the horizontal concatenation of the two routing matrices \mathbf{A}_1 and \mathbf{A}_2 , and \mathbf{Y} is the concatenated vector of $\widehat{\mathbf{X}}_1$, $\widehat{\mathbf{X}}_2$ and $\widehat{\mathbf{B}}$.

The same solving procedure as in Section 3.3.2 can be followed to obtain the LS estimators of \mathbf{X}_1 and \mathbf{X}_2 . Notice that the size of \mathbf{H} here is even larger than that in Section 3.3.2 and therefore leads to the higher computational complexity. The situation is worse when more routing changes occur during the estimation interval as the size of \mathbf{H} increases with the number of routing changes: \mathbf{H} is an $(m + n \times i) \times (n \times i)$ matrix for i routing changes. Therefore the complexity reduction technique in Section 3.3.2 becomes increasingly important here. We will show in Section 3.6.3 that we can achieve very desirable accuracy within tens of seconds with the complexity reduction technique for a reasonably sized network.

3.6 *Evaluation of Enhancements*

In this section, we present the evaluation results on the impact of the aforementioned three aspects of practical challenges on the accuracy of traffic matrix estimation, and correspondingly the performance of our proposed techniques to these challenges.

Unless specified otherwise, we use the following default parameters in this section: the noise level of SNMP link counts is 0.01; the scaling factor for the NetFlow noise is 1; the threshold for complexity reduction T is set such that around 1000 OD flows are selected in \mathbf{X}_L .

3.6.1 Incomplete NetFlow data

As we mentioned before, partial deployment of NetFlow and incomplete NetFlow measurement data are very common in operational networks. In this section, we evaluate our scheme towards estimating traffic matrices based on incomplete NetFlow data. Our evaluation also provides us insight on the effectiveness of various deployment strategies of the NetFlow measurement infrastructure for traffic matrix computation.

Recall that we have incorporated a parameter λ in our scheme in the presence of incomplete NetFlow measurement. λ captures the relative quality between the NetFlow measurement and the prior from the generalized gravity model. A larger value of λ corresponds to a more accurate NetFlow measurement in comparison to the generalized gravity model. Figure 4 evaluates the performance of our scheme with different values of λ .

In Figure 4, we study the estimation error (measured by MRE) with varying λ when the NetFlow measurement is only available at the top 20% edge routers (ranked by its total ingress traffic volume) assuming two different scaling factors, 1 or 4, of the NetFlow noise. We observe that, in both cases, there exists an optimal value for λ that achieves the minimum MRE. This value of λ best reflects the relative accuracy of sampled NetFlow and generalized gravity model defined in our solution framework. The optimal value of λ is found at around 40 when the scaling factor is 1 and at around 10 when the scaling factor is 4. This matches well with our expectation since the relative accuracy of sampled NetFlow is low when the NetFlow noise level is high (scaling factor 4), resulting in a reduced optimal penalty weight for generalized gravity model. The ratio in their optimal penalty weight (40/10) matches well with the inverse of their noise scaling factor of NetFlow (4/1). Furthermore, we observe that the performance of our approach is robust to the choice of λ at higher values (note that the x-axis is in logscale). In other words, the performance degradation due to using a λ value higher than the optimal is not dramatic. It suggests that when applying our approach in operation, one does not need to put too much effort in tuning the parameter λ to get a good performance. In the rest of the evaluation, we set $\lambda = 40$ (matching the scaling factor 1).

Now we evaluate the accuracy of our proposed scheme with NetFlow coverage on the ingress points ranking in top 20% by traffic volume (we will study other strategies later) and compare it

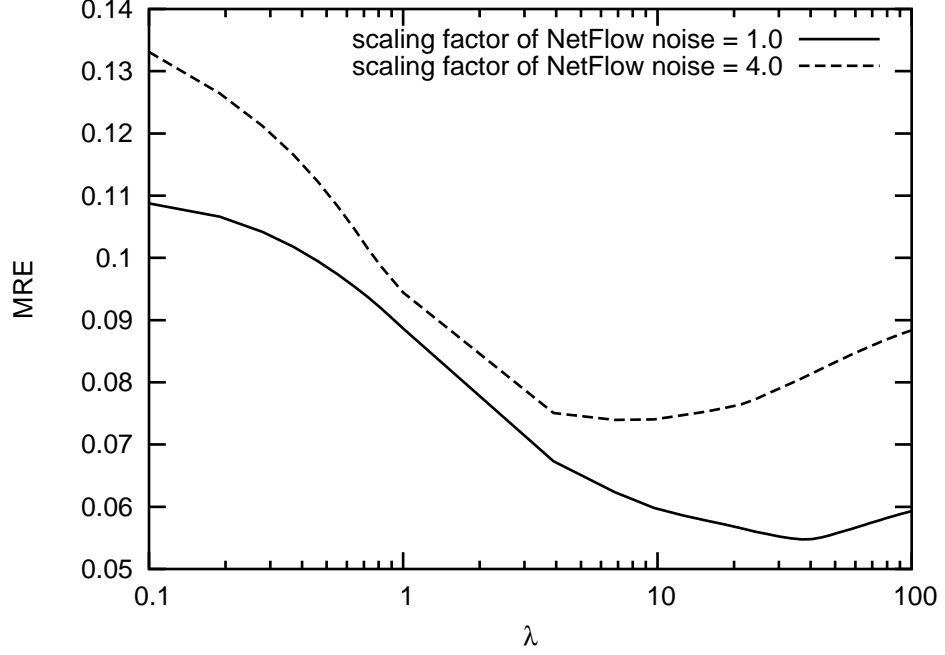


Figure 4: MRE under different values of λ (x axis is in logscale)

with those from the generalized gravity model, tomogravity solution and the NetFlow measurement amended by EGO. Figure 5 plots the CDFs of the relative errors of the estimated volume of OD flows obtained by those approaches. We observe that our approach, which makes use of the NetFlow measurement at 20% of the ingress points, already achieves a significantly better accuracy than that of the tomogravity solution, which only depends on the SNMP measurement. This improvement is mostly due to the better prior estimate from the EGO-amended NetFlow than from the generalized gravity model, as indicated by the gap between the corresponding two curves. Finally, our optimization method further improves the prior estimate from the EGO-amended NetFlow to achieve an even higher accuracy. We have also plotted in Figure 5 the result when we apply our complexity reduction technique. The curve overlaps with that of the estimated traffic matrix without complexity reduction, indicating little inaccuracy has been introduced because of the approximation in the complexity reduction.

We next study the NetFlow deployment problem: given a fixed number (e.g., 20% ingress points) of ingress points that can deploy NetFlow, how to choose ingress points where the NetFlow measurement data yield the most accurate traffic matrix estimation? This problem is briefly

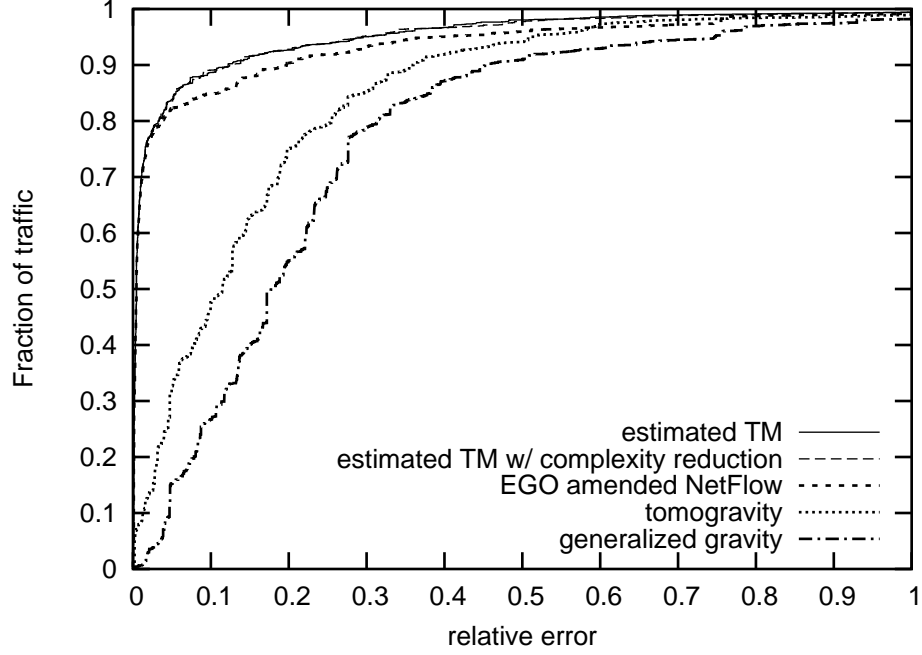


Figure 5: The fraction of traffic vs. the magnitude of the relative error.

studied in [85] and [131]. Here we revisit this problem with more comprehensive evaluation. In particular, we evaluate the following three different types of strategy that (i) randomly select x fraction of ingress points; (ii) select top x fraction of ingress points ranked by traffic volume generated from actual value, generalized gravity model and our scheme, respectively; and (iii) select top x fraction of ingress points ranked by the difference of traffic volume between the actual value and result of the generalized gravity model and between the result of our scheme and that of the generalized gravity model, respectively. Figure 6 shows the accuracy of traffic matrix estimation for various value of x . It is not surprising that random selection performs the worst. The strategies in (ii) (i.e., ranking ingress points by traffic volume) yield the best overall performance. In addition, we observe that the curves for strategies in (ii) and (iii) become flat after x is approaching 0.6. This indicates that deploying NetFlow measurement on more than 60% of ingress points only has marginal gain under careful deployment decision.

Instrumenting NetFlow measurement infrastructure on a set of ingress points to facilitate traffic matrix computation is an arduous engineering task, which involves careful testing, certification and performance tuning, etc.. We would like the set of ingress points for NetFlow deployment, which is

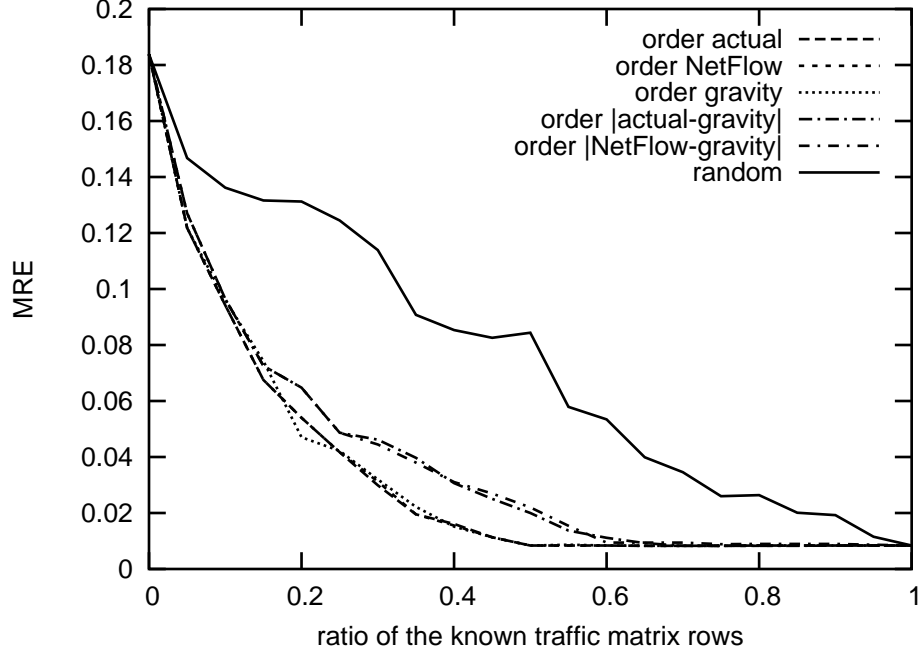


Figure 6: Impact of partial deployment of NetFlow on traffic matrix estimation.

based on most up-to-date traffic volume information, to be stable over a long time. In our evaluation, we define a stability function $f = |\alpha \cap \beta|/|\beta|$, where α and β are two sets of ingress points that are selected base on traffic information at time t_α and t_β , respectively. An f value closer to 1 indicates that the two sets share a large number of overlapping elements. Figure 7 shows the change of value of f during one week period (August 15-21, 2005). In this case, t_β is set to 0:00 AM on August 15, 2005 and t_α varies from 0:00 AM on August 15 to 23:00 PM on August 21. Two sets of ingress points are selected: the ingress points ranking at top 10% and 30% in traffic volume, respectively. We observe that the values of f for both sets are fairly high (always above 0.8). This implies that network administrators can make their deployment decision based on traffic volume without the risk of major re-deployment in a short term.

3.6.2 Dirty data

In order to evaluate our scheme in identifying and removing dirty data, we synthetically inject some dirty data into both the SNMP link loads and the sampled NetFlow measurement. In particular, we introduce two types of dirty data in SNMP link loads. In the first type, we scale up an SNMP

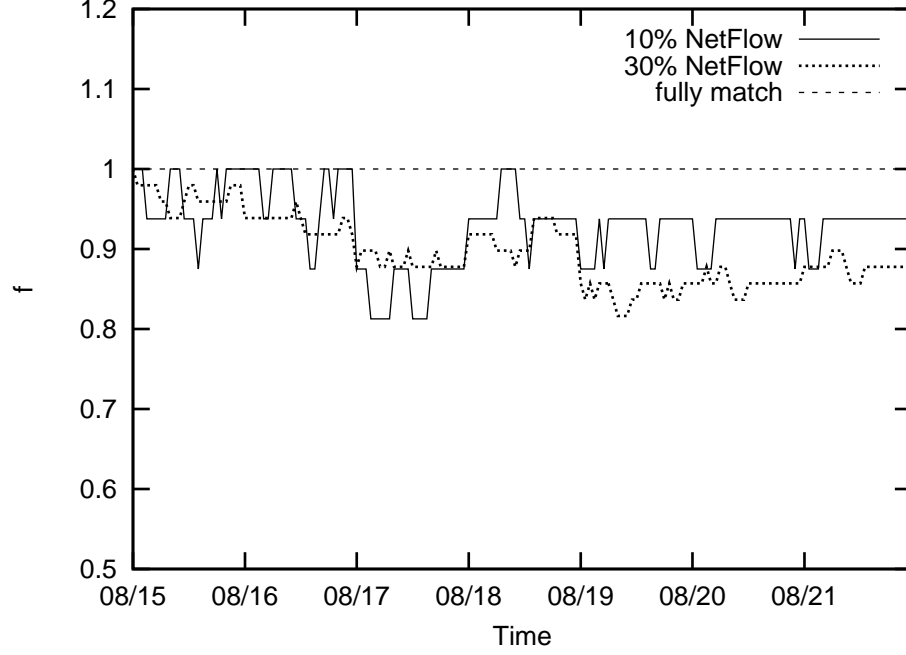


Figure 7: The stability of the NetFlow deployment decision.

measurement by a factor of 5 on a link chosen at random among the top 20% links (ranked by the traffic volume). In the second type, we scale down an SNMP measurement also by a factor of 5 on a link chosen randomly among the top 2% links. We introduce three types of dirty data into NetFlow measurements. In the first two types, we scale down a NetFlow measurement by a factor of 5 on an OD flow chosen at random among the top 0.2% and the top 2% OD flows respectively. In the third type, we scale up the NetFlow data by a factor of 10 on an OD flow chosen randomly among the top 20% of OD pairs. In the evaluation that we show next, the results are obtained when we inject one dirty data of each type into the measurement.

We plot the MREs of our estimation with and without dirty data in Figure 8 to illustrate the negative impact of the small number of dirty data. Figure 8(a) shows the results in which Netflow has complete coverage at the network periphery, while Figure 8(b) presents the case where NetFlow measurement is available at the top 20% edge routers (ranked by the total ingress traffic volume). We repeat our experiment with different choice of dirty data and present the average result in the graphs. We first look at the left three columns of these graphs. The first column represents the MRE of the estimated traffic matrix when there is no synthetically introduced dirty data; the second

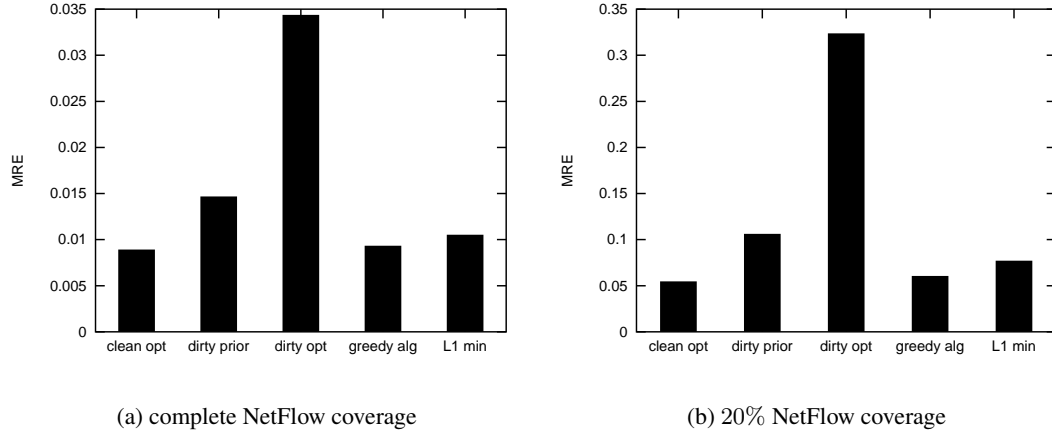


Figure 8: Impact of dirty data on traffic matrix estimation.

column represents the MRE of the Netflow or the EGO-amended gravity prior after we inject one dirty data of each type; and the third one corresponds to the MRE of the our least square estimate of the traffic matrix with those dirty data. We observe that the small number of dirty data have a strong disturbing effect to our quadratic optimization in (9): the optimization result with a handful of dirty data (the third column) is 3 times (in Figure 8(a)) or 6 times (in Figure 8(b)) worse than the result when dirty data is not present (the first column). It is interesting to observe that the contaminated Netflow or the EGO-amended gravity prior in the second column has a better accuracy than the third column. This suggests that dirty data in the SNMP measurement, as part of the optimization constraints, may steer the optimization result away from the actual traffic matrix, causing significant inaccuracy in traffic matrix estimation. In fact, dirty data phenomena has caused many problems in various applications in network management besides the traffic matrix estimation, and becomes quite a headache to network operators.

We now apply our techniques in Section 3.5.2 to remove dirty measurement data. We use a threshold of 3.09, which corresponds to 99.9 percentile of a standard Gaussian, as the cut off between dirty data and the normal measurement noise. Since for most of the applications of traffic matrix estimation, only the dirty data of significant size is of interest, we further filter the identified dirty data such that only those correspond to a data rate higher than 10Mbps are reported. When the sampled NetFlow covers the whole network, with the above configuration our scheme using the

greedy algorithm identifies all of the five injected dirty data with 3 false positive on the average, and our scheme using L_1 norm minimization identifies all five injected dirty data with 22 false positives on the average (with repeated experiments with different base traffic matrices and random choices of dirty data). When NetFlow measurement only covers the top 20% edge routers ranked by the total ingress traffic volume, we find that our greedy algorithm can typically produce no false negatives and a small number (≤ 5) of false positives, while the L_1 norm minimization has a slightly worse performance, producing occasionally one false negative and a small number (≤ 35) of false positives.

After we identify the dirty data, we correct the corresponding elements (by subtracting the dirty components) in our problem formulation and then derive an estimate of the traffic matrix. The result is shown in Figure 8(a) and (b), where the fourth columns are the results when we use the greedy algorithms in identifying the dirty data and the fifth columns are the results when we apply L_1 minimization in identifying the dirty data. We find that both algorithms achieve comparable accuracy than that when dirty data are not present. Comparing the two methods, our greedy algorithm has a slightly better performance than the L_1 minimization.

3.6.3 Routing changes

In this section, we evaluate the impact of routing changes during a measurement period and the effectiveness of our proposed solution to this issue. Our experiment scenario is constructed as follows. We assume the measurement interval is of one hour length and a routing change occurs at the end of the 10th minute. This routing change is a result of the failure of an internal link, which is chosen at random. We simulate the routing of the network before and after the failure to compute the corresponding routing matrices, \mathbf{A}_1 and \mathbf{A}_2 . To obtain the traffic demand both before and after the failure, we pick the actual traffic matrices from two consecutive hours and prorate the traffic to populate the traffic matrix in the first 10 minutes and the last 50 minutes respectively. Finally, we compute the link load measurement over the hour, \mathbf{B} , by summing up the total traffic of both the first 10 minutes and the last 50 minutes on each link. The result is fed into our algorithm in Section 3.5.3 for traffic matrix estimation. We use the proposed complexity reduction technique (i.e., targeting at top 1,000 elements in \mathbf{X}_1 and \mathbf{X}_2 respectively) to speed up the computation. In order to form the

base for our comparison, we also consider the case in which there is no routing change during the hour. This can be constructed by simply using the traffic matrix and routing matrix from the first hour of the two consecutive ones. We next present the result of one example constructed using the above settings. Results of other cases based on different traffic matrices and choices of link failures are quantitatively similar.

Figure 9 shows the CDF of the relative error of the estimated volume of the OD flows. The two solid lines are the result derived solely based on SNMP link load, i.e., the traffic matrix being estimated entirely from the EGO of the prorated generalized gravity model, with and without routing change. We observe that the accuracy of the derived traffic matrix degrades significantly when routing change occurs during the measurement interval. This is because SNMP link load measurement does not contain sufficient information to distinguish the traffic before and after the routing change. The two dotted lines in Figure 9, however, show the result of our approach when NetFlow data is available at the top 30% edge routers. We make two observations here. First, we find that the estimation accuracy is significantly improved with the additional information from NetFlow. This echoes our observation in Section 3.6.1. Second, we observe that the performance degradation due to the routing change, shown as the difference between the case with routing change and the case without routing change (gap between the two dotted lines), is much less than that of the SNMP only scenario (gap between the two solid lines). Again, this demonstrates the effectiveness of our approach, and more importantly the power of combining multiple data sources in deriving a good estimate of traffic matrix.

3.7 Combining Time Series Information

A large number of data sets in statistics, signal processing, and econometrics are discovered to own temporal correlation over time and quite a few time series models have been developed to understand such correlations and make forecasts (predictions) based on known past events. This time series information is beginning to play an important role in a lot of applications such as economic forecasting, stock market analysis, workload projection and census analysis. In the context of network traffic analysis, it is also well believed that some implicit temporal correlations exist in the

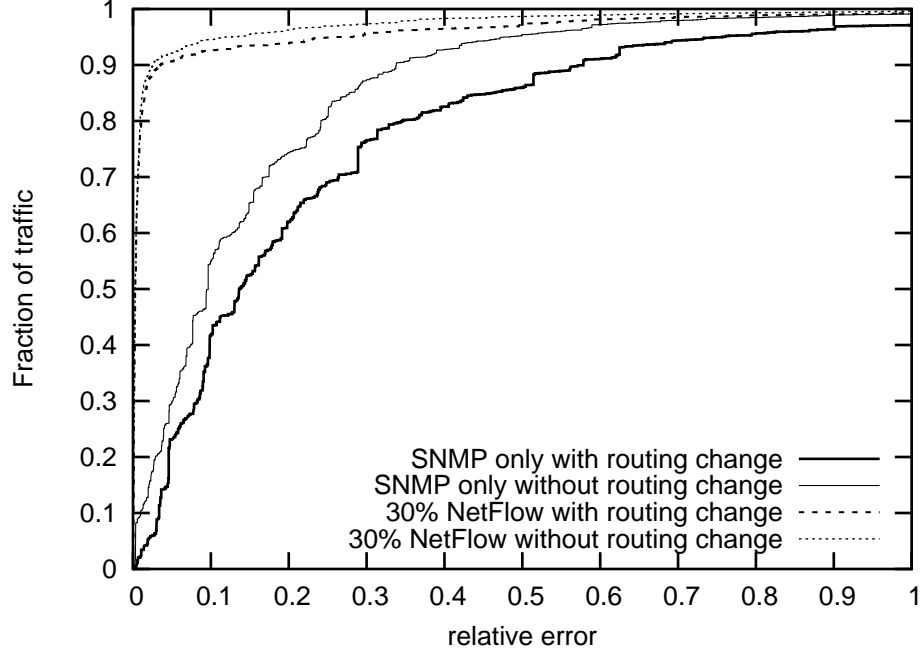


Figure 9: CDFs of the relative errors of the estimated TM elements with and without routing changes during a measurement interval

traffic demands of a given network. In this section, we carefully explore these temporal correlations. We develop the a time series predictive model and design the methodology to integrate it with other information together. In our experiments, we find that the evolution of the traffic demands over time can be modeled using our proposed model reasonably but unfortunately this part of information can only produce a marginal improvement on the estimation accuracy in our framework. In the rest of this section, we will describe our predictive model, estimation methodology and some experimental results in detail.

We treat the whole traffic matrices (organized as a vector) evolving along with time as a discrete time stochastic process $\{\mathbf{X}_t, t = 0, 1, 2, \dots\}$ where t is the index of time interval. We propose to adopt Autoregressive (AR) models with Gaussian noise for capturing the temporal correlations in traffic matrices over time in this work since it has a long record of successful application in a wide spectrum of monitoring problems. Concretely, we build the following AR model:

$$\mathbf{X}_t = \sum_{p=1}^P \mathbf{C}_p \mathbf{X}_{t-p} + \boldsymbol{\eta}_t \quad (13)$$

where P is the model order, \mathbf{C}_p is the AR coefficients and $\boldsymbol{\eta}_t$ is the driven noise which accounts for

both randomness of the traffic volume and the unpredictable elements. $\boldsymbol{\eta}_t$ is assumed to be normally distributed with covariance matrix $\boldsymbol{\Theta}$. The model order P is an important parameter in general and should be configured carefully. If P is set too large, the model prediction would require long history data and large storage space; if P is too small, the estimate cannot full enjoy the time-regression for prediction. In [79], a first-order AR (AR(1)) model is adopted by default, i.e., $P = 1$. However, there is no model order justification in that work to prove first order is the optimal choice. Here we let P and other related parameters \mathbf{C}_p and $\boldsymbol{\Theta}$ undefined in the first place. We then use history data and information criterion to determine their good optimal values.

We adapt the method in [113] for finding optimal P value and corresponding \mathbf{C}_p and $\boldsymbol{\Theta}$. This method balances the risk due to the bias when a lower model order is selected and the risk due to the increase of variance when a higher order is selected. We evaluate the performance of this model using the data collected over two continuous weeks in July of 2006 from a large tier-1 ISP IP backbone and a measurement time interval lasts for 1 hour. We use the sampled NetFlow data of the first week to compute P value and the corresponding parameters.⁵ We then use the derived model and the data in the second week to plot the model prediction values in Figure 10. We compare the prediction results based on the following three different data sets with the results from sampled NetFlow data (solid curve) and generalized gravity model (dashed curve), respectively: i) actual values; ii) sampled NetFlow; iii) generalized gravity model, in that figure. We observe that i) and ii) generate very similar result (hard to distinguished in the figure) and iii) is slightly worse. We conclude that all the three data sets can produce reasonably good predictions but the prediction may significantly deviate the actual values sometimes, which is manifested by the spikes of the curves.

Next we demonstrate how to integrate the time series information into our previous methodology framework. For simplicity, we assume that the routing matrix \mathbf{A} does not change in the experimental period. Then we can modify the previous problem formulation as follows:

⁵We also tried other choices such as the data from generalized gravity model but did not find any observable differences in the results.

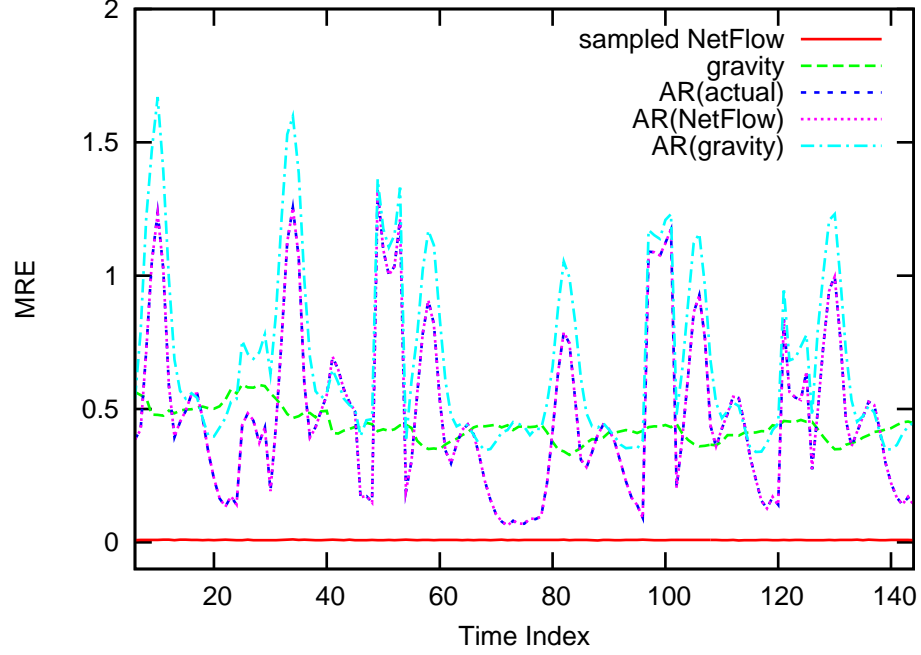


Figure 10: MRE of AR(P) model

$$\begin{cases} \hat{\mathbf{X}}_t = \mathbf{X}_t + \varepsilon_t^{\mathbf{X}} \\ \hat{\mathbf{B}}_t = \mathbf{A}_t \mathbf{X}_t + \varepsilon_t^{\mathbf{B}} \end{cases} \quad (14)$$

We again combine these two equations into a single equation as follows:

$$\hat{\mathbf{Y}}_t = \mathbf{H} \mathbf{X}_t + \mathbf{N}_t \quad (15)$$

Here all the notations are as same as the previous definition in 8 except the added time index t . Recall that $\mathbf{H} = (\mathbf{I}, \mathbf{A}^T)^T$ is an $(m+n) \times n$ matrix which vertically concatenates an identity matrix \mathbf{I} and the routing matrix \mathbf{A} , $\hat{\mathbf{Y}}_t$ is the concatenated vector of $\hat{\mathbf{B}}_t$ and $\hat{\mathbf{X}}_t$, and \mathbf{N}_t is the concatenated vector of $\varepsilon_t^{\mathbf{B}}$ and $\varepsilon_t^{\mathbf{X}}$ with covariance matrix $\mathbf{K} = E[\mathbf{N}_t \mathbf{N}_t^T]$. Because all the observations are obtained independently, covariance matrix \mathbf{K} is diagonal. The remaining problem is how to optimally combine all the information together to obtain the best possible results. The fundamental tradeoff here is between data fitting and model fitting. If we use the sampled NetFlow and SNMP data without considering the time series model, then the estimation accuracy is bounded by the variance of the measurement noises, and the estimate sequence may change dramatically from time to time

which may not match with the reality. On the other hand, if we fully or mostly depend on the time series model, then the model fitting error will restrict the estimation performance significantly as shown in Figure 10. Therefore, we need somehow to balance the confidence on these two types of information.

Based on (13), we can find that the prior of \mathbf{X}_t is also Gaussian distributed. Therefore, the maximum-likelihood (ML) estimator on \mathbf{X}_t by taking into account data and model information is the solution to minimize

$$\|\mathbf{K}^{-1/2}(\hat{\mathbf{Y}}_t - \mathbf{H}\mathbf{X}_t)\|^2 + \lambda^2 \|\boldsymbol{\Theta}^{-1/2}(\mathbf{X}_t - \sum_{p=1}^P \mathbf{C}_p \mathbf{X}_{t-p})\|^2 \quad (16)$$

where the first term corresponds to the performance of data fitting and the second term corresponds to the performance of model fitting. Recall that \mathbf{K} and $\boldsymbol{\Theta}$ are the known weight matrices which can be calculated based on (15) and (13), respectively, and λ is the parameter to trigger the tradeoff between data fitting and model fitting. When λ is equal to 0, the problem reduces to our previous one (10). The larger the value of λ , the more weight put on time series information. In our experiments we find that the choice of λ is insensitive to the overall estimation accuracy in a fairly large range from 10^{-3} to 0.6 and hereby we can choose a good value easily. In the following experiments, we set its value to 0.15. After that we can derive the WLS estimation of \mathbf{X}_t which is given by

$$\tilde{\mathbf{X}}_t = (\mathbf{H}^T \mathbf{K}^{-1} \mathbf{H} + \lambda^2 \boldsymbol{\Theta}^{-1})^{-1} (\mathbf{H}^T \mathbf{K}^{-1} \hat{\mathbf{Y}}_t + \boldsymbol{\Theta}^{-1} \sum_{p=1}^P \mathbf{C}_p \mathbf{X}_{t-p}) \quad (17)$$

Again using the data collected over two continuous weeks in July of 2006 from a large tier-1 ISP IP backbone we evaluate the performance of this solution. Figure 11(a) compares MREs of the estimation with that from our previous scheme and sampled NetFlow data respectively, when sampled NetFlow is deployed over all the ingress points of the network. We observe that our newly proposed scheme achieves the nearly same estimation accuracy as our previous scheme manifested by the indistinguished curves from them. In Figure 11(b), we run the same comparison but now sampled NetFlow only covers 20% ingress points of the network. Here we adopt a deployment strategy described in Section 3.6.1 which selects top 20% fraction of ingress points ranked by total traffic volume during the first time interval. We find the similar observation here: combining time series information only generates marginal improvement. Notice that in Figure 11(b) the MREs

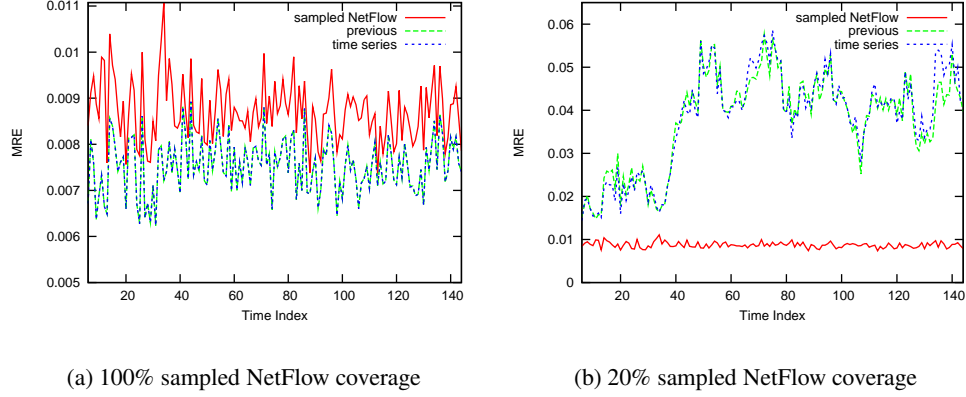


Figure 11: Comparison between this scheme and prior solutions

resulted from the schemes combining time series information are relatively small for the first around 40 measurement time intervals. This is because we select the set of ingress points to deploy sampled NetFlow based on the measurement in the first time interval but this set is changing gradually over time (recall Figure 7). When the time duration is long enough (about 40 hours in our experiment) a nontrivial fraction of the original set members should be swapped out, which downgrades the estimation accuracy for the current measurement time interval.

3.8 Conclusion

In this chapter we present several novel inference techniques for robust traffic matrix estimation with both imperfect SNMP link counts and sampled NetFlow measurement. In contrast to previous work, our schemes take advantage of both NetFlow and SNMP link loads data to obtain better estimation. We find that under the existing configuration of sampled NetFlow the result from NetFlow is quite accurate and combining SNMP link loads with it only improves the estimation accuracy slightly if NetFlow is fully deployed in the network. However, when full deployment of NetFlow is not available – a common case in operational networks, our algorithm can improve estimation accuracy significantly even with a small fraction of NetFlow data.

More importantly, we show that dirty data can contaminate a traffic matrix. We design two novel algorithms to identify and remove dirty data in sampled NetFlow and SNMP data. This would benefit a number of important network management applications including the traffic matrix

estimation. We show that by using our algorithms, the errors in traffic matrix estimation can be reduced by more than an order of magnitude. Finally, we observe that routing and topology change is also key factor that affects traffic matrix estimation. We develop a novel algorithm for estimating more accurate traffic matrices upon topology and routing changes.

We also explore the temporal correlation among the traffic matrices evolving along with time and study the possibility to combine it with other information sources to further improve traffic matrix estimation. We find out that even though traffic matrices in a large ISP network do have some temporal correlations it only produces the marginal gain on overall estimation accuracy. We are investigating some other ways for further improvement as follows: i) changing predictive model to capture temporal correlations more accurately; ii) devising some mechanism to identify and filter the abnormal traffic which deviate the times series model significantly in predicting iii) designing another framework to better combine all the information sources.

To the best of our knowledge, this work is the first to offer a comprehensive solution which fully takes advantage of using multiple readily available but imperfect data sources. The experimental results based on real data obtained from a large tier-1 ISP backbone network provide valuable insight on the effectiveness of combining multiple data sources to estimate traffic matrices.

CHAPTER IV

MEASUREMENT OF TRAFFIC AND FLOW MATRICES

4.1 Introduction

In Chapter 3 we present some signal processing techniques to combine link-level data (SNMP link counts) and path-level data (sampled NetFlow records) for alleviating “too little data” problem and improving estimation accuracy. Here we study the same problem from another angle. We know that the path-level measurements can produce an estimation of the traffic matrix based on the recorded traffic statistics directly. The most common method for path-level is sampling as discussed in Section 2.2, which suffers from the low sampling probability to make the sampling operation affordable. In this chapter, we present a data streaming algorithm to replace sampling for data reduction. We show that the proposed scheme can provide higher estimation accuracy than the sampling scheme given the same time and storage complexities.

Besides this, we notice that sometimes the traffic matrix is not yet fine-grained enough for some flow-oriented applications such as inferring the usage pattern of ISPs, detecting route-flapping, link failure, DDoS attacks, and Internet worms [46, 47, 73]. Traffic matrices only split total traffic volume among different OD pairs, not among different flows. In this work we define a new term “flow matrix” which quantifies the traffic volume of flows between OD pairs in a network. Compared with traffic matrix, flow matrix is at a finer grained level and is more useful for the flow-oriented applications. Correspondingly, we design a fast and efficient data streaming algorithm to estimate flow matrices in this work.

The main idea of our algorithms is to first perform data streaming on participating ingress and egress routers of the network to achieve data reduction. They generate streaming digests that are orders of magnitude smaller than the original traffic stream. These digests are shipped to a central server on demand, when traffic/flow matrix needs to be estimated. We show that even with such small digests our schemes can obtain estimates with high accuracy.

Two data streaming algorithms are proposed, namely, *bitmap* based and *counter-array* based

algorithms, for estimating traffic matrices and flow matrices, respectively. The data structure of the *bitmap* algorithm is extremely simple: an array of bits (bitmap) initialized to zero on each monitoring node. For each packet arrival at a participating node, the node simply sets the bit, indexed by the hash value of the part of this packet (described in Section 4.3.1), to 1 in that array. To estimate a traffic matrix element $TM_{i,j}$, two sets of bitmaps are collected from the corresponding nodes i and j , and are fed to a sophisticated estimator. Our key contribution here is the design and rigorous analysis of the accuracy of this estimator. The storage complexity of the algorithm is also reasonably low: 1~2 bits per packet¹. We will show with a similar amount of storage complexity our scheme achieves better accuracy than the sampling-based schemes such as the work in [57]. This storage complexity can be further reduced through sampling. For maximizing the estimation accuracy when sampling is used, we also propose a technique for sampling packets consistently (instead of randomly independently) at multiple nodes and develop a theory that determines the optimal sampling rate under hard storage resource constraints.

As mentioned above, for estimating any matrix element $TM_{i,j}$, only the bitmaps from nodes i and j are needed. This allows us to estimate a submatrix using the minimum amount of information possible, namely, only the bitmaps from the rows and columns of the submatrix. This feature of our scheme is practically important in two aspects. First, in a large ISP network, most applications are often interested in only a portion of elements in the traffic matrix instead of the whole traffic matrix. Second, this feature allows for the incremental deployment of our scheme since the existence of non-participating nodes does not affect the estimation of the traffic submatrix between all participating ingress and egress nodes.

Our second streaming algorithm, namely *counter-array* scheme, is proposed for estimating the aforementioned flow matrix. With an array of counters as its data structure, its operation is also very simple. For each packet arrival, the counter indexed by the hash value of its flow label is incremented by 1. We show that the medium and large elements of the flow matrix can be inferred fairly accurately through correlating the counter arrays at all ingress and egress nodes. This algorithm can also be used for estimating the traffic matrix (weaker than flow matrix), though it is less cost

¹Depending on the application that uses traffic matrix information, the bitmaps typically do not need to be stored for more than a day.

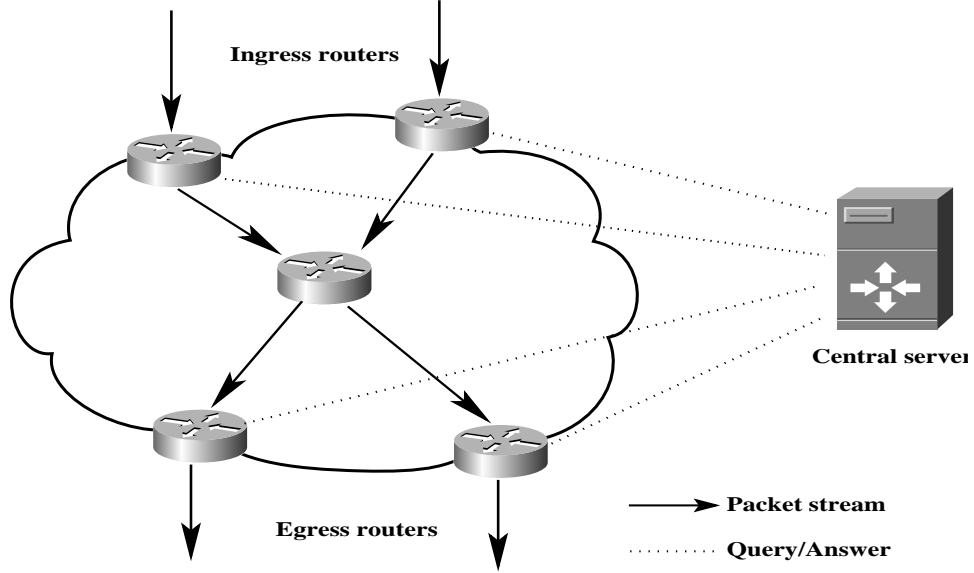


Figure 12: System model

effective than the bitmap scheme for this purpose.

The rest of this chapter is organized as follows: we start in Section 4.2 with a system overview and performance requirements we need to achieve. Sections 4.3, 4.4, and 4.5 describe our main ideas and provide a theoretical analysis. This is followed by synthetic experimental evaluation on real-world Internet traffic and actual traffic matrices in Section 4.6. Finally, we conclude this work in Section 4.7.

4.2 System Overview and Performance Requirements

The common system architecture of both the bitmap scheme and the counter array scheme is shown in Figure 12. The participating nodes will run an online streaming module that produces digests that are thousands of times smaller than the raw traffic. These digests will be stored locally for a period of time, and will be shipped to a central server on demand. A data analysis module running at the central server obtains the digests needed for estimating the traffic matrix and flow matrix through queries. Sophisticated decoding algorithms are used to obtain a very accurate estimation from these digests.

Our work is the first effort that uses data streaming algorithms for traffic (and flow) matrix estimation. The key challenge in this effort is to satisfy the following four seemingly conflicting

performance requirements simultaneously:

1. Low storage complexity. The amount of data digests generated on a single monitored node (per second) needs to be as small as possible since each node may be required to store such a digest for future inquiry for a certain period of time. Our (baseline) algorithms achieve more than three orders of magnitude reduction on the raw traffic volume. In the bitmap scheme, more reduction can be achieved using sampling (to be described in Section 4.4) and the inaccuracies of sampling is minimized through our consistent sampling method and parameter optimization theory.

2. Low memory (SRAM) complexity. As in other data streaming algorithms, our estimation accuracy becomes higher when the available memory (SRAM) becomes larger. However, we would also like to minimize this memory size since SRAM is an expensive and scarce resource on routers. We will show that both of our schemes provide very high accuracy using a reasonable amount of SRAM (e.g., 512KB).

3. Low computational complexity. The online streaming module should be fast enough to keep up with high link speeds such as 10 Gbps (OC-192) or even 40 Gbps (OC-768). We will show that our schemes are designed to address this challenge: we use efficient hardware implementation of hash functions and for each packet the bitmap scheme needs only one memory write and counter-array scheme needs only one memory read and write (to the same location). When coupled with sampling in the bitmap scheme, even higher rates can be supported.

4. High accuracy of estimation. Despite the fact that the digests produced at each node are orders of magnitude smaller than the original traffic stream, we would like our estimation of the traffic matrix in a measurement interval to be as close to the actual values as possible. We propose sophisticated “decoding” algorithms that produce estimates much more accurate than existing approaches, using these small digests.

4.3 The Bitmap Based Algorithm

We first present the online streaming module and the data analysis module of our bitmap based scheme. Then, we address the issues of clock synchronization, measurement epoch alignment, and heterogeneity in router speeds that arise in the practical operation of our scheme.

Algorithm 1: Algorithm for updating the online streaming module

```
1 Initialize  
2    $B[k] := 0, k = 1, 2, \dots, b$   
3 Update  
4   Upon the arrival of a packet  $pkt$   
5      $ind := h(\phi(pkt));$   
6      $B[ind] := 1;$ 
```

4.3.1 Online streaming module

The operations of the online streaming module are shown in Algorithm 1. The data structure is very simple: a bitmap B indexed by a hash function h . When a packet p arrives, we extract the invariant portion of the packet (denoted as $\phi(p)$ and described later) and hash it using h (we will discuss the choice of the hash function later). The result of this hashing operation is an integer which is viewed as an index into B and the bit at the corresponding index is set to 1. The bitmap is set to all 0's initially and will be paged to disk when it is filled to a threshold percentage (discussed in Section 4.4). We define this time interval as a “bitmap epoch”. This algorithm runs at all participating ingress and egress nodes, using the same hash function h and the same bitmap size b .

The invariant portion of a packet used as the input to the hash function must uniquely represent the packet and by definition should remain the same when it travels from one router to another. At the same time, it is desirable to make its size reasonably small to allow for fast hash processing. In our scheme, the invariant portion of a packet consists of the packet header, where the variant fields (e.g., TTL and checksum) are marked as 0's, and the first 8 bytes of the payload if there is any. As shown in [115], these 28 bytes are sufficient to differentiate almost all non-identical packets.

For this application we need a uniform hash function that is amenable to hardware implementation² and can be computed very fast. The H_3 family of hash functions proposed by Carter and Wegman [21] satisfies this requirement. It can produce hash result in a few nanoseconds with straightforward hardware implementation [103]. The design of the H_3 family of hash functions is described in Appendix A.2.4.

This online streaming module is extremely low in both computational and storage complexities:

²We do not use cryptographically strong hash functions such as MD5 or SHA, which are much more expensive to compute, since their security properties such as collision-resistance are not needed in this application.

Computational complexity. Each update only requires one hash function computation and one write to the memory. Using hardwired H_3 hash functions and $10ns$ SRAM, this would allow around 50 million packets per second, thereby supporting 50 Gbps traffic stream³. To support OC-192 speed (10 Gbps), we only need to use DRAM, since each packet has $100ns$ time budget.

Storage complexity. For each packet our scheme only produces a little more than one bit as its digest, which is three orders of magnitude reduction compared to the original traffic stream. For an OC-192 link, about 1~2 MB of digests will be generated and stored every second. An hour's worth of digests are about 100 MB. This can be further reduced using sampling, at the cost of reduced accuracy (to be discussed in Section 4.4).

4.3.2 Data analysis module

When we would like to know $TM_{i,j}$ during a certain time interval, the bitmaps corresponding to the bitmap epochs contained or partly contained in that interval (Figure 13) will be requested from nodes i and j and shipped to the central server. Next, we present and analyze an estimator of $TM_{i,j}$ given these bitmaps. For simplicity of discussion, we assume an ideal case where both node i and node j produce exactly one bitmap during that time interval (a measurement interval) and defer other details of the general case to Section 4.3.3.

Our estimator is adapted from [125], which was proposed for a totally different application (in database). In addition, we also provide a rigorous analysis of its standard deviation/error (not studied in [125] and is very involved). This analysis not only quantifies the accuracy of the estimator, an important result by itself, but also is an important step in identifying the optimal sampling rate when the online streaming algorithm operates under hard resource (storage) constraints.

Let the set of packets arriving at the ingress node i during the measurement interval be T_i and the resulting bitmap be B_{T_i} . Let U_{T_i} denote the number of bits in B_{T_i} that are 0's. Recall that the size of the bitmap is b . A good estimator⁴ of $|T_i|$, the number of elements (packets) in T_i , adapted from [125], is

³We assume a conservative average packet size of 1,000 bits, to our disadvantage. Measurements from real-world Internet traffic report much larger packet sizes.

⁴Note that, although D_{T_i} (as well as D_{T_j} and $D_{T_i \cup T_j}$) is an estimator, we do not put "hat" on top of it since it is just a component of the main estimator we are interested in.

$$D_{T_i} = b \ln \frac{b}{U_{T_i}} \quad (18)$$

$TM_{i,j}$, the quantity we would like to estimate, is simply $|T_i \cap T_j|$. An estimator for this quantity, also adapted from [125], is

$$\widehat{TM}_{i,j} = D_{T_i} + D_{T_j} - D_{T_i \cup T_j} \quad (19)$$

Here $D_{T_i \cup T_j}$ is defined as $b \ln \frac{b}{U_{T_i \cup T_j}}$, where $U_{T_i \cup T_j}$ denotes the number of 0's in $B_{T_i \cup T_j}$ (the result of hashing the set of packets $T_i \cup T_j$ into a single bitmap). The bitmap $B_{T_i \cup T_j}$ is computed as the bitwise-OR ⁵ of B_{T_i} and B_{T_j} . It can be shown that $D_{T_i} + D_{T_j} - D_{T_i \cup T_j}$ is a good estimator of $|T_i| + |T_j| - |T_i \cup T_j|$, which is exactly $|T_i \cap T_j|$. ⁶

The computational complexity of estimating each element of the matrix is $O(b)$ for the bitwise operation of the two bitmaps. The overall complexity of estimating the entire $m \times n$ matrix is therefore $O(mnb)$. Note that the bitmaps from other nodes are not needed when we are only interested in estimating $TM_{i,j}$. This poses significant advantage in computational complexity over existing indirect measurement approaches, in which the whole traffic matrix needs to be estimated even if we are only interested in a small subset of the matrix elements due to the holistic nature of the inference method. This feature also makes our scheme incrementally deployable, as mentioned earlier.

While this estimator has been briefly mentioned in [125], there is no rigorous analysis of its standard deviation and error, which we perform in this work. These are characterized in the following theorem. Let t_{T_i} , t_{T_j} , $t_{T_i \cap T_j}$, and $t_{T_i \cup T_j}$ denote $\frac{|T_i|}{b}$, $\frac{|T_j|}{b}$, $\frac{|T_i \cap T_j|}{b}$, and $\frac{|T_i \cup T_j|}{b}$, respectively. They are the “load factors” of the array when the corresponding set of packets are hashed into the array (of size b). Its proof is provided in Appendix A.2.2.

Theorem 1 The variance of $\widehat{TM}_{i,j}$ is given by

$$\text{Var}[\widehat{TM}_{i,j}] = b(2e^{t_{T_i \cap T_j}} + e^{t_{T_i \cup T_j}} - e^{t_{T_i}} - e^{t_{T_j}} - t_{T_i \cap T_j} - 1).$$

⁵One can easily verify the correctness of the computation with respect to the semantics of $B_{T_i \cup T_j}$.

⁶Directly using Equation 18 based on the bitmap produced by bitwise-ANDing B_{T_i} and B_{T_j} to get an estimator is problematic due to random hashing.

The average (relative) error of the estimator $\widehat{TM_{i,j}}$, which is equal to the standard deviation of the ratio $\frac{\widehat{TM_{i,j}}}{TM_{i,j}}$ since this estimator is almost unbiased (discussed in Appendix A.2.1), is given by

$$\frac{\sqrt{2e^{t_{T_i \cap T_j}} + e^{t_{T_i \cup T_j}} - e^{t_{T_i}} - e^{t_{T_j}} - t_{T_i \cap T_j} - 1}}{\sqrt{b}t_{T_i \cap T_j}}. \quad (20)$$

Equation 20 characterizes the tradeoff between memory complexity and estimation accuracy for the bitmap scheme as follows. The average error is scaled by the inverse of \sqrt{b} , which means the larger the page size the more accurate the result we get. Our experiments in Section 4.6 show that very high estimation accuracy can be achieved using a reasonable amount of SRAM (e.g., 512 KB).

4.3.3 Extension for operational issues

The above estimation procedure and its accuracy analysis are for the ideal case with the following three assumptions:

(i) *The measurement interval is exactly one bitmap epoch.* Practically some network management tasks such as capacity planning and routing configuration need the traffic matrices on the long time scales such as tens of minutes or a few hours. Each epoch in our measurements is typically much smaller especially for the high speed links. Therefore we need extend our scheme to support any time scales.

(ii) *The clocks on nodes i and j are perfectly synchronized.* Using GPS synchronization [105], or more cost-effective schemes [96], clocks at different nodes only differ by tens of microseconds. Since each bitmap epoch is typically one to several seconds with OC-192 or OC-768 speeds (even longer for lower link speeds), the effect of clock skew on our measurement is negligible.⁷ Due to the high price of GPS cards today, the standard network time protocol (NTP) is most commonly used to synchronize clocks. As we will shown later in this section, our measurements still work accurately with relative large clock skews (e.g., tens of milliseconds as one may get from clocks synchronized by using NTP).

(iii) *The bitmap epochs between nodes i and j are well aligned.* Traffic going through different nodes can have rates orders of magnitude different from each other, resulting in some bitmaps being filled up very fast (hence short bitmap epoch) and some others filled up very slowly (hence long

⁷For the same reason the impact of clock resolution on our measurements is also negligible.

bitmap epoch). We refer to this phenomena as *heterogeneity*. Because of heterogeneity the bitmap epochs on different nodes may not well aligned.

Next, we solve for the general case in which these assumptions are eliminated. We assume that the measurement interval spans exactly bitmap epochs $1, 2, \dots, k_1$ at node i and bitmap epochs $1, 2, \dots, k_2$ at node j , respectively. Then the traffic matrix element $TM_{i,j}$ can be estimated as

$$\widehat{TM}_{i,j} = \sum_{q=1}^{k_1} \sum_{r=1}^{k_2} \widehat{N}_{q,r} \times \text{overlap}(q, r) \quad (21)$$

where $\widehat{N}_{q,r}$ is the estimation of the common traffic between the page⁸ q at node i and the page r at node j , and $\text{overlap}(q, r)$ is 1, when the page q at node i overlaps temporally with page r at node j , and is 0 otherwise. To determine whether two bitmap epochs overlap with each other temporally, the timestamps of their starting times will be stored along with the pages. We name the above method “multipaging”. We show that the multipaging actually completely eliminates the aforementioned three assumptions.

Clearly the assumption (i) is eliminated because the multipaging supports the measurements over multiple epochs. Now we further adapt the multipaging to the case that the measurement interval does not necessarily align with the epoch starting times (assumption (iii)). We illustrate this using an example shown in Figure 13. A measurement interval corresponds to the rear part of epoch 1, epochs 2 and 3, and the front part of epoch 4 at node i . It also corresponds to the rear part of epoch 1, epoch 2, and the front part of epoch 3 at node j . By Equation 21, we need to add up the terms $\widehat{N}_{1,1}, \widehat{N}_{2,1}, \widehat{N}_{2,2}, \widehat{N}_{3,2}, \widehat{N}_{3,3}$, and $\widehat{N}_{4,3}$ based on their temporal overlap relationships. However, this would be more than $TM_{i,j}$ because the measurement interval only has the rear part of epoch 1 and the front part of epoch 4 at node i . Our solution is to adjust $\widehat{N}_{1,1}$ to the proportion of the epoch 1 that overlaps with the measurement interval. $\widehat{N}_{4,3}$ will also be adjusted proportionally accordingly. Since traffic matrix estimation is typically on the time scales of tens of minutes, which will span many pages, the inaccuracies resulting from this proportional rounding are negligible.

The assumption (ii) can be eliminated by combing the clock skew factor into definition of “temporal overlapping” in Equation 21. We still use an example shown in Figure 13. The epoch 1 at

⁸The notions of “page” and “bitmap” are exchangeable in the rest of this section.

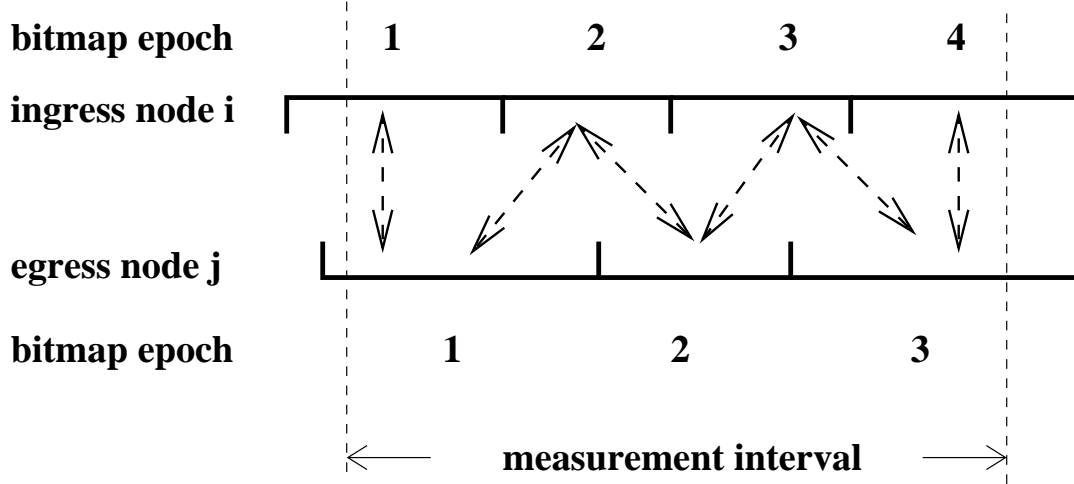


Figure 13: Example of timeline

node i does not overlap temporally with the epoch 2 at node j visually. But if the interval between the end of the epoch 1 at node i and the start of the epoch 2 at node j is smaller than an upper bound T_1 of the clock skew (e.g., $50ms$ for an NTP enabled network), we still consider they are temporally overlapping.

By now there is still a remaining problem for the extension of Equation 21: the packets in transit. Let us come back to the example in Figure 13. If there are packets departing from i in epoch 1 (at node i) and arriving at j in epoch 2 (at node j) due to nontrivial traversal time from i to j , our measurement will miss these packets because only $\widehat{N}_{1,1}$ is computed. This can be easily fixed using the same method used above to eliminate the assumption (ii), i.e., combining another upper bound T_2 of the traversal time (e.g., $50ms$) to define “temporal overlapping”. In other words if the interval between the end of epoch 1 at node i and the start of epoch 2 at node j is within $T_1 + T_2$, it should be labeled “temporal overlapping” ($overlap(1, 2) = 1$) and join the estimation.

4.4 Sampling

Sometimes the bitmaps need to be stored for a long period of time for later troubleshooting. This could result in huge storage complexity for very high speed links. Sampling can be used to reduce this requirement significantly. Also, if we would like to use DRAM to conduct online streaming for very high speed links (e.g., beyond OC-192), it is important to sample only a certain percentage of

the packets so that the DRAM speed can keep up with the data stream speed. However, we need to bear in mind that sampling comes at the cost of reduced accuracy. In this section, we rigorously analyze the impact of sampling on accuracy, and address two challenging problems that arise in minimizing this impact: *sampling versus squeezing* and *consistent sampling*.

4.4.1 Sampling versus squeezing

Suppose there is a hard resource constraint on how much storage the online streaming algorithm can consume every second. For example, the constraint can be one bitmap of 4 Mbits per second. Suppose we have 40 million packets arriving within one second. One option is that we do no sampling and hash all these packets into the bitmap, referred to as “squeezing”. But the resulting high load factor of approximately 10 would lead to high estimation error according to Equation 20. An alternative option is to sample only a certain percentage p of packets to be squeezed into the bitmap. We have many different p values to choose from. For example, we can sample 50% of the packets and thereby squeeze 20 million sampled packets into the bitmap, or we can sample and squeeze only 25% of them. This comes to the question which p is optimal. On the one extreme, if we sample at a very low rate, the bitmap will only be lightly loaded and the error of estimating the total *sampled* traffic as well as its common traffic with another node (a traffic matrix element) becomes lower. However, since the sampled traffic is only a small percentage of the total traffic, the overall error will be blown up by a large factor (discussed in Section 4.4.2). On the other extreme, if we sample with very high probability, the error from sampling becomes low but the error from estimating the sampled traffic becomes high. We establish the following principle for conducting sampling that aims at reaching a “sweet spot” between these two extremes.

Principle 1 *If the expected traffic demand in a bitmap epoch does not make the resulting load factor exceed t^* , no sampling is needed. Otherwise, sampling rate p^* should be set so that the load factor of the sampled traffic on the bitmap is approximately t^* .*

We consider the following scenario throughout the rest of this section. We assume that each ingress and egress node will coordinate to use the same load factor t after sampling. This coordination will allow us to optimize t for estimating most of the traffic matrix elements accurately. We show how to minimize the error of an arbitrary $\widehat{N_{q,r}}$ term shown in Equation 21. Recall that $N_{q,r}$

is the amount of common traffic between two overlapping bitmap pages, page q at node i and page r at node j . We simply denote it and its estimator as X and \hat{X} , respectively. Also, let α and p_α be the aforementioned page q at node i and the corresponding sampling rate, respectively. Similarly, β and p_β denote page r at node j and the corresponding sampling rate, respectively. Note that each overlapping page pair may have its own optimal t^* to achieve the optimal accuracy of estimating its common traffic. Therefore it is impossible to adapt t^* to satisfy every other node, as their needs (t^* for optimal accuracy) conflict with each other. The goal of our following analysis is to identify a default t^* for every node such that the estimation accuracy for the most common cases is high.

We first study the optimal p^* and t^* between pages α and β given the expected traffic demand in a bitmap epoch. In fact, only one of them needs to be determined since the other follows from Principle 1. In Section 4.4.2 we will propose a sampling technique called *consistent sampling* to significantly reduce the estimation error. With consistent sampling, \hat{X} , the estimator of X , is given by $\frac{\hat{N}}{p}$, where $p = \min(p_\alpha, p_\beta)$ and \hat{N} is the estimation result (by Equation 19) on the sampled traffic. We denote the total sampled traffic volume squeezed into the page with sampling rate p by T . The following theorem characterizes the variance of \hat{X} . Its proof is provided in Appendix A.2.3.

Theorem 2 The variance of \hat{X} is approximately given by

$$\frac{b}{p^2} \left(\left(e^{\frac{Tp}{b} - \frac{Xp}{2b}} - e^{\frac{Xp}{2b}} \right)^2 + e^{\frac{Xp}{b}} - \frac{Xp}{b} - 1 \right) + \frac{X(1-p)}{p}.$$

Remark: The above formula consists of two terms. We show in Appendix A.2.3 that the first term corresponds to the variance from estimating the sampled traffic (Equation 19) scaled by $\frac{1}{p^2}$ (to compensate for the sampling), and the second term corresponds to the variance of the sampling process. Since these two errors are orthogonal to each other, their total variance is the sum of their individual variances as shown in Appendix A.2.3.

The average error of \hat{X} , which is equal to the standard deviation of the ratio $\frac{\hat{X}}{X}$ since \hat{X} is an almost unbiased estimator of X (discussed in Appendix A.2.1), is given by

$$\frac{\sqrt{\frac{b}{p^2} \left(\left(e^{\frac{Tp}{b} - \frac{Xp}{2b}} - e^{\frac{Xp}{2b}} \right)^2 + e^{\frac{Xp}{b}} - \frac{Xp}{b} - 1 \right) + \frac{X(1-p)}{p}}}{X}. \quad (22)$$

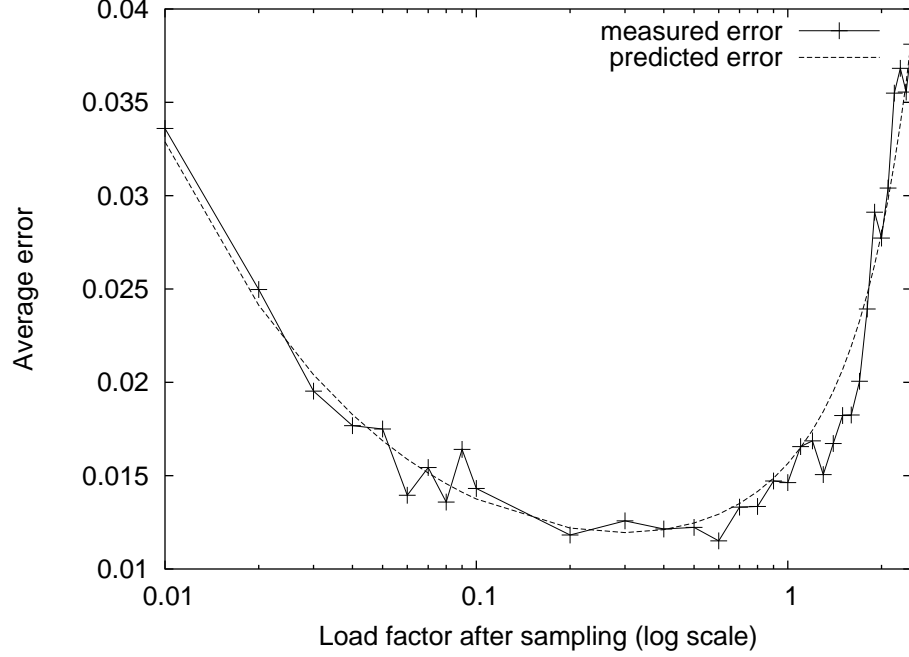


Figure 14: Measured average error from Monte-Carlo simulation vs. the value from Equation 22 ($X = 1\text{M}$ packets, $T = 10\text{M}$ packets, and $b = 1$ Mbits).

We perform Monte-Carlo simulations to verify the accuracy of the above formula since there is some approximation in its derivation (see Appendix A.2.3). The following parameter settings are used in the simulations. The size b of the bitmap is 1M bits and both the ingress page and the egress page have the same load factor 10 without sampling. In other words, 10M packets each is to be squeezed, or sampled and then squeezed, into the bitmaps. Among both sets of 10M packets, 1M packets are common between both the ingress page and the egress page (i.e., $X = 1\text{M}$ packets).

Figure 14 shows that the observed average error from Monte-Carlo simulation matches well with the value from Equation 22. The curve verifies our intuition above about the estimation error caused by sampling and squeezing. When the load factor is very low, the sampling error dominates; when the load factor becomes large the error caused by “excessive squeezing” is the main factor. At the optimal t^* (≈ 0.4) the scheme achieves the smallest average relative error (≈ 0.012).

Since the optimal t^* value is a function of T and X , setting it according to some global default value may not be optimal all the time. Fortunately we observe that t^* , the optimal load factor, does not vary much for different T and X values through our extensive experiments. In addition, we can

observe from Figure 14 that the curve is quite flat in a large range around the optimal load factor. For example, the average errors corresponding to any load factor between 0.09 and 1.0 only fluctuate between around 0.012 to 0.015. Combining the above two observations, we conclude that by setting a global default load factor t^* according to some typical parameter settings, the average error will stay very close to optimal values. Throughout this work we set the default load factor to 0.7.

4.4.2 Consistent sampling

If the sampling is performed randomly and independently, only $p_\alpha p_\beta$ of the common traffic that comes from page α to page β is recorded by both nodes on the average. To estimate X , although it is possible to get its unbiased estimate by blowing the estimation of the sampled portion up by $\frac{1}{p_\alpha p_\beta}$, it also blows the error of our estimate up by $\frac{1}{p_\alpha p_\beta}$. To address this problem, we propose a consistent sampling scheme which has the following desirable property. When $p_\alpha \leq p_\beta$, among the set of packets that come from page α to page β , we ensure that those sampled and squeezed into page α are a subset of those sampled and squeezed into page β , and vice versa. In this way, $\min(p_\alpha, p_\beta)$ of traffic between page α and page β will be sampled by both nodes and the error of our estimation will only be blown up by $\frac{1}{\min(p_\alpha, p_\beta)}$ times.

Our consistent sampling scheme works as follows. We fix a hash function h' (different from the aforementioned h which is used to generate the bitmap), that maps the invariant portion of a packet to an l -bit binary number. The range of the hash function h' is $\{0, 1, \dots, 2^l - 1\}$. If a node would like to sample packets with rate $p = \frac{c}{2^l}$, it simply samples the set of packets $\{pkt | h'(\phi(pkt)) < c\}$. We make l sufficiently large such that any desirable sampling rate p can be approximated by $\frac{c}{2^l}$ for some c between 1 and 2^l . When every node uses the same hash function h' , the above property is clearly achieved as follows. When $p_\alpha = \frac{c_1}{2^l} \leq \frac{c_2}{2^l} = p_\beta$, the set of packets sampled and squeezed into page α , $\{pkt | h'(\phi(pkt)) < c_1\}$, is clearly a subset of those sampled and squeezed into page β , $\{pkt | h'(\phi(pkt)) < c_2\}$.

4.5 The Counter-array Based Scheme

In this section, we formally introduce the concept of flow matrix, which contains finer grained information than traffic matrix, and present our counter-array based scheme for estimating flow

Algorithm 2: Algorithm for updating the online streaming module

```
1 Initialize  
2    $C[k] := 0, k = 1, 2, \dots, b$   
3 Update  
4   Upon the arrival of a packet  $pkt$   
5      $ind := h(pkt.flow\_label);$   
6      $C[ind] := C[ind] + 1;$ 
```

matrix. *Flow matrix* is defined as the traffic matrix combined with the information on how each OD element is splitted into flows of different sizes. Formally, a flow matrix element $FM_{i,j}$ is the set of sizes of flows that travel from node i to node j during a measurement interval. A traffic matrix element $TM_{i,j}$ is simply the summation of all the flow sizes in $FM_{i,j}$, that is, $TM_{i,j} = \sum_{s \in FM_{i,j}} s$. Thus our counter-array scheme also works for estimating traffic matrices.

4.5.1 Online streaming module

The online streaming algorithm (shown in Algorithm 2) uses a very simple data structure: an array of counters C . Upon arrival of a packet, its flow label⁹ is hashed to generate an index into this array, and the counter at this index is incremented by 1. Similar to the bitmap scheme, all nodes employ an array of the same size b and the same hash function h . Since for each packet, this algorithm requires only one hash operation, one memory read and one memory write (to the same location), this allows our scheme to operate at OC-768 (40 Gbps) speed with off-the-shelf 10ns SRAM and efficient hardware implementation of the H_3 family of hash functions.

Due to the delicate nature of the data analysis algorithm (discussed Section 4.5.2) for the counter array scheme, much more stringent yet still reasonable constraints are placed on the online streaming module. First, unlike in the bitmap scheme, the counter array scheme is holistic in the sense that all ingress and egress nodes have to participate. Second, unlike in the bitmap scheme, the *counter epochs* (defined next) in this scheme need to be aligned with each other, that is, all counter epochs in all ingress and egress nodes need to start and end at the approximately same time. The practical implication behind this is the counter array b needs to be large enough to accommodate the highest link speed among all nodes (i.e., the worst case). Similar to the definition of “bitmap epoch”, we

⁹Our design does not place any constraints on the definition of flow label. It can be any combination of fields from the packet header.

refer to the amount of time the highest-speed link takes to fill up the counter array to a threshold percentage as a “counter epoch”, or epoch for abbreviation.

Next, we analyze the memory (SRAM) and storage complexities of the online streaming module.

Memory complexity. Depending on the maximum link speed among all the nodes, the counter epoch ranges from 1 to a few 10’s of seconds. We show in Section 4.6 that, very accurate estimation can be achieved by setting the number of counters in the array to around the same order as the number of flows during an epoch. Therefore, for OC-192 or OC-768 link, one to a few million of counters need to be employed for a measurement interval of one to a few seconds. If each counter has a “safe” size of 64 bits to prevent the overflow¹⁰, the memory requirement would be quite high. Fortunately, leveraging our technique described in Chapter 7, this requirement is reduced to 4 bits per counter, an 94% reduction. For example, a million counters will only cost 1.1 MB SRAM. The key idea of this technique is to keep short counters in SRAM and long counters in DRAM. When a short counter in SRAM exceeds a certain threshold value due to increments, the value of this counter will be “flushed” to the corresponding long counter in DRAM.

Storage complexity. Perhaps surprisingly, at the same link speed, the storage complexity is even smaller than the bitmap scheme. In the bitmap scheme, each packet results in 1 to 2 bits of storage. Here, each flow results in 64 bits of storage. However, since most of the counter values are small, resulting in a lot of repetitions in small counter values, Huffman type of compression [66] can easily reduce the storage complexity to only a few bits per counter. Since the average flow length is about 10 packets (observed in our experiments), the average storage cost per packet is amortized to less than 1 bit.

4.5.2 Data analysis module

Once there is a need to estimate the flow matrix during a measurement interval, the counter arrays during that interval need to be shipped to the central server for analysis. If the measurement interval spans more than one epochs, digests in each epoch will be processed independently. In this section,

¹⁰Due to the “Zipfian” nature of the Internet traffic, some “elephant” flows may account for the majority of Internet traffic during an epoch.

we present our data analysis algorithm that infers the flow matrix from these counter arrays during a single epoch.

We first describe the intuition behind our algorithm. Let $I^k = \{C_{I_1}[k], C_{I_2}[k], \dots, C_{I_m}[k]\}$ where $C_{I_i}[k], i = 1, \dots, m$, is the value of the k^{th} counter at the ingress node I_i , and $E^k = \{C_{E_1}[k], C_{E_2}[k], \dots, C_{E_n}[k]\}$ where $C_{E_j}[k], j = 1, \dots, n$, is the value of the k^{th} counter at the egress node E_j . Since every node uses the same hash function, packets recorded in I^k have an approximate one-to-one correspondence with packets recorded in E^k .

$$\sum_{i=1}^m C_{I_i}[k] \approx \sum_{j=1}^n C_{E_j}[k] \quad (23)$$

This approximation comes from the fact that the clock is not perfectly synchronized at all nodes and the packet traversal time from an ingress node to an egress node is non-zero. Similar to the discussion in Section 4.3.3, both factors only have marginal impact on the accuracy of our estimation. In addition, the impact of this approximation on the accuracy of our algorithm is further alleviated due to the “elephant matching” nature of our algorithm (discussed later).

Now, let us consider an ideal case in which there is no hash collision on index k in all counter arrays, and therefore the counter value represents a flow of this size. We further assume that: (i) the number of ingress nodes is the same as the egress nodes (i.e., $m = n$); (ii) the elements in I^k are all distinct; and (iii) the flows in I^k all go to distinct elements of E^k . Then the values in E^k are simply a permutation of the values in I^k . A straightforward “one-to-one matching” will allow us to infer this permutation.

In reality, none of the above assumptions is true. At an ingress node, multiple flows can collide into one counter. Flows from multiple ingress nodes can collide into the same counter at an egress node, and m is in general not equal to n . Therefore, a “naive” matching algorithm like above will not work in general. However, since there are only a small number of medium to large flows due to the Zipfian nature of the Internet traffic, matching kangaroos (medium flows) and elephants (large flows) between ingress and egress nodes turns out to work very well. As expected, it does not work well on small flows.

Algorithm 3: Generate matching of counter values at index k

```

1   $FM_{i,j}^k := 0, i = 1, \dots, m, j = 1, \dots, n;$ 
2  do
3     $max\_i := \underset{i}{argmax}\{C_{I_i}[k], i = 1, 2, \dots, m\};$ 
4     $max\_j := \underset{j}{argmax}\{C_{E_j}[k], j = 1, 2, \dots, n\};$ 
5    if  $C_{I_{max\_i}}[k] \geq C_{E_{max\_j}}[k]$ 
6       $FM_{max\_i,max\_j}^k := FM_{max\_i,max\_j}^k + C_{E_{max\_j}}[k];$ 
7       $C_{I_{max\_i}}[k] := C_{I_{max\_i}}[k] - C_{E_{max\_j}}[k];$ 
8       $C_{E_{max\_j}}[k] := 0;$ 
9    else
10      $FM_{max\_i,max\_j}^k := FM_{max\_i,max\_j}^k + C_{I_{max\_i}}[k];$ 
11      $C_{E_{max\_j}}[k] := C_{E_{max\_j}}[k] - C_{I_{max\_i}}[k];$ 
12      $C_{I_{max\_i}}[k] := 0;$ 
13  while  $((C_{I_{max\_i}}[k] > 0) \& \& (C_{E_{max\_j}}[k] > 0))$ 

```

Algorithm 3 shows a greedy algorithm for matching elephants and kangaroos. For each index k in all the counter arrays we perform the following iteration (lines 2 to 13). We first match the largest ingress counter value $C_{I_{max_i}}[k]$ with the largest egress counter value $C_{E_{max_j}}[k]$. The smaller value of the two is considered a flow from max_i to max_j (lines 6 and 10), and this value will be subtracted from both counter values (lines 7 and 11). This clearly reduces the smaller counter value to 0. The above procedure is repeated until either all ingress counters or all egress counters at index k become 0's¹¹. When there is a tie on the maximum ingress or egress counter values, a random tie-breaking is performed. We will show that such a simple algorithm produces surprisingly accurate estimation of flow matrix for medium to large flows.

The computation complexity of the algorithm in Algorithm 3 is $O((m + n - 1)(\log m + \log n))$ because the binary searching operation (lines 3 and 4) dominates the complexity of each iteration and there are at most $m + n - 1$ iterations. Thus the overall complexity to estimate the flow matrix is $O(b(m + n - 1)(\log m + \log n))$.

Recall that an exact flow matrix can be used to indicate intrusions such as DDoS attacks in Section 4.1. Unfortunately our estimation algorithm only offers accurate estimation on the medium and large flow matrix elements, as shown in Section 4.6; some typical intrusions (e.g., DDoS attacks) consist of a large number of *small* flows. To make our algorithm still be able to provide valuable

¹¹They may not become all 0's simultaneously due to the approximation mentioned above.

information of intrusions we can adapt the flow definition in our scheme correspondingly. For example, to detect DDoS attacks, we can use the destination IP address of a packet as its flow label. Then the traffic of a DDoS attack becomes a large flow going through the network instead of a large number of small ones.

Sometimes besides obtaining a flow matrix during a time interval we need to know the identity (flow labels) of these flows, which is not provided by our scheme and could be useful in some applications. One possible method is to generate these flow labels using other sampling or streaming algorithms [50]. Since this is a challenging separate effort which is orthogonal to the problem we are solving, we do not explore it further.

Finally, the traffic matrix can also be obtained by adding up the sizes of all the flows that we determine going from node i to node j using the above algorithm. This is in fact a fairly accurate estimation of traffic matrix, as shown in Section 4.6, since our algorithm tracks kangaroos and elephants very accurately, and they account for the majority of traffic. However, for the purpose of estimating traffic matrix alone, the bitmap scheme provides much better accuracy and is clearly a better choice, also shown in Section 4.6.

4.6 Evaluation

An ideal evaluation of our traffic matrix and flow matrix estimation mechanisms would require packet-level traces collected simultaneously at hundreds of ingress and egress routers in an ISP network for a certain period of time. However, it is very expensive if not impossible to collect, ship, and store raw packet-level traces at a large number of high-speed links (OC-192). Instead, we use two data sets in our evaluation: synthetic traffic matrices generated from publicly available packet-level traces from NLANR [89] and actual traffic matrices from a tier-1 ISP. The ISP traffic matrix was produced in [130] and corresponds to traffic during a one-hour interval in a tier-1 network.

4.6.1 NLANR trace-driven experiments

We adopt two performance metrics: Root Mean Squared Error (RMSE) and Root Mean Squared Relative Error (RMSRE), which were proposed and used in [130] for the same purpose of evaluating the accuracy of estimated traffic matrix.

$$\begin{aligned}
RMSE &= \sqrt{\frac{1}{N} \sum_{i=1}^N (\hat{x}_i - x_i)^2} \\
RMSRE &= \sqrt{\frac{1}{N_T} \sum_{\substack{i=1 \\ x_i > T}}^N \left(\frac{\hat{x}_i - x_i}{x_i}\right)^2}
\end{aligned}$$

The RMSE provides an overall measure of the absolute errors in the estimates, while RMSRE provides a relative measure. Note that the relative errors for small matrix elements are usually not very important for network engineering. We take only matrix elements greater than some threshold T in the computation of RMSRE (properly normalized). In the above equation, N_T refers to the number of matrix elements greater than T , i.e., $N_T = |\{x_i | x_i > T, i = 1, 2, \dots, N\}|$.

The set of traces we used consists of 16 publicly available packet header traces from NLANR. The number of flows in these traces varies from 170K to 320K and the number of packets varies from 1.8M to 3.5M. We piece together these traces to construct a synthetic scenario that *appears* as if these traces were collected simultaneously at all ingress nodes of a network. We set up the experimental scenario as follows: there are 16 ingress nodes and 16 egress nodes in the measurement domain. Each trace corresponds to the packet stream for one ingress node. The challenge in constructing this scenario lies in assigning the flows in the input stream at an ingress node to 16 different egress nodes such that the generated matrix will reflect some properties of real traffic matrices.

Recent work [16] shows that the Internet has “hot spot” behavior. A few OD pairs have very large traffic volume, while the majority of OD pairs have substantially less volume between them. Following the observed quantitative properties of real Internet traffic matrices [16], for each ingress node, we randomly divide the 16 ingress nodes into three categories: 2 nodes belonging to *large* category, 7 nodes belonging to *medium* category, and the rest 7 nodes belonging to *small* category. For each flow at an ingress node, we assign an egress node randomly, with nodes belonging to the large category twice as likely to be picked as medium category nodes which in turn are twice as likely to be picked as small category nodes. For simplicity, we configure the size of the bitmap and the counter array to fit the data set size without adopting the enhancement techniques (i.e., multipaging and sampling). Thus, we set the size of bitmap to 2,880K bits and the size of counter

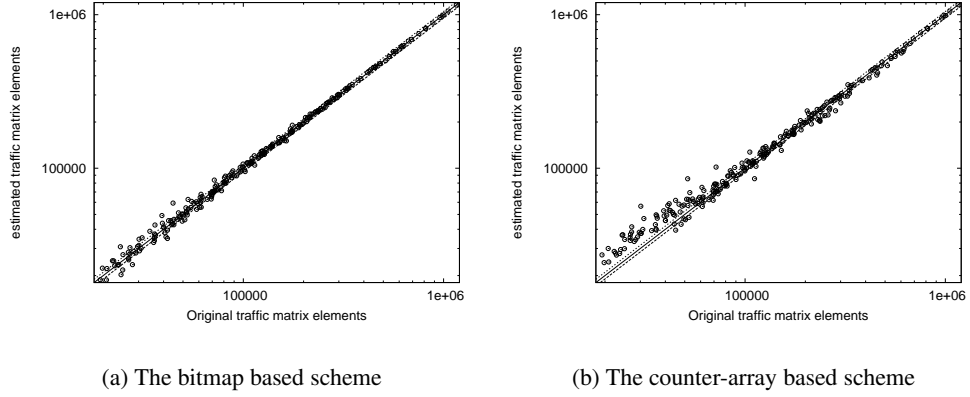


Figure 15: Original vs. estimated traffic matrix elements. Both axes are on logscale

array to 320K counters which approximately occupy 2,880K bits of fast memory (SRAM).

Figure 15 compares the estimated traffic matrix elements using the bitmap scheme and counter array scheme with the original traffic matrix elements. The solid diagonal lines denotes perfect estimation, while the dashed lines denote an estimation error of $\pm 5\%$. Clearly, the closer the points cluster around the diagonal, the more accurate the scheme is. We observe that both schemes are very accurate, and the bitmap scheme is more accurate than the counter array scheme.

Figure 16 shows the impact of varying T on RMSRE. We observe that both schemes produce very close estimates for the large and medium matrix elements. The traffic volume of the thresholded matrix elements decreases as the threshold increases, and the performance improves. For example, the RMSRE actually drops to below 0.05 for the top 70% of traffic for the counter array scheme. For the bitmap scheme, it drops even further to below 0.01 for the top 70% of traffic. In absolute terms, the RMSEs of the bitmap scheme and counter array scheme are equal to 4,136 packets and 11,918 packets, respectively, which are very small in comparison to the average traffic on a node. All of the above results confirm that the bitmap scheme achieves higher accuracy than the counter array scheme. The overall RMSRE of the bitmap scheme is below 6%, and that of the counter array scheme evolves from around 1% for large elements to 16% for the overall elements.

Note that our results reflect relative accuracy on a small time scale (one to several seconds for high speed routers), and they should not be directly compared with results reported in literature such as [117], which is on much larger time scales. Our schemes usually can achieve much higher

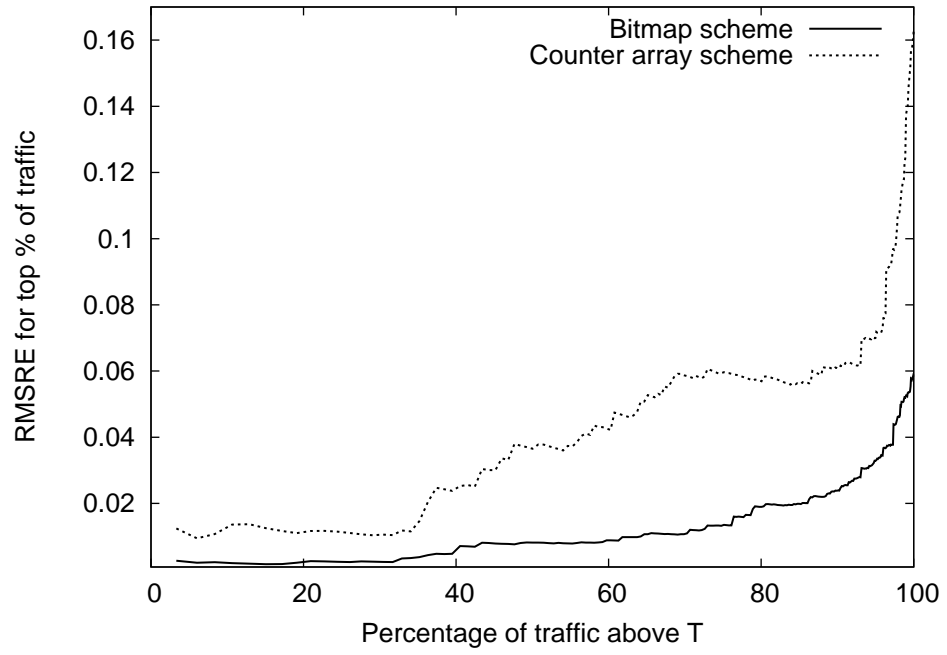


Figure 16: The RMSRE for various threshold T .

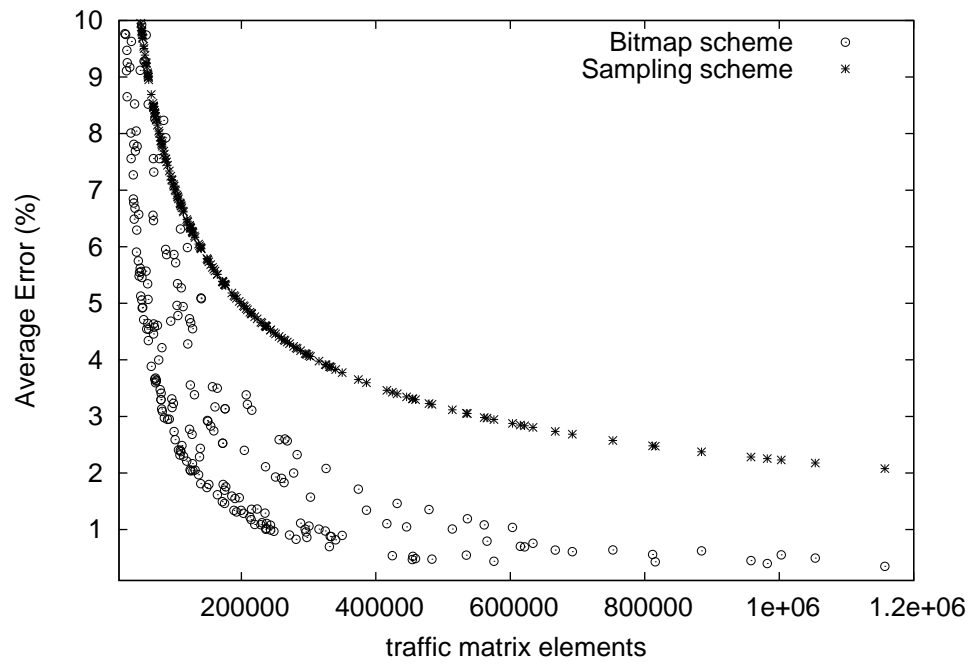


Figure 17: The average error of the bitmap scheme and the sampling based scheme.

relative accuracy on larger time scales (e.g., tens of minutes), as shown in Section 4.6.2.

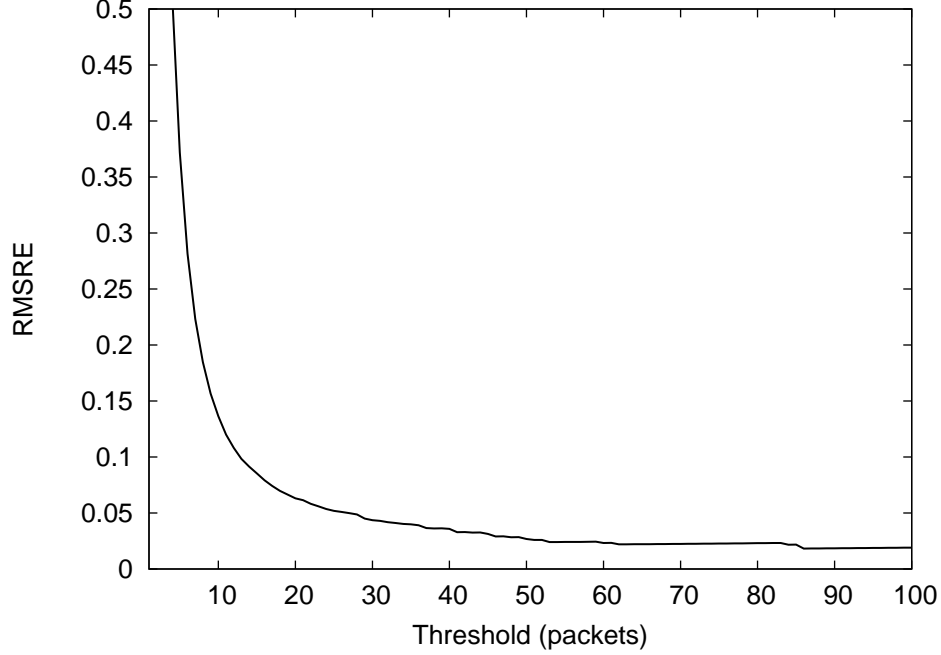


Figure 18: The RMSRE for various threshold T for flow matrix.

We also compare the average (relative) error between our bitmap scheme and the existing sampling based schemes such as NetFlow. We adopt the similar method in [57] to infer the traffic matrix by collecting the packets in each ingress node with the typical NetFlow sampling rate of $\frac{1}{500}$ (which generates a similar amount of data per second as our bitmap scheme) and inferring the traffic matrix according to the egress nodes we assigned above for each sampled flows. Here, the variance of $\widehat{TM}_{i,j}$ is given by $\frac{TM_{i,j}(1-p)}{p}$ where p is the sampling rate $\frac{1}{500}$. Figure 17 plots the average error of each element of the traffic matrix in the trace-driven experiments for both our bitmap scheme and the sampling based scheme. We observe that our bitmap scheme achieves a consistently higher accuracy than the sampling based scheme.

Next, we evaluate the accuracy of flow matrix estimation. Note that our flow matrix estimation is counter-based and cannot distinguish the flows which are hashed to the same location with the same ingress node and egress node (we call this an *indistinguishable collision*). Our goal is to accurately estimate the medium and large flow matrix elements. We observe that the indistinguishable collisions happen rarely for medium and large flows. In our experiments, among the total 4 million flows with average size about 10 packets there are only 41 out of 71,345 medium and large flows

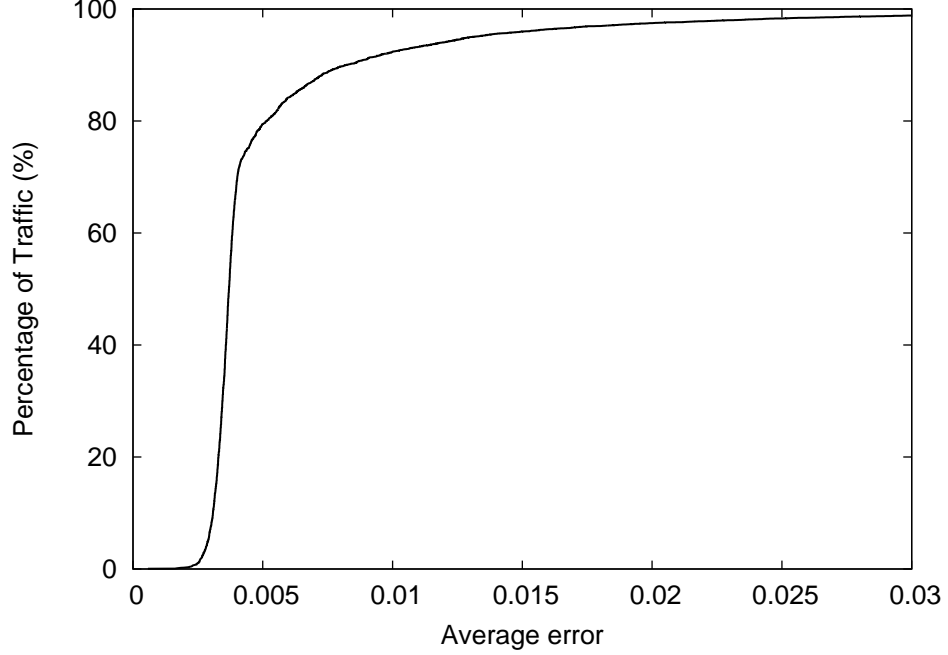


Figure 19: Cumulative distribution of traffic with certain average error.

(> 10 packets) which suffer from this collision. Thus the impact of such indistinguishable collisions on the estimation accuracy is negligible. Figure 18 shows RMSREs for various threshold T . We observe a sharp downward trend in the value of RMSRE for increasing threshold values. When the threshold is equal to 10 packets, the error drops to below 15%. The accurate estimation of these flows is very important since, in this trace, flows of size 10 and above (71,345 of them) accounts for 87% of the total traffic.

4.6.2 Experiments based on ISP traffic matrices

We use a one-hour router-level traffic matrix from a tier-1 ISP network obtained in [130] to analytically evaluate the accuracy of the bitmap scheme. We assume that traffic volume between each pair of backbone routers is evenly distributed over the one hour time period¹². Clearly an hour's traffic is too large (we assume a conservative average packet size of 200 bytes) to fit in a single bitmap, and therefore the aforementioned multipaging technique is used. Given a traffic matrix, we split the traffic on each ingress/egress node into multiple pages of 4Mbits (i.e., 512KB) with load factor 0.7

¹²Our estimation accuracy will not be impacted significantly by the specific distribution of traffic during this one-hour interval, when multipaging is used.

(the default load factor described in Section 4.4). Then, we compute the standard deviation for each pair of overlapped pages using Theorem 1. The sum of the standard deviation divided by the real matrix element value gives us the predicted error for the entire 1-hour interval. Figure 19 shows the cumulative distribution of traffic with the analytically predicted average error. We observe that our bitmap scheme provides very accurate results. Over 98% of the traffic has negligible average error (< 0.03) and the error for around 80% traffic is even below 0.005. Compared with the result in [131], our scheme improves the accuracy by more than an order of magnitude. For example, the error for around 80% traffic in [131], is about 20%. In addition, the average error across all traffic matrix elements in our estimation is around 0.5%, which is also more than an order of magnitude lower than that in [131] (i.e., 11.3%).

4.7 Conclusion

The problem of estimating traffic matrices has received considerable attention recently. In this work, we attack this problem using a brand new approach: network data streaming. Our first main contribution is a novel data streaming algorithm that can produce traffic matrix estimation at least (depending on the amount of SRAM we use) an order of magnitude better than all existing approaches. We also establish principles and techniques for optimally combining this streaming method with sampling through rigorous analysis. In addition, we propose another data streaming algorithm that very accurately estimates *flow matrix*, a finer-grained characterization than traffic matrix. Both algorithms are designed to operate at very high link speeds (e.g., 40 Gbps) using only a small amount of SRAM (e.g., 512KB) and reasonable persistent storage. The accuracy of both algorithms is rigorously analyzed and verified through extensive experiments on synthetic and actual traffic/flow matrices.

CHAPTER V

DETECTION OF SUPER SOURCES AND DESTINATIONS

5.1 Introduction

The problem of detecting *super sources and destinations* has received considerable attention recently [106, 100, 52, 122]. A super source¹ is a source that has a large *fan-out* (e.g., larger than a predefined threshold) defined as the number of distinct destinations it communicates with during a small time interval. The concepts of super destination and *fan-in* can be defined symmetrically. In this context a *source* can be any combination of “source” fields from a packet header such as source IP address, source port number, or their combination, depending on target applications. Similarly, a *destination* can be any combination of the “destination” fields from a packet header. We refer to the *source-destination pair* of a packet as the *flow label* and use these two terms interchangeably in the rest of this chapter.

This problem arises in many applications of network monitoring and security. For example, port-scans probe for the existence of vulnerable services across the Internet by trying to connect to many different pairs of destination IP address and port number. This is clearly a type of super source under our definition. Similarly, in a DDoS (Distributed Denial of Service) attack, a large number of zombie hosts flood packets to a destination. Thus the problem of detecting the launch of DDoS attacks can be viewed as detecting a super destination. This problem also arises in detecting worm propagation and estimating their spreading rates. An infected host often propagates the worm to a large number of destinations, and can be viewed as a super source. Knowing its fan-out allows us to estimate the rate at which the worm may spread. Another possible instance lies in peer-to-peer and content distribution networks, where a few servers or peers might attract a larger number of requests (for content) than they can handle while most of others in the network are relatively idle. Being able to detect such “hot spots” (a type of super destination) in real-time helps balance the

¹Super sources have also been referred to as “*superspreaders*” in literature [122].

workload and improve the overall performance of the network. A number of other variations of the above applications, such as detecting flash crowds [67] and reflector attacks [97], also motivate this problem.

Techniques proposed in the literature for solving this problem typically maintain per-flow state, and hereby suffer the “too much data” problem and cannot scale to high link speeds of 10 or 40 Gbps. For example, to detect port-scans, the widely deployed Intrusion Detection System (IDS) *Snort* [106] maintains a hash table of the distinct source-destination pairs to count the destinations each source talks to. A similar technique is used in FlowScan [100] for detecting DDoS attacks. The inefficiency in such an approach stems from the fact that most of the source-destination pairs are not a part of port scans or DDoS attacks. Yet, they result in a large number of source-destination pairs that can be accommodated only in DRAM, which cannot support the high access rates required for updates at line speed. More recent work [122] has offered solutions based on hash-based flow sampling technique. However, its accuracy is limited due to the typically low sampling rate imposed by some inherent limitations of the hash-based flow sampling technique discussed later in Section 5.3.

In this work we propose two data streaming algorithms for detecting the set of super sources more accurately and efficiently. These algorithms can be easily adapted symmetrically for detecting the super destinations. Our schemes in fact solve a strictly harder problem than making a binary decision of whether a source/destination is a super source/destination or not: They actually provide accurate estimates of the fan-outs/fan-ins of potential super sources/destinations. Their designs are based on the insight that (flow) sampling and data streaming are often suitable for capturing different and complementary regions of the information spectrum, and a close collaboration between them is an excellent way to recover the complete information. This insight leads to two novel methodologies of combining the power of streaming and sampling, namely, “filtering after sampling” and “separation of counting and identity gathering”. Our two solutions are built upon these two methodologies respectively.

Our first solution, referred to as the *simple scheme*, is based on the methodology of “filtering after sampling”. It enhances the traditional hash-based flow sampling algorithm to approximately count the fan-outs of the sampled sources. As suggested by its name, the design of this solution is very simple. Its main innovation is that the sampled traffic is further filtered by a simple data

streaming module (a bit array), which guarantees that at most one packet from each flow is processed. This allows for much higher sampling rate (hence much higher accuracy) than achievable with traditional hash-based flow sampling. Our second solution, referred to as the *advanced scheme*, is more sophisticated than the simple scheme but offers even higher accuracy. Its design is based on the methodology of “separation of counting and identity gathering”, which combines the power of streaming in efficiently estimating quantities (e.g., fan-out) associated with a given identity, and the power of sampling in generating a list of candidate identities (e.g., sources). Through rigorous theoretical analysis and extensive trace-driven experiments on real-world Internet traffic, we demonstrate these two algorithms produce very accurate fan-out estimations.

We also extend our advanced scheme for detecting the sources that have large *outstanding fan-outs*, defined as the number of distinct destinations it has contacted but has not obtained acknowledgments (TCP ACK) from. This extension has several important applications. One example is that in port-scans, the probing packets, which target a large number of destinations, will receive acknowledgments from only a small percentage of them. Another example is distributed TCP SYN attacks. In this case, the victim’s TCP acknowledgments (SYN/ACK packets) to a large number of hosts for completing the TCP handshake (the second step) are not acknowledged. Our evaluation on bidirectional traffic collected simultaneously on a link shows that our solution estimates outstanding fanout with high accuracy.

The rest of this chapter is organized as follows. We start with discussing the related work in Section 5.2. Then Sections 5.3 and 5.4 describe the design of the two schemes in detail respectively and provide a theoretical analysis of their complexity and accuracy. Section 5.5 presents an extension of our scheme for estimating outstanding fan-outs. We evaluate our solutions in Section 5.6 using packet header traces of real-world Internet traffic. We finally conclude the work in Section 5.7.

5.2 *Related Work*

The problem of detecting super sources and destinations has been studied in recent years. In general, three approaches have been proposed in the literature:

1. A straightforward approach is to keep track, for each source/destination, the set of distinct destinations/sources that it contacts, using a hash table. This approach is adopted in Snort [106] and

FlowScan [100]. It is straightforward to implement but not memory-efficient, since most of the source-destination pairs in the hash table do not come from super sources/destinations. As mentioned before, this approach is not feasible for monitoring high-speed links since the hash table typically can only fit into DRAM.

2. Data streaming algorithms are designed by Estan et al. [52] mainly for estimating the number of active flows in the Internet traffic. However, it is stated in [52], that one variant of their scheme, i.e., triggered bitmap, can be used for identifying the super sources. This algorithm maintains a small bitmap (4 bytes) for each source (subject to hash collision), for estimating its fan-out. Once the number of bits set in the small bitmap exceeds a certain threshold (indicating a large fan-out), a large multi-resolution bitmap is allocated to perform a more accurate counting of its fan-out. Since the implementation of the binding between the source and the bitmap is not elaborated in [52], we speculate that the binding is implemented as a hash table, which can be quite costly if it has to fit in SRAM (for high-speed processing). Also, its memory efficiency is further limited by allocating at least 4 bytes for each source.

3. Recently Venkataraman et al. [122] propose two flow sampling based techniques for detecting super sources/destinations. Their one-level and two-level filtering schemes both use a traditional hash-based flow sampling technique for estimating fan-outs. We explained in Section 5.3.1 that, when this scheme is used for high-speed links (e.g., 10 or 40 Gbps), the sampling rate is typically low due to the aforementioned traffic burst problem. This prevents the algorithms from achieving high estimation accuracy. In addition, the memory usage of both schemes, which use hash tables, is much higher than our advanced scheme. They only mentioned the possibility of replacing hash table with Bloom filters to save space, but did not fully specify the details of the scheme (e.g., parameter settings). This makes a head-on comparison of our schemes with theirs very difficult. In fact, after this replacement (of hash table with Bloom filters), their scheme becomes a variant of Space Code Bloom Filter (SCBF) we proposed in [76], with a slightly different decoding algorithm². Their decoding algorithm has similar computational complexity as that of SCBF, which is an order magnitude more expensive than that of our advanced scheme. This may prevent our SCBF scheme

²In [76], we decode for the exact value of the parameter to be estimated while their scheme [122] decodes for a lower bound of the parameter.

(and their scheme as well) from operating at very high link speeds (e.g., 40 Gbps).

5.3 *The Simple Scheme*

In this section we present a relatively simple scheme for detecting super sources. It builds upon the traditional hash-based flow sampling technique but can achieve a much higher sampling rate, and hence more accurate estimation. We begin with a discussion of some limitations of the traditional hash-based sampling approach, and then describe our solution that alleviates these limitations. We also present an analysis of the complexity and accuracy of the scheme.

5.3.1 Limitations of traditional hash-based flow sampling

There are two generic sampling approaches for network measurement: packet sampling and flow sampling. In the former approach, each packet is sampled independently with a certain probability, while in the latter, the sampling decision is made at the granularity of flows (i.e., all packets belonging to sampled flows are sampled). In the following, we only consider flow sampling since packet sampling is not suitable for our context of detecting super sources.³

A traditional flow sampling algorithm that estimates the fan-outs of sources works as follows. The algorithm randomly samples a certain percentage (say p) of source-destination pairs using a hashing technique (described next). The fan-out of each source in the sampled pairs is counted and then scaled by $1/p$ to obtain an estimate of the fan-out of the source in the original traffic (i.e., before sampling). This counting process is typically performed using a hash table that stores the fan-out values (after sampling) of all sources seen in the sampled traffic so far, and a newly sampled flow will increment the fan-out counter of the corresponding hash node (or trigger the creation of a new node). Since the estimation error is also scaled by $1/p$, it is desirable to make the sampling rate p as high as possible. However, we will show that, at high link speeds, the traditional hash-based flow sampling approach may prevent us from achieving high sampling rate needed for accurate estimation.

Flow sampling is commonly implemented using a simple hashing technique [43] as follows. First a hash function g that maps a flow label to a value uniformly distributed in $[0, 1)$ is fixed.

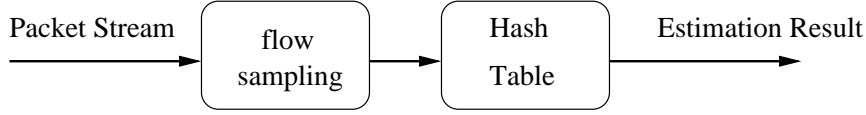
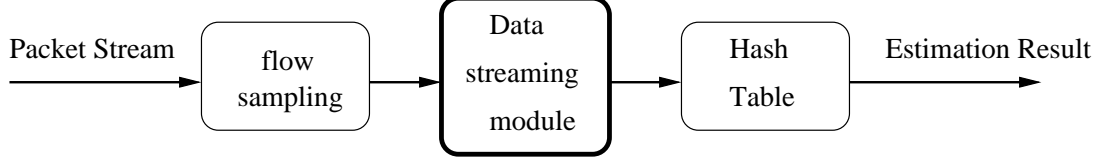
³There is no explicit inversion procedure to recover the number of flows if packet sampling is used. The technique used in [47] may be helpful but does not provide accurate answers.

When a packet arrives, its flow label is hashed by g . Given a sampling probability p , the flow is sampled if and only if the hashing result is no more than p . Recall that the purpose of flow sampling is to reduce the amount of traffic that needs to be processed by the aforementioned hash table which performs the counting. Clearly, it is desirable that a hash table that runs slightly faster than p times the link speed, can keep up with the incoming rate of the sampled traffic (with rate p). For example, we would like a hash table (in DRAM) that is able to process a packet in $400ns$ to handle all traffic sampled from a link with 10 million packets per second (i.e., one packet arrival per $100ns$ on the average) with slightly less than 25% sampling rate. Unfortunately, we cannot achieve this goal with the current hash-based flow sampling approach for the following reason.

With hash-based flow sampling, if a flow is sampled, all packets belonging to the flow need to be processed by the hash table. Internet traffic is very bursty in the sense that the packets belonging to a flow tend to arrive in bursts and do not interleave well with packets from other flows and is also known to exhibit the following characteristic [50]: a small number of elephant flows contain most of the overall traffic while the vast majority of the flows are small. If a few elephant flows are sampled, their packets could generate a long burst of sampled traffic that has much higher rate than that can be handled by the hash table⁴. Therefore, with hash-based flow sampling, the sampling rate p has to be much smaller than the ratio between the operating speed of the hash table and the arrival rate of traffic, thus leading to large estimation errors as discussed before. In the following subsection, we present an efficient yet simple solution to this problem, allowing the sampling rate to reach or even well exceed this ratio.

In [122] the authors propose a *one-level filtering algorithm* which uses the hash-based flow sampling approach described above, in conjunction with a hash table for counting the fan-out values. It does not specify whether DRAM or SRAM will be used to implement the hash table. If DRAM were used, it will not be able to achieve a high sampling rate as discussed before. If SRAM were used, the memory cost is expected to be prohibitive when the sampling rate is high. This algorithm appears to be effective and accurate for monitoring lower link speeds, but cannot deliver a high estimation accuracy when operating at high link speeds such as 10Gbps (the target link speeds are

⁴A small buffer in SRAM will not be able to smooth out such bursts since at high link speeds, such bursts can easily fill up several Megabytes of buffer in a matter of milliseconds.

Traditional flow sampling:**Filtering after sampling:****Figure 20:** Traditional flow sampling vs. filtering after sampling

not mentioned in [122]).

5.3.2 Our scheme

We design a filtering technique that completely solves the aforementioned problem. It allows the sampling rate to be very close to the ratio between the hash table speed and the link speed in the worst-case and well exceed the ratio otherwise. Its conceptual design is shown in Figure 20. Compared with the traditional flow sampling approach, our approach places a data streaming module between the hash-based flow sampling module and the hash table (for counting). *This streaming module guarantees that at most one packet from each sampled flow needs to be processed by the hash table.* This will completely smooth out the aforementioned traffic bursts in the flow-sampled traffic, since such bursts are caused by highly bursty arrivals from one or a small number of elephant flows and now only the first packets of these flows may trigger updates to the hash table.

The data structure and algorithm of the data streaming module are shown in Algorithm 4. Its basic idea is to use a bit array G to remember whether a flow label, a source-destination pair in our context, has been processed by the hash table. Let the size of the array be w bits. We fix a hash function h that maps a flow label to a value uniformly distributed in $[1, w]$. The array is initialized to all “0”s at the beginning of a measurement epoch. Upon the arrival of a packet pkt , we hash its flow label ($\langle pkt.src, pkt.dst \rangle$) using h and the result r is treated as an index into the array G . If $G[r]$ is equal to 1, our algorithm concludes that a packet with this flow label has been processed earlier, and takes no further action. Otherwise (i.e., $G[r]$ is 0), this flow label will be processed to update

Algorithm 4: Algorithm of updating data streaming module.

```
1 Initialize
2    $G[r] := 0, r=1,2,\dots, w$ 
3   /*  $w$  is the size of the array */
4    $u := w$ 
5   /* variable  $u$  keep track the number of “0”s in  $G$  */
6 Filtering after sampling
7   Upon each incoming sampled packet  $pkt$ 
8      $r := h(< pkt.src, pkt.dst >)$ 
9     if  $G[r] = 0$ 
10        $s := pkt.src$ 
11        $\widehat{N}_s := \widehat{N}_s + \frac{w}{u}$ 
12       /* The  $(s, \widehat{N}_s)$  pairs are maintained as a hash table  $L$ . */
13        $G[r] := 1$ 
14        $u := u - 1$ 
15       /* The number of “0”s is decreased by 1 */
```

the corresponding counter $N_{pkt.src}$ maintained in a hash table L . Then $G[r]$ is set to 1 to remember the fact that a packet with this flow has been seen and processed. This method clearly ensures that at most one packet from each sampled flow is processed by L . However, due to hash collisions, some sampled flows may not be processed at all since their corresponding bits in G would be set by their colliding counterparts.⁵ The update procedure of the hash table L , described next, statistically compensates for such collisions.

Now we explain our statistical estimator, which is the computation result of the hash table update procedure shown in Algorithm 4 (line 11). Suppose the number of “0” entries in G (with size w) is u right before a packet pkt with source s arrives ($s := pkt.src$ in line 10). Assume pkt belongs to a new flow and its flow label hashes to an index r . The value of $G[r]$ has value 0 with probability $\frac{u}{w}$. Therefore to obtain an unbiased estimator \widehat{N}_s of the fan-out of the source s on the sampled traffic, we should statistically compensate for the fact that with probability $1 - \frac{u}{w}$, the bit $G[r]$ has value 1 and pkt will miss the update to L due to aforementioned hash collisions. It is intuitive that if we add $\frac{w}{u}$ to \widehat{N}_s , the resulting estimator is unbiased. To be more precise, suppose in a measurement epoch, the hash table is updated by altogether K packets $\{pkt_j, j = 1, 2, \dots, K\}$ from a source s , whose flow labels hash to locations r_j ’s where $G[r_j] = 0$, and there are u_j 0’s in G right before pkt_j

⁵We can use multiple independent hash functions to reduce the probability of collisions. But it will significantly increases the overhead of updating G and does not improve the estimation result too much.

arrives, respectively. The output of the hash table L , which is an unbiased estimator of the fan-out of s on the sampled traffic, is

$$\widehat{N}_s = \sum_{j=1}^K \frac{w}{u_j} \quad (24)$$

We show in the following lemma that this is an unbiased estimator of N_s and its proof can be found in Appendix A.3.1.

Lemma 1 \widehat{N}_s is an unbiased estimator of N_s , i.e., $E[\widehat{N}_s] = N_s$.

Then an unbiased estimator of the fan-out F_s of source s is given by scaling \widehat{N}_s by $1/p$, i.e.,

$$\widehat{F}_s = \frac{1}{p} \sum_{j=1}^K \frac{w}{u_j} \quad (25)$$

where p is the sampling rate used in the flow sampling. We show in the following theorem that the estimator \widehat{F}_s is unbiased. Its proof uses Lemma 1 and is provided in Appendix A.3.2.

Theorem 3 \widehat{F}_s is an unbiased estimator of F_s , i.e., $E[\widehat{F}_s] = F_s$.

We now demonstrate that this solution will completely smooth out the aforementioned problem of traffic bursts, and allow the sampling rate to be close to the ratio between the hash table speed and the link rate, the theoretical upper limit in the worst case. The worst case for our scheme is that each flow contains only one packet (e.g., in the case of DDoS attacks)⁶. Even in this worst case, the update times to the hash table (viewed as a random process) is very close to Poisson⁷ (nonhomogeneous as the value of u varies over time) since each new flow is sampled independently. Due to the “benign” nature of this arrival process, by employing a tiny SRAM buffer (e.g., holding 20 flow labels of 64 ~ 100 bits each), a hash table that operates slightly faster than the average rate of this process will only miss a negligible fraction of updates due to buffer overflow. This process can be faithfully modeled as a Markov chain for rigorous analysis. We will elaborate it with a numerical example in Section 5.3.3.

Notice that in Algorithm 4 the variable u , the number of “0” entries in G , decreases as more and more sampled flows are processed. When more and more packets pass through the data streaming

⁶Note that the worst case for hash-based flow sampling is different. It occurs when a few of the sampled flows contain most of the traffic on a link.

⁷The inter-arrival time is in fact of geometric distribution.

module, u becomes small and hence the probability for a new flow to be recorded, $\frac{u}{w}$, decreases. Thereby the estimation error will increase. To maintain high accuracy, we specify a minimum value u_{min} for u . Once the value of u drops below u_{min} , the estimation procedure will use a new array (set to all “0”s initially) and start a new measurement epoch (with an empty hash table). Two sets of arrays and hash tables will be operated in an alternating manner so that the measurement can be performed without interruption. The parameter u_{min} is typically set to around $w/2$ (i.e., “half full”).

5.3.3 Complexity analysis

The above scheme has extremely low storage (SRAM) complexity and allows for very high streaming speed.

Memory (SRAM) consumption. Each processed flow only consumes a little more than one bit in SRAM. Thus a reasonable amount of SRAM can support very high link speeds. For example, assuming the average flow size of 10 packets [73], 512KB SRAM is enough to support a measurement epoch which is slightly longer than 2 seconds for a link with 10 million packets per second even without performing any flow sampling. With 25% flow sampling which is typically set for OC-192 links the SRAM requirement is even brought down to 128KB.⁸

Streaming speed. Our algorithm in Algorithm 4 has two branches to deal with the packets arriving at the data streaming module. If the corresponding bit is “1”, the packets only require one hash function computation and one read to SRAM. Otherwise they require one hash function computation, one read and one write (at the same location) to SRAM and an update to the hash table. Using efficient hardware implementation of hash function [103] and $5ns$ SRAM, all operations in the data streaming module can be finished in 10’s of ns in both cases. Next we will discuss the computational complexity of the procedure of updating the hash table.

Consider a hash table⁹ that can process an update in $400ns$ and a link with 10 million packets arriving per second (i.e., one packet arrival per $100ns$ on average). Assume the arriving traffic will

⁸We assume a conservative average packet size of 1,000 bits, to our disadvantage. Measurements from real-world Internet traffic report much larger packet sizes.

⁹Given a large number of hash table entries, we can guarantee that each update costs at most four memory accesses with high probability. Thus an update can be easily finished in $400ns$ using $60ns$ DRAM.

issue the update request to the hash table with the probability no more than 20%.¹⁰ Then we can model the corresponding dynamics of the hash table as follows. The update times to the hash table is a random process R with geometric inter-arrival time. The average arrival rate is 2 million updates per second ($10^6 \times 20\%$). The service time is constant ($400ns$). Clearly the hash table operates 1.25 times faster than the average arrival rate. In other words the load of the system is 80%. Then given a buffer size of b flow labels ($64 \sim 100$ bits per label) the system can be faithfully modeled as a Markov chain of $(4b + 5)$ states. If b is set to 20 flow labels (i.e., about 160 \sim 240 bytes), the hash table will only miss a small fraction ($\sim 1.8 \times 10^{-6}$) of updates due to buffer overflow.

5.3.4 Accuracy analysis

Now we establish the following theorem to characterize the variance of the estimator \widehat{F}_s in Formula 25. Its proof can be found in Appendix A.3.3

Theorem 4

$$Var[\widehat{F}_s] \approx \frac{\sum_{j=1}^{pF_s} \frac{w-u_j}{u_j}}{p^2} + \frac{F_s(1-p)}{p}$$

Remark: The above variance consists of two terms. The first term corresponds to the variance of the error term in estimating the sampled fan-out, scaled by $\frac{1}{p^2}$ (to compensate for sampling), and the second term corresponds to the variance of the error term in inverting flow sampling process. Since these two errors are approximately orthogonal to each other, their total variance is the sum of their individual variances.

5.4 The Advanced Scheme

In this section we propose the advanced scheme that is more sophisticated than the simple scheme but can offer more accurate fan-out estimations. It is based on the aforementioned design methodology of separating identity gathering from counting. Its system model is shown in Figure 21. There are two parallel modules processing the incoming packet stream. The data streaming module encodes the fan-out information for each and every source (arc 1 in Figure 21) into a very compact data structure, and the identity sampling module captures the candidate source identities which have

¹⁰This can be achieved by setting the flow sampling rate to 20% in the worst case, e.g., DDoS attacks.

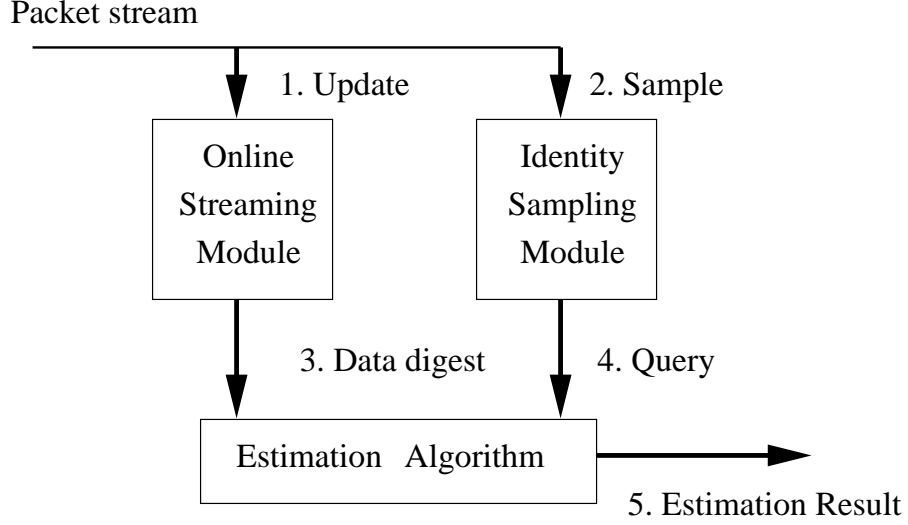


Figure 21: System model of the advanced scheme

Algorithm 5: Algorithm of online streaming module

```

1 Initialize
2    $A[i, j] := 0, \quad i = 1, 2, \dots, m \quad j = 1, 2, \dots, n$ 
3 Update
4   Upon the arrival of a packet  $pkt$ 
5      $row := h'(< pkt.src, pkt.dst >)$ 
6     for  $i := 1$  to  $k$ 
7        $col := h_i(pkt.src)$ 
8        $A[row, col] := 1$ 
  
```

potential to be super sources (arc 2). These source identities are then used by an *estimation algorithm* to look up the data structure (arc 3) produced by the data streaming module (arc 4) to get their corresponding fan-out estimates. The design of these modules are described in the following subsections.

5.4.1 Online streaming module

The data structure used in the online streaming module is quite simple: an $m \times n$ 2-dimensional bit array A . The bits in A are set to all “0”s at the beginning of each measurement epoch. The algorithm of updating A is shown in Algorithm 5. Upon the arrival of a packet pkt , $pkt.src$ is hashed by k independent¹¹ hash functions h_1, h_2, \dots, h_k with range $[1..n]$. The hashing results $h_1(pkt.src)$,

¹¹Such hash functions are referred to as k -universal hash function in literature [21]. It has been shown empirically in [21] that the H_3 family of hash functions are very close to k -universal statistically when operating on real-world data,

$h_2(pkt.src), \dots, h_k(pkt.src)$ are viewed as column indices into A . In our scheme, k is set to 3, and the rationale behind it will be discussed in Section 5.4.3. Then, the flow label $\langle pkt.src, pkt.dst \rangle$ is hashed by another independent hash function h' (with range $[1..m]$) to generate a row index of A . Finally, the k bits located at the intersections of the selected row and columns are set to “1”. When A is filled (by “1”) to a threshold percentage we terminate the current measurement epoch and start the decoding process¹². In Section 5.4.2, we show that the above process produces a very compact and accurate (statistical) encoding of the fan-outs of the sources, and present the corresponding decoding algorithm.

The proposed online streaming module has very low memory consumption and high streaming speed:

Memory (SRAM) consumption. Our scheme is extremely memory-efficient. Each source-destination pair (flow) will set 3 bits in the bit vectors to “1”s and consume a little more than 3 bits of SRAM storage¹³. We will show that the scheme provides very high accuracy using reasonable amount of SRAM (e.g., 128KB) in Section 5.6.

Streaming speed. Each update requires only 4 hash function computations and 3 writes to the SRAM. We require that these four hash functions are independent and amendable to hardware implementation. They can be chosen from the H_3 hash function family [21, 103], which, with hardware implementation, can produce a hash output within a few nanoseconds. Then with commodity 5ns SRAM our scheme would allow around 40 million packets per second, thereby supporting 40 Gbps traffic stream assuming a conservative average packet size of 1,000 bits.

5.4.2 Estimation module

For each source identity recorded by the sampling module (described later), the estimation module decodes its approximate fan-out from the 2D bit array A , the output of the data streaming module. In this section, we describe this decoding algorithm in detail.

When we would like to know F_s , the fan-out of the source s , s is hashed by the hash functions h_1, \dots, h_k , which are defined and used in the online streaming module, to obtain k column indices.

for small k values (e.g., $k \leq 4$).

¹²Again, two ping-pong modules can be used in an alternating fashion to avoid any operational interruption.

¹³This is estimated based on the typical *load factor* (defined later) we place on the bit vector.

Let $A_i, i = 1, 2, \dots, k$, be the corresponding columns (viewed as bit vectors). In the following, we derive, step by step, an accurate and almost unbiased estimator of F_s , as a function of $A_i, i = 1, 2, \dots, k$.

Let the set of packets hashed into column A_i during the corresponding measurement epoch be T_i and the number of bits in A_i that are “0”s be U_{T_i} . Note that the value U_{T_i} is a part of our observation since we can obtain U_{T_i} from A_i through simple counting, although the notation itself contains T_i , the size of which we would like to estimate. Recall the size of the column vector is m . A fairly accurate estimator of $|T_i|$, the number of packets of T_i , adapted from [125], is

$$D_{T_i} = m \ln \frac{m}{U_{T_i}} \quad (26)$$

Note that F_s , the fan-out of the source s , is equal to $|T_1 \cap T_2 \cap \dots \cap T_k|$, if during the measurement epoch, no other sources are hashed to the same k columns A_1, A_2, \dots, A_k . Otherwise $|T_1 \cap T_2 \cap \dots \cap T_k|$ is the sum of the fan-outs of all (more than 1) the sources that are hashed into A_1, A_2, \dots, A_k . We show in the next section, that the probability with which the latter case happens is very small when $k = 3$. We obtain the following estimator of F_s , which is in fact derived as an estimator for $|T_1 \cap T_2 \cap \dots \cap T_k|$.

$$\begin{aligned} \widehat{F_s} &= \sum_{1 \leq i \leq k} D_{T_i} - \sum_{1 \leq i_1 < i_2 \leq k} D_{T_{i_1} \cup T_{i_2}} \\ &\quad + \sum_{1 \leq i_1 < i_2 < i_3 \leq k} D_{T_{i_1} \cup T_{i_2} \cup T_{i_3}} \\ &\quad + \dots + (-1)^{k-1} D_{T_1 \cup T_2 \dots \cup T_k} \end{aligned} \quad (27)$$

Here $D_{T_{i_1} \cup \dots \cup T_{i_j}}$, is defined as $m \ln \frac{m}{U_{T_{i_1} \cup \dots \cup T_{i_j}}}$, where $U_{T_{i_1} \cup \dots \cup T_{i_j}}$ denotes the number of “0”s in the bit vector $B_{T_{i_1} \cup \dots \cup T_{i_j}}$ which is the result of hashing the set of packets $T_{i_1} \cup \dots \cup T_{i_j}$ into a single empty bit vector. The bit vector $B_{T_{i_1} \cup \dots \cup T_{i_j}}$ is computed as the bitwise-OR of A_{i_1}, \dots, A_{i_j} . One can easily verify the correctness of this computation with respect to the semantics of $B_{T_{i_1} \cup \dots \cup T_{i_j}}$.

We need to show that the RHS of Formula 27 is a fairly good estimator of $|T_1 \cap T_2 \cap \dots \cap T_k|$.

Note that

$$\begin{aligned}
|T_1 \cap T_2 \cap \dots \cap T_k| &= \sum_{1 \leq i \leq k} T_i - \sum_{1 \leq i_1 < i_2 \leq k} T_{i_1} \cup T_{i_2} \\
&+ \sum_{1 \leq i_1 < i_2 < i_3 \leq k} T_{i_1} \cup T_{i_2} \cup T_{i_3} \\
&+ \dots + (-1)^{k-1} T_1 \cup T_2 \dots \cup T_k
\end{aligned} \tag{28}$$

by the principle of inclusion and exclusion. Since $D_{T_i \cup \dots \cup T_j}$ is a fairly good estimator of $|T_i \cup \dots \cup T_j|$ according to [125], we obtain the RHS of Formula 27 by replacing the terms $|T_i \cup \dots \cup T_j|$ in Formula 28 by $D_{T_i \cup \dots \cup T_j}$, $1 \leq i < j \leq k$. Note that it is not correct to directly use the bitwise-AND of A_1, A_2, \dots, A_k for estimating $|T_1 \cap T_2 \cap \dots \cap T_k|$ using Formula 26, because the bit vector corresponding to the result of hashing the set of packets $T_1 \cap T_2 \cap \dots \cap T_k$ into an empty bit vector, is not equivalent to the bitwise-AND of A_1, \dots, A_k .

The estimator in Formula 27 generalizes the result in [125] which is developed for the special case $k = 2$. We will show that our scheme only needs to use the special case of $k = 3$, which is

$$\begin{aligned}
\widehat{F_s} &= D_{T_1} + D_{T_2} + D_{T_3} - D_{T_1 \cup T_2} - D_{T_1 \cup T_3} - D_{T_2 \cup T_3} \\
&+ D_{T_1 \cup T_2 \cup T_3}
\end{aligned} \tag{29}$$

The computational complexity of estimating the fan-out of a source is dominated by $2^k - k - 1$ bitwise-OR operations among k column vectors. Such vectors can be encoded as one or more unsigned integers so that the bit-parallelism can significantly reduce the execution time. Since m is typically 64 bits in our scheme, the whole vector can be held in two 32-bit integers. Therefore, in our scheme where $k = 3$, estimation of the fan-out of each source only needs 8 bitwise-OR operations between 32-bit integers. We also need to count the number of “0”s in a vector (to get U_T values). This can be sped up significantly by using a pre-computed table (in SRAM) of size 262,144 ($= 2^{16} \times 4$) bits that stores the number of “0”s in all 16-bit numbers. Our estimation of the execution time shows that our scheme is fast enough to support OC-768 operations.

5.4.3 Accuracy analysis

In this section we first briefly explain the rationale behind setting k to 3 in the estimator and then analyze the accuracy of our estimator rigorously. We set k (the number of “column” hash functions)

to 3 due to the following two considerations. First, we mentioned before that if two sources s_1 and s_2 both are hashed to the same k columns, our decoding algorithm will give us an estimate of their total fan-out, when we use s_1 or s_2 to lookup the 2D array. We certainly would like the probability with which this scenario occurs to be as small as possible. This can be achieved by making k as large as possible. However, larger k implies larger computational and storage complexities at the online streaming module. We will show that $k = 3$ makes the probability of the aforementioned hash collision very small, and at the same time keeps the computational and storage complexities of our scheme modest.

Now we derive η , the probability that at least two sources happen to hash to the same set of k columns by h_1, h_2, \dots, h_k . It is not hard to show, using straightforward combinatorial arguments, that $\eta = 1 - \frac{\binom{n}{k} S!}{n^{kS}}$, where S is the total number of the distinct sources during the measurement epoch. We observe that, given typical values for n and S , η is quite large when $k = 2$, but drops to a very low value when $k = 3$. For example, when $n = 16K$ and $S = 100,000$, η is close to 1 (around $1 - 3 \times 10^{-9}$) when $k = 2$, but drops to 0.0012 when $k = 3$.

The following theorem characterizes the variance of the estimator in Formula 29, which is also its approximate mean square error (MSE), since the estimator is almost unbiased and the impact of η (discussed above) on the estimation error is very small when $k = 3$. Its proof can be found in Appendix A.3.4. This is an extension of our previous variance analysis in [133] which is derived for the special case $k = 2$. Let t_T denote $|T|/m$, which is the load factor of the bit vector (of size m) when the corresponding set T of source-destination pairs are hashed into it, in the following theorem.

Theorem 5 *The variance of \widehat{F}_s is given by*

$$\begin{aligned} \text{Var}[\widehat{F}_s] &\approx -m \sum_{i=1}^3 f(t_{T_i}) - m \sum_{1 \leq i_1 < i_2 \leq 3} f(t_{T_{i_1} \cup T_{i_2}}) \\ &\quad + 2m(f(t_{T_1 \cup (T_2 \cap T_3)}) + f(t_{T_2 \cup (T_1 \cap T_3)}) + f(t_{T_3 \cup (T_2 \cap T_1)})) \\ &\quad + 2m \sum_{1 \leq i_1 < i_2 \leq 3} f(t_{T_{i_1} \cap T_{i_2}}) + m f(t_{T_1 \cup T_2 \cup T_3}) \\ &\quad - 2m(f(t_{T_1 \cap (T_2 \cup T_3)}) + f(t_{T_2 \cap (T_1 \cup T_3)}) + f(t_{T_3 \cap (T_2 \cup T_1)})) \end{aligned}$$

where $f(t) = e^t - t - 1$.

Note that given the size of m , our scheme is only able to accurately estimate fan-out values up to $m \ln m + O(m)$, because if the actual fan-out F_s is much larger than that, we will see all 1's in the corresponding column vectors with high probability (due to the result of the “coupon collector’s problem” [86]). In this case, the only information we can obtain about F_s is that it is no smaller than $m \ln m$. Fortunately, for the purpose of detecting super sources, this information is good enough for us to declare s a super source, as long as the threshold for super sources is much smaller than $m \ln m$. However, in some applications (e.g., estimating the spreading speed of a worm), we may also want to know the approximate fan-out value. This can be achieved using a *multi-resolution* extension of our advanced scheme. The methodology of *multi-resolution* is quite standard and has been used in several recent works [52, 76, 73]. The extension in our context is straightforward. We omit its detailed specifications here in interest of space.

5.4.4 Identity sampling module

The purpose of this module is to capture the identities of potential super sources that will be used to look up the 2D array to get their fan-out estimations. The filtering after sampling technique proposed in Section 5.3 is adopted here with a slightly different recording strategy. Instead of constructing a hash table to record the sources and their fan-out estimation, here we only record the source identities sequentially in the DRAM. Since this strategy avoids expensive hash table operations and sequential writes to DRAM can be made very fast (using burst mode), very high sampling rate can be achieved. With commodity $5ns$ SRAM and $60ns$ DRAM, this recording strategy will be able to process more than 12.5 million packets per second. At this speed, we can record 100% and 25% flow labels for OC-192 and OC-768 links respectively. With such a high sampling rate, the probability that the identity of a real super source misses sampling is very low. For example, given 25% sampling rate the probability that a source with fan-out 50 fails to be recorded is only 5.6×10^{-7} ($= (1 - 25\%)^{50}$).

5.5 Estimating Outstanding Fan-outs

In this section we describe how to extend the advanced scheme to detect the sources that have contacted but have not obtained acknowledgments from a large number of distinct destinations (i.e.,

Algorithm 6: Algorithm for updating the 2D bit array B to record ACK packets

```
1 Initialize  
2    $B[i, j] := 0, i = 1, 2, \dots, m \quad j = 1, 2, \dots, n$   
3 Update  
4   Upon the arrival of a packet  $pkt$   
5     if  $pkt$  is an acknowledgment packet  
6        $row := h'(< pkt.dst, pkt.src >)$   
7       for  $i := 1$  to  $k$   
8          $col := h_i(pkt.dst)$   
9          $B[row, col] := 1$ 
```

the sources with large outstanding fan-outs). Although both of our schemes have the potential to support this extension we focus on the advanced scheme in this work and leave the extension of the simple scheme for future research. In the following sections we show how to slightly modify the operations of the online streaming module and the estimation module of the advanced scheme for estimating outstanding fan-outs. The sampling module does not need to be modified.

5.5.1 Online streaming module

The online streaming module employs two 2D bit arrays A and B of identical size and shape. The array A encodes the fan-outs of sources in traffic in one direction (called “outbound”) in the same way as in the advanced scheme (shown in Algorithm 5). The array B encodes the fan-ins of the destinations of the acknowledgment packets in the opposite direction (called “inbound”). Its encoding algorithm is shown in Algorithm 6. It is a transposed version of the algorithm shown in Algorithm 5 in the sense that all occurrences of “ $pkt.src$ ” are replaced with “ $pkt.dst$ ” and “ $pkt.dst$ ” with “ $pkt.src$ ”. This transposition is needed since a source in the outbound traffic appears in the inbound acknowledgment traffic as a destination, and after transposing two packets that belong to a flow and its “acknowledgment flow” respectively will result in a write of “1” to the same bit locations in A and B respectively. This allows us to essentially take a “difference” between A and B to obtain the decoding of outstanding fan-outs of various sources, shown next.

5.5.2 Estimation module

We compute the bitwise-OR of A and B , denoted as $A \vee B$. For each source s , we decode its fan-out from $A \vee B$ using the same decoding algorithm as described in Section 5.4.2. Similarly, we decode

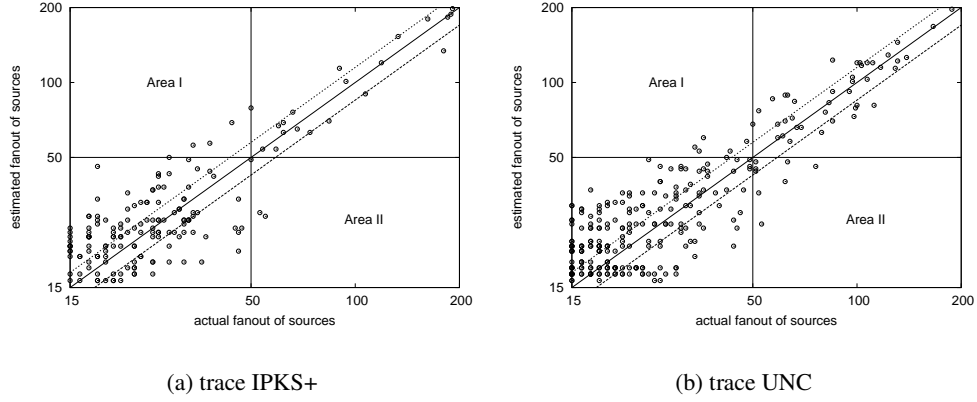


Figure 22: Actual vs. estimated fan-outs of sources by the simple scheme given the flow sampling rate 1/4. Notice both axes are on logscale.

its fan-in in the acknowledgment traffic from B . Our estimator of the outstanding fan-out of s is simply the former subtracted by the latter.

Now we explain why this estimator will provide an accurate estimate of the outstanding fan-out of a source s . Let S_1 be the set of flows whose source is “ s ” in the outbound traffic. Let S_2 be the set of flows whose destination is “ s ” in the inbound acknowledgment traffic. Clearly the quantity we would like to estimate is simply $|S_1 - S_2|$. The correctness of our estimator is evident from the following three facts: (a) $|S_1 - S_2|$ is equal to $|S_1 \cup S_2| - |S_2|$; (b) decoding from $A \vee B$ will result in a fairly accurate estimate of $|S_1 \cup S_2|$ and (c) decoding from B will result in a fairly accurate estimate of $|S_2|$.

5.6 Evaluation

In this section, we evaluate the proposed schemes using real-world Internet traffic traces. Our experiments are grouped into three parts corresponding to the three algorithms presented: the simple scheme, the advanced scheme, and its extension to estimate outstanding fan-outs. The experimental results show that our schemes allow for accurate estimation of fan-outs and hence the precise detection of super sources.

5.6.1 Traffic Traces and Flow definitions

Trying to make the experimental data as representative as possible, we use packet header traces gathered at two different locations of the Internet, namely, University of North Carolina (UNC) and NLANR. The trace from UNC was collected on a 1 Gbps access link connecting the campus to the rest of the Internet, on Thursday, April 24, 2003 at 11:00 am. The traces *IPKS+* and *IPKS-* are collected simultaneously on both directions of an OC192c link on June 1, 2004. The link connects Indianapolis (IPLS) to Kansas City (KSCY) using Packet-over-SONET. This pair of traces is especially valuable to evaluate the extended advanced scheme for estimating outstanding fan-outs. All the above traces are either publicly available or available for research purposes upon request. We will use *UNC* and *IPKS+* to evaluate our simple scheme and advanced scheme and use the concurrent traces *IPKS+* and *IPKS-* to evaluate the extension.

As mentioned before, a source/destination label can be any combination of source/destination fields from the IP header. Two different definitions of source and destination labels are used in our experiments, targeting different applications. In the first definition, source label is the tuple $\langle \text{src_IP}, \text{src_port} \rangle$ and destination label is $\langle \text{dst_IP} \rangle$. This definition targets applications such as detecting worm propagation and locating popular web servers. In the second definition, source label is $\langle \text{src_IP} \rangle$ and destination label is the tuple $\langle \text{dst_IP}, \text{dst_port} \rangle$. This definition targets applications such as detecting infected sources that conduct port scans. The experimental results presented in this section use the first definition of source and destination labels unless noted otherwise.

5.6.2 Accuracy of the simple scheme

In this section, we evaluate the accuracy of the simple scheme in estimating the fan-outs of sources and in detecting super sources. Figure 22 compares the fan-outs of the sources estimated using our simple scheme with their actual fan-outs in traces *IPKS+* and *UNC* respectively. In these experiments, a flow sampling rate of $1/4$ and a bit array of size $128K$ bits is used. The figure only plots the points whose actual fan-out values are above 15 since lower values (i.e., < 15) are not interesting for finding super sources. The solid diagonal line in each figure denotes perfect estimation, while the dashed lines denote an estimation error of $\pm 15\%$. The dashed lines are parallel to the diagonal line since both x-axis and y-axis are on the log scale. Clearly the closer the points

cluster around the diagonal, the more accurate the estimation is. We observe that the simple scheme achieves reasonable accuracy for relatively large fan-outs in all three traces. Figure 22 also reflects the false positives and negatives in detecting super sources. For a given threshold 50, the points that fall in “Area I” corresponds to *false positives*, i.e., the sources whose actual fan-outs are less than the threshold but the estimated fan-outs are larger than the threshold. Similarly, the points that fall in “Area II” corresponds to *false negatives*, i.e., the sources whose actual fan-outs are larger than the threshold but the estimated fan-outs are smaller than the threshold. We observe that in Figure 22, points rarely fall into Areas I and II (i.e., very few false positives and negatives¹⁴).

While this scheme works well with 1/4 sampling rate, this sampling rate should not be decreased further. Figure 23 plots the results when the flow sampling rate becomes 1/16 in the trace *IPKS+*. However such lower sampling rates might be necessary to keep up with very high link speeds such as 40 Gbps (OC-768). Compared with the results in Figure 22(a) the points in the figure deviate much more widely from the diagonal and the points falling in Area I and II also increase. Therefore the simple scheme does not work well with the small sampling rate (e.g. $< 1/4$).

We repeat the above experiment under the aforementioned second definition of source and destination, in which the source label is $\langle \text{src_IP} \rangle$ and destination label is $\langle \text{dst_IP}, \text{dst_port} \rangle$. Figure 24 plots the estimated fan-outs of sources in trace *IPKS+*. With this definition the trace *IPKS+* has 9,359 sources and 140,140 distinct source-destination pairs. We can see from the figure that our estimation is also quite accurate with this second definition of source and destination.

5.6.3 Accuracy of the advanced scheme

In this section we evaluate the accuracy of the advanced scheme using both trace-driven simulation and theoretical analysis. The estimation accuracy of the advanced scheme is a function of the various design parameters, including the size and shape of the 2D bit array A (i.e., the number of rows m and columns n) and the number of hash functions (k).

In the experiments we set the size of A to 128KB (64 rows \times 16,384 columns), $k = 3$ and the flow sampling rate to 1. This configuration is very space-efficient. For example it only uses 7 bits

¹⁴One shall not simply compare this false positive and negative ratios with the results in [122] since there only when the scheme fails to detect a source whose fan-out is several (say 5) times larger than the threshold will a false negative be declared.

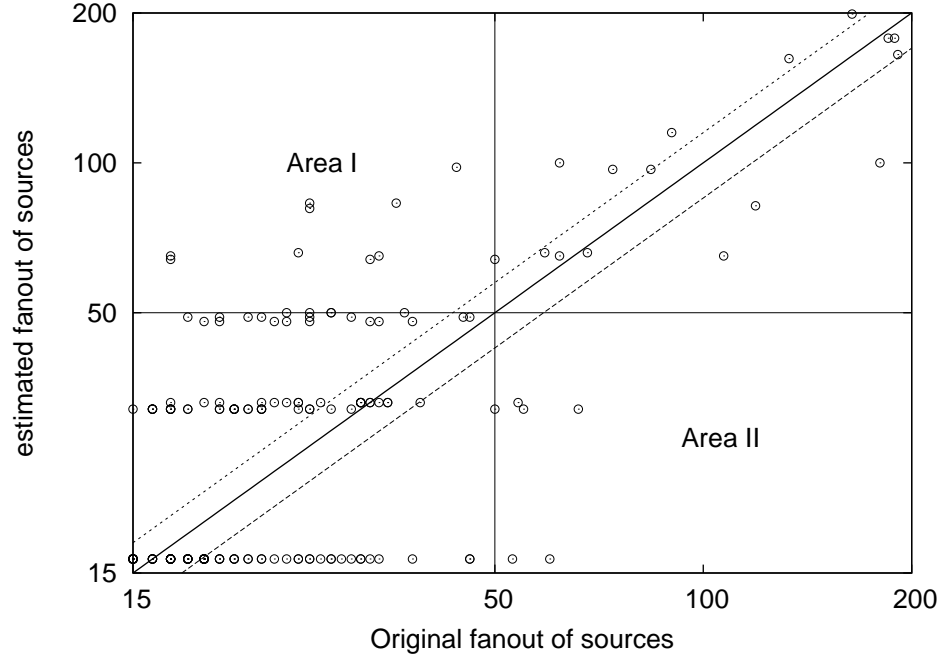


Figure 23: Original vs. estimated fan-out of sources for trace *IPKS+* by the simple scheme given the flow sampling rate 1/16. Notice both axes are on logscale.

per flow on the average for the trace *IPKS+*.

5.6.3.1 Trace-driven experiments

Figure 25 compares the fan-out values estimated using the advanced scheme with the actual fan-outs of the corresponding sources given three different traces. Compared with the corresponding plots in Figure 22, the points are much closer to the diagonal lines, which means that the advanced scheme is much more accurate than the simple scheme.

In Figure 26, we repeat the experiments with source and destination labels defined as `<src_IP>` and `<dst_IP,dst_port>`, respectively. Compared with the result of the simple scheme (Figure 24) the points are much closer to the diagonal again, indicating the higher accuracy achieved by the advanced scheme.

Note that in the experiments above we set the flow sampling rate to 1 instead of 1/4 used in the experiments of the simple scheme since as we described in Section 5.3.3 and Section 5.4.4 respectively for a fully utilized OC-192 link the simple scheme requires 1/4 flow sampling rate but the identity sampling module of the advanced scheme can record 100% flow labels.

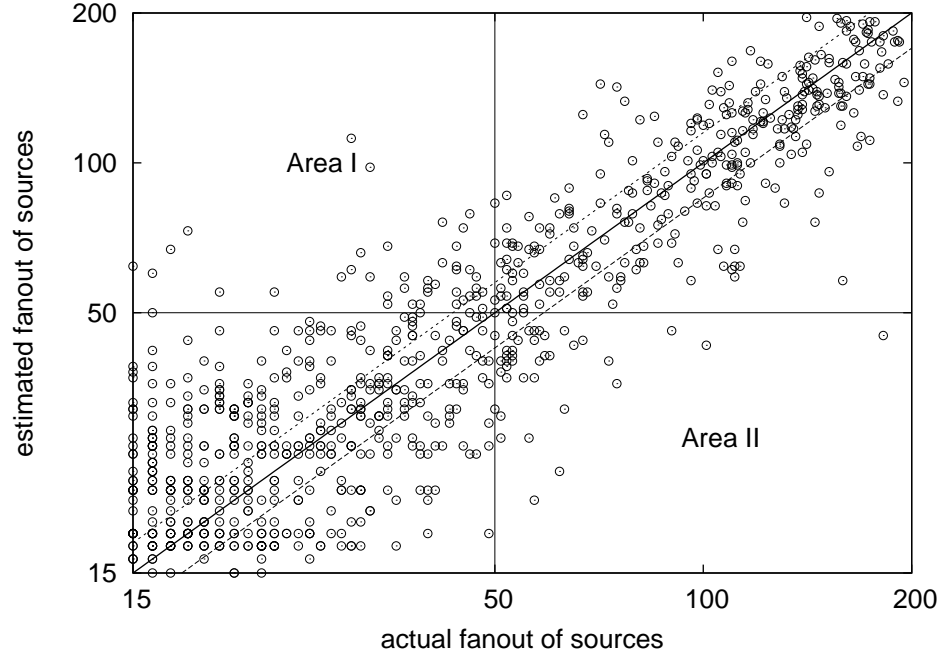


Figure 24: Actual vs. estimated fan-out of sources for trace *IPKS+* with flow sampling rate 1/4. The aforementioned second definition of source and destination labels is used here. Note that both x-axis and y-axis are on logscale.

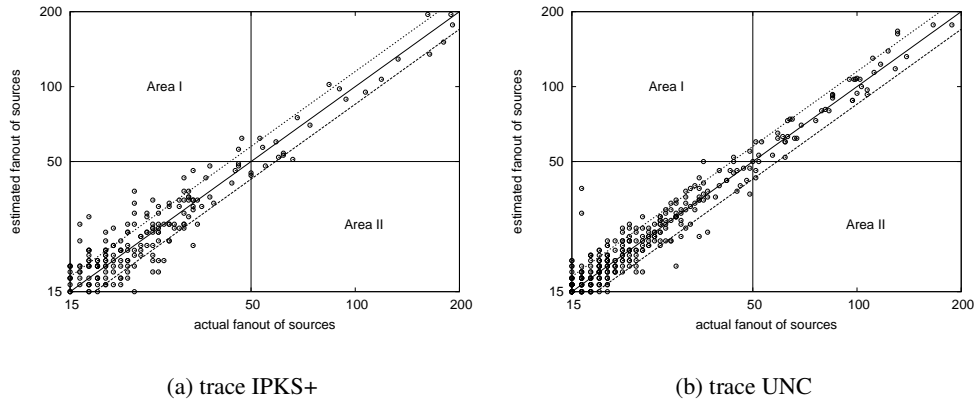


Figure 25: Actual vs. estimated fan-out of sources by the advanced scheme. Notice both axes are on logscale.

5.6.3.2 Theoretical accuracy

The accuracy of the estimation can be characterized by the average relative error of the estimator, which is equal to the standard deviation of the ratio $\frac{\widehat{F}_s}{F_s}$ which can be computed by Theorem 5.

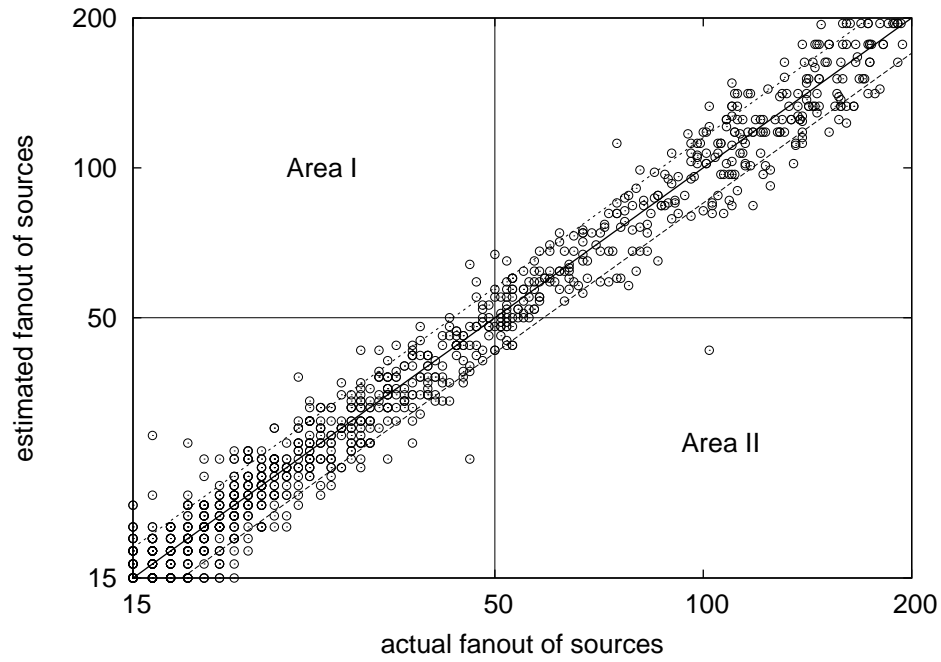


Figure 26: Actual vs. estimated fan-out of sources for trace *IPKS+* under the second flow definition by the advanced scheme. Notice both axes are on logscale.

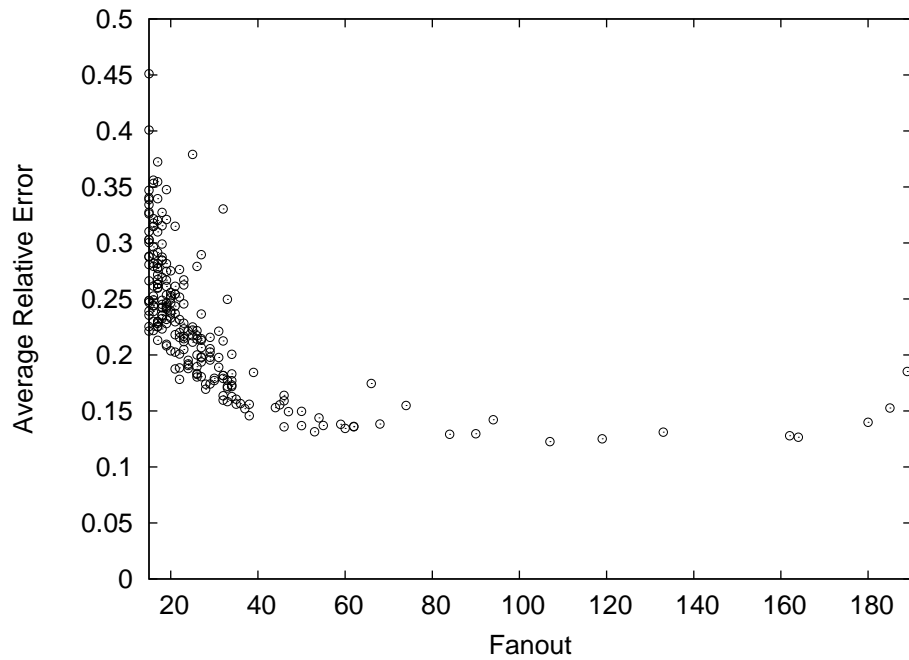


Figure 27: Average relative error for various fan-out values in the trace *IPKS+*.

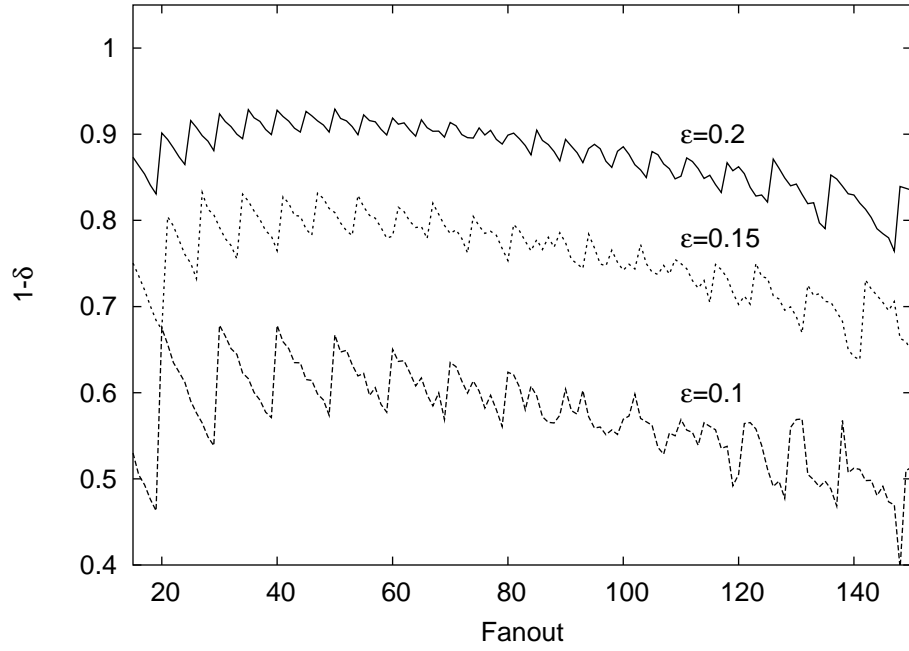


Figure 28: Probability that the estimate \widehat{F}_S is within a factor of $(1 \pm \epsilon)$ of the actual fan-out F_s for various values of ϵ .

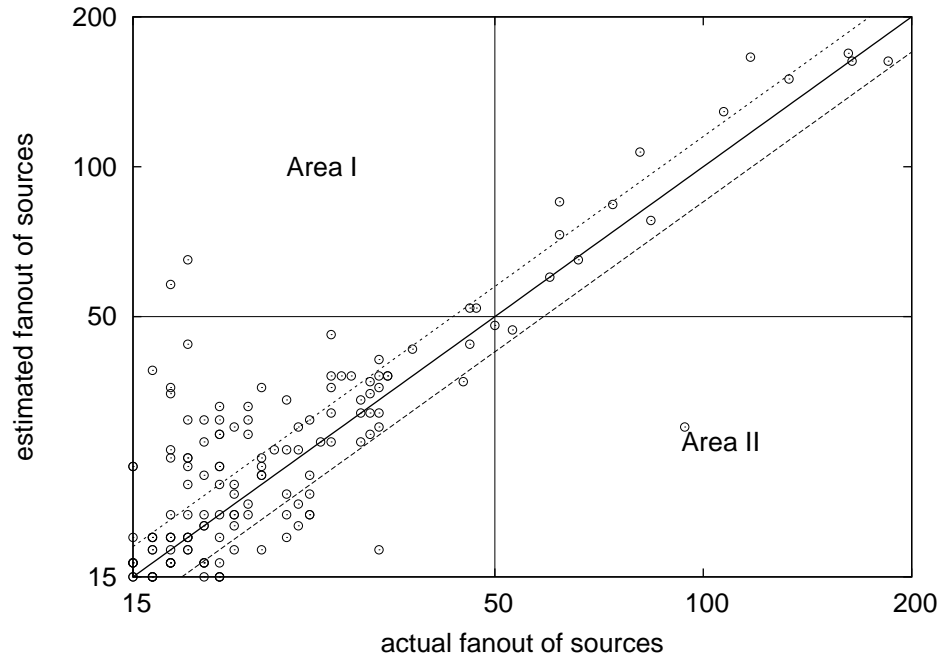


Figure 29: Actual vs. estimated fan-out of sources by extension of the advanced scheme including deletions. Notice both axes are on logscale.

Figure 27 shows the average relative error plotted against estimated fan-outs for the sources in the trace *IPKS+*. Experiments on other traces produced similar results. The average relative error shows a sharply downward trend when the estimated value of fan-out increases in Figure 27. This is a very desirable property as we would like our mechanism to be more accurate when estimating larger fan-outs. Towards the right extreme of the figure, the average relative error starts to increase. This is because the selected bit vectors become almost full (“saturation”) when the fan-out value is close to 266 ($m \ln m$). As we discussed in Section 5.4.3 the accuracy of our estimator would degrade when the corresponding column vectors become saturated¹⁵. It does not affect the accuracy of our scheme for detecting super sources, but to accurately estimate the exact fan-out values that are large, the aforementioned multi-resolution extension [52, 73, 76] is needed.

The accuracy of the estimator can also be characterized by the probability of the estimated values \widehat{F}_s falling into the interval $[(1 - \epsilon)F_s, (1 + \epsilon)F_s]$, where F_s is actual fan-out of the source s . This quantity can be numerically computed by Monte-Carlo Simulation as follows. We first use the trace *UNC* to construct the 2D bit array A (serving as “background noise”). Then we synthetically generate a source that has fan-out value F_s and insert it into A by randomly selecting 3 different columns. The estimator (Formula 29) is used to obtain the \widehat{F}_s . The above operations are repeated 100,000 times to compute the probabilities shown in Figure 27.

Figure 28 shows the plot of $(1 - \delta)$ for different values of F_s , where $1 - \delta = Prob[(1 - \epsilon)F_s \leq \widehat{F}_s \leq (1 + \epsilon)F_s]$. Each curve corresponds to a specific level of relative error tolerance, i.e., a specific choice of ϵ , and represents the probability that the estimated value is within this factor of the actual value. For example, the curve for $\epsilon = 0.2$ shows that around 85% of the time the estimate is within 20% of the actual value. Notice how the curves in the figure have an upward trend first and then show a downward trend as the fan-out increases further. This corresponds exactly to the aforementioned “saturation” situation.

5.6.4 Accuracy of the extension to estimate outstanding fan-outs

To evaluate the extension of the advanced scheme to estimate outstanding fan-outs we use the pair of traces, *IPKS+* and *IPKS-*, collected simultaneously on both directions of a link. We extract

¹⁵For more details about this please refer to [125].

all the acknowledgment packets from *IPKS*— to produce the 2D bit array B using the transposed update algorithm (Algorithm 6). The same parameters are configured for both 2D bit arrays A and B . Figure 29 shows the scatter diagram of the fan-out estimated using our proposed scheme (y axis) vs. actual outstanding fan-out (x axis). The fact that most points are concentrated within a narrow band of fixed width along the diagonal line indicates that our estimator is accurate on estimating outstanding fan-outs.

5.7 Conclusion

Efficient and accurate detection of super sources and destinations at high link speeds is an important problem in many network security and measurement applications. In this work we attack the problem with a new insight that sampling and streaming are often suitable for capturing different and complementary regions of the information spectrum, and a close collaboration between them is an excellent way to recover the complete information. This insight leads to two novel methodologies of combining the power of streaming and sampling, namely, “filtering after sampling” and “separation of counting and identity gathering”, upon which our two solutions are built respectively. The first solution improves the estimation accuracy of hash-based flow sampling by allowing for much higher sampling rate, through the use of an embedded data streaming module for filtering/smoothing the bursty incoming traffic. Our second solution combines the power of data streaming in efficiently retaining and estimating fan-out/fan-in associated with a given source/destination, and the power of sampling in generating a list of candidate source/destination identities. Mathematical analysis and trace-driven experiments on real-world Internet traffic show that both solutions allow for accurate detection of super sources and destinations.

CHAPTER VI

FINDING GLOBAL ICEBERGS OVER DISTRIBUTED DATA SETS

6.1 Introduction

Finding items whose frequency of occurrence is greater than a certain threshold is a well-explored problem [24, 54]. Most of the existing work focuses on iceberg queries at a single node [9, 39, 69, 82]. However, in many real-life applications, data sets are physically distributed over a large number of nodes. We study the problem of finding items whose frequencies of occurrences across these nodes add up to exceed a certain threshold, i.e., finding global icebergs over aggregate data.

Identifying global icebergs has applications in wide range of areas ranging from network monitoring to biosurveillance. For example, applications that detect DDoS attacks [81] try to find destination IP addresses (victims of a DDoS attack) that occur frequently in IP traffic aggregated over many network ingress points. Since the attacking packets may come from a large number of hosts (called Zombie hosts) and may traverse many different Internet paths, an individual ISP (Internet service provider) may not see a large number of packets destined to any victim. In other words, such global icebergs (IP addresses that are targets of the DDoS attacks) may not emerge as local icebergs anywhere, and therefore may not be detectable without correlating data from multiple monitoring points. Another network monitoring application [81] is the problem of finding frequently accessed objects/URLs (icebergs) in a Content Delivery Network (CDN) that contains many CDN nodes (e.g., Akamai [1] has thousands of nodes), where multiple nodes can cache and serve each object/URL. To find the globally frequently accessed objects/URLs, we must monitor and correlate the frequency counts on all CDN nodes. Global iceberg query also has applications in system monitoring. It is suggested to us [124] that finding DLLs (Dynamically Linked Libraries) that have been modified on a large number of hosts inside an organization may help detect the spread of an unknown worm or spyware. Recently, the possibility of bioterrorist acts has highlighted the need of biosurveillance, which monitors a large population for changes against a predetermined norm. To achieve this goal, we need effective linking of data from a large number of different sources so that we can catch the

changes that are locally subtle, but globally emerging.

Two naïve approaches might be considered for finding global icebergs. First, we may ship the data from all the nodes to a central server, which takes their aggregation and then applies the algorithms which are designed for detection of icebergs on a single node [9, 39, 69, 82]. This method is clearly inefficient and faced with the “too much data” problem, in the sense that when there are large number of nodes having a large amount of data the communication cost between the server and the nodes is prohibitive.

In the second approach, each node applies local iceberg algorithms on its own data set to find local icebergs, and the central server aggregates the local icebergs to find the global icebergs. This approach reduces communication overhead, but unfortunately may suffer the “too little data” problem. It may miss items which are infrequent in local nodes, but their aggregated frequencies across all nodes exceed the threshold of global icebergs. For instance, suppose that having 10,000 accesses globally per day well qualifies a URL to be an global iceberg. However, these 10,000 accesses may be distributed among many different ISPs so that the qualified URL is not a local iceberg in any ISP. To find the global icebergs of this kind, the aforementioned naïve solution was shown to be inefficient since messages that contain only the local icebergs are lossy. Thus, the core problem of finding global icebergs is to design an appropriate scheme for summarizing local data. In other words, each node should send a small yet informative message to the server so that the server can derive global icebergs in an efficient manner. In this work, we propose two schemes for this purpose.

6.1.1 Problem statement

We formulate the problem as follows. Consider a system or network that consists of N distributed nodes. The data set S_i at node i contains a list of $\langle item, count \rangle$ pairs, corresponding to items and their exact frequency counts in the local dataset. We want to find the set of items whose total counts across all the nodes add up to exceed a certain threshold T .

In addition, instead of making a binary decision of whether an item is an iceberg or not, we want to provide accurate estimates of the total counts for all the potential icebergs. In other words, we want to find item x along with its estimated total count c , such that $c \geq (1 - \epsilon)T$, where ϵ is a small constant (say 0.1). We are thus tackling an estimation problem. While this problem is strictly

harder than the standard binary decision version, we still refer to it as the “global iceberg problem”.

We are interested in solutions for the above problem that require a minimum amount of communication between these nodes and the central monitoring server. In particular, we are interested in one-way¹ (unidirectional) protocols, in which each node needs to send only a significantly reduced amount of data to the central server, and the server does not need to send any message back to the local nodes (except perhaps for TCP ACK’s) and/or request additional data. Conceptually, each message from a node contains a highly space-efficient encoding of the frequency counts of the items at the node. The server will decode messages from all of the nodes to find the global icebergs. The challenge is to find the right encoding and decoding schemes.

Note that we distinguish between two types of global iceberg problems, namely, finding icebergs over *distributed bags* and finding icebergs over *distributed streams* [69]. In both types, we assume that each node will process a stream of data items. The difference is in how the data is processed at each node. In the case of *distributed bags*, each node has enough memory and computation power to “digest” its local stream with zero information loss. In other words, after processing the stream, each node obtains a list of items and their exact frequency counts in the local stream. In the case of iceberg query over *distributed streams*, completely accurate processing is not possible for high-speed data streams because of memory and computation constraints. In this case, we may have to find icebergs over multiple streams where each of these streams is processed with some information loss concerning the frequencies of items. Clearly, it is in general more challenging to find icebergs over distributed streams than over distributed bags. In this work we solve for the bag case², leaving the stream case for future work.

6.1.2 Our solutions and contributions

We propose two solutions to the global iceberg problem: a sampling-based approach and a counting-sketch-based approach. In our sampling-based approach, each node samples a list of data items along with their frequency counts in the local data bag and send them to the server, which for

¹Note that the efficiency and accuracy of the protocol may be improved if multiple back-and-forth data exchanges are allowed (i.e., more than one-way). However, our preliminary study finds that one-way communication can be made very accurate and efficient, thus the benefit of additional interaction is not immediately clear.

²The solution proposed by Manjhi et al. [81] works for the stream case. However, their solution is for the hierarchical communication infrastructure and will work well for neither bag nor stream case on flat infrastructure, which our solutions target. We will discuss this in detail in Section 6.2.

each distinct item, aggregates its sampled frequency counts scaled by the inverse of their respective sampling rates to obtain an estimate of its total frequency. An obvious question is “what is a good sampling strategy?” More specifically, consider that at a node, one item, A , has count 1 and another item, B , has count 100. Should we sample the former with the same probability as the latter? Intuitively, the answer is “no” because the count of B has bigger potential to help push the aggregate count of B over the threshold T than that of A . Therefore we should sample the pair $\langle B, 100 \rangle$ with higher probability than $\langle A, 1 \rangle$, and more generally, the probability with which a node samples an item should be an increasing function g of its frequency count c . But what kind of function should this $g(c)$ be? We answer this question in a comprehensive way. We obtain a near-optimal sampling strategy that results in almost the lowest possible estimation error under certain resource (communication cost) constraints. We also uncover the fundamental mathematical structures behind the global iceberg problem (over flat infrastructure). In addition, we develop a new large deviation theorem that is not only critical for establishing a tight accuracy bounds in both our sampling and sketch based solutions, but may also be useful in other distributed data streaming applications.

Our second solution, the sketch-based approach, detects icebergs in a much more communication-efficient way than the sampling-based solution³. It is inspired by the observation that the sampling process in our sampling-based approach implicitly serves two purposes simultaneously. The first purpose is to obtain the identities of the items that may potentially become icebergs. The second purpose is to use it as a counting device, encoding the frequency counts of items (subject to information loss due to sampling). We observe that although sampling is great for gathering identities of the items, it is not a great counting device. In fact, a counting sketch, such as the one in [77], may be able to encode the frequency counts of various items in a much more resource-efficient way than sampling. Based on this observation, our approach works as follows. Each node will not only summarize its data into a counting sketch, but also sample a small⁴ percentage of identities of the items, and then send both the summary and the sampled identities to the server. For each sampled identity, the server will use it to query all the collected counting sketches and add up the approximate counts

³The presentation of our sampling-based solution, however, is necessary since the near-optimal sampling strategy and the underlying mathematical structure are not only new discoveries but also implicitly used in our sketch-based solution.

⁴Compared to the pure sampling approach, the sampling rate here can be much smaller, since we only need to ensure that each item that has large overall frequency count has *at least one of its occurrences sampled*.

to obtain an estimate of the total frequency count of the item.

One challenge we face in this approach is to find the right counting sketch. As we will discuss later, existing counting sketches such as Space-Code Bloom Filter [77], Spectral Bloom Filter [30] and CM sketch [34] are not well suited for this application. We design a novel counting sketch that is especially accurate and efficient for detecting icebergs, based on insights into the mathematical structure of the global iceberg learned in designing our sampling-based approach.

Through rigorous mathematical analysis, we show that both of our solutions achieve high estimation accuracy with a communication cost at least an order of magnitude smaller than the naive approach of shipping all data to the server. In addition, we evaluated the performance of both solutions on two real-world data sets obtained from Abilene network and IBM. Experimental results confirm the high accuracy and efficiency of both solutions predicted by the analysis.

The rest of this chapter is organized as follows. We start with a discussion of the related work in Section 6.2. Then we describe the design of our sampling-based scheme and develop the theory of good sampling strategies, in Section 6.3. Next we present a more accurate scheme, i.e., the counting-sketch-based scheme and its theoretical analysis in Section 6.4. Section 6.5 evaluates the sketch-based scheme using real-world data sets. We conclude our work in Section 6.6.

6.2 Related Work

The term *iceberg queries* was first introduced to describe database queries that find records whose aggregate values over an attribute are above a certain threshold [54]. Many applications compute a simple type of iceberg query — identifying in a multiset items with frequency above a certain threshold.

Traditional iceberg queries are computed over data bags. For example, a number of heuristics proposed by Fang *et al.* [54] make multiple scans of the data. Recently, finding icebergs over streaming data has attracted much research effort [22, 9, 39, 69, 32, 22, 82]. Among them, Karp *et al.* [69] gave the space and time lower bound for computing exact iceberg queries for both streams and bags. Charikar *et al.* [22] developed a randomized algorithm to identify frequent items in a stream. Cormode *et al.* [32] proposed a method for finding frequent items in a dataset that undergoes deletions and insertions.

All the above work focuses on finding icebergs at a single node, and the central issue is how to design a synopsis data structure called counting sketches to tradeoff accuracy against space and time. One may wonder that since almost all of these counting sketches are linearly composable in the sense that the counting sketch of the overall data set $\bigcup_{i=1}^N S_i$ can be computed as the aggregate of the counting sketches of the individual data sets S_i , $i = 1, 2, \dots, N$, some of them may provide a better solution to the global iceberg problem than our approach. This is not the case for two reasons. The first reason is that these counting sketches typically allocate similar amount of storage/communication resource to an item of large or small count alike, which may not be resource-efficient for the purpose of detecting global icebergs. Our approach, on the contrary, allocates the right amount of storage/communication resource to an item according to its potential contribution to push its total count over the threshold. The second reason is that the accuracy guaranty in the counting sketches is typically in the form of ϵ times the L_1 norm of whole data stream (i.e., the sum of all counts of all items). In our approach, the accuracy guarantee is in the form of ϵ times the total count of an item across all the nodes. In Appendix A.4.6, we compare our approach with the CM sketch [34], a representative counting sketch, with respect to accuracy and communication overhead, and show that our approach requires much less communication overhead than the CM sketch to achieve the same accuracy in detecting global icebergs.

Distributed streaming has recently been studied in the theoretical computer science and database contexts [55, 10, 27]. The iceberg problem is first introduced to the distributed environment in [81]. Manjhi *et al.* [81] proposed a method to find icebergs in the union of multiple distributed data streams. It arranges nodes in a multi-level communication structure. Each node uses traditional synopsis data structures (e.g., [82]) to find “local icebergs,” and then sends the synopsis to its parent in the communication hierarchy. The paper shows that, by giving each node an error tolerance according to its level in the hierarchy, the approximate synopses computed at lower levels can be combined to recover icebergs at a higher level within certain accuracy. This algorithm, however, does not solve our problem of finding icebergs using a one-level (flat) communication infrastructure.

By combining local icebergs to find global icebergs, the above approach essentially assumes that a globally frequent item is also frequent locally somewhere. This assumption works well if the global iceberg threshold is very high relative to the number of distributed nodes. However, when

the number of nodes is very large, a globally frequent item may not be frequent in any local site⁵. For example, if we cut up an iceberg of size 10,000 and distribute it evenly across 10,000 nodes, the algorithm of Manjhi *et al.* [81] will not be able to identify this iceberg unless the leaf nodes send all their data to their parents⁶. In our problem, where a one-level (flat) communication infrastructure is assumed, this is translated into every node sending all its data to the server, which is unacceptable. Therefore, the approach of Manjhi *et al.* [81] does not offer an efficient solution to our problem even for the bag case. A key contribution of our work is a novel counting sketch that is specially optimized for finding global icebergs.

Another problem related to answering iceberg queries is to find the most frequent items in data streams, known as top-K. Techniques for answering top-K queries have been proposed for a single stream as well as a distributed set of data sets (bags) [24, 12, 91, 36]. Babcock *et al.* [12] studied the problem of monitoring top- k items in a distributed environment. The work is extended by Olston *et al.* [91] to support both sum and average queries on data spread over multiple sources. The work is extended also by Das *et al.* [36] to support estimation of set-expression cardinality in a distributed streaming environment. These approaches all try to keep local top-K items at each node in alignment with the global top-K items. In Babcock *et al.*'s approach [12], for instance, each local site is assigned a factor that represents an allowed local skew, and the sum of the local skew is kept within the global error tolerance. As long as the local skew is within the assigned factor, the global top-K will not change and no communication is required between local sites and the central server. In this sense, the techniques used in the above approaches for solving the top-K problem are similar to the techniques used by Manjhi *et al.* [81] for the iceberg problem. Techniques for solving top-K queries generally cannot be directly used for solving iceberg queries since the size of an iceberg can be much smaller than top-K items, making it much harder to find.

6.3 The Sampling-based Scheme

An obvious approach to find global icebergs over distributed bags is sampling. In this approach, each item-count pair $\langle I, c \rangle$ at a node is sampled with some probability and the list of sampled pairs

⁵Many real-life applications fall into this category.

⁶This may be acceptable in Manjhi *et al.* [81] due to the distributed nature of the hierarchical communication infrastructure and the fact that much less data need to be transferred when the level becomes higher.

are sent to the server; after collecting all the samples from all nodes, the server estimates the total count of each item that appears at least once in the aggregate list by aggregating the counts in the samples and compensating for sampling. We state earlier that the probability with which a node samples an item in general should be an increasing function g of its frequency count c . But what kind of function should this $g(c)$ be?

To answer this question, we need to come up with a mathematically sound measure to compare different sampling strategies, determined by their choices of the sampling function g , and then we pick the one that optimizes this measure. We show that one such measure is the *worst-case* mean square error (MSE). Let \hat{X} be the estimator for a random variable X . Then its MSE is defined as $E[(\hat{X} - X)^2]$. The notion of minimizing the worst-case MSE can be explained by the following adversarial minimax model. Imagine that an adversary tries to split an iceberg into small pieces and hide it among many nodes. Given our choice of g and the corresponding estimator, the adversary would choose the split pattern that maximizes the MSE of our estimator. The only thing we can do is to minimize this maximum (worst-case) MSE, since in general neither the nodes nor the server have a control over the split pattern.

In the following, we will describe our sampling process (Section 6.3.1), characterize the type of g we will use that attempts to minimize the aforementioned worst-case MSE (Section 6.3.2), and then establish tight tail bounds of the estimation based on an improved large deviation theorem (Section 6.3.3) developed by us.

6.3.1 Sampling process and the estimator

Recall that there are total N nodes in the system. Suppose that the frequency count of an item I at a node i is $c_i \geq 0$, $i = 1, \dots, N$. Our goal is to estimate $f_I = \sum_{i=1}^N c_i$. Due to sampling only a part of these (c_i) 's may appear in the aggregate list at the server. Suppose that the server received k of them, denoted as x_1, \dots, x_k , from nodes j_1, \dots, j_k respectively. In other words, $x_i = c_{j_i}$ for $1 \leq i \leq k$.

Since each count c is independently selected with probability $g(c)$, for each occurrence of c , on the average $\frac{1}{g(c)} - 1$ other nodes have $\langle I, c \rangle$ but did not sample it; that is, each occurrence of c can

be thought of as representing $\frac{1}{g(c)}$ occurrences of c . Thus, we estimate the total count f_I of I as

$$\hat{f}_I = \frac{x_1}{g(x_1)} + \frac{x_2}{g(x_2)} + \cdots + \frac{x_k}{g(x_k)} \quad (30)$$

The following theorem shows that this estimator is unbiased and establishes its MSE.

Theorem 6 *The estimator \hat{f}_I is unbiased, and its MSE is $\sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)}$.*

Proof: Consider the random variables y_i , $1 \leq i \leq N$:

$$y_i = \begin{cases} c_i & \text{with probability } g(c_i), \\ 0 & \text{with probability } 1 - g(c_i). \end{cases}$$

Then the estimator in (30) is rewritten as $\sum_{i=1}^N \frac{y_i}{g(c_i)}$, by treating $\frac{0}{0}$ as 0 (for (c_i) 's that is equal to 0). For all i , $1 \leq i \leq N$, $E[y_i] = c_i g(c_i)$ so $E[\frac{y_i}{g(c_i)}] = \frac{E[y_i]}{g(c_i)} = c_i$. This means that the estimator is unbiased, and therefore, the MSE of f_I is equal to its variance: $\text{Var}[f_I] = \text{Var}[\sum_{i=1}^N \frac{y_i}{g(c_i)}] = \sum_{i=1}^N \frac{\text{Var}[y_i]}{g(c_i)^2} = \sum_{i=1}^N \frac{c_i^2 g(c_i)(1-g(c_i))}{g(c_i)^2} = \sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)}$.

6.3.2 The worst-case MSE and the near-optimal sampling strategy

We obtain from Theorem 6 that the MSE of f_I is $\sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)}$. Its value clearly depends on how the quantity f_I is split into c_1, \dots, c_N . As mentioned above, for each fixed function g , we think of the worst-case split of f_I by an adversary, which is the one that maximizes the MSE with respect to g . We would like to choose g that minimizes this worst-case MSE. Note that we can always reduce the worst-case MSE by increasing the sampling rates (i.e., making $g(c)$ larger for all c values), albeit at higher communication cost. Therefore, g should be constrained to a family \mathcal{G} that has the same communication cost (elaborated next). In other words, we wish to solve the optimization problem:

$$\min_{g \in \mathcal{G}} \max_{c_1, \dots, c_N: c_1 + \dots + c_N = f_I} \sum_{i=1}^N \frac{c_i^2(1-g(c_i))}{g(c_i)} \quad (31)$$

The minimax problem in (31) is in fact not as well-formed as it looks, as most of the complications are hidden in the term “ $g \in \mathcal{G}$ ”. We intend to choose g from a family \mathcal{G} that has the same communication cost. However, it is very hard (if not impossible) to come up with a right measure of the communication cost that both is mathematically sound and allows the above minimax problem to be solved exactly. An obvious and mathematically sound measure is the expected total size of

the sampled items. For example, suppose there are n items of sizes s_1, s_2, \dots, s_n in a bag. Then the average communication cost under this measure is $\sum_{i=1}^n g(s_i)$. However, this measure does not lead to a clean solution since the communication cost is a function of the sizes of all the items at all the nodes, which should not be a part of the equation since it is not known when a node applies sampling. Therefore, we would like the communication cost to be defined as a function of just g and the split pattern. However, the problem with definitions like this is that they are often at odds with our intuitions of “communication cost”, making the resulting mathematical solution (if any) practically unappealing (no matter how clean it is).

Our approach is to stick with the above intuitive definition of the communication cost, but to solve for a fairly good solution instead of the optimal one. Our idea is to design our sampling strategy such that the MSE is independent of the split pattern, i.e., each split pattern is equally bad (or equally good) as any other split patterns. In this case, every case is equivalent to the worst-case. This completely eliminates the power of the adversary to increase the MSE by manipulating the split pattern, although there is some room for it to impose a slightly larger communication cost. We assume that the adversary only has control over the split patterns of its own few items. Therefore, the increase of the overall communication cost is usually negligible. The remaining question is how this independence between the MSE and split pattern can be enforced. We have the following exact answer to this question.

Proposition 1 *The independence condition is satisfied if and only if $g(c) = \frac{c}{d+c}$ for some constant $d \geq 0$.*

Proof: Let us first solve for the “if” part. For each i , $1 \leq i \leq N$, $\frac{c_i^2(1-g(c_i))}{g(c_i)}$ is equal to dc_i , then the MSE is $d \sum_{i=1}^N c_i = df_I$ since $f_I = c_1 + \dots + c_N$, and the condition is satisfied. For the “only if” part, let us first consider the split pattern in which all the “mass” f_I is concentrated on one node. The MSE of our estimator with this split pattern is $F(f_I)$, where $F(c)$ denotes the function $\frac{c^2(1-g(c))}{g(c)}$. Then we consider another split pattern in which the mass is evenly distributed over f_I different nodes. The MSE in this case is $f_I F(1)$. According to the independence condition, we have $F(f_I) = f_I F(1)$. Since this relation holds for arbitrary f_I values, it can be shown that the function $F(x)$ has the form dx for any positive integer value x , where $d = F(1)$. By resolving

$$\frac{c_i^2(1-g(c_i))}{g(c_i)} = dc_i, \text{ we obtain } g(c_i) = \frac{c_i}{d+c_i}.$$

6.3.3 Tight tail bounds through a new Large deviation theorem

So far we know the variance of our estimator, i.e., $\text{Var}[\hat{f}_I]$ is equal to df_I no matter how f_I is split. We now wish to obtain a tight tail bound on the estimator; that is, a tight upper bound on $\Pr[|\hat{f}_I - f_I| > b]$, the probability that the estimation error has deviation larger than b . Since we know the MSE of \hat{f}_I , a standard tail bound can be obtained using Chebyshev's inequality, but it offers only a loose bound since it does not take into consideration that \hat{f}_I is the sum of independent random variables $y_1/g(c_1), \dots, y_N/g(c_N)$ (defined in the proof of Theorem 6). A much tighter bound can be obtained using Chernoff–Hoeffding's inequality [28], but it is not ideal since it does not take advantage of the fact that the estimator has small variance.

We establish the following tail bound theorem that takes advantage of both the independence and the small variance. This is a one-sided bound and a similar two-sided bound can be easily obtained (with the RHS doubled). This theorem improves Theorem A.1.19 in [7, pp.270] by sharpening ϵ to $\frac{\epsilon}{3}$ on the RHS of the inequality. We are able to obtain tail bounds of $\hat{f}_I - f_I$ from this Theorem that are much tighter than obtainable from Chernoff–Hoeffding's inequality [28]. Its proof can be found in Appendix A.4.1. This theorem will also be used to establish tail bounds in our counting sketch based scheme.

Theorem 7 *For every $\theta > 0$ and $\epsilon > 0$, the following holds: Let $W_i, 1 \leq i \leq m$, m arbitrary, be independent random variables with $E[W_i] = 0$, $|W_i| \leq \theta$ and $\text{Var}[W_i] = \sigma_i^2$. Let $W = \sum_{i=1}^m W_i$ and $\sigma^2 = \sum_{i=1}^m \sigma_i^2$ so that $\text{Var}[W] = \sigma^2$. Let $\delta = \ln(1 + \epsilon)/\theta$. Then for $0 < a \leq \delta\sigma$,*

$$\Pr[W > a\sigma] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$$

We now map this large deviation theorem to our scheme. The error of our estimator (a random variable) corresponds to $W (= \hat{f}_I - f_I)$. The error in our estimation of each c_i ($y_i/g(c_i)$ where y_i is defined in the proof of Theorem 6) corresponds to W_i , i.e., $W_i = y_i/g(c_i) - c_i$ for all c_i . The MSE/variance of our (unbiased) estimator corresponds to σ^2 in Theorem 7. Theorem 7 essentially states that the probability that the estimation error W is larger than a times standard deviation is no more than $e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$. However, this is true only for $a \leq \delta\sigma$, and δ, ϵ and θ are related by

$\delta = \ln(1 + \epsilon)/\theta$. Therefore, we need to bound θ , which is the upper bound for $|W_i|$, $i = 1, 2, \dots, n$, since otherwise either ϵ has to be large or a has to be very small, and both cases lead to trivial or loose tail bounds. This is achieved by setting a cutoff point for sampling as follows.

We know that for all⁷ c_i equal to 0, $W_i = 0 - 0 = 0$, and for all $c_i > 0$, $W_i = c_i / \frac{c_i}{c_i + d} - c_i = d$ if c_i is sampled, and $W_i = 0 - c_i = -c_i$ otherwise. Therefore, $|W_i| \leq \max\{d, c_i\}$. So if we do not set a cutoff point, θ has to be larger than or equal to $\max\{d, \max\{c_1, c_2, \dots, c_N\}\}$ which may be a very large value considering some big value of c_i and hence lead to loose tail bounds. Instead we set a cutoff point M in the sense that for any item-count pair $\langle I, c \rangle$, when $c > M$, we sample it with probability 1 instead of $g(c)$. These pairs would not introduce any error to our estimation. So we only need to consider the left item-count pairs. In this case θ can be set to $\max\{d, M\}$. The remaining question is what the value of M is. We set it to d . The reason is that if we set M less than d , there is no improvement on θ (and hence the bound) because $\theta \geq d$, but if we set M larger than d , θ becomes larger than d , resulting in a worse bound. Notice that an item-count pair whose count is greater than d will have at least $1/2$ chance of being sampled even if there is no cutoff point. Therefore, use of cutoff point d will increase the average communication cost of some item-count pairs by at most a factor of two. This should not impact the communication bandwidth too much considering the small number of items whose (local) frequencies are larger than d in real data.

6.3.4 Numerical results

In this section, we illustrate the worst-case MSE and tail bound of our estimator using a numerical example. We assume that our system consists of $N = 10,000$ nodes. We compute the numerical results for $f_I = 10,000$ and $f_I = 5,000$. We want to send no more than a small fraction (say 2%) of the data to the server. Recall that an item with count c is sampled with probability $\frac{c}{c+d}$. We set $d = 49$, which translates into a 2% sampling rate for items with count value 1. By Theorem 6, the standard deviation of the estimator is 495 and 700 in the cases of $f_I = 5,000$ and $f_I = 10,000$ respectively. By Theorem 7, we establish that $\Pr[4,030 \leq \hat{f}_I \leq 5,970] \geq 76\%$ when $f_I = 5,000$. In other words, the probability that the error (when $f_I = 5,000$) is within 20% is about 0.76. When $f_I = 10,000$, we get $\Pr[8,647 \leq \hat{f}_I \leq 11,353] \geq 76\%$. In other words, the probability

⁷Here we treat $\frac{0}{0}$ as 0 by convention.

that the error (when $f_I = 10,000$) is within 13% is about 0.76. It shows that the value of d , which determines the sampling rate, crucially affects the tradeoff between communication cost and accuracy.

6.4 Counting-sketch-based Scheme

We observe that the sampling-based method is not very accurate because the sampling rate is typically low due to limited communication bandwidth. More importantly, sampling is not efficient for counting. For example, suppose the adversary cuts up an iceberg of size 10,000 into 10,000 1's and distributes them evenly among 10,000 nodes. With a sampling rate of 10%, on the average 1,000 nodes will send in the identity and the frequency (which is 1) of this item to the server. The identity of the item, which can be hundreds of bits long in many applications, will be repeated for approximately 1,000 times. Such repetitions result in poor use of communication bandwidth.

These observations lead us to consider the use of counting sketches. A carefully designed counting sketch such as those proposed in [30, 77] is likely to offer much higher accuracy in estimating counts, but it alone cannot complete the task of finding icebergs. This is because a counting sketch is essentially a query interface, that is, to inquire about the count of an item its identity has to be known first. This means that the server would require a separate list of item identities L for which it will inquire about their global counts in the counting sketches. Our scheme makes each node sample some candidate identities and send them to the server, which the server will merge into L . Note that the sampling rate here for generating candidate identities can be significantly smaller than that of the sampling-based approach, for we only have to ensure that each item that has a large global count is *sampled at least once*. In contrast, in the sampling-based scheme we need a relatively large percentage of the item identities sampled to achieve good accuracy in counting.

In this section we design a novel counting sketch that is particularly efficient and accurate for finding global icebergs. We optimize its parameters based on the principle of minimizing worst-case MSE (described earlier in the sampling-based approach). We show that with the same communication overhead as the sampling-based scheme, this scheme has much smaller variance and produces much more accurate estimates. Before designing our own counting sketch, we investigated the possibility of using existing ones such as Spectral Bloom Filter (SBF) [30] and the Space-Code Bloom

Filter (SCBF) [77]. We found that SCBF is not very efficient in encoding a bag since it introduces unnecessary noises through a random “balls into bins” process that is necessary for encoding a high-speed stream. SBF is not suitable for our purpose either since neither it is optimized for finding icebergs nor does it afford us a tight tail bound on the estimation error.

In the following, we first present the scheme for a special case in which the count for every item at each node is either 0 or 1 (Section 6.4.1). This scheme is relatively easy to analyze and understand. We then extend this approach to a more complicated general case (Section 6.4.2).

6.4.1 0/1 case

This is the case in which the frequency count for every item at each node is either 0 or 1. In this case the data at each node can be viewed as a set of items, and thus, the iceberg problem can be viewed as the problem of finding all the items appearing in more than T nodes. This special case occurs in real applications, such as the spyware detection problem discussed in Section 6.1.

The counting sketch we use to solve the special case is simply Bloom filters [17] combined with sampling. The Bloom filter is an ideal synopsis data structure for this 0/1 case since it is known to provide near-perfect lossy encoding of sets (see [74]). In our scheme, each node i samples its items with probability p and encodes the sampled items into a Bloom filter, denoted as B_i . As mentioned before, each node also samples a list of items A_i uniformly with probability p' , where p' is much smaller than p . After collecting (A_i, B_i) from all the nodes $i, i = 1, 2, \dots, N$, for each $I \in \cup_{i=1}^N A_i$, the server queries each Bloom filter whether it contains I and then calculates the total number of occurrences of I by scaling the total number of positive answers by $1/p$ and discounting the number of false positives that these filters might have provided.

We impose a natural homogeneous resource constraint on our system: the average amount of communication cost from each node i to the server, amortized over the number of items at node i , is approximately a constant D . We will show that this approach achieves a much smaller worst-case MSE and has much tighter tail bounds than the pure sampling-based approach when the same communication constraint is imposed. In the following, we first present a quick overview of the basic facts of the Bloom filter relevant to our encoding algorithm (Section 6.4.1.1). Then we present our unbiased estimator and derive its MSE (Section 6.4.1.2). Finally, we obtain a method for minimizing

the MSE (Section 6.4.1.3).

6.4.1.1 Bloom filter basics

A Bloom filter is an approximate representation of a set S , which given an arbitrary element x , allows for the membership query “ $x \in S$?”. A Bloom filter employs an array B of m bits, initialized to all 0’s, and k independent hash functions h_1, h_2, \dots, h_k with range $\{1, \dots, m\}$. During *insertion*, given an element x to be inserted into a set S , the bits $B[h_i(x)]$, $i = 1, 2, \dots, k$, are set to 1. To *query* for an element y , i.e., to check if y is in S , we check the values of the bits $B[h_i(y)]$, $i = 1, 2, \dots, k$. The answer to the query is *yes* if all these bits are 1, and *no* otherwise.

A Bloom filter guarantees not to have any false negative, i.e., returning “no” even though the set actually contains the item. However, it may contain false positives, i.e., returning “yes” while the item is not in the set. Let t be the number of items stored in the filter. Then the probability that the filter returns a false positive to the query about an item that is not stored is approximately

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kt}\right)^k \approx \left(1 - e^{-\frac{kt}{m}}\right)^k$$

(see [53]). The RHS is minimized to 2^{-k} when⁸ k is set to $m \ln 2/t$. We refer to the amount of storage consumed by each element, namely m/t , denoted as C . Then the optimal k value is equal to $C \ln 2$.

Our Bloom filter configuration for each node is homogeneous in the sense that they will sample their items with the same probability p , consume the same amount of storage per sampled item (translating into communication cost per item) C , and use the same (optimal) number of hash functions $k = C \ln 2$. This is consistent with our intention to impose the same constraint of communication cost per item D on each node. It is not hard to see that $D = C \times p$. Therefore, given D , there is a clear tradeoff between C , bits per sampled item, and p , the sampling rate. We will show in Section 6.4.1.3 how to configure C and p so that our estimator, to be discussed next, has the minimum MSE.

⁸In this case, the Bloom filter contains approximately half 1’s and half 0’s.

6.4.1.2 The estimator

After receiving the Bloom filter B_i and the sampled identities A_i from each node i , the server estimates the global count f_I of an item $I \in \cup_{i=1}^N A_i$ as follows: let α be the total number of Bloom filters that assert to contain I and let p be the sampling rate for generating (B_i) 's. Let $q = 2^{-k}$ be the false positive rate of an individual Bloom filter. Since the sampling rate is p , out of the f_I occurrences pf_I are expected to be in the Bloom filters. All such Bloom filters should return a positive answer. The remaining $N - pf_I$ Bloom filters may return a false positive with false positive rate q . So, the expect number of positive answers is $pf_I + (N - pf_I)q$. Since α positive answers are returned, by solving $\alpha = pf_I + (N - pf_I)q$ for f_I , we obtain an estimator \hat{f}_I of f_I as

$$\hat{f}_I = \frac{\alpha - Nq}{p(1 - q)} \quad (32)$$

The following theorem shows that \hat{f}_I is unbiased, i.e., $E[\hat{f}_I] = f_I$. Its proof can be found in Appendix A.4.2.

Theorem 8 (i) The estimator \hat{f}_I is unbiased. (ii) $MSE[f_I] = \text{Var}[\hat{f}_I] = \frac{f_I(1-2q-p+pq)}{p(1-q)} + \frac{qN}{p^2(1-q)}$.

6.4.1.3 Configuring for optimal performance and tail bound analysis

Recall when the communication cost per item is set to D , we need to configure parameters p and $C = D/p$ so that the MSE of our estimator is minimized. The interplay between p and C can be illustrated as follows. Larger C means that we devote more bits to each sampled item use a larger number of hash functions for encoding an item (recall that $k = C \ln 2$), which will reduce the false positive rate q (recall that $q = 2^{-k}$). However, this implies smaller sampling rate p , which will increase the estimation error when sampling is compensated through scaling. Intuitively there is a sweet spot between two extremes. In this section, we develop the theory for finding this optimal spot.

Let $F(p)$ denote the MSE of our estimator \hat{f}_I as a function of p . The following theorem, the proof of which is quite involved and can be found in Appendix A.4.3, shows how to minimize $F(p)$.

Theorem 9 (i) The first derivative of $F(p)$ has at most two roots in $[0, 1]$. (ii) The optimal p is attained either at the smaller (or unique) root, or at $p = 1$.

Like in the 0/1 case, a tight tail bound can be obtained by using Theorem 7. The parameter mapping is as follows. Again let S be the set of nodes at which I appears. For each $i \in S$, we set W_i in Theorem 7 to $\frac{Y_i - q}{p - pq} - 1$. Here Y_i takes value 1 if the Bloom filter B_i gives the positive answer regarding I , and takes value 0 otherwise. For each $i \notin S$, we set W_i to $\frac{Z_i - q}{p - pq}$. Here Z_i takes value 1 if B_i gives the positive answer regarding I , and takes value 0 otherwise. Thus, $|\frac{Y_i - q}{p - pq} - 1| \leq \frac{1 - q}{p - pq} - 1 = \frac{1}{p} - 1$ and $\frac{Z_i - q}{p - pq} \leq \frac{1 - q}{p - pq} = \frac{1}{p}$. Since $\frac{1}{p} > \frac{1}{p} - 1$ we set θ to $\frac{1}{p}$ in Theorem 7.

6.4.1.4 Numerical results

Here we use an example to illustrate how much better this counting-sketch-based works than the pure sampling scheme. In this example, the length of an item identifier is assumed to be 100 bits (e.g., network flow label or URL), and there are $N = 10,000$ different nodes (same as the example in Section 6.3.3). We assume that the amount of bits sent to the server by any node can only be 0.02 of the raw data size at the node (i.e., 2 bits per item after compression on the average or $D = 2$ bits). Suppose there is one item I that occurs at exactly 5,000 nodes.

In the pure sampling scheme, each node samples 2.0% of the item (i.e., $d = 49$). In the sketch-based scheme, the sampling part samples 0.2% of the item identities, which is equivalent to the cost of 0.2 bits per item (each identity is 100 bits long). The Bloom filter part will cost 1.8 bits per item (recall that the total has to be 2). Compared to the sampling-based scheme the estimator here has much tighter tail bounds and smaller variance. For example, we have $\Pr[4887 \leq \hat{f}_I \leq 5,113] \geq 76\%$. In other words, with probability 0.76 the relative error is no more than 2.26%. In comparison, the sampling-based scheme can only guarantee, with probability 0.76 the relative error is no more than 20%. In addition, the standard deviation of the estimator here is only 60.3, about an order of magnitude smaller than in the sampling-based scheme (about 495).

6.4.2 The general case

In the general case, the frequency count of an item could be any positive integer. Our approach is to extend the sketch-based method in Section 6.4.1 to obtain a solution to the general case. A naïve solution here will be to divide the items into those having the same count and conquer each group using the scheme for the 0/1 case (the Bloom filter). It is not hard to see that this solution can be configured to generate an unbiased estimate as before. Unfortunately, preparing a Bloom filter for

every possible count may not be very economical since more Bloom filters may result in more false positives and hence higher MSE.

We propose to resolve this by rounding each count to a point in a short series of quantization points $\{b_j\}_{j=1}^n$, $b_1 < \dots < b_n$. Then, at each quantization point, we build a Bloom filter to encode these (rounded) counts. Although this rounding (quantization) introduces an error, it helps to reduce the error caused by the false positives of the Bloom filters. The MSE of our estimator is in fact the summation of these two types of errors. In this section, we analyze the interplay between these two types of errors to arrive at a near-optimal tradeoff.

In the 0/1 case, there is only one possible split pattern: split an aggregate count of x into x 1's. In the general case, however, there are many possible split patterns, since the count at each node may be greater than 1. Just like in our sampling solution, we need to consider the existence of an adversary that attempts to maximize the MSE by splitting the overall count. Our solution should be able to minimize the maximum (worst-case) MSE by carefully choosing the quantization points. Again, we resolve this by making all the split patterns equally bad for the adversary, in terms of having the same small overall MSE. The solution, however, is much more sophisticated than in the sampling case.

In the following we will provide the details of the scheme. We begin with a description of the encoding algorithm (Section 6.4.2.1). We then describe the decoding algorithm at the server, analyze its accuracy rigorously (Section 6.4.2.2), and show how to configure the system to minimize the worst-case MSE bound (Section 6.4.2.3). Finally we describe how the sampling module works to obtain the candidate item identities (Section 6.4.2.4).

6.4.2.1 The encoding algorithm

Suppose we are given a series of $n + 1$ “quantization points” $\{b_j\}_{j=0}^n$, $0 = b_0 < b_1 < b_2 < \dots < b_n = M$, chosen from the range of the global frequency counts⁹. These points are in fact algorithmically determined for the purpose of minimizing the worst-case MSE, but we will postpone the discussion of the algorithm (and the definition of M) until Section 6.4.2.3. A count of value v between 1 and b_n that is not equal to any quantization point, is probabilistically rounded to one of

⁹The point b_0 is a virtual point to simplify our description.

the quantization points as follows. Suppose $b_{j-1} \leq v \leq b_j, j \geq 1$. Then v is rounded to b_{j-1} with probability $\frac{b_j - v}{b_j - b_{j-1}}$ and to b_j with probability $\frac{v - b_{j-1}}{b_j - b_{j-1}}$. For example, if count 4 sits between two neighboring quantization points 3 and 7, then it is assigned to 3 with probability $\frac{3}{4}$ and to 7 with probability $\frac{1}{4}$. The rationale for adopting this quantization method is that an unbiased estimation of the original count can still be made after the quantization. If the count c of an identity I is larger than b_n , the largest value of the quantization points, the pair $\langle I, c \rangle$ will be sent to the server as it is (without any lossy encoding).

At each node i , for each $j, 1 \leq j \leq n$, a Bloom filter $B_i^{(j)}$ is constructed to encode the items that are rounded to the quantization point b_j . Like in the 0/1 case, items rounded to b_j may first be subject to sampling and only the sampled items are encoded into the Bloom filter $B_i^{(j)}$. The question now is “What should be the resource consumption (in bits) of each sampled item and what should be the sampling rate in constructing $B_i^{(j)}$?”. To answer this question, we need to first decide “What should be the resource consumption of each item on the average, denoted as D_{b_j} , in constructing $B_i^{(j)}$?”. We refer to the mathematical rationale behind this decision (for $B_i^{(j)}, j = 1, 2, \dots, n$) as our resource consumption model. We need to establish this model for the general case, because unlike in the 0/1 case, item counts after quantization could take n possible positive values $b_1 < b_2 < \dots < b_n$, and clearly an item of larger count deserves more generous resource consumption. In other words, the larger the b_j value is, the larger the D_{b_j} value should be. But how should D_{b_j} grow as a function of b_j ?

Interestingly, the answer to this second question is intertwined (but without “chicken and egg problem”) with the answer to the first question. The intuition of our resource consumption model is again to make each split pattern result in the same overall MSE (the independence principle) like in the sampling-based scheme. In other words, we set D_{b_j} to such a value that the MSE in estimating a fragment of size b_j from the Bloom filter $B_i^{(j)}$ is exactly $d * b_j$ (i.e., proportional to b_j), where d is a constant. This MSE in turn is computed from the optimal tradeoff between the resource consumption of a sampled item, denoted as C_j , and the sampling probability, denoted as p_j , under the constraint $C_j * p_j = D_{b_j}$. As a consequence, the optimal sampling rate p_j may be different at different (possible) quantization values b_j 's. The technique in determining this optimal tradeoff is similar to that we described in Section 6.4.1.3. Since we do not know the values of b_j at this point

yet, when we compute this resource consumption model, we assume trivial quantization, i.e., every count value is a quantization point. In other words, we compute D_c for every possible quantization value c based on the above independence principle and obtain the aforementioned optimal tradeoff point on this quantization value c as a byproduct of this computation. These computation results will serve as constant parameters (hence no “chicken and egg” problem) to our algorithm in determining the optimal quantization points b_1, b_2, \dots, b_n , shown later in Algorithm 7.

6.4.2.2 The decoding algorithm

The central server will collect from each node i the n Bloom filters $\{B_i^{(j)}\}_{j=1}^n$ and the set A_i of sampled item identities (produced by a separate sampling module to be described later). As in the 0/1 case, each item in $\cup_{i=1}^N A_i$ is used to query the Bloom filters and the results are used to estimate the global count of the item. Let I be an item belonging to $\cup_{i=1}^N A_i$. For each j , $1 \leq j \leq n$, let g_j be the random variable that takes as value the number of nodes at which the count for I (its global count denoted as f_I) is rounded to the quantization point b_j . We are able to obtain an unbiased estimator \hat{g}_j of g_j derived from Equation (32) in the 0/1 case, when we consider the Bloom filters at the quantization point b_j across all the nodes. These (g_j) 's, viewed as a random vector, is in fact a (random) function of the split pattern of f_I . We obtain an estimator of f_I as a functions of (g_j) 's:

$$\hat{f}_I = \sum_{j=1}^n b_j \hat{g}_j. \quad (33)$$

This is an unbiased estimator. The proof can be found in Appendix A.4.4.

Theorem 10 $\hat{f}_I = \sum_{j=1}^n b_j \hat{g}_j$ is unbiased.

6.4.2.3 Configuring parameters for minimizing worst-case MSE

The following theorem establishes a tight bound of the MSE of our estimator as a function of the quantization points and the split pattern of f_I . Its proof can be found in Appendix A.4.5.

Theorem 11 $\text{Var}[\hat{f}_I] \leq \sum_{j=1}^n F_j(e_j)$, where for each j , $1 \leq j \leq n$, $F_j(e_j)$ is a linear function of

the form $a_j e_j + d_j$, such that

$$\begin{aligned} e_j &= b_j \mathbb{E}[g_j] \\ a_j &= \frac{b_j(1 - 2q_j - p_j + p_j q_j)}{p_j(1 - q_j)} \\ &\quad + \frac{\max\{(b_j - b_{j-1})^2, (b_{j+1} - b_j)^2\}}{4b_j} \\ d_j &= \frac{q_j N b_j^2}{p_j^2(1 - q_j)} \end{aligned}$$

where b_{n+1} is set to b_n by convention (for computing a_n).

Now based on Theorem 11, we tune the quantization points to achieve the near-optimal accuracy. Note that e_1, e_2, \dots, e_n can be viewed as a “quantized split pattern”, since they correspond to the average number of fragments that are quantized to b_1, b_2, \dots , and b_n by rounding, respectively, and $\sum_{j=1}^n e_j = f_I$. The variance of our estimator, as we show in Theorem 11, is bounded by $\sum_{j=1}^n (a_j e_j + d_j)$. Again like in Section 6.3.2, we imagine that an adversary will try to split f_I into $(e_i)'s$ such a way that maximizes this MSE. Note that for any such split, the values $(a_j)'s$ and $(d_j)'s$ remain constant. Suppose the $(a_j)'s$ are ordered as $a_{\pi(1)} \geq a_{\pi(2)} \geq \dots \geq a_{\pi(n)}$, where π is a permutation defined over $\{1, 2, \dots, n\}$. Then the best strategy for the adversary is to manipulate the split pattern so that $e_{\pi(1)}$ is first made as large as possible, and then $e_{\pi(2)}, e_{\pi(3)}$ and so on, until the total f_I has been reached (i.e., the greedy strategy). We would like to configure our parameters to make this worst MSE as small as possible.

Our strategy, like in the sampling case, is to first make the constant factor a_i on each e_i piece approximately equal to a . Then no matter how the adversary splits f_I , the MSE of our estimation is approximately equal to $a f_I + \sum_{j=1}^n d_j$, a linear function of f_I . There is a fundamental tradeoff between a , the slope of the function, and $\sum_{j=1}^n d_j$, the intercept of the function. This tradeoff is in fact the result of the interplay between the quantization error, and the errors caused by Bloom filter false positives and sampling. When the distance between neighboring quantization point pairs decreases, the slope of the function, which corresponds to the quantization error, also decreases, but the intercept, which corresponds to the errors caused by Bloom filter false positives and sampling, increases since more quantization points result in more false positives. In the following, we develop an algorithm to further tune the quantization points to find the best tradeoff point between a and

Algorithm 7: Algorithm for Computing Quantization Points.

```
1 Input:  $N, M, T$ 
2 Output:  $b_j, j = 1, 2, \dots$ 
3 Optimize:
4    $b_0 \leftarrow 0; b_1 \leftarrow 1$ 
5   for each  $k, 2 \leq k \leq M$ 
6      $b_2 \leftarrow k$ 
7     Compute  $a_1$  and  $d_1$ 
8     /*  $a_j$  and  $d_j, j = 1, 2, \dots$ , are computed by Theorem 11 */
9      $j \leftarrow 3$ 
10     $l \leftarrow b_2$ 
11    while  $l < M$ 
12       $b_j \leftarrow \operatorname{argmin}_{l < b_j \leq M} |a_1 - a_{j-1}|$ 
13       $l \leftarrow b_j$ 
14       $j \leftarrow j + 1$ 
15     $v_i \leftarrow \max\{a_1, a_2, \dots\} \times T + \sum_j d_j$ 
16    Find  $i$  whose  $v_i$  is the smallest of all
17    Output the  $(b_j)$ 's with respect to  $i$  as the quantization points
```

(d_j) 's so that $af_I + \sum_{j=1}^n d_j$ is minimized (while keeping (a_i) 's close to each other).

The pseudo-code of the algorithm is shown above (Algorithm 7). The intuition behind this algorithm is as follows. It takes as input three parameters, namely, N , the number of nodes, T , the iceberg threshold, and M , the cutoff point such that all items with counts larger than the point will be reported to the server (together with their counts) in the raw form (without Bloom filter encoding). Recall that our goal is to make the slope on each quantization point approximately same. The value a_1 will be determined by the choice of b_0 (fixed to 0), b_1 (fixed to 1), and b_2 . So, as soon as b_2 is fixed to some value ≥ 2 , the value of a_1 is set. The next quantization point b_3 is chosen so that a_2 is close to a_1 . We repeat this to find b_4, b_5, \dots until we reach the cutoff point M . These particular choices were obtained by fixing b_2 . So, by changing the value of b_2 a difference series of quantization points can be obtained. We try each possible value (from 2 to M) for b_2 and compute the series. We choose the value for b_2 that minimizes the linear function $af_I + \sum_{j=1}^n d_j$ (computed according to Theorem 11). The complexity of the above algorithm is $O(M^2 \log M)$ (there are two nested loops and the computation of argmin uses binary search algorithm.). Notice that this algorithm needs to be executed only once during the system configuration. Since M is usually no more than a few hundred in most of the targeted applications, the actual running time of

this algorithm is very short. We will show two examples of spectrum of the quantization points in Section 6.5 generated by this algorithm using real-world data sets.

6.4.2.4 Sampling module

Recall that we must define a method for sampling item identities to generate lists A_1, \dots, A_N . As in our sampling-based scheme we use a size-dependent sampling technique so as to give a high overall sampling rate to an item with a higher total count. To do so, we propose to sample an item with count v at a node with probability $1 - e^{-\gamma v}$ for a constant γ . If the counts of an item I in the nodes are x_1, \dots, x_N , then the probability that it is not sampled at any node is $\prod_{i=1}^N e^{-\gamma x_i} = e^{-\gamma f_I}$. So, the probability it is sampled at at least one node is $1 - e^{-\gamma f_I}$. We can tune the parameter γ so as to achieve a certain success probability. For example, suppose there is a distributed iceberg whose global count is 10,000. If we set γ to 0.002, the probability its identity is not sampled at any node is only $e^{-20} = 2.06 \times 10^{-9}$. Finally, note that in the 0/1 case the value of v is either 0 or 1 this sampling scheme becomes uniform sampling.

6.5 Evaluation

In this section we evaluate our proposed counting-sketch-based scheme using real-world data sets. We do not evaluate the sampling approach since it has been shown to be less accurate through analysis earlier.

6.5.1 Data sets

Our motivating applications include detecting DDoS attacks and monitoring “hot spots” in large scale distributed systems. For the first type of the application, we apply our scheme on Internet2 traffic logs [2] to identify hosts (destination IP addresses) that receive large number of flows. For the second type, we study the system event logs of an IBM network to identify frequent system events generated by a number of different hosts.

The Internet2 traffic traces [2] were anonymized NetFlow [88] data collected from nine core routers in the Abilene network. The data represents one full day of router operation, broken into 288 five-minute epochs. we divided the data from each router in a random fashion to simulate an environment of 216 different nodes. The system event logs were collected from 32 hosts in

Table 1: Data sets used in our experiments.

Trace	# of nodes	# of item-count pairs
NetFlow traces	216	2,927,878
IBM system logs	32	1,982

a production network at IBM. Each machine logs the name of the event and the timestamp of its occurrence. Table 1 summarizes all the data sets used in the evaluation. All of our experiments were carried out on a 2.8 GHz Dell PowerEdge workstation running Linux Kernel version 2.2.21.

6.5.2 Experiment setup and results

For both experiments we set the largest quantization point to 100 and remove all occurrences of items whose local frequency counts are over 100 at each node, since such $\langle item, count \rangle$ pairs will be delivered to the server directly (instead of being hashed into the Bloom filters) with zero information loss. Therefore, our experiments reflect precisely the accuracy of our counting sketch estimator.

For the experiments on the Internet2 NetFlow traces, we set the communication cost to 2 bits for each item-count pair whose count is 1. The iceberg threshold T is set at 2,160, which corresponds to 10 occurrences per node on the average. According to the aforementioned resource constraint model, we compute the communication “budget” for items with higher count values. The cost per pair gradually increases from 3 bits to 15.1 bits. Then using Algorithm 7 we compute the spectrum of quantization points as $\{1, 4, 10, 19, 32, 48, 67, 89, 100\}$. The slopes ((a_j) ’s) generated by the algorithm are very close to each other (all around 1.63). Small discrepancies among slopes are unavoidable due to our constraint that quantization points have to fall on integers. Similarly, for the experiments on the IBM system logs, we set the iceberg threshold T to 96, which corresponds to 3 occurrences per node on the average. The communication cost per item-count pair gradually increases from 5 bits to 21.6 bits. Then the spectrum of quantization points is computed as $\{1, 3, 6, 11, 17, 25, 35, 46, 59, 74, 90, 100\}$ and the slopes are all around 0.90.

In Internet2 traces each item-count pair occupies 64 bits. This is because each item is identified by a destination IP address, which is 32 bits, and the count for each item also needs 32 bits. In

Table 2: Communication cost of our scheme.

Trace	original (KB)	our scheme (KB)
NetFlow traces	22,203.4	1576.4
IBM system logs	35.8	2.05

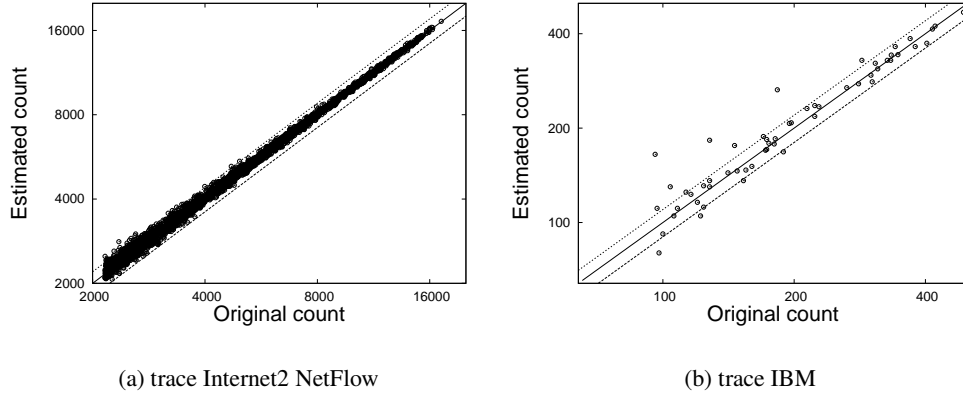


Figure 30: Original (x-axis) vs. estimated (y-axis) global frequency counts by the counting-sketch-based scheme. Notice both axes are on logscale.

IBM system logs each system event is denoted by a long character string (around 16 characters on average in the logs we have) and each count also has 32 bits. Thus each item-count pair in the logs is 160 bits on the average. Table 2 compares the size of the original data bags (not original data streams) with the communication cost our scheme uses. It is observed that our scheme achieves very efficient communication bandwidth usage compared with the naive approach of shipping all the data bags to the central server. The data reduction ratio of the Internet2 NetFlow traces is about 14 to 1 and the ratio is about 17 to 1 for IBM system logs. We will show next that even with such a small communication cost our scheme allows us to achieve very accurate estimations.

Figures 30(a) and 30(b) compare the global frequency counts estimated using our proposed sketch-based scheme with their actual values. In both figures, the solid diagonal line denotes perfect estimation, while the dashed lines that parallels the solid line denote estimations of relative error $\pm 10\%$. Clearly, the closer the points cluster around the diagonal line, the more accurate the estimations are. We only plot the items whose frequency counts are estimated to be larger than or equal to the pre-defined thresholds. We observe that estimations are very accurate in both experiments, and that the estimations are more accurate for relatively large frequency count values (in

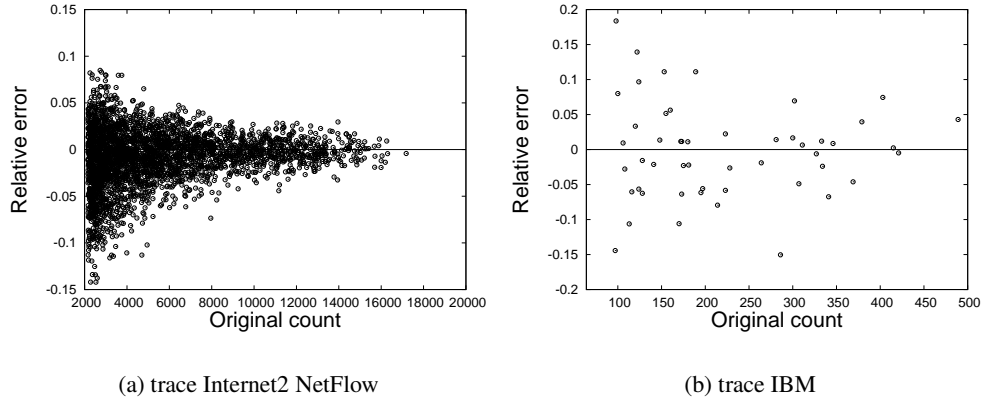


Figure 31: Relative errors of our estimations.

Figure 30(a) the points cluster closer around the diagonal line when their values become large.)

Figures 31(a) and 31(b) quantify the accuracy of our estimations shown in Figure 30 respectively, using the relative error ($\frac{f_I - \hat{f}_I}{f_I}$) as the evaluation metric. In both figures, points above and below the solid line in the middle (perfect estimation) denote overestimates and underestimates respectively. In Figure 31(a), the points are scattered around the solid line in an approximately symmetric way, demonstrating the unbiased nature of our estimator. The same symmetry also can be found in Figure 31(b) but is not so clear as in Figure 31(a) since there are not many points there. The relative errors are mostly within ± 0.05 in Figure 31(a), and within ± 0.1 in Figure 31(b). In both experiments, the relative error generally decreases when the actual count value increases. This is more obvious in Figure 31(a) since it has more points.

6.6 Conclusion

Identifying icebergs in distributed datasets with minimum communication overhead is an open problem. The problem has applications in many areas ranging from network monitoring to biosurveillance. To the best of our knowledge, this is the first work on finding global icebergs in a distributed environment without assuming that a globally frequent item is also locally frequent. We propose two solutions to the global iceberg problem, one based on sampling, and the other based on a novel counting sketch. We show that our sampling strategy is near-optimal, resulting in almost the lowest possible estimation error under certain resource (communication cost) constraints. Our second approach uses a novel counting sketch to help detect icebergs in a much more communication-efficient way than the sampling-based solution. Theoretical and experimental analysis demonstrate

the statistical properties of our proposed algorithms and their high accuracy.

CHAPTER VII

DESIGN OF A NOVEL STATISTICS COUNTER ARCHITECTURE WITH OPTIMAL SPACE AND TIME EFFICIENCY

7.1 *Introduction*

Hardware enhancement is another approach to support detailed measurements for today's measurement. In this chapter we focus on a very useful memory architecture organized as a large number (say millions) of statistics counters. How to efficiently store and maintain these counters in memory that need to be incremented at very high speed has been recognized as an important research problem [111, 102]. In this problem, tens of millions of increments need to be performed every second, each of which can happen to any of these counters¹. In addition, the size of each counter needs to be as large as 64 bits [102], since the values of some of these counters can become very high. The above speed requirement precludes the storage and maintenance of these counters in slower yet inexpensive memory such as DRAM. While fitting these counters entirely in fast yet more expensive SRAM meets the speed requirement, a large amount of SRAM may be needed with such large counter sizes, and hence the high cost. The common research issue in both the prior work [111, 102] and this work, is whether we can design a counter architecture that satisfies the above speed and size requirements, yet use much less SRAM than storing the counters entirely in SRAM.

7.1.1 **Motivation**

The need to maintain a large array of counters arises in various router management algorithms and data streaming algorithms, where a large array of counters is used to track various network statistics and to implement various counting sketches respectively.

As described in the prior work [111, 102], maintaining a large number of statistics counters is essential for a range of statistical accounting operations (e.g., performing SNMP link counts) at Internet packet switches and routers (e.g., IP routers, ATM switches, and Ethernet switches).

¹Lack of locality in counter accesses prevents the traditional caching approach from being effective.

Such operations are needed in network performance monitoring, management, intrusion detection, tracing, traffic engineering, etc. [111, 102]. As discussed in [102], the number of statistics counters can be as large as millions when routers would like to support traffic accounting based on various traffic filters such as source and/or destination IP prefixes, traffic types, AS (Autonomous System) pairs, and their combinations. In a typical application scenario, each incoming packet triggers an increment (typically by 1) to one or more of the counters depending on the traffic filter(s) that the packet matches. For high speed links such as OC-768 (40 Gbps), such an increment needs to be performed within several nanoseconds to keep up with the packet arrival rates.

The need to maintain a large number of high speed counters is also motivated by the recent advances of data streaming algorithms such as [50, 30, 71, 73, 132, 133, 75]. As defined in [87, 76], data streaming is concerned with processing a long stream of data items in one pass using a small working memory in order to approximately estimate certain statistics of the stream. A data streaming algorithm typically organizes its working memory into a synopsis data structure called sketch, which is specialized to capture as much information pertinent to the statistics it intends to estimate, as possible. While different sketches are proposed for estimating various statistics about the data stream, they often consist of one (e.g., [71, 73, 133, 75]) to several (e.g., [50, 33, 132]) arrays of counters and have a common online operation called “hash and increment”. In these algorithms, an incoming data item (e.g., a packet) is fed to a hash function and the hash result is treated as the index into an array of counters. The corresponding counter is then incremented (often by 1). In network and even some database applications, data items can arrive at a very high speed, and in many sketches, each data item can trigger an increment to multiple counters². In addition, several of these algorithms need to use up to millions of counters in various application scenarios. Therefore, techniques to significantly reduce the amount of SRAM needed to maintain these counters will benefit all these data streaming algorithms in considerably reducing their implementation costs.

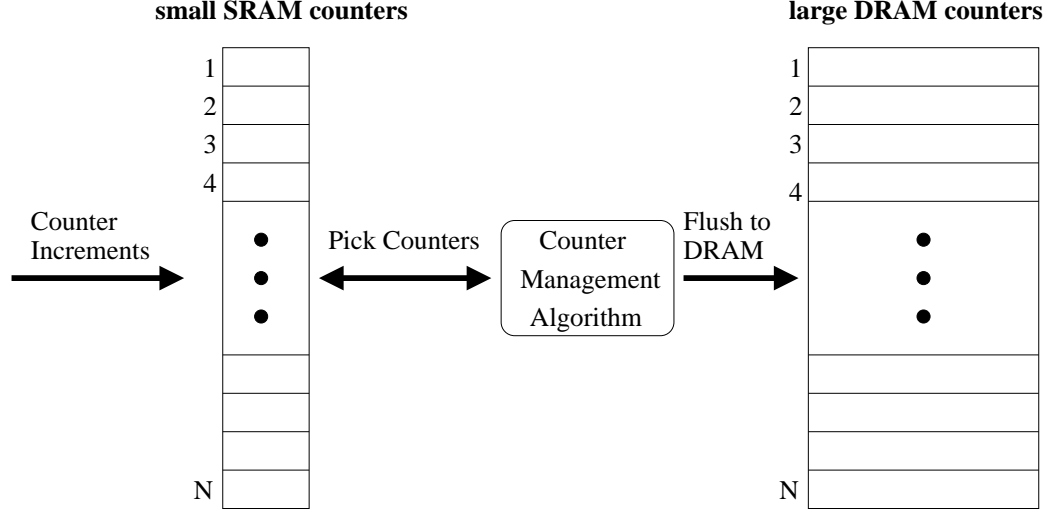


Figure 32: Hybrid SRAM/DRAM counter architecture

7.1.2 Hybrid SRAM/DRAM counter architectures

A hybrid SRAM/DRAM counter architecture is proposed in the seminal work of [111] as a more SRAM-efficient way of maintaining a large counter architecture, and this architecture is further improved in the followup work of [102]. Figure 32 shows the generic architecture for maintaining a large number of counters using a hybrid SRAM/DRAM design. The baseline idea of this architecture is to store some lower order bits (e.g., 9 bits) of each counter in SRAM, and the full-size counter (e.g., 64 bits) in DRAM. The increments are made only to these SRAM counters, and when the value of a SRAM counter becomes close to overflow, it will be scheduled to be *flushed* to the corresponding DRAM counter. The “flush” operation here is defined as adding the value of the SRAM counter to the corresponding DRAM counter and resetting the SRAM counter to 0. The key research challenge in this architecture is the design of a counter management algorithm (CMA) that flushes the right set of SRAM counters to DRAM at the right time.

Both prior works achieve impressive reductions in SRAM usage through this hybrid SRAM/DRAM approach. Since the result in [102] is strictly better than that of [111], here we only compare our work with [102], deferring a careful comparison with both schemes to later sections. As we will show, with a SRAM/DRAM speed difference of 30 times, the architecture in [102] reduces the

²They often reside in different logical arrays, but these arrays are often stored in a single SRAM chip due to various design constraints.

SRAM usage from 64 bits per counter to only 11 bits per counter. Among these 11 bits, each SRAM counter takes 9 bits and the other 2 bits per counter are used by the CMA. However, 9 bits are far from the minimum number of SRAM bits per counter that is theoretically possible, which we will show to be 5 in this case (with SRAM/DRAM speed difference of 30). Moreover, the CMA control logic in [102] is fairly complicated to implement, which requires the maintenance of a tree-like data structure in pipelined hardware and consumes 2 bits per counter, although it is simpler and more efficient than that in [111], where a heap has to be maintained in hardware and the control logic consumes about 20 bits per counter.

7.1.3 Our approach and contributions

In this work, we present a novel hybrid SRAM/DRAM statistics counter architecture that is provably optimal in terms of SRAM consumption yet has extremely simple control logic. With the same assumption as above, our scheme only requires $5 + \epsilon$ bits per counter, where each SRAM counter consumes 5 bits and our CMA consumes ϵ bits per counter. Here ϵ is typically a small number (e.g., 0.01). Note that reducing the SRAM counter size from 9 bits per counter to 5 bits per counter is to a certain extent $2^{9-5} = 16$ times harder, since overflows from an SRAM counter happen 16 times faster with the smaller overflow threshold (2^5 as compared to 2^9), requiring the “flushing” mechanism to operate 16 times more efficiently. What is more remarkable is that this improvement in efficiency is achieved with a CMA that is extremely simple and consumes only a small fraction of bits per counter, as compared to 2 bits per counter in [102].

There is a tiny price to pay for the above significant improvements in terms of both operational efficiency and implementation complexity. Our CMA uses a randomized algorithm, which with an extremely small yet nonzero probability, may lose some increments to the counters, while both previous approaches are deterministic, guaranteeing no such loss. However, in practice there is no need to worry about this probability since it can be made so small that even if a router operates continuously for billions of years (say from Big Bang to now), the probability that a single loss of increment happens is less than 1 over a billion. Note that router software/hardware failures and other unexpected or catastrophic events (e.g., power outage or earthquake) that may disable a router happen with a probability many orders of magnitude higher.

In short, our solution works as follows. Each logical counter is represented by a 5 bit counter in SRAM, and a larger 64 bit counter in DRAM. Increments to a logical counter happen to its 5 bit SRAM counter until it reaches the overflow value 32, at which point the value of this SRAM counter (i.e., 32) needs to be flushed to the corresponding DRAM counter. Since updates to DRAM counters take much longer than to SRAM counters, several SRAM counters may overflow during the time it takes to update just one DRAM counter. Our solution is to install a small SRAM FIFO buffer between the SRAM counters and the DRAM counters to temporarily hold the “flush requests” that need to be made to the DRAM in the future. However, this solution as stated so far will not work well in the worst case, where a large number of counter overflows may happen during a short period of time so that the SRAM FIFO buffer has to be very large (thereby negating the advantage of our scheme). Our solution to this problem lies in a simple randomization technique with which we can statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up that small SRAM FIFO buffer, even in the worst case. This randomization technique is the key innovation of this work.

Through a rigorous analytical modeling of the proposed solution, we show that a small SRAM buffer (several hundred slots) is large enough to ensure that the probability for it to be filled up by flush requests is vanishingly small, when there are millions of counters. Here each slot holds a counter index to be flushed, which is usually shorter than 4 bytes. This translates into the aforementioned ϵ bits per counter when the cost of hundreds of buffer slots are amortized over millions of counters.

Our analytical modeling of the proposed solution is a combination of worst case analysis and a novel tail bound technique. The purpose of the worst case analysis is to characterize the worst-case workload (counter increment sequence) to the counter array, which is modeled as the best workload generation strategy of an adversary (explained later) that has complete knowledge of our algorithm but not the random values generated by our algorithm during execution. This analysis allows us to establish a tight bound on the variance of the number of counter overflows during a measurement interval. The next step is to bound the probability that the FIFO buffer is overwhelmed by the overflowed counters using some well-known tail bound theorems. However, traditional tail bound techniques such as Chernoff bound will not allow us to take advantage of the variance bound

obtained through the above worst case analysis. We develop a novel tail bound theorem, which takes full advantage of the variance bound, to establish probability bounds which is able to improve the Chernoff bound significantly.

Simulations of this architecture using real-world Internet traces demonstrate that the actual buffer size needed is much smaller than the bounds derived through our analysis. This is not surprising given that the analytical bounds work for the worst case while typical traffic patterns observed in a real-world network tend to be fairly “benign” (not in the security sense). One could design a more specific solution targeting observed patterns in real world traffic. We advise against this extension, however, because the savings on SRAM will be very small (in terms of percentage) and it will limit the applicability of our scheme to certain traffic arrival patterns that may not hold true in general.

Note that our hybrid SRAM/DRAM counter architecture is designed for “increment by 1” and will not work well for other increment sizes. While this is sufficient for counting the number of packets, in which each incoming packet triggers the increment of one or more counters by 1, it will not work for counting the number of bytes, in which each incoming packet needs to increment one or more counters by hundreds or thousands (e.g., performing SNMP link counts). A simple extension of the architecture to accommodate other increment sizes is proposed in [102]. Its idea is to statistically quantize an increment of size S to a Bernoulli random variable with mean S/M , where M is the maximum packet size. After this quantization, a counter only needs to be incremented by 1, but the tradeoff is that some estimation error will be introduced. This extension can be readily used in combination with our architecture to handle increments of other sizes. We refer readers to [102] for details of this extension. In the rest of the chapter we focus on the problem of “increment by 1”.

The rest of the chapter is organized as follows. Section 7.2 describe our architecture in detail. In Section 7.3 we provide a rigorous analysis on the performance of our architecture in the worst case. In Section 7.4 we present the numerical results of this analysis and simulate the performance of our architecture using real world Internet traces. Section 7.5 concludes the work.

7.2 Our Scheme

The basic intuition behind our scheme is as follows. Like in prior approaches [111, 102], we maintain l -bit counters in SRAM and full-size counters in DRAM. The SRAM counters will handle increments at very high speed, and once an SRAM counter reaches value 2^l (overflow), its value needs to be flushed to its corresponding DRAM counter. The CMA in our scheme is extremely simple. We maintain a small FIFO queue in SRAM that holds the indices of the SRAM counters that have overflowed but have not yet been flushed to DRAM. Note that the access speed of DRAM (departure rate from the queue) should be faster than the average rate of counter overflow (arrival rate to the queue), since otherwise, the queue will fill up no matter how large it is. This implies that if the ratio of DRAM access speed to that of SRAM is μ (< 1), the SRAM counter size l should be larger than $\log_2 \frac{1}{\mu}$ bits, which we referred to earlier as the theoretically minimum SRAM counter size.

A queue is still needed even when $l > \log_2 \frac{1}{\mu}$, however, since the instantaneous counter overflow rate could be much faster than the DRAM access speed *in the worst case*. From queuing theory, we know that the queue size is small when the arrival process is “smooth”. A key innovation of our scheme is to guarantee that the arrival process (the arrival of the counter overflows) is fairly smooth even in the worst case through a simple randomization scheme. This ensures that a small queue can guarantee no loss of the indices (to be flushed to DRAM) with overwhelming probability.

The pseudo-code of the algorithm is shown in Algorithm 8. The counter arrays A and B correspond to l -bit SRAM counters and full-size DRAM counters respectively. Let N be the number of counters in A as well as in B . An increment to an arbitrary counter i is handled in lines 1 through 5. The SRAM counter $A[i]$ will first be incremented, and if this increment causes it to overflow (line 3), its content needs to be flushed to DRAM. In this case, its index i is placed into the FIFO queue Q (line 4) and $A[i]$ is reset to 0 (line 5). The actual flush operation is shown in lines 6 through 9. When DRAM finishes the previous write (flushing), a controller will fetch a counter index i from the head of the queue if it is not empty, and increment the DRAM counter $B[i]$ by 2^l (line 9). In terms of implementation, we only require that the array A and the queue Q reside in different SRAM modules, as we will explain later in this section.

Algorithm 8: The pseudo-code of the algorithm

```
1 UpdateSRAM(i)
2    $A[i] := A[i] + 1;$ 
3   if ( $A[i] == 2^l$ ) /* overflow happens */
4      $Q.enqueue(i);$ 
5      $A[i] := 0;$ 

6 FlushToDRAM()
7   while ( $Q$  is not empty)
8      $i := Q.dequeue();$ 
9      $B[i] := B[i] + 2^l;$ 

10 Initialize()
11   for  $i:=1$  to  $N$ 
12      $A[i] := uniform(0, 2^l - 1);$ 
13      $B[i] := -A[i];$ 
```

The specification of our scheme so far is not complete, as we have not assigned initial values to $A[i]$ and $B[i]$, $i = 1, 2, \dots, N$. These initial assignments turn out to be the most critical part of our scheme. While simply setting $A[i]$ and $B[i]$, $i = 1, 2, \dots, N$, to 0 at the beginning of a measurement (counting) interval is a standard practice, it will not work well in our scheme for the following reason. In the worst case, an adversary (explained below) could choose the sequence of the indices of the counters to be incremented to be $1, 2, \dots, N, 1, 2, \dots, N, \dots$ (i.e., the repetition of the subsequence “ $1, 2, 3, \dots, N$ ” over and over). At the end of the $(2^l - 1)_{th}$ repetition, the values of $A[i]$, $i = 1, 2, \dots, N$, will all become $2^l - 1$. Then during the next repetition, $A[i]$, $i = 1, 2, \dots, N$ will overflow one by one after each increment, resulting in a “burst arrival” of size $O(N)$ to the queue. The queue has to be made very large (in fact much larger than the SRAM counter array A) to be able to accommodate this burst, which negates the purpose of our scheme to save SRAM.

Our solution to this problem, specified in lines 10 through 13 in Algorithm 8, is again very simple. For each index i , we generate a random integer number uniformly distributed over $\{0, 1, 2, \dots, 2^l - 1\}$, and assign this number to $A[i]$ (line 12). We need to somehow remember this value since it should be subtracted from the observed counter value the end of a measurement (counting) interval. This is achieved in line 13, by setting the initial value of $B[i]$ to $-A[i]$, for $i = 1, 2, \dots, N$.³ We

³This needs 1 extra DRAM bit per counter for the sign. An alternative method is to use an additional DRAM counter array (each counter has l bits) to remember the initial values. Both methods increases the DRAM cost only slightly (i.e., N and $l \times N$ bits respectively).

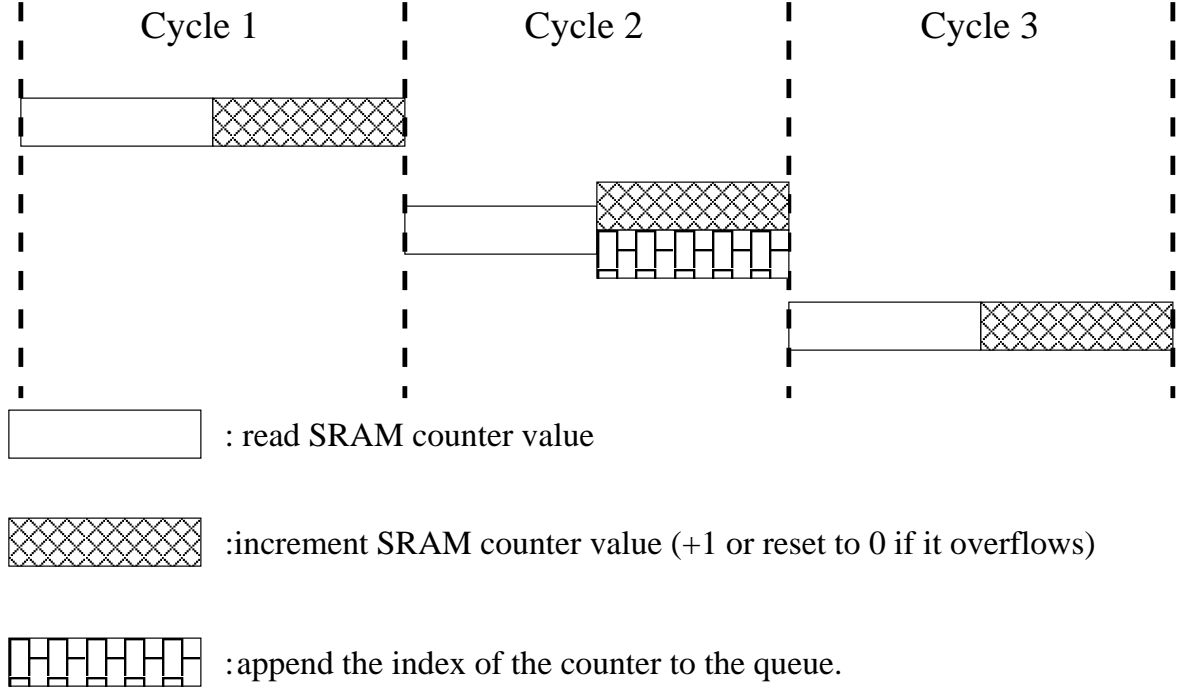


Figure 33: Timing diagram for our algorithm

assume that these initial values of $A[i]$ and $B[i]$ are not known to the adversary (explained next). Intuitively, this randomization ensures that given any workload (counter increment sequence), the counter overflow process will be quite smooth. We show that the queue size only needs to be around several hundreds to guarantee that the queue Q will not fill up with very high probability.

Note that here this adversary is defined entirely in the well-established context of randomized online algorithm design [86], and has nothing to do with its connotation in security and cryptography; It is defined as a deterministic or randomized algorithm that aims at maximizing the size of our queue, and has complete knowledge of our algorithm except (i.e., oblivious to) the initial values of $A[i]'$ s and $B[i]'$ s. The sole purpose of introducing this adversary is to model the worst-case workload (i.e., counter increment sequence) to our architecture. It has nothing to do with security or cryptography.

We show the timing diagram of our scheme in Figure 33. Each cycle is the time it takes for a counter to be read from SRAM, incremented, and written back to SRAM. It is only slightly longer than two memory accesses since the increment operation takes much less time with hardware implementation. Therefore Figure 33 only shows a read step and a write step, omitting the tiny

increment step in the middle. When a counter overflows, it is set to zero during the write step, and simultaneously its index is written into the queue. We only require the SRAM counter array and the queue to be implemented on two different SRAM modules so that these two writes can happen simultaneously; No pipelining logic is needed. Note that this requirement is much weaker than used in prior approaches [111, 102], in which different parts of a data structure need to be placed on different memory modules and nontrivial pipelining logic needs to be placed between them. Note that even in the worst case when there is one counter overflow every cycle, the read/write bandwidth of the queue is only 50% utilized because the insertion of an index (of an overflowed counter) only happens in the second part of a cycle. The first part of a cycle can always be used to move an index at the top of the queue to a DRAM write buffer, as soon as DRAM finishes the previous write operation. To summarize, we show that our architecture is able to handle one increment per cycle. Therefore, this concept of *cycle* will be used as the basic unit of time for our analysis, the topic of the next section.

Note that while our improved CMA scheme allows for the use of fewer SRAM bits per counter, it significantly increases the amount of traffic to DRAM (through the system bus). In fact, for every bit we save on SRAM counter size, the amount of traffic to DRAM is doubled. Although this problem also exists in other CMA schemes [111, 102], it is not as severe there since they use longer SRAM counter and therefore generate less traffic to DRAM. This increase in DRAM traffic may become a serious concern in today's network processors where system bus and DRAM bandwidth is heavily utilized already for packet processing. We acknowledge that this problem has not been addressed in this work. We plan to study and hopefully solve the problem in our future research.

7.3 Analysis

In this section, we prove that even with a small buffer size, the probability that the FIFO queue Q overflows during a fairly long time interval is extremely small in the worst case. Some numerical examples will be shown in Section 7.4.1.

7.3.1 Notations and summary of results

In the previous section, we have defined the concept of a cycle (see Figure 2) and explained that our architecture can handle one increment per cycle. Therefore, we will use “cycle” as the basic unit of time. Throughout the following analysis, we assume there is an increment in each and every cycle (i.e., being continuously busy). It is intuitive that this assumption indeed represents the worst-case in the sense that the probability bounds derived for this case will be no better than allowing certain cycles to be idle (i.e., no increment during these cycles). We omit the proof of this fact here since it would be a tedious application of the elementary stochastic ordering theory [107]. We also assume that the sequence of array indices to be incremented is arbitrary. In other words, the probability bounds we derive in this section will apply to any increment sequence.

Let K be the number of slots, each of which stores an index to be flushed to DRAM, in the FIFO queue Q . Let N be the number of counters in the array and l be the size of each SRAM counter as defined before. Let μ be the ratio of DRAM access time to SRAM access time. For example, if SRAM is 30 times faster than DRAM, then μ is equal to $\frac{1}{30}$. Recall from the previous section that a cycle is approximately one SRAM read and one SRAM write (with the time needed for “increment by 1” omitted). Note that to flush an index i from Q to DRAM, we need to read the corresponding DRAM entry $B[i]$, add 2^l to it, and writes it back to DRAM. Again omitting the amount of time to perform “increment by 2^l ”, this transaction takes approximately two DRAM accesses. Therefore, it takes $1/\mu$ cycles to flush an index to the DRAM. Equivalently we can say that μ flushes are finished within one cycle, and μ can be viewed as the departure rate (per cycle) from the queue Q . The average arrival rate to the queue Q is 2^{-l} , since it takes 2^l increments on the average to guarantee a counter overflow⁴. Clearly, the average arrival rate to the queue has to be smaller than the departure rate (for the queue to be stable), and hence $2^{-l} < \mu$ or $2^l > \frac{1}{\mu}$ as we have stated in the previous section.

Let D_n be the event that one or more “flushes to DRAM” requests are dropped because Q is full when they came during the time interval of a total of n cycles (e.g., n increments). Again, in

⁴One may feel and even we felt that since the counter values are initialized to be uniformly distributed between 0 and $2^l - 1$, it takes an adversary on the average 2^{l-1} to overflow a counter. This is not true because the adversary does not know the initial values of the counters and therefore may waste some effort (increments) on counters that have already overflowed once.

the following, all time parameters and values such as s and t are in the units of cycles. We shall establish a tight bound on the probability of this event D_n , as a function of aforementioned system parameters K , N , l , μ , and n . In this section, we shall fix n and will therefore shorten D_n to D .

We first show that $\Pr[D]$ is bounded by the summation of probabilities $\Pr[D_{s,t}]$, $0 \leq s \leq t \leq n$, i.e.,

$$\Pr[D] \leq \sum_{0 \leq s \leq t \leq n} \Pr[D_{s,t}]$$

Here $D_{s,t}$ represents the event that the number of arrivals during the time interval $[s, t]$ is larger than the maximum possible number of departures in Q (if serving continuously), by more than the queue size K . Formally letting $b(s, t)$ denote the number of “flush to DRAM” requests generated during time interval $[s, t)$, then we have

$$\Pr[D_{s,t}] \equiv \Pr[b(s, t) - \mu(t - s) > K].$$

The inequality above is a direct consequence of the following lemma, which states that if the event D happens, at least one of the events $\{D_{s,t}\}_{0 \leq s < t \leq n}$ must happen.

Lemma 2 $D \subseteq \bigcup_{0 \leq s \leq t \leq n} D_{s,t}$

Proof: Given an outcome $\omega \in D$, suppose an overflow happens at time z . The queue is clearly in the middle of a busy period at time z . Now suppose this busy period starts at y . Then the number of departures from y to z is equal to $\lfloor \mu(z - y) \rfloor$. Since a “flush to DRAM” request happens at time z to find the queue of size K full, $b(y, z)$, the total number of arrivals during time $[y, z]$ is at least $K + 1 + \lfloor \mu(z - y) \rfloor \geq K + \mu(z - y)$. In other words, $D_{y,z}$ happens and $\omega \in D_{y,z}$. This means for any outcome ω in the probability space, if $\omega \in D$, then $\omega \in D_{s,t}$ for some $0 \leq s < t \leq n$.

Remark: As a consequence,

$$\Pr[D] \leq \Pr\left[\bigcup_{0 \leq s \leq t \leq n} D_{s,t}\right] \leq \sum_{0 \leq s \leq t \leq n} \Pr[D_{s,t}]$$

The rest of the section is devoted to deriving tight tail bounds for individual $\Pr[D_{s,t}]$ terms. We develop two main techniques in these derivations, both of which are based on the properties of the sum of independent random variables. These two bounds are described in detail in Section 7.3.2 and 7.3.3 respectively. We first provide a brief summary of these results in the following.

1. Our first bound, stated as Theorem 13, is derived using a simplified form of the well-known Chernoff bound. It has the following form:

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu(t-s) > K] \\ < e^{-2(K + \mu(t-s) - 2^{-l}(t-s))^2 / \min\{t-s, N\}}$$

This is proven by showing that the centered random variable $b(s,t) - \mathbb{E}[b(s,t)]$ is the summation of $\min\{t-s, N\}$ independent random variables, each of which is a Poisson trial (explained later), which allows a Chernoff-type theorem optimized for the summation of Poisson trials to be applied. This bound shows that a small number of slots in the queue will allow us to achieve very small (queue) overflow probability.

2. The first bound still leaves considerable room for improvement because the Chernoff bound does not take advantage of the second moment information of $b(s,t)$ which can be derived analytically in our context. We will show in Section 7.3.3 that the variance of $b(s,t)$ is bounded by

$$\text{Var}[b(s,t)] \leq \begin{cases} \frac{N}{4} & t-s \geq 2^{l-1}N, \\ \frac{(2^l - \frac{t-s}{N})(t-s)}{2^{2l}} & N \leq t-s < 2^{l-1}N, \\ \frac{(2^l-1)(t-s)}{2^{2l}} & 0 < t-s < N. \end{cases}$$

We derive a novel tail bound theorem that takes advantage of such additional information (the bound on the variance), which significantly improves the first bound for most s and t values, especially when the queue size K is relatively small. Let $\tau = t-s$ and let σ be the standard deviation of $b(s,t)$. Using this new theorem, we obtain the following tail bound for $\Pr[D_{s,t}]$.

$$\Pr[D_{s,t}] \equiv \Pr[b(s,t) - \mu\tau > K] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})},$$

where

$$a = \min\{a_0, \frac{K + (\mu - 2^{-l})\tau}{\sigma}\}, \epsilon = e^{\frac{a}{\sigma}} - 1,$$

and a_0 is the unique root of $g(a) = 4 - e^{\frac{a}{\sigma}} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma}$ for $a \in (0, \sigma \ln 4)$.

3. we propose a hybrid overall bound which is obtained when we use the minimum of the first and the second bounds on each $D_{s,t}$ term based on the comparison of the above two bounds. The detailed analytical comparison of the bounds can be found in Appendix A.5.2.

We will show some numerical results of these three types of bounds in Section 7.4.1. We find that when the queue size K is relatively small, the second bound is orders of magnitude better than the first one. Using the hybrid bound only provides negligible improvement. However, when the queue size becomes large, the first bound can become better than the second one gradually. In this case, the second bound still proves its usefulness by making the hybrid case several times smaller than the first bound.

7.3.2 Bounding the probability of $D_{s,t}$ using simplified Chernoff bound

In this section, we use a variant of Chernoff bound optimized for a family of random variables called Poisson trials to bound $\Pr[b(s, t) - \mu(t - s) > K]$. We first state a technical lemma that will be used in the later derivation. Its proof can be found in Appendix A.5.1.

Lemma 3 *Let $V_j(t)$ be the value of counter j at time t . Given an arbitrary counter increment sequence, we have (i) $V_1(t), V_2(t), \dots, V_N(t)$ are mutually independent random variables and (ii) each of them has the following distribution:*

$$V_j(t) = \begin{cases} 0 & \text{with probability } 2^{-l}, \\ 1 & \text{with probability } 2^{-l}, \\ \dots & \\ 2^l - 1 & \text{with probability } 2^{-l}. \end{cases}$$

Let b_j be the number of “flush to DRAM” requests generated by the counter j during the time interval $[s, t]$. Clearly we have $\sum_{j=1}^N b_j = b(s, t)$. Let c_j be the number of increments to counter j during time period $[s, t]$, $j = 1, 2, \dots, N$. In all our derivations $c'_j s$ are allowed to take arbitrary values, and are considered constants once the values are chosen. We first prove some properties of b_j and c_j in the following theorem, which will be used in the later derivations. In that theorem we use a new notation: for any real number x , we define $\{x\}$ as $x - \lfloor x \rfloor$. In other words, $\{x\}$ is the fraction part of x .

Theorem 12 (i) b_1, b_2, \dots, b_N are mutually independent random variables during any time interval $[s, t]$; (ii) $E[b_j] = c_j 2^{-l}$; (iii) $\text{Var}[b_j] = \{c_j 2^{-l}\}(1 - \{c_j 2^{-l}\})$; (iv) $E[b(s, t)] = \frac{t-s}{2^l}$

Proof: It is easy to obtain that

$$b_j = \left\lfloor \frac{c_j + V_j(s)}{2^l} \right\rfloor$$

that is, b_j is a function of the counter value V_j at cycle s and the constant c_j . Since $V_1(s), V_2(s), \dots, V_N(s)$ are mutually independent due to Lemma 3, b_j are mutually independent as well for any possible s . So (i) holds.

Since by Lemma 3, we know that $V_j(s)$ is uniformly distributed over $\{0, 1, \dots, 2^l - 1\}$, the distribution of b_j can be simplified as

$$b_j = \begin{cases} \lfloor \frac{c_j}{2^l} \rfloor & \text{with probability } 1 - \{2^{-l}c_j\}, \\ \lfloor \frac{c_j}{2^l} \rfloor + 1 & \text{with probability } \{2^{-l}c_j\}. \end{cases}$$

For simplicity let α and β denote $\frac{c_j}{2^l}$ and $\lfloor \frac{c_j}{2^l} \rfloor$ respectively. $E[b_j] = \beta(\beta + 1 - \alpha) + (\beta + 1)(\alpha - \beta) = \alpha$. So (ii) holds.

We can also obtain that $E[b_j^2] = \beta^2(\beta + 1 - \alpha) + (\beta + 1)^2(\alpha - \beta) = 2\alpha\beta - \beta^2 + \alpha - \beta$. Therefore

$$\begin{aligned} \text{Var}[b_j] &= E[b_j^2] - (E[b_j])^2 \\ &= \alpha - \beta - (\alpha - \beta)^2 = (\alpha - \beta)(1 - \alpha + \beta) \end{aligned}$$

Thus we finish the proof of (iii).

Finally, we have $E[b(s, t)] = \sum_{j=1}^N E[b_j] = \frac{\sum_{j=1}^N c_j}{2^l} = \frac{(t-s)}{2^l}$. Thus (iv) holds.

A direct consequence of the above theorem is that $b_j - E[b_j]$, $j = 1, 2, \dots, N$, are mutually independent random variables and

$$b_j - E[b_j] = \begin{cases} -\{2^{-l}c_j\} & \text{with probability } 1 - \{2^{-l}c_j\}, \\ 1 - \{2^{-l}c_j\} & \text{with probability } \{2^{-l}c_j\}. \end{cases}$$

Such random variables are called (centered) *Poisson trials*. Note that $b(s, t) - E[b(s, t)]$ is the summation of these independent Poisson trails $b_j - E[b_j]$, $j = 1, 2, \dots, N$. The following variant of Chernoff bound theorem can be used to derive a tail bound on $\Pr[b(s, t) - \mu(t - s) > K]$.

Lemma 4 (cited from [7]) *Let X_1, X_2, \dots, X_m be mutually independent random variable such that, for $1 \leq j \leq m$, $\Pr[X_j = 1 - p_j] = p_j$ and $\Pr[X_j = -p_j] = 1 - p_j$, where $0 < p_j < 1$. Then, for $X = \sum_{j=1}^m X_j$ and $a > 0$,*

$$\Pr[X > a] < e^{-2a^2/m}$$

To apply this theorem we map all the parameters to our context. m corresponds to $\min\{t-s, N\}$ because during the $t-s$ cycles at most $t-s$ counters are updated and X_j corresponds to $b_j - 2^{-l}c_j$ so that $p_j = \{2^{-l}c_j\}$ according to Theorem 12. Thus X corresponds to $b(s, t) - 2^{-l}(t-s)$. To simplify the formula, we use τ to replace $t-s$ in the following derivation when it is more convenient.

Therefore

$$\begin{aligned} \Pr[b(s, t) - \mu(t-s) > K] &= \Pr[X + 2^{-l}\tau - \mu\tau > K] \\ &= \Pr[X > K + \mu\tau - 2^{-l}\tau] \end{aligned}$$

We set a in Lemma 4 to $K + \mu\tau - 2^{-l}\tau$ and finally obtain

Theorem 13 *For any $s < t$, let $\tau = t - s$.*

$$\Pr[D_{s,t}] \equiv \Pr[b(s, t) - \mu\tau > K] < e^{-2(K+\mu\tau-2^{-l}\tau)^2 / \min\{\tau, N\}}$$

The computational complexity to obtain the overall bound of $\Pr[D]$ is $O(n)$ because the bound on $\Pr[D_{s,t}]$ is shift-invariant in the sense that it is same as the bound on $\Pr[D_{s+\Delta, t+\Delta}]$, i.e., $\Pr[D_{s,t}]$ is only a function of $\tau = t - s$. So we only need to compute such a bound once and multiply it by $n - \tau + 1$ to account for the overall bound on $\Pr[D]$. This complexity may be further reduced since $\Pr[D_{s,t}]$ is monotonically decreasing when τ increases. Then if the value of $\Pr[D_{s,t}]$ already decreases to a negligible level (e.g., 2.2251×10^{-308} in MATLAB 7.0), we need not compute the following terms.

7.3.3 Using second moment information to obtain new bound of $\Pr[D_{s,t}]$

So far we obtain an upper bound of $\Pr[D_{s,t}]$ using Theorem 13. However it does not fully take advantage of the available information such as a bound on the variance of $b(s, t)$ which we can derive analytically. Thus it is possible to obtain a better bound in some cases by combining this

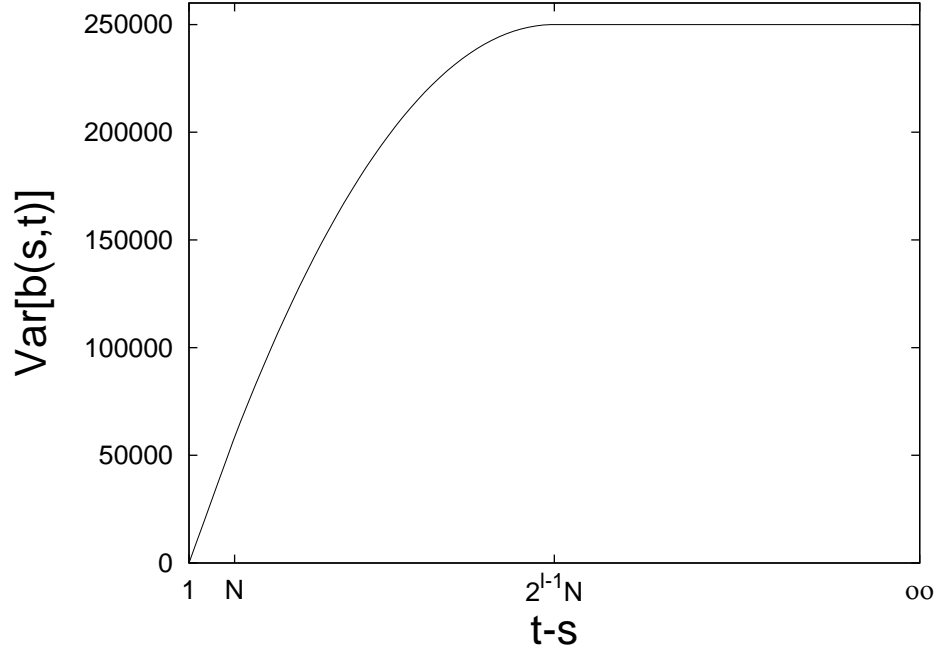


Figure 34: variance of $b(s, t)$.

variance information. Next we first show (in Theorem 14) how to derive a bound on the variance of $b(s, t)$. Then we propose a new tail bound which takes advantage of both the fact that $b(s, t)$ is the sum of independent random variables $b_j, j = 1, 2, \dots, N$ and the fact that we have a bound on its variance. We will show how to adapt this theorem to our context to obtain another often stronger upper bound of $\Pr[D_{s,t}]$.

7.3.3.1 Bounding the variance of $b(s, t)$

Theorem 14

$$\text{Var}[b(s, t)] \leq \begin{cases} \frac{N}{4} & t - s \geq 2^{l-1}N, \\ \frac{(2^l - \frac{t-s}{N})(t-s)}{2^{2l}} & N \leq t - s < 2^{l-1}N, \\ \frac{(2^l - 1)(t-s)}{2^{2l}} & 0 < t - s < N. \end{cases}$$

Proof: Because b_1, b_2, \dots, b_N are mutually independent by Theorem 12, $\text{Var}[b(s, t)] = \sum_{j=1}^N \text{Var}[b_j]$.

In the following we prove the theorem in three different cases:

i) By Theorem 12, we have

$$\begin{aligned}\text{Var}[b_j] &= \left\{\frac{c_j}{2^l}\right\}(1 - \left\{\frac{c_j}{2^l}\right\}) \\ &\leq \frac{(\left\{\frac{c_j}{2^l}\right\} + 1 - \left\{\frac{c_j}{2^l}\right\})^2}{4} = \frac{1}{4}\end{aligned}$$

The equality is satisfied if and only if $\frac{c_j}{2^l} - \lfloor \frac{c_j}{2^l} \rfloor = 0.5$ which means c_j is equal to $2^l \times d + 2^{l-1}$ where d is a constant. So if $t - s \geq 2^{l-1}N$, i.e., it is large enough to spread to N different counters with 2^{l-1} increments per counter, $\text{Var}[b(s, t)] \leq \sum_{j=1}^N \frac{1}{4} = \frac{N}{4}$. The equality is satisfied only when $c_j = 2^l d_j + 2^{l-1}$, $j = 1, 2, \dots, N$, where $d_j \geq 0$.

ii) When $t - s < 2^{l-1}N$ the total number of increments (i.e., $t - s$) is not large enough to spread out to all N counters with 2^{l-1} increments each. So the question is how to allocate the total $t - s$ increments to different counters to maximize the variance. We show that the best strategy is spreading the increments to as many different counters as possible. Because each c_j can be represented by $2^l d_j + y_j$ where $0 \leq y_j \leq 2^l - 1$ and $\text{Var}[b_j]$ is maximized ($= \frac{1}{4}$) when $y_j = 2^{l-1}$ as we showed in the above, we conclude that the variance is maximized when all $d_j = 0$, $j = 1, 2, \dots, N$ and $y_j \leq 2^{l-1}$.

Then $c_j = y_j \leq 2^{l-1}$ such that $\lfloor 2^{-l} c_j \rfloor = 0$. Therefore

$$\begin{aligned}\text{Var}[b(s, t)] &= \sum_{j=1}^N \text{Var}[b_j] = \frac{1}{2^{2l}} \sum_{j=1}^N y_j(2^l - y_j) \\ &= 2^{-l}(t - s) - 2^{-2l} \sum_{j=1}^N y_j^2\end{aligned}$$

Then when $N \leq t - s < 2^{l-1}N$, $(t - s)^2 = (\sum_{j=1}^N y_j)^2 \leq N \sum_{j=1}^N y_j^2$. So $\sum_{j=1}^N y_j^2 \geq \frac{(t-s)^2}{N}$.

The equality is satisfied, i.e., $\sum_{j=1}^N y_j^2$ is minimized, if and only if $y_1 = y_2 = \dots = y_N = \frac{t-s}{N}$. Thus $\text{Var}[b(s, t)] \leq \frac{(2^l - \frac{t-s}{N})(t-s)}{2^{2l}}$.

iii) When $t - s < N$, at most $t - s$ counters may be incremented in $t - s$ cycles. Assume these counters are $\psi_1, \psi_2, \dots, \psi_{t-s}$. Similarly to the above derivation we have

$$(t - s)^2 = \left(\sum_{j=1}^{t-s} y_{\psi_j}\right)^2 \leq (t - s) \sum_{j=1}^{t-s} y_{\psi_j}^2$$

The last inequality is due to Cauchy Schwartz's inequality. Therefore $\sum_{j=1}^{t-s} y_{\psi_j}^2 \geq t - s$ where the equality is satisfied if and only if $y_{\psi_1} = y_{\psi_2} = \dots = y_{\psi_{t-s}} = 1$. Thus $\text{Var}[b(s, t)] \leq \frac{(2^l - 1)(t-s)}{2^{2l}}$.

For example, setting $N = 1,000,000$ SRAM counters and $l = 4$ bits we plot the curve of the bound on $\text{Var}[b(s, t)]$ by varying the value of $t - s$ in Figure 34. To make it clearer we divide it

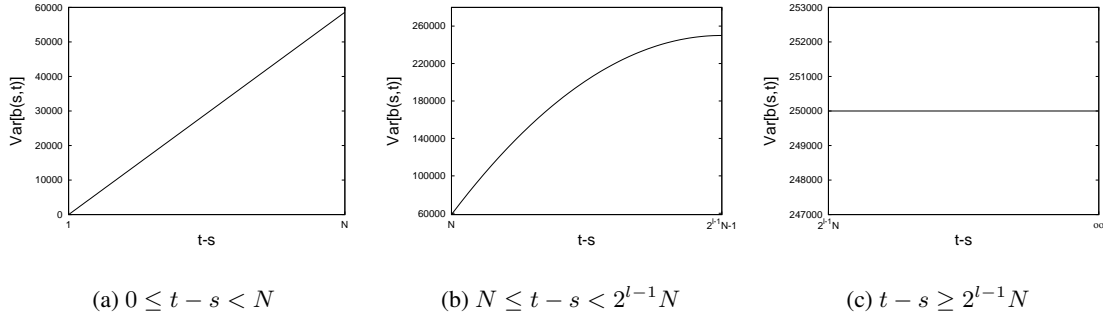


Figure 35: Expanded depiction of variance of $b(s, t)$ for three cases.

into three parts (shown in Figure 35) according to three different cases of the variance. It is clear that if $t - s < N$ the corresponding variance bound is linearly increasing with $t - s$ (Figure 35(a)); if $N \leq t - s < 2^{l-1}N$, the variance bound is monotonically increasing with $t - s$ (Figure 35(b)); otherwise, the corresponding variance bound stays the same everywhere (Figure 35(c)), i.e., $\frac{N}{4}$. In all three cases, the variance bound is always a function of $t - s$. Therefore, for simplicity we again use τ to replace $t - s$ in the following derivations where it is more convenient.

7.3.3.2 Another tail bound theorem

Here we again adopt Theorem 7 in Chapter 6. We are able to obtain tail bounds from this Theorem that are much tighter than obtainable from Theorem 13 in some cases (e.g., when the queue size K is small.) and the numerical results will be shown in Section 7.4.1.

7.3.3.3 Mapping the new tail bound to our context

We now map this tail bound theorem to our context as follows. W corresponds to the deviation of $b(s, t)$ from its expectation ($= b(s, t) - E[b(s, t)]$). The deviation of each b_j from its expectation corresponds to W_j , i.e., $W_j = b_j - E[b_j]$.⁵ And σ_j^2 and σ^2 correspond to $\text{Var}[b_j]$ and $\text{Var}[b(s, t)]$ respectively. Theorem 7 essentially states that the probability that W is larger than a times standard deviation is less than $e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$. However, this is true only for $a \leq \delta\sigma$, and δ , ϵ and θ are related by $e^{\delta\theta} \leq 1 + \epsilon$. Therefore we need to bound θ , which is the upper bound for $|W_j|$, $j = 1, 2, \dots, N$, since otherwise either ϵ has to be large or a has to be very small, and both cases lead to trivial or

⁵here we use notation j to replace notation i in Theorem 7 for convenience.

loose tail bounds. Clearly $|W_j| \leq \max\{\frac{c_j}{2^l} - \lfloor \frac{c_j}{2^l} \rfloor, \lfloor \frac{c_j+2^l-1}{2^l} \rfloor - \frac{c_j}{2^l}\} < 1$. So we set θ to 1. So far we know that

$$\begin{aligned}\Pr[b(s, t) - \mu(t - s) > K] &= \Pr[W + 2^{-l}\tau - \mu\tau > K] \\ &= \Pr[W > K + \mu\tau - 2^{-l}\tau]\end{aligned}$$

To apply Theorem 7 we need to find the corresponding a given τ to make $\Pr[W > K + \mu\tau - 2^{-l}\tau] \leq \Pr[W > a\sigma]$. Therefore, we obtain another constraint, which is, $a\sigma \leq K + \mu\tau - 2^{-l}\tau$. Now we take out this constraint for a moment and look at how we optimize the rest. We certainly want a to be as large as possible. However, we have the constraint $a \leq \delta\sigma$ and δ is in turn constrained by $e^{\theta\delta} \leq 1 + \epsilon$. Note that a large a will result in large ϵ and $1 - \epsilon/3$ can be small or even negative. Intuitively there is an optimal tradeoff point in between. So we can formulate the problem as follows:

$$\begin{aligned}\text{maximize} \quad & \frac{a^2}{2}(1 - \frac{\epsilon}{3}) \\ \text{subject to} \quad & 0 < a \leq \delta\sigma \\ & e^\delta - 1 \leq \epsilon < 3 \\ & a\sigma \leq K + \mu\tau - 2^{-l}\tau\end{aligned}$$

In the following we show how to resolve this optimization problem. We know that

$$\frac{a^2}{2}(1 - \frac{\epsilon}{3}) \leq \frac{a^2}{2}(1 - \frac{e^{\frac{a}{\sigma}} - 1}{3}) = \frac{a^2}{6}(4 - e^{\frac{a}{\sigma}})$$

since $a \leq \delta\sigma$. The RHS of the above equation can be viewed as a function of a , denoted as $f(a)$.

Its first derivative $f'(a)$ is equal to

$$\frac{4a}{3} - \frac{ae^{\frac{a}{\sigma}}}{3} - \frac{a^2e^{\frac{a}{\sigma}}}{6\sigma}$$

The maxima of $f(a)$ should be achieved at some of the roots of $f'(a) = 0$, or at the boundary. We first compute and check out these roots (to see if it is a maxima or minima). Since $a > 0$, $f'(a) = 0$ can be simplified as $g(a) = 0$ where

$$g(a) = 4 - e^{\frac{a}{\sigma}} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma}$$

It is clear that $g'(a) = -\frac{3e^{\frac{a}{\sigma}}}{2} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma} < 0$ since $\frac{a}{\sigma} > 0$. Therefore $g(a)$ is a strictly monotonically decreasing function of a . Note that when we tune the parameters to optimize our tail bound, we always keep ϵ under 3 (otherwise the tail bound will be trivial). Since $e^{\frac{a}{\sigma}} \leq e^{\delta} \leq \epsilon + 1 < 4$ we also have $a < \sigma \ln 4$. Because $g(0) = 3 > 0$ and $g(\sigma \ln 4) = -2 \ln 4 < 0$ the equation $g(a) = 0$ has a unique root denoted as a_0 between $(0, \sigma \ln 4)$ given σ . We can compute a_0 numerically since there is no closed form solution for it. Recall that $f'(a) = \frac{a}{3}g(a)$. So within $(0, a_0)$, $f'(a) > 0$ and within $(a_0, \sigma \ln 4)$, $f'(a) < 0$. Thus we conclude that $f(a)$ is first monotonically increasing and begins to decrease after a reaches a_0 . Therefore $f(a)$ exhibits a maximal value achieved at a_0 which can be calculated numerically.

Combining the third constraint in the formulated problem, we obtain the following solution for the optimization problem. If $K + (\mu - 2^{-l})\tau \geq a_0\sigma$, then the maximization is achieved at $a = a_0$ and $\epsilon = e^{\frac{a}{\sigma}} - 1$. If, however, $K + (\mu - 2^{-l})\tau < a_0\sigma$, then, a must be smaller than a_0 due the third constraint. We know $f(a)$ is monotonically increasing when $a \in (0, a_0]$. Therefore we set a to be closest possible to a_0 under this constraint so that the objective is maximized: $a = \frac{K + (\mu - 2^{-l})\tau}{\sigma}$ and $\epsilon = e^{\frac{a}{\sigma}} - 1$.

Therefore, we obtain

Theorem 15 *For any $s < t$, let $\tau = t - s$ and let σ be the standard deviation bound of $b(s, t)$ as derived in Theorem 14 (RHS of the inequality),*

$$\Pr[D_{s,t}] \equiv \Pr[b(s, t) - \mu\tau > K] < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})},$$

where $a = \min(a_0, \frac{K + (\mu - 2^{-l})\tau}{\sigma})$, $\epsilon = e^{\frac{a}{\sigma}} - 1$, and a_0 is the unique root of $g(a) = 4 - e^{\frac{a}{\sigma}} - \frac{ae^{\frac{a}{\sigma}}}{2\sigma}$ for $a \in (0, \sigma \ln 4)$.

The computational complexity to obtain the overall bound of $\Pr[D]$ is $O(N)$, which is much smaller than that using the variant of the Chernoff bound, i.e., $O(n)$, when $N \ll n$, due to the following two facts. Like the bound from Theorem 13, this bound of $\Pr[D_{s,t}]$ is again shift-invariant, i.e., $\Pr[D_{s,t}] = \Pr[D_{s+\Delta, t+\Delta}]$. So it only needs to be computed once and multiplied by $n - \tau + 1$ to account for the overall bound on $\Pr[D]$. The other fact we should thank to is that when $\tau \geq 2^{l-1}N$ the standard deviation of σ keeps the same ($= \frac{\sqrt{N}}{2}$) no matter how large

τ is. Hence the root a_0 of $f(a)$ also keeps the same for all $2^{l-1}N \leq \tau \leq n$ because $g(a)$ is always the same. And so do the bounds of $\Pr[D_{s,t}]$ when $\tau > 2^{l-1}N$ (denoted as ρ). This means $\Pr[D] = \sum_{0 \leq t-s < 2^{l-1}N} \Pr[D_{s,t}] + \rho \sum_{2^{l-1}N \leq t-s \leq n} 1$.

So far we have two different tail bound on $\Pr[D]$ by Theorem 13 and 15 respectively. Carefully analyzing and comparing these two bounds, we found that the bound on $\Pr[D_{s,t}]$ from Theorem 13 is generally better but the one from Theorem 15 outperforms for some $t - s$ values. The detailed analysis is shown in Appendix A.5.2. This suggests a hybrid bound which is obtained when we apply the smaller bound between the bounds from Theorem 13 and Theorem 15 for each $\Pr[D_{s,t}]$ to achieve the tightest bound. In other words, when we denote the bounds from Theorem 13 and Theorem 15 on term $D_{s,t}$ as $\Omega_1(s, t)$ and $\Omega_2(s, t)$ respectively, the hybrid bound is

$$\Pr[D_{s,t}] \leq \min\{\Omega_1(s, t), \Omega_2(s, t)\}$$

The numerical results shown in Section 7.4.1 will provide some examples for comparison of these three bounds. We observe that although $\Omega_1(s, t)$ is better on the most of range $\Omega_2(s, t)$ is able to improve the overall bound of $\Pr[D]$ significantly.

7.4 Evaluation

In this section, we evaluate the operational performance of our design. We first present in Section 7.4.1 numerical examples of tail bounds derived in the previous section under a set of typical parameter configurations. Note that these tail bounds are derived for the worst case scenarios. In practice, as demonstrated by experiments on real-world traffic traces described in Section 7.4.2, typical distributions in Internet traffic are nowhere close to this worst case. This translates to much smaller values of the maximum queue length being observed in practice. While the analysis predicts that a queue with a few hundred slots would be required to achieve a negligibly small probability of overflow, in practice the queue length never exceeded 18 for experiments with hundreds of millions of counter increments.

Table 3: Probability of overflow when $N = 10^6$, $n = 10^{12}$, $\mu = 1/12$ and $l = 4$ bits

K	Theorem 13	Theorem 15	Hybrid
200	trivial (≥ 1)	trivial (≥ 1)	5.0×10^{-5}
300	1.4×10^{-6}	trivial (≥ 1)	2.4×10^{-14}
4,000	7.8×10^{-274}	1.4×10^{-10}	$\leq 2.2251 \times 10^{-308}$

Table 4: Probability of overflow when $N = 10^6$, $n = 10^{12}$, $\mu = 1/30$ and $l = 5$ bits

K	Theorem 13	Theorem 15	Hybrid
500	trivial (≥ 1)	trivial (≥ 1)	1.1×10^{-11}
3033	1.4×10^{-6}	trivial (≥ 1)	8.7×10^{-142}

7.4.1 Numerical examples of the tail bounds

In this section we present a set of numerical results computed from the tail bound theorems derived in the previous section using MATLAB 7.0. We only use a set of representative parameters; other parameter settings result in similar observations. Like before, we assume that there are $N = 1,000,000$ counters in our architecture. We also assume that the measurement/counting interval has $n = 10^{12}$ cycles (one counter increment per cycle as discussed in Section 2). This interval is about several hours long when we assume that each packet arrival triggers the increment to one counter and the link speed is 40 Gbps (OC-768), and it is longer (in terms of actual time) with lower link speeds. We construct two sets of numerical examples by setting the ratio of DRAM access speed to SRAM access speed μ to $\frac{1}{12}$ and $\frac{1}{30}$ respectively. Recall that μ has to be larger than 2^{-l} to make the queue stable. Therefore we set l to 4 and 5 bits respectively in above two examples, the aforementioned minimum SRAM counter size that is theoretically possible.

Table 3 shows the probabilities of queue overflow computed by the three aforementioned tail bounds (Theorem 13, Theorem 15, and the hybrid bound) given three different queue buffer sizes (200 slots, 300 slots and 4,000 slots). The size of each slot in this case is 20 bits ($= \log_2 1,000,000$). In the table "trivial" means that the resulting tail bound is larger than 1 so that it makes no sense. We observe that the bound from Theorem 13 is generally better than that from Theorem 15 (see the case of 4,000 slots to make it clear). But the bound from Theorem 15 is still valuable because

Table 5: Cost-benefit comparison for different schemes for the reference system with a million 64-bit counters, $\mu = 1/30$, and $K = 500$ slots.

	Naive	LCF	$LR(b)$	Ours
Counter memory	64Mb SRAM	9Mb SRAM 64Mb DRAM	9Mb SRAM 64Mb DRAM	5Mb SRAM 65Mb DRAM
Control memory	None	20Mb SRAM	2Mb SRAM	10Kb SRAM
Control logic	None	hardware heap	aggregated bitmap	FIFO queue
Implementation Complexity	Very low	High	Low	Very Low

it can improve the first bound significantly. For example, although the first bound is trivial in the case of 200 slots the resulting hybrid bound reaches a good level with the help of the second bound. It is clear that the hybrid bound is orders of magnitude better than any of separate bounds and achieves very desirable level when the buffer size is quite small (around 300 slots). We observe the similar result in Table 4. We also find that the bound from Theorem 15 can improve the bound from Theorem 13 more significantly when the service rate ($1/2^l$) is closer to the average arrival rate (μ) by comparing the second rows of the tables.

Table 5 compares the proposed scheme with the other three existing schemes, i.e., LCF in [111], $LR(b)$ in [102] and the naïve approach of implementing all counters in SRAM, in terms of memory (SRAM and DRAM) consumption and implementation complexity. We only show the data for the case of $\mu = 1/30$ due to interest of space. The data for the other case can be easily computed. Our scheme can be implemented using $5 + \epsilon$ (ϵ around 0.01) bits of SRAM per counter.⁶ When compared to the best one of the previous approaches, i.e., $LR(b)$, our scheme achieves a 2.2-fold reduction in SRAM usage. In addition, our scheme, which uses a simple FIFO queue as its control logic, is much easier to implement than $LR(b)$, which needs a pipelined implementation of a tree-like structure (called aggregated bitmap).

⁶In that table, our scheme uses 1 extra bit (in DRAM) per counter to remember the sign of the counter value as we described before.

7.4.2 Simulation using real-world Internet traffic

In this section, we evaluate our statistics counter architecture using real-world Internet traffic traces, with each packet arrival triggering one or more counter increments. The experimental results show that the maximum queue length over time is more than two orders of magnitude smaller than the values we set previously in Section 7.4.1 to obtain the desirable bound of the overflow probability. This reflects the fact that the analytical results derived for the worst case are not typically observed in practice.

For our evaluations, we would like to use long packet header traces to capture low probability events. Most publicly available traces are divided into small time scales and anonymized separately so that the small pieces cannot be merged into a big one. Fortunately, we were able to obtain two large traces that are publicly available for research purposes. They were collected at different locations in the Internet, namely University of Southern California (USC) and University of North Carolina (UNC) respectively. The trace from USC was collected at their Los Nettos tracing facility on Feb. 2, 2004 and the trace from UNC was collected on a 1 Gbps access link connecting to the campus to the rest of the Internet, on April 24, 2003. Both traces are quite large: the trace from USC has 120,773,099 packet headers and around 8.6 million flows; the trace from UNC has 198,944,706 packet headers and around 13.5 million flows.

We use the same parameter settings as in Section 7.4.1 where $N = 1,000,000$ and run the experiments for both traces with $l=4$ bits, $\mu = 1/12$ and $l = 5$ bits, $\mu = 1/30$, respectively. For each packet header in the trace we hash its flow label (i.e., \langle source IP address, destination IP address, source port, destination port, protocol \rangle) and the result is viewed as the index to the counter array. This operation is performed in a number of networking applications [133, 73, 75]. Table 6 shows the maximal and average buffer sizes during the experiments. For comparison we also compute the theoretical hybrid bounds of overflow for every experiment we run by setting the buffer size to the maximal observed buffer sizes in the above experiments. We find that all the resulting hybrid bounds are trivial (larger than 1). We also tune the buffer size and observe that buffer size need to be set to 154 (when SRAM counter is 4 bits each) and 299 (when SRAM counter is 5 bits each) in order to make the bound nontrivial. Both numbers are much larger than the maximum we observe

Table 6: Statistics of buffer size with single counter increment

Trace	SRAM counter size (in bits)	μ	Buffer Size	
			Max	Average
USC	4	1/12	21	1.6
	5	1/30	61	6.0
UNC	4	1/12	23	1.7
	5	1/30	72	7.0

Table 7: Statistics of buffer size with 4 counter increments

Trace	SRAM counter size (in bits)	μ	Buffer Size	
			Max	Average
USC	4	1/12	22	1.7
	5	1/30	75	6.6
UNC	4	1/12	30	1.8
	5	1/30	82	7.3

in the experiments. These facts implies two possibilities or their combination: i) the derived hybrid bound bounds the worst case and the real-world traffic does not exhibit worst-case behavior; ii) there is still considerable room for tightening this bound. We are investigating the second possibility (Section 8.2.3) as one of our future research directions.

In the next set of experiments, we let each packet arrival result in increments to multiple counters, which is typical in counting sketches or other applications such as [50, 33, 132], as we described before. Since this generates a longer counter increment sequence and hence may make the process more “bursty”, we would like to know its impact on the buffer size. Note that during each cycle we still only perform one increment. Using the same traces (USC and UNC) above, each packet arrival results in increments to 4 different counters, indexed by four independent hash functions computed over the flow label fields from the packet header. The statistics of the observed buffer size are shown in Table 7. We also repeat the previous calculations and comparisons for the theoretical hybrid bounds of overflow. We observe the same results as in the earlier study. In addition, compared with the statistics in Table 6 it seems the longer increment sequence caused by multiple counter increments per packet arrival does not impact the practical maximal buffer size much.

7.5 Conclusion

Supporting high-speed increments to a large number of counters using limited amounts of fast memory is the problem addressed in this work. Solutions proposed in recent works have used hybrid architectures where small counters in SRAM are incremented at high speed, and occasionally written back (“flushed”) to larger counters in DRAM. In this work, we present a novel hybrid SRAM/DRAM counter architecture that uses the optimal amount of fast memory, while minimizing the complexity of the counter management algorithm. Our design uses a small write-back buffer (in SRAM) that stores indices of the overflowed counters (to be flushed to DRAM) and an extremely simple randomized algorithm to statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up the write-back buffer even in the worst case. The statistical guarantee of the algorithm is proven through a combination of worst-case analysis for characterizing the worst case counter increment sequence and a new tail bound theorem for bounding the probability of filling up the write-back buffer. Experiments with real Internet traffic traces show that the actual queue lengths observed in practice are orders of magnitude smaller than the analytical bounds derived for the worst case.

CHAPTER VIII

CONTRIBUTION AND FUTURE WORK

8.1 Conclusion

Network measurement plays a crucial role in managing IP networks and understanding the properties of traffic traversing them. It suffers two fundamental problems in today's Internet: "too little data" due to the lack of importance in original Internet design and reinforced by best effort service model. This problem is reflected by the fact that the network can only generate some highly aggregated measurements about the traffic statistics of interest sometimes. The other problem is "too much data" which is due to the explosive growth of the Internet in size, speed and complexity. The ever-increasing network link speeds generate huge amounts of data for the post-mortem measurement and analysis, which challenges the limited communication, storage or processing resources in the instrumented network measurement devices. Both problems necessitate the new advanced software and hardware technologies. In this dissertation, we propose a novel *statistical algorithmic* approach to alleviate these challenges.

Our work mainly consists of three complementary methodologies and several concrete schemes guided by these methodologies:

Network data inference with multiple data sources studies how to fully utilize the existing information collected from multiple independent measurement infrastructures in the network to alleviate "too little data" problem. This methodology is highly likely to generate better estimation accuracy than obtainable from a single infrastructure since the more information may be provided. Another important advantage is to identify accidental errors occurring in measurement results ("dirty data"). We find that although different measurement infrastructures work independently they may collect some common information within the full spectrum. This information redundancy can be compared and verified between different measurement results and then the errors can be identified and removed.

Using this methodology, we devise a set of methods for robust traffic matrix estimation and detection of dirty data by correlating SNMP and sampled NetFlow data, allowing for much more accurate estimation than obtainable from either alone, . Our techniques are practically important and useful since both SNMP and NetFlow are now widely supported by vendors and deployed in most of the operational IP networks. We also explore the possibilities to model the temporal correlation existing in traffic matrices over time and combine it with other spacial information to further improve the performance of traffic matrix estimation. We find that our linear predictive model is able to produce reasonably good prediction result but unfortunately combining it with sampled NetFlow and SNMP data only generates very marginal improvement.

Network data streaming processes a long data stream (e.g., network traffic) in a single pass using a small working memory to answer a class of queries regarding the stream. It has been recognized as a potential alternative solution to sampling for “too much data” problem. Using this methodology we design several data streaming algorithms for estimation of some important traffic statistics that have traditionally been considered hard to monitor at high-speed network links and routers:

- **Traffic and flow matrix estimation:** In this work we propose a novel data streaming algorithm that can process traffic stream and produce traffic digests that are orders of magnitude smaller than the original traffic stream. By correlating the digests collected at any OD pair, the volume of traffic flowing between OD pairs can be accurately determined. We also establish principles and techniques for optimally combining this streaming method with sampling, when sampling is necessary due to stringent resource constraints. In addition, we propose another data streaming algorithm that estimates a *flow matrix*, a finer-grained characterization than traffic matrix. A flow matrix is concerned with not only the total traffic between an OD pair (traffic matrix), but also how it splits into flows of various sizes.
- **Detection of super sources and destinations:** Super sources and destinations are defined as sources or destinations that have communicated with a large number of distinct destinations or sources during a short time interval. We propose two solutions to this problem. Their designs are based on two distinct novel methods to collaborate sampling and data streaming. Sampling and data streaming are often suitable for capturing different and complementary

regions of the information spectrum, and a close collaboration between them is a desirable way to recover the complete information. This insight has been subsequently applied to some of our other works and is expected to be useful for solving many other network measurement and monitoring problems. The “filtering after sampling” scheme builds on the standard hash-based flow sampling algorithm. Its main innovation is that the sampled traffic is further filtered by a data streaming module before entering the back-end hash table for information recording. This allows for much higher sampling rate (hence much higher accuracy) than achievable with standard hash-based flow sampling. The “separation of counting and identity gathering” scheme is more sophisticated but offers higher accuracy. It combines the power of data streaming in efficiently estimating quantities associated with a given identity, with the power of sampling in collecting a list of candidate identities.

- **Finding global iceberg over distributed data sets:** A global iceberg is defined as an item whose frequency of occurrence is above a certain threshold, over distributed massive data sets. In this work, we propose two accurate and efficient solutions to this problem with modest communication cost: a sampling-based scheme and a counting-sketch-based scheme. In the sampling-based scheme, each node samples a list of data items along with their frequency counts in the local data bag and sends them to the server, which for each distinct item, aggregates its sampled frequency counts scaled by the inverse of their respective sampling rates to obtain an estimate of its total frequency. And our second solution, the sketch-based approach, detects icebergs in a much more communication-efficient way than the sampling-based solution. We let each node not only summarize its data into a counting sketch, but also sample a small percentage of identities of the items, and send both to the server. For each sampled identity, the server will use it to query all the collected counting sketches and add up the approximate counts to obtain an estimate of the total frequency count of the item.

Hardware enhancement is another way to alleviate the challenge imposed by the ever-developing Internet. In this dissertation, we focus on a fundamental tool of measurement: counting. This tool need maintain a large number (e.g., millions) of counters economically and accurately at high speed. We present a novel hybrid SRAM/DRAM counter architecture that consumes much less SRAM and

has a much simpler design of the CMA than previous work. We show, in fact, that our design is optimal in the sense that for a given difference of access rates between SRAM and DRAM, our design uses the theoretically minimum number of bits per counter in SRAM. Our design uses a small FIFO write-back buffer (in SRAM) that stores indices of the overflowed counters (to be flushed to DRAM) and an extremely simple randomized algorithm to statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up the write-back buffer even in the worst case. The statistical guarantee of the algorithm is proven using a combination of worst case analysis for characterizing the worst case counter increment sequence and some tail bound theorems for bounding the probability of filling up the write-back buffer.

8.2 *Future Work*

In this section, we identify some research directions for future work located in these themes. Whenever possible, we try to emphasize the main difficulties and possible solutions to address the proposed problems.

8.2.1 **Network Data Inference with Multiple Data Sources**

Besides traffic matrix which has been well studied in networking research community, *delay matrix* is another important performance matrix which represents of the average delay of every OD flow. We define it as an $m \times n$ matrix \mathbf{D} where each column represents the vector of the link delays of the corresponding OD flow. Our goal could be the overall delay of each OD flow, i.e., computing $sum(\mathbf{D})$ instead of computing \mathbf{D} directly where the function $sum(.)$ treats the columns of the input as vectors, returning a row vector of the sums of each column. Again by the routing matrix \mathbf{R} we can immediately mark part of elements in \mathbf{D} as 0's directly which would significantly reduce the computational complexity. The resulting \mathbf{D} after marking can be easily computed by $D \times R$ where \times is the element-by-element matrix multiplication.

Here again we want to fully utilize all the data sources available in a large tier-1 ISP network to resolve this problem. There are three data sources so far in AT&T backbone: (i) the column vector of the estimated average per-link delay $\mathbf{Z} = \{z_1, z_2, \dots, z_m\}$ of layer 3. This could be computed from the combined information of the average link utilization and the drop probability in the recent

SNMP polling interval using a proper queuing model; (ii) the routing matrix and (iii) the vector of the estimated end-to-end delay for each OD flow. Again this can be obtained by active probing. Then the resulting formulated problem is a linear system based on the above information because an end-to-end delay of a packet is exactly the sum of the delays of all the links on its traversal path. Some statistical methods need to be carefully designed to resolve the problem.

8.2.2 Network Data Streaming

Traditional data streaming algorithms are usually designed for a single specific goal. However, in future Internet environments, we may have to face situations in which several different goals have to be achieved at the same time. In such situations, one can certainly use independent streaming modules designed for each of the goals. However, as the number of goals increases the resource requirement of such a strategy will quickly become prohibitive. Even for achieving just two goals, the tight constraints that are usually imposed in practical applications make such an approach extremely wasteful. One of our future direction is to study how streaming algorithms designed for different goals or for some unlocalized parts of data can share information and data structures among themselves so that overall storage requirement can be greatly reduced.

8.2.3 Statistics Counter Architecture

In Chapter 7, we present a novel hybrid SRAM/DRAM counter architecture that consumes much less SRAM and has a much simpler design of the scheduler than previous approaches. Our design implements the CMA as a small FIFO buffer (in SRAM) that stores addresses of the overflowed counters (to be flushed to DRAM) and an extremely simple randomized algorithm to statistically guarantee that SRAM counters do not overflow in bursts large enough to fill up the FIFO buffer even in the worst case. We also prove a very small FIFO buffer (say hundreds of slots) is enough to handle the overflow from millions of SRAM counters. However, we find that there is considerable room for improvement on this tail bound using intuitive arguments and simulations. Our future direction is to develop new and general stochastic analysis techniques, in particular the combination of stochastic ordering and large deviation theory, that can not only significantly improve this tail bound, but also be widely applicable to some other network and queuing analysis problems. More importantly, we would like to fully understand the deep mathematical structure underneath this tail

bound problem, and with this understanding, more variants of such techniques can be discovered to solve many other network and queuing analysis problems that look totally different, but indeed share some common mathematical structures in a subtle way.

APPENDIX A

A.1 Derivation in Chapter 3

We start to derive $\Pr[L_0 = l | L_2 = 0]$ as follows. We first use Bayes formula to derive the following equations.

$$\begin{aligned}
 \Pr[L_0 = l | L_2 = 0] &= \sum_{j=0}^{\min\{l, r_i z/s\}} \Pr[L_0 = l, L_1 = j | L_2 = 0] \\
 &= \sum_{j=0}^{\min\{l, r_i z/s\}} \Pr[L_0 = l | L_1 = j] \Pr[L_1 = j | L_2 = 0] \\
 &= \sum_{j=0}^{\min\{l, r_i z/s\}} \frac{\Pr[L_0 = l, L_1 = j]}{\sum_{m=j}^{\infty} \Pr[L_1 = j, L_0 = m]} \frac{\Pr[L_1 = j, L_2 = 0]}{\sum_{j=0}^{r_i z/s} \Pr[L_1 = j, L_2 = 0]} \\
 &= \sum_{j=0}^{\min\{l, r_i z/s\}} \frac{\binom{l}{j} r_i^j (1 - r_i)^{l-j}}{\sum_{m=j}^{\infty} \binom{m}{j} r_i^j (1 - r_i)^{m-j}} \frac{1 - \frac{j}{r_i z/s}}{\sum_{j=0}^{r_i z/s} (1 - \frac{j}{r_i z/s})} \\
 &= \sum_{j=0}^{\min\{l, r_i z/s\}} \frac{\binom{l}{j} (1 - r_i)^{l-j}}{\sum_{m=j}^{\infty} \binom{m}{j} (1 - r_i)^{m-j}} \frac{2 - \frac{2j}{r_i z/s}}{r_i z/s + 1} \\
 &= \sum_{j=0}^{\min\{l, r_i z/s\}} \binom{l}{j} r_i^{j+1} (1 - r_i)^{l-j} \frac{2 - \frac{2j}{r_i z/s}}{r_i z/s + 1}
 \end{aligned}$$

Recall that r_i is the sampling rate of the sampled NetFlow and z is the threshold of the smart sampling. The last equation is due to the binomial theorem for negative integers. Given the fact that it is sampled at all (mean value is equal to 0), we can numerically compute its MSE based on the above pdf .

A.2 Proofs in Chapter 4

A.2.1 Preliminaries

Given a random variable X , we denote $X - E[X]$ as X^c . X^c is often referred to as a *centered random variable*. It can easily be verified that $E[X^c] = 0$ and $Var[X] = Var[X^c]$.

Next, we state without proof some lemmas and facts which will be used for proving Theorem 1. Here T is an arbitrary set of distinct packets hashed to a bitmap.

Lemma 5 (proved in [125])

$$\begin{aligned}
(I) \quad & E[U_T] = be^{-t_T} \\
(II) \quad & Var(U_T) = be^{-t_T}(1 - (1 + t_T)e^{-t_T}) \\
(III) \quad & E[D_T] = |T| + \frac{e^{t_T} - t_T - 1}{2} \\
(IV) \quad & Var(D_T) = b(e^{t_T} - t_T - 1) \\
(V) \quad & D_T^c \approx -e^{t_T} U_T^c
\end{aligned}$$

Remark: We can see that from Lemma 5(III) that D_T is a biased estimator of $|T|$ with bias $\frac{e^{t_T} - t_T - 1}{2}$. However, this bias is very small compared to the parameter $|T|$, since $|T|$ is typically in millions and the t_T (load factor) we are working with is no more than 0.7, resulting in a bias no more than 0.16. We omit this bias from all following derivations since this omission results in negligible differences in the values of affected terms (confirmed by Monte-Carlo simulation), and the final result would be unnecessarily complicated without this omission. With this understanding, we will use equal signs instead of approximation signs when the only approximation involved is the omission of the bias.

Next, we prove a lemma that will be used extensively in the proof of Theorem 1.

Lemma 6 Let Y and Z be two sets of packets and $Y \cap Z = \emptyset$. Then we have (i) $E[U_Y^c U_Z^c] = 0$ and (ii) $E[U_{Y \cup Z}^c U_Y^c] \approx e^{-t_Z} Var[U_Y]$.

Proof: (i) Note that U_Y^c and U_Z^c are independent random variables since $Y \cap Z = \emptyset$. Therefore $E[U_Y^c U_Z^c] = E[U_Y^c]E[U_Z^c] = 0$.

(ii) Recall that B_T denotes the corresponding bitmap after “inserting” T into an empty array, and U_T is the number of 0’s in the resulting array. Then U_T^c can be thought of as “extra” 0’s in addition to the average (this “extra” can go negative). Then

$$\begin{aligned}
E[U_{Y \cup Z}^c U_Y^c] &= \sum_{i=-\infty}^{\infty} E[U_{Y \cup Z}^c U_Y^c | U_Y^c = i] Pr[U_Y^c = i] \\
&= \sum_{i=-\infty}^{\infty} i E[U_{Y \cup Z}^c | U_Y^c = i] Pr[U_Y^c = i] \quad (*)
\end{aligned}$$

We would like to show that $E[U_{Y \cup Z}^c | U_Y^c = i] \approx ie^{-t_Z}$, denoted as (**). It suffices to prove that for any particular Y^* (a constant set) such that $U_{Y^*} - E[U_Y] = i$ and $Y^* \cap Z = \emptyset$, we have

$E[U_{Y^* \cup Z}] = (i + be^{-t_Y})e^{-t_Z}$, denoted as (***). This is because, when (***) holds, we have

$$\begin{aligned} E[U_Y^c \cup Z | U_Y^c = i] &= E[U_Y \cup Z | U_Y^c = i] - E[U_Y \cup Z] \\ &= E[U_{Y^* \cup Z}] - E[U_Y \cup Z] \\ &\approx (i + be^{-t_Y})e^{-t_Z} - E[U_Y \cup Z] \end{aligned}$$

Since $E[U_Y \cup Z] = be^{-t_Y}e^{-t_Z}$, we have $E[U_Y^c \cup Z | U_Y^c = i] \approx ie^{-t_Z}$.

It remains to prove (***). Note that in the bitmap B_{Y^*} approximately $be^{-t_Y} + i$ entries (or $e^{-t_Y} + i/b$ percentage) are 0 since $U_{Y^*}^c = i$. Since in B_Z by definition U_Z entries (or U_Z/b percentage) are 0, in $B_{Y^* \cup Z}$ on the average $(e^{-t_Y} + i/b)(U_Z/b)$ percentage of entries is 0. In other words, $E[U_{Y^* \cup Z} | U_Z] \approx U_Z(e^{-t_Y} + i/b)$. Therefore,

$$\begin{aligned} E[U_{Y^* \cup Z}] &= E[E[U_{Y^* \cup Z} | U_Z]] \\ &= E[U_Z(e^{-t_Y} + i/b)] = e^{-t_Z}(e^{-t_Y} + i/b) \end{aligned}$$

which is exactly (***). Now applying (**) to (*) we have

$$\begin{aligned} E[U_Y^c \cup Z U_Y^c] &= \sum_{i=-\infty}^{\infty} i^2 e^{-t_Z} \Pr[U_Y^c = i] \\ &= e^{-t_Z} \text{Var}[U_Y^c] = e^{-t_Z} \text{Var}[Y] \end{aligned}$$

Note however that $E[U_Y^c U_Z^c] = 0$ in general does not hold where Y and Z are not disjoint. The value of $E[U_Y^c U_Z^c]$ is characterized in the following lemma. We omit its proof since it is similar to that of lemma 6 (ii).

Lemma 7 $E[U_Y^c U_Z^c] \approx e^{-t(Y-Z)-t(Z-Y)} \text{Var}[U_Y \cap Z]$

A.2.2 Proof of Theorem 1

Proof: To simplify the notations in the following proof, we use R, S, N, \hat{N} to replace $T_i, T_j, TM_{i,j}$, and $\widehat{TM}_{i,j}$ respectively. Recall that our estimator becomes $\hat{N} = D_R + D_S - D_{R \cup S}$. We

have

$$\begin{aligned}
Var(\hat{N}) &= Var(\hat{N}^c) \\
&= E[(D_R^c + D_S^c - D_{R \cup S}^c)^2] \\
&= E[(D_R^c)^2] + E[(D_S^c)^2] + E[(D_{R \cup S}^c)^2] \\
&\quad + 2E[D_R^c D_S^c] - 2E[D_R^c D_{R \cup S}^c] \\
&\quad - 2E[D_S^c D_{R \cup S}^c]
\end{aligned}$$

These six terms are calculated as follows. By Lemma 5, we have

$$E[(D_R^c)^2] = Var(D_R) = b(e^{t_R} - t_R - 1) \quad (34)$$

$$E[(D_S^c)^2] = Var(D_S) = b(e^{t_S} - t_S - 1) \quad (35)$$

$$\begin{aligned}
E[(D_{R \cup S}^c)^2] &= Var(D_{R \cup S}) \\
&= b(e^{t_{R \cup S}} - t_{R \cup S} - 1)
\end{aligned} \quad (36)$$

$$\begin{aligned}
E[D_R^c D_S^c] &\approx E[e^{t_R} U_R^c e^{t_S} U_S^c] && \text{by Lemma 5} \\
&\approx e^{t_R + t_S - t_{R \cap S} - t_{S \cap R}} Var(U_{R \cap S}) && \text{by Lemma 7} \\
&= b(e^{t_{R \cap S}} - (1 + t_{R \cap S}))
\end{aligned} \quad (37)$$

$$\begin{aligned}
E[D_R^c D_{R \cup S}^c] &\approx E[e^{t_R} U_R^c e^{t_{R \cup S}} U_{R \cup S}^c] && \text{by Lemma 5} \\
&= e^{t_R + t_{R \cup S}} E[U_R^c U_{R \cup (S-R)}^c] \\
&\approx e^{t_R + t_{R \cup S}} e^{-t_{S-R}} Var[U_R] && \text{by Lemma 6} \\
&= b(e^{t_R} - (1 + t_R))
\end{aligned} \quad (38)$$

Similarly we have

$$E[D_S^c D_{R \cup S}^c] = b(e^{t_S} - (1 + t_S)) \quad (39)$$

Adding up these terms (Equation. 34-39), we get

$$Var(\hat{N}) = b(2e^{t_{R \cap S}} + e^{t_{R \cup S}} - e^{t_R} - e^{t_S} - t_{R \cap S} - 1)$$

A.2.3 Proof of Theorem 2

Proof: It is not hard to verify that our estimator $\hat{X} = \hat{N}/p$ is approximately unbiased, i.e., $E[\hat{X}] \approx X$, provided that we consider \hat{N} as unbiased (discussed in Appendix A.2.1). Therefore, we have

$$\begin{aligned} Var(\hat{X}) &= E[(\frac{\hat{N}}{p} - X)^2] = \frac{1}{p^2} E[(\hat{N} - pX)^2] \\ &= \frac{1}{p^2} E[(\hat{N} - N + N - pX)^2] \\ &= \frac{1}{p^2} E[(\hat{N} - N)^2] + \frac{1}{p^2} E[(N - pX)^2] \\ &\quad + \frac{2}{p^2} E[(\hat{N} - N)(N - pX)] \end{aligned}$$

We first prove that $E[(\hat{N} - N)(N - pX)] \approx 0$, i.e., $\hat{N} - N$ and $N - pX$ are approximately orthogonal¹. To show this, we note that for any fixed value n , $\widehat{N}(n) - n$ and $n - pX$ are independent random variables because they represent errors from two independent measurements. Here $\widehat{N}(n)$ denotes the value of the estimator when the actual number is n . Therefore $E[(\widehat{N}(n) - n)(n - pX)] = E[\widehat{N}(n) - n]E[n - pX] = (E[\widehat{N}(n)] - n)(n - E[pX]) \approx 0$ since $\widehat{N}(n)$ is approximately unbiased (discussed in Appendix A.2.1). Thus $E[(\hat{N} - N)(N - pX)] = E[E[(\hat{N} - N)(N - pX)|N = n]] \approx 0$. Therefore, we have $Var(\hat{X}) = \frac{1}{p^2} E[(\hat{N} - N)^2] + \frac{1}{p^2} E[(N - pX)^2]$. We claim that $E[(\hat{N} - N)^2] \approx E[(\hat{N} - pX)^2]$, denoted as (*). To prove this, note that with high probability, N does not fluctuate too far away its mean pX due to large deviation theory. Therefore we obtain $E[(\hat{N} - N)^2] \approx E[(\hat{N} - pX)^2]$ using mean value approximation. Finally, since N is a binomial random variable, we have $E[(N - pX)^2] = Var(N) = Xp(1 - p)$, denoted as (**). Combining (*), (**), and Theorem 1, we obtain the variance of \hat{X} as approximately

$$\frac{b}{p^2} \left((e^{\frac{Tp}{b} - \frac{Xp}{2b}} - e^{\frac{Xp}{2b}})^2 + e^{\frac{Xp}{b}} - \frac{Xp}{b} - 1 \right) + \frac{X(1 - p)}{p}$$

A.2.4 H_3 hash function

Each hash function in H_3 class [21] is a linear transformation $B^T = QA^T$ that maps a w -bit binary string $A = a_1a_2 \cdots a_w$ to an r -bit binary string $B = b_1b_2 \cdots b_r$ as follows:

¹Note that they are dependent since N is a random variable.

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_r \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1w} \\ q_{21} & q_{22} & \cdots & q_{2w} \\ \cdots & \cdots & \cdots & \cdots \\ q_{r1} & q_{r2} & \cdots & q_{rw} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_w \end{pmatrix}$$

Here a k -bit string is treated as a k -dimensional vector over $\text{GF}(2) = \{0,1\}$ and T stands for transposition. Q is a $r \times w$ matrix defined over $\text{GF}(2)$ and its value is fixed for each hash function in H_3 . The multiplication and addition in $\text{GF}(2)$ is boolean AND (denoted as \circ) and XOR (denoted as \oplus), respectively. Each bit of B is calculated as:

$$b_i = (a_1 \circ q_{i1}) \oplus (a_2 \circ q_{i2}) \oplus \cdots \oplus (a_w \circ q_{iw}) \quad i = 1, 2, \dots, r$$

Since computation of each output bit goes through $\log_2 w$ stages of Boolean circuitry [127], and all output bits can be computed in parallel, it can finish well within 10ns.

A.3 Proofs in Chapter 5

A.3.1 Proof of Lemma 1

Proof: Let us consider a series of indicator variables $Y_i, i = 1, \dots, N_s$

$$Y_i = \begin{cases} 1 & \text{with } p_i, \\ 0 & \text{with } 1 - p_i. \end{cases}$$

Here p_i is the probability that $G(h(f_i)) = 0$, which is equal to $\frac{u_i}{w}$ where u_i is the number of “0”s in the array G when the i^{th} sampled flow of the source s arrives at G . Then \widehat{N}_s can be rephrased by $\sum_{i=1}^{N_s} \frac{Y_i}{p_i}$.

$$E[\widehat{N}_s] = \sum_{i=1}^{N_s} \frac{E[Y_i]}{p_i} = \sum_{i=1}^{N_s} \frac{1 \times p_i + 0 \times (1 - p_i)}{p_i} = \sum_{i=1}^{N_s} 1 = N_s$$

Therefore \widehat{N}_s is an unbiased estimator.

A.3.2 Proof of Theorem 3

Proof: Because \widehat{N}_s is an unbiased estimator (Lemma 1, we have $E[\widehat{F}_s] = E[E[\widehat{F}_s|N_s]] = E[E[\widehat{N}_s/p|N_s]] = \frac{E[N_s]}{p}$. Because N_s is a binomial random variable, we have $E[N_s] = F_s p$. Therefore $\frac{E[N_s]}{p} = F_s$, which proves that \widehat{F}_s is an unbiased estimator.

A.3.3 Proof of Theorem 4

We first introduce a useful lemma before giving the proof of Theorem 4.

Lemma 8 $Var[\widehat{N}_s] = \sum_{i=1}^{N_s} \frac{w-u_i}{u_j}$

Proof:

$$Var[\widehat{N}_s] = \sum_{i=1}^{N_s} \frac{Var[Y_i]}{p_i^2} = \sum_{i=1}^{N_s} \frac{p_i(1-p_i)}{p_i^2} = \sum_{i=1}^{N_s} \frac{w-u_i}{u_i}$$

where $p_i = \frac{u_i}{w}$ and Y is modeled as a Bernoulli random variable.

Now we prove Theorem 4 as follows:

Proof:

$$\begin{aligned} Var(\widehat{F}_s) &= \frac{1}{p^2} E[(\widehat{N}_s - pF_s)^2] = \frac{1}{p^2} E[(\widehat{N}_s - N_s + N_s - pF_s)^2] \\ &= \frac{1}{p^2} E[(\widehat{N}_s - N_s)^2] + \frac{1}{p^2} E[(N_s - pF_s)^2] \\ &\quad + \frac{2}{p^2} E[(\widehat{N}_s - N_s)(N_s - pF_s)] \end{aligned}$$

We first prove that $E[(\widehat{N}_s - N_s)(N_s - pF_s)] = 0$, i.e., $\widehat{N}_s - N_s$ and $N_s - pF_s$ are approximately orthogonal². To show this, we note that for any fixed value n , $\widehat{N}_s(n) - n$ and $n - pF_s$ are independent random variables because they represent errors from two independent measurements. Here $\widehat{N}_s(n)$ denotes the value of the estimator when the actual number is n . Therefore $E[(\widehat{N}_s(n) - n)(n - pF_s)] = E[\widehat{N}_s(n) - n]E[n - pF_s] = (E[\widehat{N}_s(n)] - n)(n - E[pF_s]) = 0$ since $\widehat{N}_s(n)$ is unbiased by Lemma 1. Thus $E[(\widehat{N}_s - N_s)(N_s - pF_s)] = E[E[(\widehat{N}_s - N_s)(N_s - pF_s)|N_s = n]] = 0$. Therefore, we have $Var(\widehat{F}_s) = \frac{1}{p^2} E[(\widehat{N}_s - N_s)^2] + \frac{1}{p^2} E[(N_s - pF_s)^2]$. We claim that $E[(\widehat{N}_s - N_s)^2] \approx E[(\widehat{N}_s - pX)^2]$, denoted as (*). To prove this, note that with high probability, N_s does not fluctuate too far away its mean pF_s due to large deviation theory. Therefore we obtain $E[(\widehat{N}_s - N_s)^2] \approx E[(\widehat{N}_s - pF_s)^2]$ using mean value approximation. Finally, since N_s is a binomial random variable, we have $E[(N_s - pF_s)^2] = Var(N_s) = F_s p(1 - p)$, denoted as (**). Combining (*), (**), and Lemma 1, we finish the proof.

²Note that they are dependent since N_s is a random variable.

A.3.4 Proof of Theorem 5

We first introduce some preliminaries before giving the proof of Theorem 5. Given a random variable X , we define X^c as $X - E[X]$. X^c is often referred to as a *centered random variable*. It can easily be verified that $E[X^c] = 0$ and $Var[X] = Var[X^c]$. Assuming T is an arbitrary set of distinct packets hashed to a bitmap, We have the following lemma.

Lemma 9 (proved in [125])

$$\begin{aligned}
 (I) \quad & E[U_T] = be^{-t_T} \\
 (II) \quad & Var(U_T) = be^{-t_T}(1 - (1 + t_T)e^{-t_T}) \\
 (III) \quad & E[D_T] = |T| + \frac{e^{t_T} - t_T - 1}{2} \\
 (IV) \quad & Var(D_T) = b(e^{t_T} - t_T - 1) \\
 (V) \quad & D_T^c \approx -e^{t_T} U_T^c
 \end{aligned}$$

Remark: We can see that from Lemma 5(III) that D_T is a biased estimator of $|T|$ with bias $\frac{e^{t_T} - t_T - 1}{2}$. However, this bias is very small compared to the parameter $|T|$, since $|T|$ is typically in tens and the t_T (load factor) we are working with is no more than 0.5 on average, resulting in a bias no more than 0.07. We omit this bias from all following derivations since this omission results in negligible differences in the values of affected terms (confirmed through Monte-Carlo simulation), and the final result would be unnecessarily complicated without this omission. With this understanding, we will use equal signs instead of approximation signs when the only approximation involved is the omission of the bias.

Next, we provide a lemma that will be used extensively in the proof of Theorem 5. We omit its proof here due to limit of space. The related proof can be found in [133]

Lemma 10 $E[U_Y^c U_Z^c] \approx e^{-t(Y-Z)} e^{-t(Z-Y)} Var[U_Y \cap Z]$

We now prove Theorem 5 in the following.

Proof: We have $Var[\widehat{F}_A^s] = E[(D_{T_1}^c + D_{T_2}^c + D_{T_3}^c - D_{T_1 \cup T_2}^c - D_{T_1 \cup T_3}^c - D_{T_2 \cup T_3}^c + D_{T_1 \cup T_2 \cup T_3}^c)^2] = \sum_{i=1}^3 E[(D_{T_i}^c)^2] + \sum_{1 \leq i_1 < i_2 \leq 3} E[(D_{T_{i_1} \cup T_{i_2}}^c)^2] + E[(D_{T_1 \cup T_2 \cup T_3}^c)^2] + 2 \sum_{1 \leq i_1 < i_2 \leq 3} E[D_{T_{i_1}}^c D_{T_{i_2}}^c] +$

$$2 \sum_{i=1}^3 E[D_{T_1 \cup T_2 \cup T_3}^c D_{T_i}^c] - 2 \sum_{i=1}^3 \sum_{1 \leq i_1 < i_2 \leq 3} E[D_{T_i}^c D_{T_{i_1} \cup T_{i_2}}^c] - 2 \sum_{1 \leq i_1 < i_2 \leq 3} E[D_{T_1 \cup T_2 \cup T_3}^c D_{T_{i_1} \cup T_{i_2}}^c] + 2 \sum_{i=1}^3 \sum_{1 \leq i_1 < i_2 \leq 3, i_1 \neq i, i_2 \neq i} E[D_{T_i \cup T_{i_1}}^c D_{T_i \cup T_{i_2}}^c]$$

In the following we derive all the items of the RHS. According to the lemma 9, we have $E[(D_{T_i}^c)^2] = \text{Var}[D_{T_i}] = m(e^{t_{T_i}} - t_{T_i} - 1)$ for $i = 1, 2, 3$, $E[(D_{T_i \cup T_j}^c)^2] = \text{Var}[D_{T_i \cup T_j}] = m(e^{t_{T_i \cup T_j}} - t_{T_i \cup T_j} - 1)$ when $i \neq j$ and $E[(D_{T_1 \cup T_2 \cup T_3}^c)^2] = \text{Var}[D_{T_1 \cup T_2 \cup T_3}] = m(e^{t_{T_1 \cup T_2 \cup T_3}} - t_{T_1 \cup T_2 \cup T_3} - 1)$.

Now we let A, B, C denote any permutation of T_1, T_2, T_3 . According to lemma 10

$$\begin{aligned} E[D_{A \cup B}^c D_{A \cup C}^c] &\approx e^{t_{A \cup B} + t_{A \cup C}} \text{Var}[U_{A \cup B}^c U_{A \cup C}^c] \\ &\approx e^{t_{A \cup B} + t_{A \cup C} - t_{A \cup B - A \cup C} - t_{A \cup C - A \cup B}} \text{Var}[U_{(A \cup B) \cap (A \cup C)}] \\ &= m e^{t_{A \cup (B \cap C)}} (1 - (1 + t_{A \cup (B \cap C)}) e^{-t_{A \cup (B \cap C)}}) \\ &= m(e^{t_{A \cup (B \cap C)}} - t_{A \cup (B \cap C)} - 1) \end{aligned}$$

Still by lemma 10

$$\begin{aligned} E[D_A^c D_{A \cup B \cup C}^c] &\approx e^{t_A + t_{A \cup B \cup C}} \text{Var}[U_A^c U_{A \cup B \cup C}^c] \\ &\approx e^{t_A + t_{A \cup B \cup C} - t_{A - A \cup B \cup C} - t_{A \cup B \cup C - A}} \text{Var}[U_A] \\ &= m e^{t_A} (1 - (1 + t_A) e^{-t_A}) = m(e^{t_A} - t_A - 1) \\ E[D_{A \cup B}^c D_{A \cup B \cup C}^c] &= m(e^{t_{A \cup B}} - t_{A \cup B} - 1) \\ E[D_A^c D_B^c] &= m(e^{t_{A \cap B}} - t_{A \cap B} - 1) \\ E[D_A^c D_{B \cup C}^c] &= m(e^{t_{A \cap (B \cup C)}} - t_{A \cap (B \cup C)} - 1) \\ E[D_A^c D_{A \cup B}^c] &= m(e^{t_A} - t_A - 1) \end{aligned}$$

Thus by now we figure out all the items and it is easy to see the theorem holds.

A.4 Proofs in Chapter 6

A.4.1 Proof of Theorem 7

Proof: We set $\lambda = a/\sigma$ so that $0 \leq \lambda \leq \delta$. Then

$$E[e^{\lambda W_i}] = \sum_{k=0}^{\infty} E\left[\frac{\lambda^k}{k!} W_i^k\right] = 1 + \frac{\lambda^2}{2} \sigma_i^2 + \sum_{k=3}^{\infty} \frac{\lambda^k}{k!} E[W_i^k]$$

Since $|W_i^k| \leq \theta^{k-2} W_i^2$, we have

$$\mathbb{E}[W_i^k] \leq \mathbb{E}[|W_i^k|] \leq \theta^{k-2} \mathbb{E}[W_i^2] = \theta^{k-2} \sigma_i^2$$

For $k \geq 3$ we have $\frac{2}{k!} \leq \frac{1}{3(k-2)!}$, so

$$\begin{aligned} \mathbb{E}[e^{\lambda W_i}] &\leq 1 + \frac{\lambda^2}{3} \sigma_i^2 + \frac{\lambda^2}{6} \sigma_i^2 \left[1 + \sum_{k=3}^{\infty} \frac{(\theta \lambda)^{k-2}}{(k-2)!} \right] \\ &= 1 + \frac{\lambda^2}{3} \sigma_i^2 + \frac{\lambda^2}{6} \sigma_i^2 e^{\lambda \theta} \end{aligned}$$

Note that we have chosen δ to satisfy $e^{\theta \delta} = 1 + \epsilon$. Since $\lambda \leq \delta$,

$$\begin{aligned} \mathbb{E}[e^{\lambda W_i}] &\leq 1 + \frac{\lambda^2}{3} \sigma_i^2 + \frac{\lambda^2}{6} \sigma_i^2 (1 + \epsilon) \\ &= 1 + \frac{\lambda^2}{2} \sigma_i^2 \left(1 + \frac{\epsilon}{3} \right) \\ &< \exp \left[\frac{\lambda^2}{2} \sigma_i^2 \left(1 + \frac{\epsilon}{3} \right) \right] \end{aligned}$$

This inequality holds for all i , $1 \leq i \leq m$. So,

$$\mathbb{E}[e^{\lambda W}] = \prod_{i=1}^m \mathbb{E}[e^{\lambda W_i}] < \exp \left[\frac{\lambda^2}{2} \sigma^2 \left(1 + \frac{\epsilon}{3} \right) \right]$$

and thus, $\Pr[W > a\sigma] \leq \mathbb{E}[e^{\lambda W}] e^{-\lambda a\sigma} < e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$.

A.4.2 Proof of Theorem 8

Proof: Let S be the set of nodes at which I appears. Then $|S| = f_I$. For each i , $1 \leq N$, let y_i be the random binary variable that takes the value 1 if B_i returns “yes” to query I and 0 otherwise. Then y_1, \dots, y_N are independent variables and $\alpha = y_1 + \dots + y_N$. Also, for every i , $1 \leq i \leq N$, $\Pr[y_i = 1] = q$ if $i \notin S$ and $\Pr[y_i = 1] = 1 - (1 - p)(1 - q) \in S$. So,

$$\begin{aligned} \mathbb{E}[\alpha] &= f_I(1 - (1 - p)(1 - q)) + (N - f_I)(1 - q) \\ &= f_I p(1 - q) + Nq. \end{aligned}$$

Then, by (32),

$$\mathbb{E}[\widehat{f_I}] = \frac{\mathbb{E}[\alpha] - N \times q}{p - p \times q} = f_I.$$

Thus, (i) holds.

To prove (ii), note that $\text{Var}[\hat{f}_I] = \frac{\text{Var}[\alpha]}{p^2(1-q)^2}$ and that

$$\begin{aligned}\text{Var}[\alpha] &= f_I(1 - (1 - p(1 - q))(1 - p(1 - q))) \\ &\quad + (N - f_I)q(1 - q) \\ &= f_I p(1 - q)(1 - 2q - p + pq) + Nq(1 - q).\end{aligned}$$

Also note that $MSE[\hat{f}_I] = \text{Var}[\hat{f}_I]$ since \hat{f}_I is unbiased as proved in (i).

A.4.3 Proof of Theorem 9

Proof: Let $f = f_I$, $V = \text{Var}[\hat{f}_I]$, $\alpha = f/N$ and $\tilde{V} = V/N$. Then,

$$\tilde{V} = \frac{\alpha(1 - 2q - p + pq)}{p(1 - q)} + \frac{q}{p^2(1 - q)}.$$

Let $r = 1/p$ (where $r \geq 1$) and $\beta = c \ln(2)$. Here c is the compression factor of the Bloom filter, given as $\frac{m \ln(2)}{x}$, where m is the bit length of the Bloom filter and x is the number of possible keys.

Then $q = e^{-\beta r}$ and $\frac{dq}{dr} = -\beta q$. We have

$$\tilde{V} = \frac{\alpha(r - 2qr - 1 + q)}{(1 - q)} + \frac{qr^2}{(1 - q)}.$$

Then $\frac{d\tilde{V}}{dr}$ is equal to

$$\begin{aligned}&\alpha \left(\frac{1 - 2q + 2\beta qr - \beta q}{1 - q} - \frac{(r - 2qr - 1 + q)\beta q}{(1 - q)^2} \right) \\ &+ \frac{-\beta qr^2 + 2qr}{(1 - q)} - \frac{qr^2\beta q}{(1 - q)^2}.\end{aligned}$$

Let $\Delta_1 = (1 - q)^2 \frac{d\tilde{V}}{dr}$. Then Δ_1 is of the form $\alpha X + Y$, where

$$\begin{aligned}X &= (1 - 2q + 2\beta qr - \beta q)(1 - q) \\ &\quad - (r - 2qr - 1 + q)\beta q \\ &= 1 - 2q + 2\beta qr - \beta q - q + 2q^2 - 2\beta q^2 r + \beta q^2 \\ &\quad - \beta qr + 2\beta q^2 r + \beta q - \beta q^2 \\ &= 1 - 3q + \beta qr + 2q^2 \\ Y &= (-\beta qr^2 + 2qr)(1 - q) - \beta q^2 r^2 \\ &= -\beta qr^2 + 2qr + \beta q^2 r^2 - 2q^2 r - \beta q^2 r^2 \\ &= -\beta qr^2 + 2qr - 2q^2 r \\ &= qr(2 - \beta r - 2q).\end{aligned}$$

Since $(1 - q)^2 > 0$, we have only to study the points $r \geq 1$ at which $\Delta_1 = 0$. Let $x = \beta r$, $\gamma = \alpha\beta$, and $\Sigma_1 = \beta\Delta_1$. We have

$$\Sigma_1(x) = \gamma(1 - e^{-x}) - e^{-x}(x - \gamma)(x - 2 + 2e^{-x}).$$

We will study the changes in the sign of $\Sigma_1(x)$. The first derivative of $\Sigma_1(x)$ is:

$$\begin{aligned} & e^{-x}\gamma + e^{-x}(x - \gamma)(x - 2 + 2e^{-x}) \\ & - e^{-x}(x - 2 + 2e^{-x}) - e^{-x}(x - \gamma)(1 - 2e^{-x}). \end{aligned}$$

Let $\Sigma_2(x)$ be this first derivative divided by e^{-x} . Then,

$$\Sigma_2(x) = 2(1 - e^{-x}) + (x - \gamma)(x - 4 + 4e^{-x}).$$

If $\Sigma_2(x) \geq 0$ for $x \geq \beta$, we have that Δ_1 is monotonically increasing for $r \geq 1$.

For any $A > 0$, $x - A + Ae^{-x} = 0$ has two roots. One is 0 and the other is a positive real less than A . Let μ_4 be the positive root for $A = 4$ and μ_2 the positive root for $A = 2$. Note that since $\gamma \leq \beta$ and $x \geq \beta$, $x - \gamma$ can be thought of as non-negative. So, for $x \geq \max\{\beta, \mu_4\}$, $\Sigma_2(x) > 0$. So, if $\mu_4/\beta \leq 1$, i.e., $c \geq \mu_4/\ln(2)$, then $\Sigma_2(x) > 0$.

We will thus consider below the case when $\beta < \mu_4$. Let $\Sigma_3(x)$, $\Sigma_4(x)$, and $\Sigma_5(x)$ be respectively the first, the second, and the third derivatives of Σ_2 with respect to x :

$$\begin{aligned} \Sigma_3(x) &= 2x - (4 + \gamma) + (-4x + 4\gamma + 6)e^{-x}, \\ \Sigma_4(x) &= 2 + (4x - 4\gamma - 10)e^{-x}, \text{ and} \\ \Sigma_5(x) &= (-4x + 4\gamma + 14)e^{-x}. \end{aligned}$$

We have $\Sigma_5(x) = 0$ at $x = \gamma + \frac{7}{2}$ and so $\Sigma_5(x)$ is positive for $x < \gamma + \frac{7}{2}$ and negative for $x > \gamma + \frac{7}{2}$. We have $\Sigma_4(0) = -4\gamma - 8 < 0$ and $\lim_{x \rightarrow \infty} \Sigma_4(x) = 2$. So, there exists some ρ_1 , $0 < \rho_1 < \gamma + \frac{7}{2}$, such that $\Sigma_4(x)$ is negative in $(0, \rho_1)$ and positive in (ρ_1, ∞) . So, $\Sigma_3(x)$ monotonically decreases in $(0, \rho_1)$ and monotonically increases in (ρ_1, ∞) . We have $\Sigma_3(0) = 3\gamma + 2 > 0$ and $\lim_{x \rightarrow \infty} \Sigma_3(x) = \infty$. If $\Sigma_3(\rho_1) \geq 0$ then $\Sigma_3(x) \geq 0$ for $x > 0$. $\Sigma_2(0) = 0$ and $\lim_{x \rightarrow \infty} \Sigma_2(x) = 2$, if $\Sigma_3(\rho_1) \geq 0$ then $\Sigma_2(x) \geq 0$ for $x \geq 0$.

So, suppose that $\Sigma_3(\rho_1) < 0$. Then there exist some ρ_2 and ρ_3 such that $0 < \rho_2 < \rho_1 < \rho_3$, $\Sigma_3(\rho_2) = \Sigma_3(\rho_3) = 0$, $\Sigma_3(x)$ is positive in $(0, \rho_2)$ and in (ρ_3, ∞) negative in (ρ_2, ρ_3) . Then $\Sigma_2(x)$

monotonically increases in $(0, \rho_2)$ and in (ρ_3, ∞) and monotonically decreases in (ρ_2, ρ_3) . Since $\Sigma_2(0) = 0$ and $\lim_{x \rightarrow \infty} \Sigma_2(x) = 2$, If $\Sigma_2(\rho_3) \geq 0$, then $\Sigma_2(x) \geq 0$ for $x > 0$.

So, suppose that $\Sigma_2(\rho_3) < 0$. Then there exist some ρ_4 and ρ_5 such that $\rho_2 < \rho_4 < \rho_3 < \rho_5$, $\Sigma_2(\rho_4) = \Sigma_2(\rho_5) = 0$, and $\Sigma_3(x)$ is positive in $(0, \rho_4)$ and in (ρ_5, ∞) negative in (ρ_4, ρ_5) . Then $\Sigma_1(x)$ monotonically increases in $(0, \rho_4)$ and in (ρ_5, ∞) and monotonically decreases in (ρ_4, ρ_5) . Since $\Sigma_1(0) = 0$ and $\lim_{x \rightarrow \infty} \Sigma_1(x) = \gamma$, If $\Sigma_1(\rho_5) \geq 0$, then $\Sigma_1(x) \geq 0$ for $x > 0$.

So, suppose that $\Sigma_1(\rho_5) < 0$. Then there exist some ρ_6 and ρ_7 such that $\rho_4 < \rho_6 < \rho_5 < \rho_7$, $\Sigma_1(\rho_6) = \Sigma_1(\rho_7) = 0$, and $\Sigma_1(x)$ is positive in $(0, \rho_6)$ and in (ρ_7, ∞) negative in (ρ_6, ρ_7) . If $\rho_6 \leq \beta$, then clearly the sign of $\Sigma_1(x)$ changes only once in $x \geq \beta$, from negative to positive. So, there are two sign changes only if $\rho_6 > \beta$. We have $\Sigma_2(x) > 0$ for $0 < x \leq \gamma$ and $x \geq \mu_4$. So, $\rho_4 > \gamma$ and $\rho_5 < \mu_4$. Also, $\Sigma_1(x) > 0$ for $\min\{\gamma, \mu_2\} \leq x \leq \max\{\gamma, \mu_2\}$. This implies that one of $\rho_7 < \min\{\gamma, \mu_2\}$ $\rho_6 > \max\{\gamma, \mu_2\}$ holds. Since $\beta \geq \gamma$ and $\rho_6 < \rho_7$, our supposition that $\rho_6 \leq \beta$ implies that the former is impossible. So, $\rho_6 > \max\{\gamma, \mu_2\}$. Thus, $\rho_6 > \max\{\beta, \mu_2\}$.

A.4.4 Proof of Theorem 10

Proof: By definition we have $Y_I = \sum_{i=1}^N y_i = \sum_{j=1}^n b_j g_j$. From $Y_I = \sum_{j=1}^n b_j g_j$, we have $E[Y_I] = \sum_{j=1}^n b_j E[g_j]$. We know from the discussion in the 0/1 case that \hat{g}_j is an unbiased estimator of g_j ; that is, $E[\hat{g}_j] = E[g_j]$. So, $\sum_{j=1}^n b_j E[\hat{g}_j] = \sum_{j=1}^n b_j E[g_j]$. The LHS is equal to $E[\sum_{j=1}^n b_j \hat{g}_j]$, which is $E[f_I]$.

Also, from $Y_I = \sum_{i=1}^N y_i$, we have $E[Y_I] = \sum_{i=1}^N E[y_i] = \sum_{i=1}^N \left(b_{j-1} \frac{b_j - x_i}{b_j - b_{j-1}} + b_j \frac{x_i - b_{j-1}}{b_j - b_{j-1}} \right) = \sum_{i=1}^N x_i = f_I$. Thus, the estimator is unbiased.

A.4.5 Proof of Theorem 11

We first prove a lemma which will be used in the proof of Theorem 11.

Lemma 11 $\text{Var}[Y_I]$ is at most

$$\sum_{j=1}^n \frac{\max\{(b_j - b_{j-1})^2, (b_{j+1} - b_j)^2\}}{4} E[g_j]$$

where $b_{n+1} = b_n$.

Proof: Let $1 \leq i \leq N$. Suppose that $b_{j-1} \leq x_i \leq b_j$ for some j , $1 \leq j \leq n$. Then

$$\begin{aligned}\text{Var}[y_i] &= \frac{(x_i - b_{j-1})^2(b_j - x_i) + (b_j - x_i)^2(x_i - b_{j-1})}{b_j - b_{j-1}} \\ &= (x_i - b_{j-1})(b_j - x_i) \\ &= \frac{(b_j - b_{j-1})^2}{4} - \left(x_i - \frac{b_j + b_{j-1}}{2}\right)^2\end{aligned}$$

So, the variance is maximized when $x_i = \frac{b_j + b_{j-1}}{2}$ to, and the maximum is $\frac{(b_j - b_{j-1})^2}{4}$. Note that for each i , $1 \leq i \leq N$, if count x_i is assigned to b_j then x_i belongs to either the interval $(b_{j-1}, b_j]$ or the interval $[b_j, b_{j+1})$, so

$$\text{Var}[y_i] \leq \max \left\{ \frac{(b_j - b_{j-1})^2}{4}, \frac{(b_{j+1} - b_j)^2}{4} \right\}.$$

This implies

$$\text{Var}[\hat{f}_I] \leq \sum_{i=1}^N \max \left\{ \frac{(b_j - b_{j-1})^2}{4}, \frac{(b_{j+1} - b_j)^2}{4} \right\}.$$

For all j , $1 \leq j \leq n$, the expected number of counts that are assigned to b_j is $E[g_j]$. So, we have

$$\text{Var}[Y_i] \leq \sum_{j=1}^n E[g_j] \max \left\{ \frac{(b_j - b_{j-1})^2}{4}, \frac{(b_{j+1} - b_j)^2}{4} \right\},$$

where $b_{n+1} = b_n$.

Then we prove Theorem 11 in the following.

Proof: By noting that $\hat{f}_I - f_I = (\hat{f}_I - Y_I) + (Y_I - f_I)$, we have

$$\begin{aligned}\text{Var}[\hat{f}_I] &= E[(\hat{f}_I - f_I)^2] \\ &= E[(\hat{f}_I - Y_I) + (Y_I - f_I)]^2 \\ &= E[(\hat{f}_I - Y_I)^2] + E[(Y_I - f_I)^2] \\ &\quad + 2E[(\hat{f}_I - Y_I)(Y_I - f_I)].\end{aligned}$$

Note that $\hat{f}_I - Y_I = \sum_{j=1}^n b_j(\hat{g}_j - g_j)$ and $Y_I - f_I = \sum_{i=1}^N (y_i - x_i)$. So, the last term in the last formula can be written as

$$2E \left[\sum_{i=1}^N (y_i - x_i) \sum_{j=1}^n b_j(\hat{g}_j - g_j) \right].$$

For each fixed assignment of the values of y_1, \dots, y_N , the distribution of $\hat{g}_j - g_j$ is determined for all j , $1 \leq j \leq n$. These distributions are independent and have expectation of 0 each. So, the term

in question is 0. Thus,

$$\text{Var}[\hat{f}_I] = \mathbb{E}[(\hat{f}_I - Y_I)^2] + \mathbb{E}[(Y_I - f_I)^2].$$

The second term is equal to $\text{Var}[Y_I]$. By Lemma 11 this is at most

$$\sum_{j=1}^n \frac{\max\{(b_j - b_{j-1})^2, (b_{j+1} - b_j)^2\}}{4} \mathbb{E}[g_j].$$

By Theorem 8 and the mean value approximation, the first term is bounded by

$$\sum_{j=1}^n \left(\frac{\mathbb{E}[g_j] b_j^2 (1 - 2q_j - p_j + p_j q_j)}{p_j (1 - q_j)} + \frac{q_j N b_j^2}{p_j^2 (1 - q_j)} \right)$$

Then the bounded in the statement follows by combining the two and taking out $\mathbb{E}[g_j]$ for each j .

A.4.6 Comparison with CM sketch

In the following we use the CM sketch [34] as an example to show that our data sketch (i.e., the Bloom filters) achieves better accuracy than other counting sketches with the same overall communication overhead. Using the same scenario stated in Section 6.4.1.4, we setup an example with $N = 10,000$ different nodes and an item I that occurs at exactly 5,000 nodes. We further assume that there are $1M$ items at each and every node. Recall that if each item costs 2 bits on the average in our sketch-based scheme (0/1 case), we can obtain the accuracy bound $\Pr[|\hat{f}_I - f_I| > 113] \leq 24\%$. Suppose that a CM sketch uses the same communication cost per node, i.e., $2M$ bits ($1.8M$ bits for the sketch part). We can first obtain the number of rows in the CM sketch, which is equal to $2 (= \lceil \ln \frac{1}{0.24} \rceil)$ according to Theorem 1 in [34]. Then we can compute the number of columns of the CM sketch and the size of cells which are approximately equal to $0.3M$ and 3 bits respectively according to [34]. But in this configuration of the CM sketch, the final estimate only achieves the following accuracy guaranty by Theorem 1 in [34]: $\Pr[|\hat{f}_I - f_I| > 90,000] \leq 24\%$. This bound is much worse than ours and is trivial because the total frequency count cannot exceed 10,000 in 0/1 case anyway.

In the above example, we assume that the CM sketches sent to the server are aggregated into a single sketch with the same shape, i.e., it has the same number of arrays and counters as the individual sketches (the counter size is however much larger due to aggregation). Then the estimation procedure described in [34], namely “count-min”, will be applied to the aggregated sketch to obtain

the count. An alternative estimation technique is to estimate the frequency from each CM sketch using “count-min” and then add them together to obtain the total frequency. We expect this technique will result in better estimation accuracy, but its accuracy will not be as high as our technique for the two reasons explained in Section 6.2. The CM sketch theory is not designed for global iceberg detection in the first place. A head-on comparison would require us to significantly develop the CM sketch theory along with large deviation techniques.

A.5 Proofs in Chapter 7

A.5.1 Proof of Lemma 3

Recall that the values of $A[i]$, $i = 1, 2, \dots, N$ are initialized to values uniformly distributed over $\{0, 1, \dots, 2^l - 1\}$ independently. In other words, $V_1(0), V_1(0), \dots, V_N(0)$ are mutually independent and each of them has the following distribution:

$$V_j(0) = \begin{cases} 0 & \text{with probability } 2^{-l}, \\ 1 & \text{with probability } 2^{-l}, \\ \dots & \\ 2^l - 1 & \text{with probability } 2^{-l}. \end{cases}$$

In other words, the lemma holds at cycle 0. Now assuming that at cycle k the Lemma holds, we prove that the Lemma also holds at cycle $k + 1$ in the following. If cycle $k + 1$ is idle, i.e., there is no counter update at this cycle, the values of all counters remain the same, so do the independence and the distributions. If there is an counter increment to counter j at that cycle, clearly the values of all other counters remain the same. And $V_j(t + 1)$ has the same distribution as $V_j(t)$ due to the “modular arithmetic” in lines 3 through 5 in Algorithm 8. Therefore at cycle $k + 1$, values of all counters are still mutually independent and the distribution of each counter value is uniform over $\{0, 1, \dots, 2^l - 1\}$.

A.5.2 Analytical Comparison of two bounds

In this section we study and compare the two different bounds on $\Pr[D_{s,t}]$ analytically. We will show that neither of them would dominate the other in the whole τ (defined as $t - s$) range. Given

a fixed K value the bound from Theorem 13 is generally better while for some τ values the bound from Theorem 15 is better.

The problem we solve in the rest of this section is to find out which τ values make the bound from Theorem 15 better than that from Theorem 13, i.e., $e^{\frac{-2(K+\mu\tau-2^{-l}\tau)^2}{\min\{N,\tau\}}} \geq e^{-\frac{a^2}{2}(1-\frac{\epsilon}{3})}$ given a fixed K value. Notice that in the parameter mappings, X_j in Theorem 13 is exactly W_j in Theorem 15. This fact makes our comparison straightforward. Recall that when applying Theorem 7 to our context the strategy to set the value of a is that when $K + \mu\tau - 2^{-l}\tau < a_0\sigma$, we set a to $\frac{K+\mu\tau-2^{-l}\tau}{\sigma}$; otherwise $a = a_0$. It is also easy to see that aforementioned $g(a)$ can also be viewed as a function h of $\frac{a}{\sigma}$, i.e., $h(y) = 4 - e^y - \frac{ye^y}{2}$. Clearly $\frac{a_0}{\sigma}$ is a root of $h(y) = 0$, denoted as C . Next we will study the above problem in these two different settings of a respectively:

Case (i) when $K + \mu\tau - 2^{-l}\tau < a_0\sigma$, i.e., $\sigma^2 > \frac{K+\mu\tau-2^{-l}\tau}{C}$. Therefore the problem becomes for which τ values

$$\begin{aligned} \frac{2(K + \mu\tau - 2^{-l}\tau)^2}{\min\{N, \tau\}} &\leq \frac{a^2}{2} \left(1 - \frac{\epsilon}{3}\right) \\ &= \frac{(K + \mu\tau - 2^{-l}\tau)^2}{2\sigma^2} \left(1 - \frac{e^{\frac{K+\mu\tau-2^{-l}\tau}{\sigma^2}} - 1}{3}\right) \end{aligned}$$

Using linear approximation of exponential function, the above inequality becomes

$$\frac{2}{\min\{N, \tau\}} \geq \frac{1}{2\sigma^2} \left(1 - \frac{K + \mu\tau - 2^{-l}\tau}{2\sigma^2}\right)$$

Using χ to denote $\frac{1}{\sigma^2}$ we have the following function $F(\chi)$,

$$F(\chi) = \frac{K + \mu\tau - 2^{-l}\tau}{6} \chi^2 - \frac{\chi}{2} + \frac{2}{\min\{N, \tau\}} \leq 0 \quad (40)$$

Solving the equation $F(\chi) = 0$, we can obtain its roots, i.e.,

$$\frac{\frac{3}{2} \pm 3\sqrt{\frac{1}{4} - \frac{4}{3\min\{N,\tau\}}(K + \mu\tau - 2^{-l}\tau)}}{K + \mu\tau - 2^{-l}\tau}$$

when $\frac{K}{\frac{3}{16} - \mu + 2^{-l}} \leq \tau \leq \frac{\frac{3N}{16} - K}{\mu - 2^{-l}}$; otherwise there is no (real number) root at all. Therefore in this case if $\frac{K}{\frac{3}{16} - \mu + 2^{-l}} \leq \tau \leq \frac{\frac{3N}{16} - K}{\mu - 2^{-l}}$, the bound from Theorem 15 is better when σ^2 is between the reciprocals of the two roots. Otherwise because $F(\chi)$ does not have any roots and $\frac{K+\mu\tau-2^{-l}\tau}{6}$ is larger than 0 in our context, $F(\chi) \geq 0$ is always false.

Considering the variance σ^2 shown in Theorem 14, we can easily find out the corresponding values of τ given the configuration of the related parameters. We use an example to clarify this further. In Figure 36 we set $N = 1,000,000$ SRAM counters and $l = 4$ bits for each. The queue size K is set to 600 and μ is set to $1/12$. We only plot the part where $\tau \geq \frac{K}{1-\mu}$ because τ at least needs to satisfy $K < b(s, t) - \mu\tau \leq \tau - \mu\tau$, otherwise there is no way that $D_{s,t}$ would happen. The solid curve represents variance (i.e., σ^2), the dashed and dashed-dotted curves represent the reciprocals of two roots of $F(\chi) = 0$ and the dotted curve represents $\frac{K+\mu\tau-2^{-l}\tau}{C}$. The merging points of the dashed curve and the dashed-dotted curve corresponds to $\tau = \frac{K}{\frac{3}{16}-\mu+2^{-l}}$ and $\frac{\frac{3 \min\{N, \tau\}}{16}-K}{\mu-2^{-l}}$ respectively where $F(\chi)$ has only one root. We only need to focus on the segment of the solid curve beyond the dotted curve (i.e., $\sigma^2 > \frac{K+\mu\tau-2^{-l}\tau}{C}$). It is easy to locate the part of this segment that is between the dashed curve and the dashed-dotted curve (i.e., between the reciprocals of these two roots), which is the range of τ in which the bound from Theorem 15 is better than that from Theorem 13.

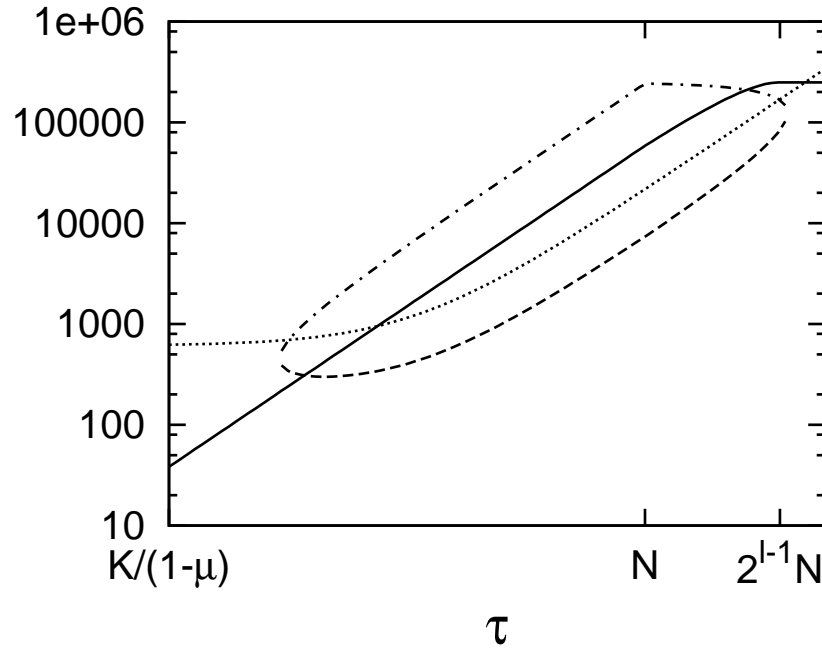


Figure 36: Case i by setting $N = 1,000,000$, $K = 600$, $l = 4$, $u = 1/12$. Notice that both x and y axis are logscale.

Case (ii) when $K + \mu\tau - 2^{-l}\tau \geq a_0\sigma$, i.e., $\sigma^2 \leq \frac{K+\mu\tau-2^{-l}\tau}{C}$. We know that $a = C\sigma$. So the

problem becomes for which τ

$$\begin{aligned} \frac{2(K + \mu\tau - 2^{-l}\tau)^2}{\min\{N, \tau\}} &\leq \frac{a^2}{2} \left(1 - \frac{\epsilon}{3}\right) \\ &= \frac{C^2\sigma^2}{2} \left(1 - \frac{e^C - 1}{3}\right) \end{aligned}$$

Thus we know that σ^2 has to fall in

$$\left[\frac{12(K + \mu\tau - 2^{-l}\tau)^2}{\min\{N, \tau\}C^2(4 - e^C)}, \frac{K + \mu\tau - 2^{-l}\tau}{C} \right]$$

Again we plot Figure 37 for this case using the same parameter setting in Figure 36 to show an example. Again, the solid curve represents variance (i.e., σ^2); the dotted and dashed-dotted curves represent the function $\frac{K + \mu\tau - 2^{-l}\tau}{C}$ and $\frac{12(K + \mu\tau - 2^{-l}\tau)^2}{\min\{N, \tau\}C^2(4 - e^C)}$ respectively. We only need to focus on the segments (2 of them in Figure 37) of the solid curve that are below the dotted curve (i.e., $\sigma^2 \leq \frac{K + \mu\tau - 2^{-l}\tau}{C}$). It is easy to locate one of these two segments that is above the dashed-dotted curve. This segment captures the range of τ in which the bound from Theorem 15 is better than that from Theorem 13.

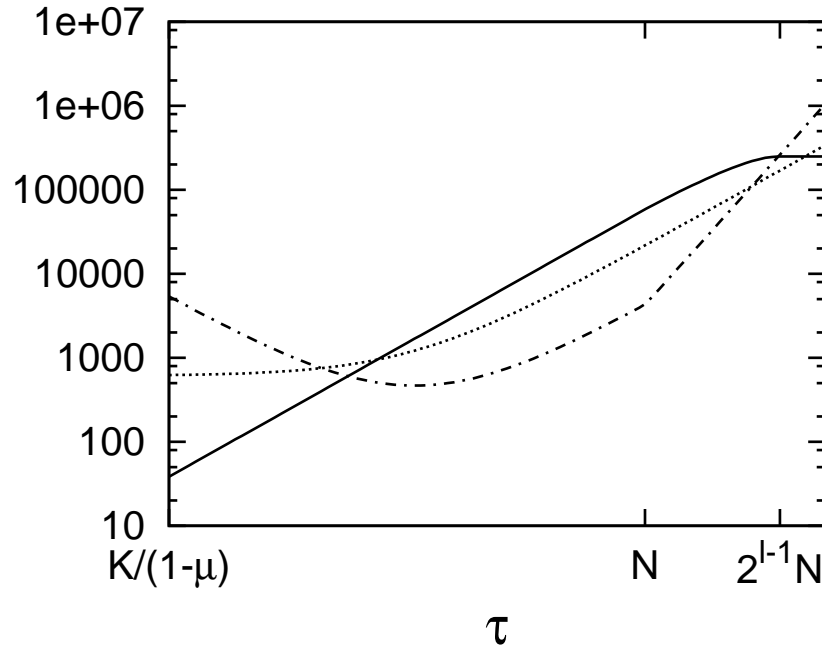


Figure 37: Case ii by setting $N = 1,000,000$, $K = 600$, $l = 4$, $u = 1/12$. Notice that both x and y axis are logscale.

REFERENCES

- [1] “Akamai technologies inc. <http://www.akamai.com/>, mar., 2004.”
- [2] “Internet2 abilene network. <http://abilene.internet2.edu/>, aug., 2005.”
- [3] “Multicast-based inference of network-internal characteristics.” available at <http://gaia.cs.umass.edu/minc>, Mar., 2004.
- [4] AGGARWAL, C., HAN, J., WANG, J., and YU, P. S., “A framework for clustering evolving data streams,” in *Proc. Int. Conf. on Very Large Data Bases(VLDB)*, Sept. 2003.
- [5] ALON, N., GIBBONS, P. B., MATIAS, Y., and SZEGEDY, M., “Tracking join and self-join sizes in limited storage,” in *Proc. ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, May 1999.
- [6] ALON, N., MATIAS, Y., and SZEGEDY, M., “The space complexity of approximating the frequency moments,” *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 137–143, 1999.
- [7] ALON, N. and SPENCER, J. H., *The Probabilistic Method*. A Wiley-Interscience Publication, second ed., 2000.
- [8] ARASU, A. and MANKU, G. S., “Approximate counts and quantiles over sliding windows,” in *Proc. ACM PODS*, June 2004.
- [9] ARASU, A. and MANKU, G. S., “Approximate quantiles and frequency counts over sliding windows,” in *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2004.
- [10] BABCOCK, B., BABU, S., DATAR, M., and MOTWANI, R., “Chain : Operator scheduling for memory minimization in data stream systems,” in *Proc. ACM SIGMOD*, June 2003.
- [11] BABCOCK, B., DATAR, M., MOTWANI, R., and O’CALLAGHAN, L., “Maintaining variance and k-medians over data stream windows,” in *Proc. ACM PODS*, June 2003.
- [12] BABCOCK, B. and OLSTON, C., “Distributed Top-K monitoring,” in *Proc. ACM SIGMOD*, June 2003.
- [13] BABCOCK, B., BABU, S., DATAR, M., MOTWANI, R., and WIDOM, J., “Models and issues in data stream systems,” in *PODS ’02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 1–16, ACM Press, 2002.
- [14] BERTERO, M., “Ill-posed problems in early vision,” *Proc. the IEEE*, pp. 869–889, 1988.
- [15] BHAGWAN, R. and LIN, B., “Fast and scalable priority queue architecture for high-speed network switches,” in *Proc. IEEE INFOCOM*, Mar. 2000.

- [16] BHATTACHARYYA, S., DIOT, C., JETCHEVA, J., and TAFT, N., “Geographical and temporal characteristics of inter-pop flows: View from a single pop,” *European Transactions on Telecommunications*, 2000.
- [17] BLOOM, B., “Space/time trade-offs in hash coding with allowable errors,” *Communications of the Association for Computing Machinery*, vol. 13, no. 7, pp. 422–426, 1970.
- [18] CACERES, R., DUFFIELD, N., HOROWITZ, J., and TOWSLEY, D., “Multicast-based inference of network-internal loss characteristics,” *IEEE Trans. Inform. Theory*, 1999.
- [19] CAO, J., DAVIS, D., WIEL, S. V., and YU, B., “Time-varying network tomography: router link data,” *Journal of American Statistics Association*, pp. 1063–1075, 2000.
- [20] CAO, J., WIEL, S. V., YU, B., and ZHU, Z., “A scalable method for estimating network traffic matrices from link counts,” in *preprint*, 2000.
- [21] CARTER, J. and WEGMAN, M., “Universal classes of hash functions,” *Journal of Computer and System Sciences*, pp. 143–154, 1979.
- [22] CHARIKAR, M., CHEN, K., and FARACH-COLTON, M., “Finding frequent items in data streams,” in *Proc. 29th ICALP*, July 2002.
- [23] CHARIKAR, M., O’CALLAGHAN, L., and PANIGRAHY, R., “Better streaming algorithms for clustering problems,” in *Proc. ACM STOC*, June 2003.
- [24] CHEN, C. and LING, Y., “A sampling-based estimator for top-k query,” in *Proc. International Conference on Data Engineering (ICDE)*, 2002.
- [25] CHEN, Y., DONG, G., HAN, J., PEI, J., WAH, B. W., and WANG, J., “Online analytical processing stream data: Is it feasible?,” in *Proc. ACM-SIGMOD Int. Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, June 2002.
- [26] CHEN, Y., DONG, G., HAN, J., WAH, B. W., and WANG, J., “Multi-dimensional regression analysis of time-series data streams,” in *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, Aug. 2002.
- [27] CHERNIACK, M., BALAKRISHNAN, H., BALAZINSKA, M., CARNEY, D., CETINTEMEL, U., XING, Y., and ZDONIK, S., “Scalable distributed stream processing,” in *Proc. CIDR - Biennial Conference on Innovative Data Systems Research*, Jan. 2003.
- [28] CHERNOFF, H., “A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations,” *Ann. Math. Stat.*, vol. 23, pp. 493–509.
- [29] COATES, M. and NOWAK, R., “Network loss inference using unicast end-to-end measurement,” in *Proc. of ITC Seminar on IP traffic, measurement and modelling*, 2000.
- [30] COHEN, S. and MATIAS, Y., “Spectral bloom filters,” in *Proc. ACM SIGMOD Conference on Management of Data*, 2003.
- [31] COPPERSMITH, D. and KUMAR, R., “An improved data stream algorithm for frequency moments,” in *Proc. ACM-SIAM SODA*, Jan. 2004.
- [32] CORMODE, G. and MUTHUKRISHNAN, S., “What’s hot and what’s not: Tracking most frequent items dynamically,” in *Proc. ACM PODS*, 2003.

- [33] CORMODE, G. and MUTHUKRISHNAN, S., “What’s new: Finding significant differences in network data streams,” in *Proc. IEEE INFOCOM*, Mar. 2004.
- [34] CORMODE, G. and MUTHUKRISHNAN, S., “An improved data stream summary: the count-min sketch and its applications,” *Journal of Algorithms*, Apr. 2005.
- [35] CRAIG, I. and BROWN, J., *Inverse problems in astronomy: a guide to inversion strategies for remotely sensed data*. Adam Hilger, 1986.
- [36] DAS, A., GANGULY, S., GAROFALAKIS, M., and RASTOGI, R., “Distributed set-expression cardinality estimation,” in *Proc. VLDB*, 2004.
- [37] DAS, A., GEHRKE, J., and RIEDEWALD, M., “Approximate join processing over data streams,” in *Proc. ACM SIGMOD*, June 2003.
- [38] DATAR, M., GIONIS, A., INDYK, P., and MOTWANI, R., “Maintaining stream statistics over sliding windows,” in *Proc. ACM-SIAM SODA*, June 2003.
- [39] DEMAINE, E., MUNRO, J., and LOPEZ-ORTIZ, A., “Frequency estimation of internet packet streams with limited space,” in *European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany*, 2002.
- [40] DONOHO, D., “For most large underdetermined systems of equations, the minimal ℓ_1 -norm near solution approximates the sparsest near-solution,” in <http://www-stat.stanford.edu/~donoho/Reports/>, 2004.
- [41] DOWNEY, A., “Using pathchar to estimate internet link characteristics,” in *Proc. ACM SIGCOMM*, Aug. 1999.
- [42] DUFFIELD, N., GREENBERG, A., GROSSGLAUSER, M., REXFORD, J., CHIOU, D., CLAISE, B., MARIMUTHU, P., and SADASIVAN, G., “A framework for packet selection and reporting,” in *Internet Draft, under review*, Jan. 2004.
- [43] DUFFIELD, N. and GROSSGLAUSER, M., “Trajectory sampling for direct traffic observation,” *IEEE transaction of Networking*, pp. 280–292, June 2001.
- [44] DUFFIELD, N. and GROSSGLAUSER, M., “Trajectory sampling with unreliable reporting,” in *Proc. IEEE INFOCOM*, Mar. 2004.
- [45] DUFFIELD, N. and LUND, C., “Predicting resource usage and estimation accuracy in an ip flow measurement collection infrastructure,” in *Proc. ACM SIGCOMM IMC*, Oct. 2003.
- [46] DUFFIELD, N., LUND, C., and THORUP, M., “Properties and prediction of flow statistics from sampled packet streams,” in *Proc. ACM SIGCOMM IMW*, Aug. 2002.
- [47] DUFFIELD, N., LUND, C., and THORUP, M., “Estimating flow distribution from sampled flow statistics,” in *Proc. ACM SIGCOMM*, Aug. 2003.
- [48] DUFFIELD, N., PRESTI, F. L., PAXSON, V., and TOWSLEY, D., “Inferring link loss using striped unicast probes,” in *Proc. of IEEE INFOCOM*, Mar. 2001.
- [49] ESTAN, C. and VARGHESE, G., “New directions in traffic measurement and accounting,” in *Proc. ACM SIGCOMM*, Aug. 2002.

- [50] ESTAN, C. and VARGHESE, G., “New Directions in Traffic Measurement and Accounting,” in *Proc. ACM SIGCOMM*, Aug. 2002.
- [51] ESTAN, C. and VARGHESE, G., “Automatically inferring patterns of resource consumption in network traffic,” in *Proc. ACM SIGCOMM*, Aug. 2003.
- [52] ESTAN, C. and VARGHESE, G., “Bitmap algorithms for counting active flows on high speed links,” in *Proc. ACM/SIGCOMM IMC*, Oct. 2003.
- [53] FAN, L., CAO, P., ALMEIDA, J., and BRODER, A., “Summary cache: a scalable wide-area Web cache sharing protocol,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.
- [54] FANG, M., SHIVAKUMAR, N., GARCIA-MOLINA, H., MOTWANI, R., and ULLMAN, J., “Computing iceberg queries efficiently,” in *Proc. VLDB*, 1998.
- [55] FEIGENBAUM, J. and KANNAN, S., “Streaming algorithms for distributed, massive data sets,” in *In proceedings of FOCS*, 1999.
- [56] FEIGENBAUM, J., KANNAN, S., STRAUSS, M., and VISWANATHAN, M., “An approximate 11-difference algorithm for massive data streams,” *SIAM Journal on Computing*, vol. 32, pp. 131–151, 2002.
- [57] FELDMANN, A., GREENBERG, A., LUND, C., REINGOLD, N., REXFORD, J., and F.TRUE, “Deriving Traffic Demand for Operational IP Networks: Methodology and Experience,” in *Proc. ACM SIGCOMM*, Aug. 2000.
- [58] FELDMANN, A., GREENBERG, A., LUND, C., REINGOLD, N., REXFORD, J., and TRUE, F., “Deriving traffic demands for operational IP networks: Methodology and experience,” *IEEE transaction on Networking*, June 2001.
- [59] GANGULY, S., GAROFALAKIS, M., and RASTOGI, R., “Processing set expressions over continuous update streams,” in *Proc. ACM SIGMOD*, June 2003.
- [60] GIANNELLA, C., HAN, J., PEI, J., YAN, X., and YU, P., “Mining frequent patterns in data streams at multiple time granularities,” in *H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, AAAI/MIT Press*, 2003.
- [61] GUHA, S., KOUDAS, N., and SHIM, K., “Data-streams and histograms,” in *Proc. ACM STOC*, July 2001.
- [62] GUHA, S., MISHRA, N., MOTWANI, R., and O’CALLAGHAN, L., “Clustering data streams,” in *Proc. IEEE Symposium on Foundations of Computer Science*, Nov. 2000.
- [63] GUNNAR, A., JOHANSSON, M., and TELKAMP, T., “Traffic matrix estimation on a large ip backbone– a comparison on real data,” in *Proc. USENIX/ACM SIGCOMM IMC*, Oct. 2004.
- [64] GUPTA, A. and SUCIU, D., “Stream processing of xpath queries with predicates,” in *Proc. ACM SIGMOD*, June 2003.
- [65] HARFOUSH, K., BESTAVROS, A., and BYERS, J., “Robust identification of shared losses using end-to-end unicast probes,” in *Proc. of IEEE ICNP*, Nov. 2000.

- [66] HUFFMAN, D., "A method for the construction of minimum-redundancy codes," *Proc. I.R.E.*, pp. 1098–1102, 1952.
- [67] JUNG, J., KRISHNAMURTHY, B., and RABINOVICH, M., "Flash crowds and denial of service attacks: Characterization and implications for cdn and web sites," in *Proc. World Wide Web Conference*, May 2002.
- [68] "Configure traffic sampling." <http://www.juniper.net/techpubs/software/junos/junos63/swconfig63-policy/html/sampling-config4.html>.
- [69] KARP, R. M., SHENKER, S., and PAPADIMITRIOU, C. H., "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems (TODS)*, vol. 28, pp. 51–55, 2003.
- [70] KAY, S., *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, 1993.
- [71] KRISHNAMURTHY, B., SEN, S., ZHANG, Y., and CHEN, Y., "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. of ACM SIGCOMM IMC*, Oct. 2003.
- [72] KRISHNAMURTHY, B., SEN, S., ZHANG, Y., and CHEN, Y., "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. Internet Measurement Conference (IMC)*, 2003.
- [73] KUMAR, A., SUNG, M., XU, J., and WANG, J., "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *Proc. ACM SIGMETRICS*, 2004.
- [74] KUMAR, A., SUNG, M., XU, J., and WANG, J., "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *Proc. ACM SIGMETRICS*, June 2004.
- [75] KUMAR, A., SUNG, M., XU, J., and ZEGURA, E., "A data streaming algorithms for estimating subpopulation flow size distribution," in *Proc. ACM SIGMETRICS*, June 2005.
- [76] KUMAR, A., XU, J., WANG, J., SPATSCHECK, O., and LI, L., "Space-Code Bloom Filter for Efficient per-flow Traffic Measurement," in *Proc. IEEE INFOCOM*, Mar. 2004.
- [77] KUMAR, A., XU, J., WANG, J., SPATSCHEK, O., and LI, L., "Space-Code Bloom Filter for Efficient per-flow Traffic Measurement," in *Proc. IEEE INFOCOM*, Mar. 2004. Extended abstract appeared in *Proc. ACM IMC* 2003.
- [78] LAKHINA, A., PAPAGIANNAKI, K., and CROVELLA, M., "Diagnosing network-wide anomalies," in *Proc. ACM SIGCOMM*, Aug. 2004.
- [79] LIANG, G., TAFT, N., and YU, B., "A fast lightweight approach to origin-destination ip traffic estimation using partial measurements," in *Technical Report, Department of Statistics, University of California, Irvine*, June 2005.
- [80] LIANG, G. and YU, B., "Pseudo likelihood estimation in network tomography," in *Proc. IEEE INFOCOM*, Mar. 2003.
- [81] MANJHI, A., SHKAPENYUK, V., DHAMDHERE, K., and OLSTON, C., "Finding (recently) frequent items in distributed data streams," in *Proc. International Conference on Data Engineering (ICDE)*, 2005.

- [82] MANKU, G. and MOTWANI, R., "Approximate frequency counts over data streams," in *Proc. 28th International Conference on Very Large Data Bases (VLDB)*, 2002.
- [83] MANKU, G. S., RAJAGOPALAN, S., and LINDSAY, B. G., "Approximate medians and other quantiles in one pass and with limited memory," in *Proc. ACM SIGMOD*, June 1998.
- [84] MANKU, G. S., RAJAGOPALAN, S., and LINDSAY, B. G., "Random sampling techniques for space efficient online computation of order statistics of large datasets," in *Proc. ACM SIGMOD*, June 1999.
- [85] MEDINA, A., TAFT, N., SALAMATIAN, K., BHATTACHARYYA, S., and DIOT, C., "Traffic matrix estimation: existing techniques and new directions," in *Proc. ACM SIGCOMM*, Aug. 2002.
- [86] MOTWANI, R. and RAGHAVAN, P., *Randomized Algorithms*. Cambridge University Press, 1995.
- [87] MUTHUKRISHNAN, S., "Data streams: algorithms and applications." available at <http://athos.rutgers.edu/~muthu/>, Aug., 2004.
- [88] "White paper-netflow services and applications." http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neft/tech/napps_wp.htm.
- [89] "<http://pma.nlanr.net>."
- [90] NUCCI, A., CRUZ, R., TAFT, N., and DIOT, C., "Design of igp link weight changes for estimation of traffic matrices," in *Proc. IEEE INFOCOM*, Mar. 2004.
- [91] OLSTON, C., JIANG, J., and WIDOM, J., "Adaptive filters for continuous queries over distributed data streams," in *Proc. ACM SIGMOD*, June 2003.
- [92] O'SULLIVAN, F., "A statistical perspective on ill-posed inverse problems (with discussion)," *Statistical Science*, pp. 502–527, 1986.
- [93] PADMANABHAN, V., QIU, L., and WANG, H., "Server-based inference of internet link lossiness," in *Proc. IEEE INFOCOM*, Mar. 2003.
- [94] PAPAGIANNAKI, K., TAFT, N., and LAKHINA, A., "A distributed approach to measure traffic matrices," in *Proc. ACM/SIGCOMM IMC*, Oct. 2004.
- [95] PARKER, R., *Geophysical inverse theory*. Princeton University Press, 1994.
- [96] PASZTOR, A. and VEITCH, D., "Pc based precision timing without gps," in *Proc. ACM SIGMETRICS*, June 2002.
- [97] PAXON, V., "An analysis of using reflectors for distributed denial-of-service attacks," *Computer Communication Review*, 2001.
- [98] PENG, F. and CHAWATHE, S., "Xpath queries on streaming data," in *Proc. ACM SIGMOD*, June 2003.
- [99] PHATAK, D. S. and GOFF, T., "A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments," in *Proc. IEEE INFOCOM*, June 2002.

- [100] PLONKA, D., “Flowscan: A network traffic flow reporting and visualization tool,” in *Proc. USENIX LISA*, 2000.
- [101] PRESTI, F. L., DUFFIELD, N., HOROWITZ, J., and TOWSLEY, D., “Multicast-based inference of network-internal delay distributions,” *IEEE/ACM Transaction on Networking*, pp. 761–775, 2002.
- [102] RAMABHADRAN, S. and VARGHESE, G., “Efficient implementation of a statistics counter architecture,” in *Proc. ACM SIGMETRICS*, June 2003.
- [103] RAMAKRISHNA, M., FU, E., and BAHCEKAPILI, E., “Efficient hardware hashing functions for high performance computers,” *IEEE Transactions on Computers*, pp. 1378–1381, 1997.
- [104] RATNASAMY, S. and MCCANNE, S., “Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurement,” in *Proc. of IEEE INFOCOM*, Mar. 1999.
- [105] “<http://www.ripe.net>.”
- [106] ROESCH, M., “Snort—lightweight intrusion detection for networks,” in *Proc. USENIX Systems Administration Conference*, 1999.
- [107] ROSS, S., *Stochastic Processes*. Wiley, 1995.
- [108] SAKS, M. and SUN, X., “Space lower bounds for distance approximation in the data stream model,” in *Proc. ACM STOC*, May 2002.
- [109] SCHWELLER, R., GUPTA, A., PARSONS, E., , and CHEN, Y., “Reversible sketches for efficient and accurate change detection over network data streams,” in *Proc. IMC*, Oct. 2004.
- [110] “White paper-traffic monitoring using sflow.” <http://www.inmon.com/technology/papers.php>.
- [111] SHAH, D., IYER, S., PRABHAKAR, B., and MCKEOWN, N., “Maintaining statistics counters in router line cards,” in *IEEE Micro*, 2002.
- [112] SHAIKH, A. and GREENBERG, A., “Ospf monitoring: Architecture, design and deployment experience,” in *Proc. USENIX NSDI*, 2004.
- [113] SHIBATA, R., “Selection of the order of an autoregressive model by akaike’s information criterion,” *Biometrika*, vol. 63, pp. 117–126, Apr. 1976.
- [114] SHIH, M. and HERO, A., “Unicast inference of network link delay distributions from edge measurements,” in *Proc. of IEEE ICASSP*, 2001.
- [115] SNOEREN, A., PARTRIDGE, C., SANCHEZ, L., JONES, C., TCHAKOUNTIO, F., KENT, S., and STRAYER, W., “Hash-based ip traceback,” in *Proc. ACM SIGCOMM*, Aug. 2001.
- [116] SOULE, A., LAKHINA, A., TAFT, N., and PAPAGIANNAKI, K., “Traffic matrices: Balancing measurements, inference and modeling,” in *Proc. ACM SIGMETRICS*, Aug. 2005.
- [117] SOULE, A., NUCCI, A., CRUZ, R., LEONARDI, E., and TAFT, N., “How to identify and estimate the largest traffic matrix elements in a dynamic environment,” in *Proc. ACM SIGMETRICS*, June 2004.
- [118] TEBALDI, C. and WEST, M., “Bayesian inference on network traffic using link count data,” *Journal of American Statistics Association*, pp. 557–576, 1998.

- [119] TSANG, Y., COATES, M., and NOWAK, R., "Passive network tomography using em algorithms," in *Proc. of IEEE ICASSP*, May 2001.
- [120] TSANG, Y., COATES, M., and NOWAK, R., "Network delay tomography," *IEEE/ACM Transaction on Signal Process*, pp. 2125–2136, 2003.
- [121] VARDI, Y., "Internet tomography: estimating source-destination traffic intensities from link data," *Journal of American Statistics Association*, pp. 365–377, 1996.
- [122] VENKATARAMAN, S., SONG, D., GIBBONS, P., and BLUM, A., "New streaming algorithms for fast detection of superspreaders," in *Proc. NDSS*, 2005.
- [123] WANG, H., FAN, W., YU, P. S., and HAN, J., "Mining concept-drifting data streams using ensemble classifiers," in *Proc. ACM SIGKDD*, Aug. 2003.
- [124] WANG, Y., "Personal communication." Dec 2004.
- [125] WHANG, K., VANDER-ZANDEN, B., and TAYLOR, H., "A linear-time probabilistic counting algorithm for database applications," *IEEE transaction of Database Systems*, pp. 208–229, June 1990.
- [126] WU, J., MAO, Z., REXFORD, J., and WANG, J., "Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network," in *Proc. USENIX NSDI*, 2005.
- [127] XU, J., SINGHAL, M., and DEGROAT, J., "A novel cache architecture to support layer-four packet classification at memory access speeds," in *Proc. of IEEE INFOCOM*, Mar. 2000.
- [128] YU, H., YANG, J., and HAN, J., "Classifying large data sets using svm with hierarchical clusters," in *Proc. ACM SIGKDD*, Aug. 2003.
- [129] ZHANG, Y., GE, Z., GREENBERG, A., and ROUGHAN, M., "Network anomography," in *Proc. USENIX/ACM SIGCOMM IMC*, Oct. 2005.
- [130] ZHANG, Y., ROUGHAN, M., DUFFIELD, N., and GREENBERG, A., "Fast accurate computation of large-scale ip traffic matrices from link loads," in *Proc. ACM SIGMETRICS*, June 2003.
- [131] ZHANG, Y., ROUGHAN, M., LUND, C., and DONOHO, D., "An information-theoretic approach to traffic matrix estimation," in *Proc. ACM SIGCOMM*, Aug. 2003.
- [132] ZHANG, Y., SINGH, S., SEN, S., DUFFIELD, N., and LUND, C., "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application," in *Proc. Internet Measurement Conference (IMC)*, 2004.
- [133] ZHAO, Q., KUMAR, A., WANG, J., and XU, J., "Data streaming algorithms for accurate and efficient measurement of traffic and flow matrices," in *Proc. ACM SIGMETRICS*, June 2005.
- [134] ZIOTOPOLOUS, A., HERO, A., and WASSERMAN, K., "Estimation of network link loss rates via chaining in multicast trees," in *Proc. of IEEE ICASSP*, 2001.

VITA

Qi Zhao received his Bachelor of Science and Master of Engineering both in computer science and engineering from Southeast University, Nanjing, China in 1998 and 2001 respectively. Qi joined the College of Computing at the Georgia Institute of Technology, as a Ph.D. student in August 2002. His thesis work was conducted under the able guidance of Dr. Jun (Jim) Xu. At Georgia Tech, Qi was a recipient of the Outstanding Graduate Research Assistant in 2007.