EFFICIENT STATISTICAL MEASUREMENT METHODS IN WIRED AND WIRELESS SYSTEMS

By

TAO LI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2012

To my parents and my wife

ACKNOWLEDGMENTS

First of all, I would like to gratefully and sincerely thank my adviser, Prof. Shigang Chen, for his great support, guidance, understanding, and most importantly, his friendship during my graduate study at University of Florida. He is an incredible adviser, a passionate scientist, and a terrific person. His consistent support and encouragement helped me to overcome many crisis situations and finish my Ph.D. study. For everything you have done for me, Prof. Chen, I want to say thank you from the bottom of my heart.

I am grateful to Prof. Sartaj Sahni, Prof. Yuguang Fang, Prof. Ahmed Helmy, and Prof. Tan Wong for their advice and support during my study at University of Florida. I would also like to thank all the members in Prof. Chen's group for their help. They are Liang Zhang, MyungKeun Yoon, Ying Jian, Wen Luo, Yan Qiao, Zhen Mo, Yian Zhou, Yangbae Park, Min Chen, and especially for Ming Zhang, who offers me a lot of suggestions and encouragements.

I would also like to thank my friends in University of Florida. They are Ying Xuan, Ting Chen, Bing Jian, Yan Li, Lu Chen, Zhuo Huang, Xuelian Xiao, Xiao Li, Wenjie Yuan, Jianmin Chen, Jingyi Wang, Long Yu, Yuchen Xie, Hechen Liu, Meizhu Liu, Yixing Yang, Shuang Zhao, Junjie Li, Mingmin Zhu, Lin Qi, Yan Deng, Shuang Lin, and Xiangguo Li. You make my life full of fun throughout my graduate study and I really enjoy being friends with you.

Finally, and most importantly, I am thankful to each member of my family: my parents, my brother, my sister-in-law, my little niece, and especially my wife Xiaoqian. Thank you all for your support, understanding, encouragement, and love for so many years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

EFFICIENT STATISTICAL MEASUREMENT METHODS IN WIRED AND WIRELESS SYSTEMS

By

Tao Li

May 2012

Chair: Shigang Chen
Major: Computer Engineering

Traffic measurement provides critical real-world data for service providers and network administrators to perform capacity planning, accounting and billing, anomaly detection, and service provision. We observe that in many measurement functions, statistical methods play important roles in system designing, model building, formula deriving, and error analyzing. In this dissertation, we first propose several novel online measurement functions in high-speed networks. We then notice that statistical methods for measurement problems are generic in many network systems. They can be also applied to wireless systems such as RFID (radio frequency identification) systems, which have been gaining popularity for inventory control, object tracking, and supply chain management in warehouses, retail stores, hospitals, etc. The second part of the dissertation studies the RFID estimation problem and designs two probabilistic algorithms for it.

One of the greatest challenges in designing an online measurement module is to minimize the per-packet processing time in order to keep up with the line speed of the modern routers. To meet this challenge, we should minimize the number of memory accesses per packet and implement the measurement module in the on-die SRAM, which is fast but expensive. Because many other essential routing/security/performance functions may also run from SRAM, it is expected that the amount of high-speed

memory allocated for the module will be small. Hence, it is critical to make the measurement module's data structure as compact as possible.

The first work of this dissertation focuses on a particularly challenging problem, the measurement of per-flow information in high-speed networks. We design a fast and compact measurement function that estimates the sizes of all flows. It achieves the optimal processing speed: 2 memory accesses per packet. In addition, it provides reasonable measurement accuracy in a tight space where the best existing methods no longer work. Our design is based on a new data encoding/decoding scheme, called *randomized counter sharing*. This scheme allows us to mix per-flow information together in storage for compactness and, at the decoding time, separate the information of each flow through statistical removal of the error introduced during information mixing from other flows. The effectiveness of our online per-flow measurement approach is analyzed and confirmed through extensive experiments based on real network traffic traces. We also propose several methods to increase the estimation range of flow sizes.

Our second work studies the scan detection problem, which is one of the most fundamental functions in intrusion detection systems. We propose an efficient scan detection scheme based on *dynamic bit sharing*, which incorporates probabilistic sampling and bit sharing for compact information storage. We design a maximum likelihood estimation method to extract per-source information from the shared bits in order to determine the scanners. Our new scheme ensures that the false positive/false negative ratios are bounded with high probability. Moreover, given an arbitrary set of bounds, we develop a systematic approach to determine the optimal system parameters that minimize the amount of memory needed to meet the bounds. Experiments based on a real Internet traffic trace demonstrate that the proposed scan detection scheme reduces memory consumption by three to twenty times when comparing with the best existing work.

The origin-destination flow measurement is the focus of our third work. An origin-destination (OD) flow between two routers is the set of packets that pass both routers in a network. We design a new measurement method that employs a compact data structure for packet information storage and uses a novel statistical inference approach for OD-flow size estimation. Not only does the proposed method require smaller per-packet processing overhead, but also it achieves much better measurement accuracy, when comparing with existing approaches. We perform both simulations and experiments to demonstrate the effectiveness of our method.

Our last work focuses on estimating the number of RFID tags deployed in a large area, which has many important applications in inventory management and theft detection. Prior works focus on designing time-efficient algorithms that can estimate tens of thousands of tags in seconds. We observe that, for a RFID reader to access tags in a large area, active tags are likely to be used due to their longer operational ranges. These tags are battery-powered and use their own energy for information transmission. However, recharging batteries for tens of thousands of tags is laborious. Hence, conserving energy for active tags becomes critical. Some prior works have studied how to reduce energy expenditure of a RFID reader when it reads tag IDs. We study how to reduce the amount of energy consumed by active tags during the process of estimating the number of tags in a system. We design two energy-efficient probabilistic estimation algorithms that iteratively refine a control parameter to optimize the information carried in transmissions from tags, such that both the number and the size of transmissions are reduced. These algorithms can also take time efficiency into consideration. By tuning a contention probability parameter ω, the new algorithms can make tradeoff between energy cost and estimation time.

CHAPTER 1
INTRODUCTION

1.1 Online Network Functions

Modern high-speed routers forward packets from incoming ports to outgoing ports via switching fabric, bypassing main memory and CPU. New technologies are pushing line speeds beyond OC-768 (40Gb/s) to reach 100Gb/s or even tera bits per second [47]. The line cards in core routers must therefore forward packets at a rate exceeding 150Mpps [104]; that leaves no more than 6.7ns to process each packet. Parallel processing and pipeline are used to speed up packet switching to a few clock cycles per packet [51]. In order to keep up with such high throughput, online network functions for traffic measurement, packet scheduling, access control, and quality of service will also have to be implemented using on-chip cache memory and bypassing main memory and CPU almost entirely [75, 104, 129]. However, fitting these network functions in fast but small on-chip memory represents a major technical challenge today [51, 95].

The commonly-used cache memory on network processor chips is SRAM, typically a few megabytes. Further increasing on-chip memory to more than 10MB is technically feasible, but it comes with a much higher price tag and access time is longer. There is a huge incentive to keep on-chip memory small because smaller memory can be made faster and cheaper. Off-chip SRAM is larger. For example, QDR-III SRAM has 36MB [91]. But it is slower to access. Hence, on-chip memory remains the first choice for online network functions that are designed to match the line speeds.

On-chip memory is limited in size. To make the matter even more challenging, it may have to be shared by security [57], measurement [75], routing [20], and performance [55] functions that are implemented on the same chip. When multiple network functions share the same memory, each of them can only use a fraction of the available space. Depending on their relative importance, some functions may be allocated tiny portions of the limited memory, whereas the amount of data they have

to process and store can be extremely large in high-speed networks. The disparity in memory demand and supply requires us to implement online functions as compact as possible [106, 114]. Furthermore, when different functions share the same memory, they may have to take turns to access the memory, making memory access the performance bottleneck. Since most online functions require only simple computations that can be efficiently implemented in hardware, their throughput will be determined by the bottleneck in memory access. Hence, we must also minimize the number of memory accesses made by each function when it processes a packet. The challenge is that compactness (in terms of space requirement) and speed (in terms of memory accesses) are sometimes conflicting objectives.

1.2 Fundamental Primitives

We observe that the implementations of many online functions heavily rely on several fundamental building blocks for data processing and storage. The first part of this dissertation studies three important fundamental online functions: per-flow estimators, spread estimators, and origin-destination flow estimators.

Per-flow estimators are used to measure per-flow information for high-speed links. The goal is to estimate the size of each flow (in terms of number of packets). A flow is identified by a label that can be a source address, a destination address, or any combination of addresses, ports, and other fields in the packet header. Measuring the sizes of individual flows has important applications. For example, if we use the addresses of the users as flow labels, per-flow traffic measurement provides the basis for usage-based billing and graceful service differentiation, where a user's service priority gracefully drops as he over-spends his resource quota. Studying per-flow data over consecutive measurement periods may help us discover network access patterns and, together with user profiling, reveal geographic/demographic traffic distributions among users. Such information will help Internet service providers and application developers to align network resource allocation with the majority's needs.

Spread estimators [70] are designed for the measurement of *distinct* elements in each flow, where "flows" can be per-source flows, per-destination flows, TCP flows, P2P flows, and so on. And "elements" can be source addresses, destination addresses, or any other application-specific addresses. For example, if we treat all packets that are from the same source address as a flow, a spread estimator that measures the number of distinct destinations for each flow can be used to detect port scans [106]. Spread estimators may be used to detect DDoS attacks when too many hosts send traffic to a receiver [92], i.e., the spread of a destination is abnormally high. They can be used to estimate the infection rate of a worm by monitoring how many addresses each infected host contacts over a period of time.

Origin-destination (OD) flow estimators are used to measure *OD flow sizes*. Consider two routers r_1 and r_2. We define the set of packets that first pass r_1 and then pass r_2 or first pass r_2 and then pass r_1 as an *origin-destination (OD) flow* of the two routers. The cardinality of the packet set is called the *OD flow size*. The OD flow measurement is also an important topic in many network management applications [39, 46, 80, 82, 99]. For example, Internet service providers may use the OD-flow information between points of interest as a reference to align traffic distribution within the network. They may also study the OD-flow traffic pattern and identify anomalies that deviate significantly from the normal pattern. In the event of a persistent congestion, OD-flow data may help point out the source of the congestion.

One of the greatest challenges in designing an online measurement module is to minimize the per-packet processing time in order to keep up with the line speed of the modern routers. To meet this challenge, we should minimize the number of memory accesses per packet and implement the measurement module in the on-die SRAM, which is fast but expensive. Because many other functions may also run from SRAM, it is expected that the amount of high-speed memory allocated for the module will be

small. Hence, it is critical to make the measurement module's data structure as compact as possible.

We observe that when studying these measurement problems, statistical methods play important roles in system designing, model building, formula deriving, and error analyzing. We first propose several novel schemes which employ classical statistical methods such as maximum likelihood estimation method. We then learn that statistical methods for measurement problems are generic in many network systems. They can also be applied to wireless systems such as RFID (radio frequency identification) systems, which have been gaining popularity for inventory control, object tracking, and supply chain management in warehouses, retail stores, hospitals, etc. The second part of this dissertation studies the RFID estimation problem and designs several probabilistic algorithms for it.

1.3 Per-Flow Traffic Measurement through Randomized Counter Sharing

This work focuses on a particularly challenging problem, the measurement of per-flow information for a high-speed link without using per-flow data structures [69]. It has been shown in [40] that maintaining per-flow counters cannot scale for high-speed links. Even for efficient counter implementations [96, 102, 130], SRAM will only be able to hold a small fraction of per-flow state (including counters and indexing data structures such as pointers and flow identities for locating the counters). The *counter braids* avoid per-flow counters and achieve near-optimal memory efficiency [75, 76]. This method maps each flow to three arbitrary counters; they are all incremented by one for every packet of the flow. Many flows may be mapped to the same counter, which stores the sum of the flow sizes. Essentially, the counters represent linear equations, which can be solved for the flow sizes. Two levels of counters are used to reduce the memory overhead. The counter braids require slightly more than 4 bits per flow and are able to count the exact sizes of all flows. But it also has two limitations. First, it performs 6 or occasionally 12 memory accesses per packet. Second, when the memory allocated

to a measurement function is far less than 4 bits per flow, our experiments show that the message passing decoding algorithm of counter braids cannot converge to any meaningful results. When the available memory is just 1$\sim$2 bits per flow, the *exact measurement* of the flow sizes is no longer possible. We have to resort to *estimation methods*. The key is to efficiently utilize the limited space to improve the accuracy of the estimated flow sizes, and do so with the minimum number of memory accesses per packet. This is what this work tries to achieve.

We design a fast and compact per-flow traffic measurement function that achieves three main objectives: i) It shares counters among flows to save space, and does not incur any space overhead for mapping flows to their counters. This distinguishes our work from [96, 102, 130]. ii) It updates exactly one counter per packet, which is optimal. This separates our work from the counter braids that update three or more counters per packet. Updating each counter requires two memory accesses for read and then write. iii) It provides estimation of the flow sizes, as well as the confidence intervals that characterize the accuracy, even when the available memory is too small such that other exact-counting methods including [75, 76] no longer work. We believe our work is the first one that achieves all these objectives. It complements the existing work by providing additional flexibility for the practitioners to choose when other methods cannot meet the speed and space requirements.

The design of our measurement function is based on a new data encoding/decoding scheme, called *randomized counter sharing*. It splits the size of each flow among a number of counters that are randomly selected from a counter pool. These counters form the *storage vector* of the flow. For each packet of a flow, we randomly select a counter from the flow's storage vector and increment the counter by one. Such a simple online operation can be implemented very efficiently. The storage vectors of different flows share counters uniformly at random; the size information of one flow in a counter is the noise to other flows that share the same counter. Fortunately, this noise can be

quantitatively measured and removed through statistical methods, which allow us to estimate the size of a flow from the information in its storage vector. We propose two estimation methods whose accuracies are statistically guaranteed. They work well even when the total number of counters in the pool is by far smaller than the total number of flows that share the counters. Our experimental results based on real traffic traces demonstrate that the new methods can achieve good accuracy in a tight space. We also propose several methods to increase the range of flow sizes that the estimators can measure.

The randomized counter sharing scheme proposed in this work for per-flow traffic measurement has applications beyond the networking field. It may be used in the data streaming applications to collect per-item information from a stream of data items.

1.4 Scan Detection in High-Speed Networks

Internet security is a fundamental problem and has received considerable attention for years [23, 24, 56]. Many network-based attacks are preceded with a reconnaissance phase, in which the attacker or its zombies scan the hosts in a network to identify vulnerability. As a result, scan detection is one of the most fundamental functions in almost any network intrusion detection system (IDS). Cisco has been pushing for years to build security functions into its high-end routers. Scan detection is increasingly performed by routers with security modules or firewalls that inspect packets [32].

We define a *contact* as a source-destination pair, for which the source sends a packet to the destination. The source or destination can be an IP address, a port number, or a combination of them together with other fields in the packet header. The *spread* of a source is the number of *distinct destinations* contacted by the source during a measurement period. A source is classified as a scanner if its spread exceeds a certain threshold. Therefore, scan detection is fundamentally an online traffic measurement problem.

A great challenge for scan detection is that the data volume to be stored can be huge. For example, the main gateway at our campus observes more than 10 million distinct source-destination pairs on an average day. Suppose each measurement period is one day long (in order to catch stealthy low-rate scanners). If we simply store all distinct source/destination address pairs for scan detection, it will require more than $80MB$ of SRAM, which is too much. A major thrust in the scan detection research is to reduce the memory consumption [13, 41, 110, 114, 129].

Reducing memory consumption does not come for free. The prior research sacrifices detection accuracy for memory saving. The basic idea is to compress the contact information in limited memory space. The compressed information allows us to estimate the spreads of the sources, instead of counting them exactly. However, the estimated spread values may cause false positives (in which a non-scanner is mistakenly reported as a scanner) and false negatives (in which a scanner is not reported). Consequently, the following questions become important for any practical security system: How serious is the false positive/false negative problem? Can the system be configured such that the false positive/false negative ratios are bounded? To date, few papers directly addressed these questions.

The prior work follows two general methods for memory reduction: *probabilistic sampling* and *storage sharing*. The probabilistic sampling method is to record only a certain percentage of randomly sampled contacts. An example is the one-level/two-level algorithms proposed by Venkataraman et al. [110]. These algorithms store the source/destination addresses of the sampled contacts in hash tables. Their main contribution is to derive the optimal sampling probability that ensures with high probability that the false positive/false negative ratios do not exceed certain pre-defined bounds.

However, it is not memory-efficient to directly store the addresses of the contacts made by each source. A naive solution is to use per-source counters to record the

number of packets from each source. Near-optimal counter architectures such as counter braids [75] require only a few bits per source. The problem is that counters cannot remove duplicates: A thousand packets from the same source to the same destination should count as one contact, instead of a thousand. In order to remove duplicates, one may use Bloom filters [110] or bitmap algorithms [41]. They encode the contacts made by each source in a separate bitmap, which automatically filters duplicates. However, per-source bitmaps still take too much space. Cao et al. use a series of Bloom filters and a hash table to reduce the number of sources that need bitmaps [13].

Instead of using a separate bitmap for each source, an interesting space-saving method is to allow *storage sharing*, where each data structure is no longer dedicated to a single source but shared among multiple sources. This is particularly necessary when the number of sources is more than the number of available bits. Zhao et al. [129] encodes each contact in three shared bitmaps using a technique similar to Bloom filters. Yoon et al. [114] design another storage sharing method with superior performance. Although both methods can be used for scan detection, none of them provides any means to ensure that the false positive/false negative ratios are bounded. Moreover, our experiments show that these existing methods [13, 114, 129] take far more memory than the one proposed in this study.

This study proposes an efficient scan detection scheme based on a new storage sharing method, called *dynamic bit sharing*, which shares the available bits uniformly at random among all sources, such that the memory space is fully utilized for storing contact information. It employs a maximum likelihood estimation method to extract per-source information from the shared bits in order to determine the scanners. It also enhances security through a private key. Our new method ensures that the false positive/false negative ratios are bounded. Moreover, given an arbitrary set of bounds, we show analytically how to choose the optimal system parameters such that

the amount of memory needed to satisfy the bounds is minimized. We also perform experiments based on a real traffic trace and demonstrate that, using these optimal parameters, we can reduce the memory consumption by three to twenty times when comparing with the best existing work.

1.5 Origin-Destination Flow Measurement in High-Speed Networks

Our third work focuses on the problem of *origin-destination (OD) flow measurement* [71]. The goal is to design an efficient method to measure the number of packets that traverse between two routers during a measurement period. It generally consists of two phases: One for online packet information storage and the other for offline OD-flow size computation. In the first phase, routers record information about arrival packets. In the second phase, each router reports its stored information to a centralized server, which performs the measurement of each OD flow based on the information sent from the origin/destination router pair.

Measurement efficiency and accuracy are two main technical challenges. In terms of efficiency, we want to minimize the per-packet processing overhead to accommodate future routers that forward packets at extremely high rates. More specifically, the function should minimize the computational complexity and the number of memory accesses for each packet.

Accuracy is another important design goal. In high-speed networks, we have to deal with a very large volume of packets. And it is unrealistic to store all packet-level information in order to achieve 100% accuracy. To solve this problem, some past research [119–121] uses data such as link load, network routing, and configuration data to indirectly measure the OD flows. Cao, Chen and Bu [11] propose a quasi-likelihood approach based on a continuous variant of the Flajolet-Martin sketches [43]. However, none of them is able to achieve both efficiency and accuracy at the same time.

To meet these challenges, we design a novel OD flow measurement method, which uses a compact bitmap data structure for packet information storage. At the end

of a measurement period, bitmaps from all routers are sent to a centralized server, which examines the bitmaps of each origin/destination router pair and uses a statistical inference approach to estimate the OD flow size. The proposed method has three elegant properties. First, its processing overhead is small and constant, only one hash operation and one memory access per packet. Second, it is able to achieve excellent measurement results, which will be demonstrated by both simulations and experiments. Finally, its data storage is very compact. The memory allocation is less than 1 bit for each packet on average.

1.6 Size Estimation Problem in RFID Systems

Radio-frequency identification (RFID) technology has been widely used in various commercial applications. RFID tags (each storing a unique ID) are attached to merchandizes at retail stores, equipment at hospitals, or goods at warehouses, allowing an authenticated RFID reader to quickly access properties of each individual item or collect statistical information about a large group of items.

This work focuses on a RFID-enabled function that is very useful in inventory management. Imagine a large warehouse with thousands of laptops, cell phones, electronics, apparel, bags, or furniture pieces. A national retail survey showed that administration error, vendor fraud and employee theft caused about 20 billion dollars lost a year [52]. Hence, it is desirable to have a quick way of counting the number of items in the warehouse or in each section of the warehouse. To timely detect theft or management errors, such counting may be performed frequently.

If each item is attached with a RFID tag, the counting problem can be solved by a RFID reader that receives the IDs transmitted (or backscattered) from the tags [112]. However, reading the actual tag IDs can be time-consuming because so many of them have to be delivered in the same low-rate channel and collisions caused by simultaneous transmissions by different tags make the matter worse. To address this problem, Kodialam and Nandagopal [63, 64] showed that reading time can be

greatly reduced through probabilistic methods that estimate the number of tags. This is called the *RFID estimation problem*. The follow-up work by Qian et al. [93] significantly reduces estimation time when comparing with [63]. It can be shown that even for applications that require reading the actual tag IDs, estimating the number of tags as a pre-processing step will help make the main procedure of reading tag IDs much more efficient [63]. Another advantage of estimating the number of tags without reading the IDs is that it ensures anonymity of the tags, which may be useful in privacy-sensitive scenarios involving RFID-enhanced passports or driver's licences, where counting the number of people present is needed but revealing their identities is not necessary.

Is time efficiency the only performance metric for the estimation problem in large-scale RFID systems that use active tags? We argue that energy cost is also an important issue that must be carefully dealt with. For any application that requires a RFID reader to access tags in a large area, it is likely that battery-powered *active tags* will be used. *Passive tags* harvest energy from radio signal of a reader and use such a minute amount of energy to deliver information back to the reader. Their typical reading range is only several meters, which do not fit well with the big warehouse scenario. Active tags use their own power to transmit. A longer reading range can be achieved by transmitting at higher power. They are also richer in resources for implementing advanced functions. Their price becomes less of a concern if they are used for expensive merchandizes or reused many times as goods moving in and out of the warehouse. But active tags also have a problem. They are powered by batteries. Recharging batteries for tens of thousands of tags is a laborious operation, considering that tagged products may be stacked up, making tags not easily accessible. To prolong the lifetime of tags and reduce the frequency of battery recharge, all functions that involve large-scale transmission by many tags should be made energy-efficient. Prior works focus on energy-efficient anti-collision protocols that minimize energy consumption of a mobile reader [61, 85] when the reader collects tag IDs. To the best of

our knowledge, this work is the first to study energy-efficient solutions for the estimation problem in large-scale RFID systems that use active tags.

This work has four major contributions. First, we observe that there exists an asymmetry in energy cost. Solving the RFID estimation problem incurs energy cost both at the RFID reader and at active tags. The asymmetry is that energy cost at tags should be minimized while energy cost at the reader is relatively less of a concern because the reader's battery can be replaced easily or it may be powered by an external source. To exploit this asymmetry, our new algorithms follow a common framework that trades more energy cost at the reader for less cost at the tags. The reader will continuously refine and broadcast a control parameter called *contention probability*, which optimizes the amount of information the reader can extract from transmissions by tags. This in turn reduces the number of transmissions by tags that are necessary to achieve a certain estimation accuracy.

Second, the design of our estimation algorithms is based on the maximum likelihood estimation method (MLE) that is different from the probabilistic counting methods [113] used by [63, 64]. Our estimation algorithms optimize their performance by iteratively applying MLE with continuously refined parameters. These new algorithms not only require fewer transmissions by tags but also minimize the size of each transmission. The number of transmissions made by tags in our best algorithm is less than one fourth achieved by the state-of-the-art algorithms. In terms of the total number of bits transmitted by tags, it is more than an order of magnitude smaller.

Third, we formally analyze the confidence intervals of estimations made by our new algorithms and establish the termination conditions for any given accuracy requirement. We perform extensive simulations to demonstrate that the measured results match well with the analytical results and that the new algorithms perform far better in terms of energy saving than the best existing algorithms.

Fourth, our algorithms are generalized with a tunable parameter ω, specifying the contention probability that tags use to decide whether they will transmit. By modifying this parameter, the generalized algorithms can make tradeoff between energy cost and estimation time (i.e., the time it takes to complete the process of estimating the number of tags). Even though our main goal is to reduce energy cost, the ability for performance tradeoff makes our algorithms more adaptable in practical setting that are sensitive not only to energy cost but also to estimation time.

In the broad context of computer networks, there are many other important topics that have drawn extensive attention from researchers. They are QoS and maxmin routing [19, 21, 22, 77, 84, 103, 109], P2P networks [59, 125, 126], distributed computing [6, 18], etc.

1.7 Outline of the Dissertation

The rest of the dissertation is organized as follows: Chapter 2 presents a fast and compact per-flow traffic measurement approach through randomized counter sharing. In this section, we design of a novel data encoding/decoding scheme, which mixes per-flow information randomly in a tight SRAM space for compactness. Chapter 3 proposes an efficient scan detection scheme based on a new method called *dynamic bit sharing*, which optimally combines probabilistic sampling, bit-sharing storage, and maximum likelihood estimation. Chapter 4 designs a new method for OD flow measurement which employs the bitmap data structure for packet information storage and uses statistical inference approach to compute the measurement results. Chapter 5 proposes two probabilistic algorithms for estimating the number of RFID tags in a region. Chapter 6 draws the conclusion.

CHAPTER 2
PER-FLOW TRAFFIC MEASUREMENT THROUGH RANDOMIZED COUNTER SHARING

This chapter studies the measurement of per-flow information for high-speed links. It is a particularly difficult problem because of the need to process and store a huge amount of information, which makes it difficult for the measurement module to fit in the small but fast SRAM space (in order to operate at the line speed). We propose a novel measurement function that estimates the sizes of all flows. It delivers good performance in tight memory space where the best existing approaches no longer work. The effectiveness of our online per-flow measurement approach is analyzed and confirmed through extensive experiments based on real network traffic traces.

The rest of this chapter is organized as follows: Section 2.1 discusses the performance metrics. Section 2.2 gives an overview of our system design. Section 2.3 discusses the state of the art. Section 2.4 presents the online data encoding module. Sections 2.5-2.6 propose two offline data decoding modules. Section 2.7 discusses the problem of setting counter length. Section 2.8 addresses the problem of collecting flow labels. Section 2.9 presents the experimental results. Section 2.10 extends our estimators for large flow sizes. Section 2.11 gives the summary.

2.1 Performance Metrics

We measure the number of packets in each flow during a measurement period, which ends every time after a certain number (e.g., 10 millions) of packets are processed. The design of per-flow measurement functions should consider the following three key performance metrics.

2.1.1 Processing Time

The per-packet processing time of an online measurement function determines the maximum packet throughput that the function can operate at. It should be made as small as possible in order to keep up with the line speed. This is especially true when multiple

routing, security, measurement, and resource management functions share SRAM and processing circuits.

The processing time is mainly determined by the number of memory accesses and the number of hash computations (which can be efficiently implemented in hardware [97]). The counter braids [75, 76] update three counters at the first level for each packet. When a counter at the first level overflows, it needs to update three additional counters at the second level. Hence, it requires 3 hashes and 6 memory accesses to read and then write back after counter increment. But in the worse case, it requires 6 hashes and 12 memory accesses. The multi-resolution space-code Bloom filters [66] probabilistically select one or more of its 9 filters and set 3$\sim$6 bits in each of the selected ones. Each of those bits requires one memory access and one hash computation.

Our objective is to achieve a constant per-packet processing time of one hash computation and two memory accesses (for updating a single counter). This is the minimum processing time for any method that uses hash operations to identify counters for update.

2.1.2 Storage Overhead

The need to reduce the SRAM overhead has been discussed in Chapter 1. One may argue that because the amount of memory needed is related to the number of packets in a measurement period, we can reduce the memory requirement by shortening the measurement period. However, when the measurement period is smaller, more flows will span multiple periods and consequently the average flow size in each period will be smaller. When we measure the flow sizes, we also need to capture the flow labels [76], e.g., a tuple of source address/port and destination address/port to identify a TCP flow. The flow labels are too large to fit in SRAM. They have to be stored in DRAM. Therefore, in a measurement period, each flow incurs at least one DRAM access to store its flow label. If the average flow size is large enough, the overhead of this DRAM access will be amortized over many packets of a flow. However,

if the average flow size is too small, the DRAM access will become the performance bottleneck that seriously limits the throughput of the measurement function. This means the measurement period should not be too small. Our experiments in Section 2.9 set a measurement period such that the average flow size is about 10.

2.1.3 Estimation Accuracy

Let s be the size of a flow and $\hat{s}$ be the estimated size of the flow based on a measurement function. The estimation accuracy of the function can be specified by a confidence interval: the probability for s to be within $[\hat{s} \cdot (1-\beta), \hat{s} \cdot (1+\beta)]$ is at least a pre-specified value α, e.g., 95%. A smaller value of β means that the estimated flow size is more accurate (in a probabilistic sense).

There is a tradeoff between the estimation accuracy and the storage overhead. If the storage space and the processing time are unrestricted, we can accurately count each packet to achieve perfect accuracy. However, in practice, there will be constraints on both storage and processing speed, which make 100% accurate measurement sometimes infeasible. In this case, one has to settle with imperfect results that can be produced with the available resources. Within the bounds of the limited resources, we must explore novel measurement methods to make the estimated flow sizes as accurate as possible.

2.2 System Design

2.2.1 Basic Idea

We use an example to illustrate the idea behind our new measurement approach. Suppose the amount of SRAM allocated to one of the measurement functions is 2Mb (2×2^{20} bits), and each measurement period ends after 10 million packets, which translate into about 8 seconds for an OC-192 link (10+ Gbps) with an average packet size of 1,000 bytes. The types of flows that the online functions may measure include per-source flows, per-destination flows, per-source/destination flows, TCP flows, WWW

flows (with destination port 80), etc. Without losing generality, suppose the specific function under consideration in this example measures the sizes of TCP flows.

Fig. 2-1 shows the number of TCP flows that have a certain flow size in log scale, based on a real network trace captured by the main gateway of our campus. If we use 10 bits for each counter, there will be only 0.2 million counters. The number of concurrent flows in our trace for a typical measurement period is around 1 million. It is obvious that allocating per-flow state is not possible and each counter has to store the information of multiple flows. But if an "elephant" flow is mapped to a counter, that counter will overflow and lose information. On the other hand, if only a couple of "mouse" flows are mapped to a counter, the counter will be under-utilized, with most of its high-order bits left as zeros.

To solve the above problems, we not only store the information of multiple flows in each counter, but also store the information of each flow in a large number of counters, such that an "elephant" is broken into many "mice" that are stored at different counters. More specifically, we map each flow to a set of l randomly-selected counters and split the flow size into l roughly-equal shares, each of which is stored in one counter. The value of a counter is the sum of the shares from all flows that are mapped to the counter. Because flows share counters, they introduce noise to each other's measurement. The key to accurately estimate the size of a flow is to measure the noise introduced by other flows in the counters that the flow is mapped to.

Fortunately, this can be done if the flows are mapped to the counters uniformly at random. Any two flows will have the same probability of sharing counters, which means that each flow will have the same probability of introducing a certain amount of noise to any other flow. If the number of flows and the number of counters are very large, the combined noise introduced by all flows will be distributed across the counter space about uniformly. The statistically uniform distribution of the noise can be measured and

removed. The above scheme of information storage and recovery is called *randomized counter sharing*.

We stress that this design philosophy of "splitting" each flow among a large number of counters is very different from "replicating" each flow in three counters as the counting Bloom filter [27] or counter braids [75] do — they add the size of each flow as a whole to three randomly selected counters. Most notably, our method increments one counter for each arrival packet, while the counting Bloom filter or counter braids increment three counters. We store the information of each flow in many counters (e.g., 50), while they store the information of each flow in three counters.

2.2.2 Overall Design

Our online traffic measurement function consists of two modules. The online data encoding module stores the information of arrival packets in an array of counters. For each packet, it performs one hash function to identify a counter and then updates the counter with two memory accesses, one for reading and the other for writing. At the end of each measurement period, the counter array is stored to the disk and then reset to zeros.

The offline data decoding module answers queries for flow sizes. It is performed by a designated offline computer. We propose two methods for separating the information about the size of a flow from the noise in the counters. The first one is called the *counter sum estimation method* (CSM), which is very simple and easy to compute. The second one is called the *maximum likelihood estimation method* (MLM), which is more accurate but also more computationally intensive. The two complementary methods provide flexibility in designing a practical system, which may first use CSM for rough estimations and then apply MLM to the ones of interest.

2.3 State of the Art

A related thread of research is to collect statistical information of the flows [36, 65], or identify the largest flows and devote the available memory to measure their

sizes while ignoring the smaller ones [34, 40, 58, 60]. For example, RATE [62] and ACCEL-RATE [50] measure per-flow rate by maintaining per-flow state, but they use a *two-run sampling* method to filter out small-rate flows so that only high-rate flows are measured.

Another thread of research is to maintain a large number of counters to track various networking information. One possible solution [30, 107] can be statistically update a counter according to the current counter size. This approach is fit for the applications with loose measurement accuracy. In order to enhance the accuracy performance, Zhao et al. [127] propose a statistical method to make a DRAM-based solution practical, which uses a small cache and request queues to balance the counter values. Since DRAM is involved and wirespeed is achieved, this approach is able to achieve decent measurement accuracy.

Also related is the work [110] that measures the number of *distinct* destinations that each source has contacted. Per-flow counters cannot be used to solve this problem because they cannot remove duplicate packets. If a source sends 1,000 packets to a destination, the packets contribute only one contact, but will count as 1,000 when we measure the flow size. To remove duplicates, bitmaps (instead of counters) should be used [13, 41, 114, 115, 128]. From the technical point of view, this represents a separate line of research, which employs a different set of data structures and analytical tools. Attempt has also been made to use bitmaps for estimating the flow sizes, which is however far less efficient than counters, as our experiments will show.

2.4 Online Data Encoding

The flow size information is stored in an array C of m counters. The ith counter in the array is denoted as $C[i]$, $0 \leq i \leq m-1$. The size of the counters should be set so that the chance of overflow is negligible; we will discuss this issue in details in Section 2.7. Each flow is mapped to l counters that are randomly selected from C through hash functions. These counters logically form a *storage vector* of the flow,

denoted as C_f, where f is the label of the flow. The ith counter of the vector, denoted as $C_f[i]$, $0 \leq i \leq l-1$, is selected from C as follows:

$$C_f[i] = C[H_i(f)], \tag{2–1}$$

where $H_i(...)$ is a hash function whose range is $[0, m)$. We want to stress that C_f is *not* a separate array for flow f. It is merely a logical construction from counters in C for the purpose of simplifying the presentation. In all our formulas, one should treat the notation $C_f[i]$ simply as $C[H_i(f)]$. The hash function H_i, $0 \leq i \leq l-1$, can be implemented from a master function $H(...)$ as follows: $H_i(f) = H(f|i)$ or $H_i(f) = H(f \oplus R[i])$, where '|' is the concatenation operator, '$\oplus$' is the XOR operator, and $R[i]$ is a constant whose bits differ randomly for different indices i.

All counters are initialized to zeros at the beginning of each measurement period. The operation of online data encoding is very simple: When the router receives a packet, it extracts the flow label f from the packet header, randomly selects a counter from C_f, and increases the counter by one. More specifically, the router randomly picks a number i between 0 and $l-1$, computes the hash $H_i(f)$, and increases the counter $C[H_i(f)]$, which is physically in the array C, but logically the ith element in the vector C_f.

2.5 Offline Counter Sum Estimation

2.5.1 Estimation Method

At the end of a measurement period, the router stores the counter array C to a disk for long-term storage and offline data analysis. Let n be the combined size of all flows, which is $\sum_{i=0}^{m-1} C[i]$. Let s be the true size of a flow f during the measurement period. The estimated size, $\hat{s}$, based on our counter sum estimation method (CSM) is

$$\hat{s} = \sum_{i=0}^{l-1} C_f[i] - l\frac{n}{m}. \tag{2–2}$$

The first item is the sum of the counters in the storage vector of flow f. It can also be interpreted as the sum of the flow size s and the noise from other flows due

to counter sharing. The second item captures the expected noise. Below we formally derive (2–2).

Consider an arbitrary counter in the storage vector of flow f. We treat the value of the counter as a random variable X. Let Y be the portion of X contributed by the packets of flow f, and Z be the portion of X contributed by the packets of other flows. Obviously, $X = Y + Z$.

Each of the s packets in flow f has a probability of $\frac{1}{l}$ to increase the value of the counter by one. Hence, Y follows a binomial distribution:

$$Y \sim Bino(s, \frac{1}{l}). \tag{2–3}$$

Each packet of another flow f' has a probability of $\frac{1}{m}$ to increase the counter by one. That is because the probability for the counter to belong to the storage vector of flow f' is $\frac{l}{m}$, and if that happens, the counter has a probability of $\frac{1}{l}$ to be selected for increment. Assume there is a large number of flows, the size of each flow is negligible when comparing with the total size of all flows, and l is large such that each flow's size is randomly spread among many counters. We can approximately treat the packets independently. Hence, Z approximately follows a binomial distribution:

$$Z \sim Bino(n - s, \frac{1}{m}) \approx Bino(n, \frac{1}{m}), \text{ because } s \ll n. \tag{2–4}$$

We must have

$$E(X) = E(Y + Z) = E(Y) + E(Z) = \frac{s}{l} + \frac{n}{m}. \tag{2–5}$$

That is,

$$s = l \times E(X) - l\frac{n}{m}. \tag{2–6}$$

From the observed counter values $C_f[i]$, $E(X)$ can be measured as $\frac{\sum_{i=0}^{l-1} C_f[i]}{l}$. We have the following estimation for s:

$$\hat{s} = \sum_{i=0}^{l-1} C_f[i] - l\frac{n}{m}. \tag{2–7}$$

If a flow shares a counter with an "elephant" flow, its size estimation can be skewed. However, our experiments show that CSM works well in general because the number of "elephants" is typically small (as shown in Fig. 2-1) and thus their impact is also small, particularly when there are a very large number of counters and flows. Moreover, our next method based on maximum likelihood estimation can effectively reduce the impact of an outlier in a flow's storage vector that is caused by an "elephant" flow.

2.5.2 Estimation Accuracy

The mean and variance of $\hat{s}$ will be given in (2–9) and (2–10), respectively. They are derived as follows: Because $X = Y + Z$, we have

$$\begin{aligned} E(X^2) &= E((Y+Z)^2) = E(Y^2) + 2E(YZ) + E(Z^2) \\ &= E(Y^2) + 2E(Y)E(Z) + E(Z^2) \\ &= \frac{s^2}{l^2} - \frac{s}{l^2} + \frac{s}{l} + 2 \cdot \frac{s}{l} \cdot \frac{n}{m} + \frac{n^2}{m^2} - \frac{n}{m^2} + \frac{n}{m}. \end{aligned}$$

The following facts are used in the above mathematical process: $E(Y^2) = \frac{s^2}{l^2} - \frac{s}{l^2} + \frac{s}{l}$ because $Y \sim Bino(s, 1/l)$. $E(YZ) = E(Y)E(Z)$ since Y and Z are independent. $E(Z^2) = \frac{n^2}{m^2} - \frac{n}{m^2} + \frac{n}{m}$ because $Z \sim Bino(n, 1/m)$.

$$\begin{aligned} Var(X) &= E(X^2) - (E(X))^2 \\ &= \frac{s}{l}(1 - \frac{1}{l}) + \frac{n}{m}(1 - \frac{1}{m}). \end{aligned} \tag{2–8}$$

In (2–7), $C_f[i]$, $0 \le i \le l-1$, are independent samples of X. We can interpret $\hat{s}$ as a random variable in the sense that a different set of samples of X may result in a different value of $\hat{s}$. From (2–7), we have

$$\begin{aligned} E(\hat{s}) &= l \times E(X) - l\frac{n}{m} \\ &= l(\frac{s}{l} + \frac{n}{m}) - l\frac{n}{m} \quad = s, \end{aligned} \tag{2–9}$$

which means our estimation is unbiased. The variance of $\hat{s}$ can be written as

$$\begin{aligned} Var(\hat{s}) &= I^2 \times Var(X) = I^2\left(\frac{s}{I}(1-\frac{1}{I}) + \frac{n}{m}(1-\frac{1}{m})\right) \\ &= s(I-1) + I^2\frac{n}{m}(1-\frac{1}{m}). \end{aligned} \tag{2–10}$$

2.5.3 Confidence Interval

The confidence interval for the estimation will be given in (2–13), and it is derived as follows: The binomial distribution, $Z \sim Bino(n, 1/m)$, can be closely approximated as a Gaussian distribution, $Norm(\frac{n}{m}, \frac{n}{m}(1-\frac{1}{m}))$, when n is large. Similarly, the binomial distribution, $Y \sim Bino(s, \frac{1}{I})$, can be approximated by $Norm(\frac{s}{I}, \frac{s}{I}(1-\frac{1}{I}))$. Because the linear combination of two independent Gaussian random variables is also normally distributed [14], we have $X \sim Norm(\frac{s}{I}+\frac{n}{m}, \frac{s}{I}(1-\frac{1}{I})+\frac{n}{m}(1-\frac{1}{m}))$. To simplify the presentation, let $\mu = \frac{s}{I}+\frac{n}{m}$ and $\Delta = \frac{s}{I}(1-\frac{1}{I})+\frac{n}{m}(1-\frac{1}{m})$.

$$X \sim Norm(\mu, \Delta), \tag{2–11}$$

where the mean μ and the variance Δ agree with (2–5) and (2–8), respectively.

Because $\hat{s}$ is a linear function of $C_f[i]$, $0 \le i \le I-1$, which are independent samples of X, $\hat{s}$ must also approximately follow a Gaussian distribution. From (2–7) and (2–11), we have

$$\hat{s} \sim Norm(s, s(I-1) + I^2\frac{n}{m}(1-\frac{1}{m})). \tag{2–12}$$

Hence, the confidence interval is

$$\hat{s} \pm Z_\alpha\sqrt{s(I-1) + I^2\frac{n}{m}(1-\frac{1}{m})}, \tag{2–13}$$

where α is the confidence level and Z_α is the α percentile for the standard Gaussian distribution. As an example, when $\alpha = 95\%$, $Z_\alpha = 1.96$.

2.6 Maximum Likelihood Estimation

In this section, we propose the second estimation method that is more accurate but also more computationally expensive.

2.6.1 Estimation Method

We know from the previous section that any counter in the storage vector of flow f can be represented by a random variable X, which is the sum of Y and Z, where $Y \sim Bino(s, \frac{1}{l})$ and $Z \sim Bino(n, 1/m)$. For any integer $z \in [0, n)$, the probability for the event $Z = z$ to occur can be computed as follows:

$$Pr\{Z = z\} = \binom{n}{z}(\frac{1}{m})^z(1 - \frac{1}{m})^{n-z}.$$

Because n and m are known, $Pr\{Z = z\}$ is a function of a single variable z and thus denoted as $P(z)$.

Based on the probability distribution of Y and Z, the probability for the observed value of a counter, $C_f[i]$, $\forall i \in [0, l)$, to occur is

$$\begin{aligned} & Pr\{X = C_f[i]\} \\ &= \sum_{z=0}^{C_f[i]}(Pr\{Z = z\} \cdot Pr\{Y = C_f[i] - z\}) \\ &= \sum_{z=0}^{C_f[i]}\binom{s}{C_f[i] - z}(\frac{1}{l})^{C_f[i]-z}(1 - \frac{1}{l})^{s-(C_f[i]-z)}P(z). \end{aligned} \tag{2–14}$$

Let $y = C_f[i] - z$ to simplify the formula. The probability for all observed values in the storage vector of flow f to occur is

$$\begin{aligned} L &= \prod_{i=0}^{l-1} Pr\{X = C_f[i]\} \\ &= \prod_{i=0}^{l-1}\left(\sum_{z=0}^{C_f[i]}\binom{s}{y}(\frac{1}{l})^y(1 - \frac{1}{l})^{s-y}P(z)\right). \end{aligned} \tag{2–15}$$

The maximum likelihood method (MLM) is to find an estimated size $\hat{s}$ of flow f that maximizes the above likelihood function. Namely, we want to find

$$\hat{s} = \arg\max_{s}\{L\}. \tag{2–16}$$

To find $\hat{s}$, we first apply logarithm to turn the right side of the equation from product to summation.

$$\ln(L) = \sum_{i=0}^{I-1} \ln\left(\sum_{z=0}^{C_f[i]} \binom{s}{y} (\frac{1}{I})^y (1-\frac{1}{I})^{s-y} P(z)\right). \tag{2–17}$$

Because $\frac{d\binom{s}{y}}{ds} = \binom{s}{y}(\psi(s+1) - \psi(s+1-y))$, where $\psi(\ldots)$ is the polygamma function [4], we have

$$\frac{d(\binom{s}{y}(1-\frac{1}{I})^{s-y})}{ds} = \binom{s}{y}(1-\frac{1}{I})^{s-y}\left(\psi(s+1) - \psi(s+1-y) + \ln(1-\frac{1}{I})\right).$$

To simplify the presentation, we denote the right side of the above equation as $O(s)$. From (2–17), we can compute the first-order derivative of $\ln(L)$ as follows:

$$\frac{d\ln(L)}{ds} = \sum_{i=0}^{I-1} \frac{\sum_{z=0}^{C_f[i]}\left(O(s)(\frac{1}{I})^y P(z)\right)}{\sum_{z=0}^{C_f[i]}\binom{s}{y}(\frac{1}{I})^y(1-\frac{1}{I})^{s-y}P(z)}. \tag{2–18}$$

Maximizing L is equivalent to maximizing $\ln(L)$. Hence, by setting the right side of (2–18) to zero, we can find the value for $\hat{s}$ through numerical methods. Because $\frac{d\ln(L)}{ds}$ is a monotone function of s, we have used the bisection search method in all our experiments in Section 2.9 to find the value $\hat{s}$ that makes $\frac{d\ln(L)}{ds}$ equal to zero.

2.6.2 Estimation Accuracy

The estimation confidence interval will be given in (2–26), and it is derived as follows: The estimation formula is given in (2–16). According to the classical theory for MLM, when I is sufficiently large, the distribution of the flow-size estimation $\hat{s}$ can be

approximated by

$$Norm(s, \frac{1}{\mathcal{I}(\hat{s})}), \tag{2–19}$$

where the *fisher information* $\mathcal{I}(\hat{s})$ [67] of L is defined as follows:

$$\mathcal{I}(\hat{s}) = -E\left(\frac{d^2 \ln(L)}{ds^2}\right). \tag{2–20}$$

In order to compute the second-order derivative, we begin from (2–11) and have the following:

$$\begin{aligned} Pr\{X = C_f[i]\} &= \frac{1}{\sqrt{2\pi\Delta}} e^{-\frac{(C_f[i]-\mu)^2}{2\Delta}} \\ \ln(Pr\{X = C_f[i]\}) &= -\ln(\sqrt{2\pi\Delta}) - \frac{(C_f[i]-\mu)^2}{2\Delta}, \end{aligned} \tag{2–21}$$

where $0 \leq i \leq l-1$. Performing the second-order differentiation, we have

$$\begin{aligned} \frac{d^2 \ln(Pr\{X = C_f[i]\})}{ds^2} &= -\frac{\mu'}{l\Delta} + \frac{(\frac{1}{2}(1-\frac{1}{l}) + \mu - C_f[i])\Delta'}{l\Delta^2} \\ &+ \frac{1}{l\Delta^3}(1-\frac{1}{l})\left((\mu - C_f[i])\mu'\Delta - (\mu - C_f[i])^2\Delta'\right), \end{aligned} \tag{2–22}$$

where $\mu' = \frac{1}{l}$ and $\Delta' = \frac{1}{l}(1-\frac{1}{l})$. Therefore,

$$\begin{aligned} &E(\frac{d^2 \ln(Pr\{X = C_f[i]\})}{ds^2}) \\ &= -\frac{\mu'}{l\Delta} + \frac{\frac{1}{2}(1-\frac{1}{l})\Delta'}{l\Delta^2} + \frac{1}{l\Delta^3}(1-\frac{1}{l})E(\mu - C_f[i])^2\Delta' \\ &= -\frac{1}{l^2\Delta} + \frac{3(1-\frac{1}{l})^2}{2l^2\Delta^2}, \end{aligned} \tag{2–23}$$

where we have used the following facts: $E(\mu - C_f[i]) = 0$ and $E(\mu - C_f[i])^2 = \Delta$. Because $L = \prod_{i=0}^{l-1} Pr\{X = C_f[i]\}$, we have

$$\begin{aligned} \mathcal{I}(\hat{s}) = -E\left(\frac{d^2 \ln(L)}{ds^2}\right) &= \sum_{i=0}^{l-1} E(\frac{d^2 \ln(Pr\{X = C_f[i]\})}{ds^2}) \\ &= \frac{1}{l\Delta} - \frac{3(1-\frac{1}{l})^2}{2l\Delta^2}. \end{aligned} \tag{2–24}$$

From (2–19), the variance of $\hat{s}$ is

$$Var(\hat{s}) = \frac{1}{\mathcal{I}(\hat{s})} = \frac{2/\Delta^2}{2\Delta - 3(1-\frac{1}{l})^2}. \tag{2–25}$$

Hence, the confidence interval is

$$\hat{s} \pm Z_\alpha \cdot \sqrt{\frac{2/\Delta^2}{2\Delta - 3(1-\frac{1}{l})^2}}, \tag{2–26}$$

where Z_α is the α percentile for the standard Gaussian distribution.

2.7 Setting Counter Length

So far, our analysis has assumed that each counter has a sufficient number of bits such that it will not overflow. However, in order to save space, we want to set the counter length as short as possible. Suppose each measurement period ends after a pre-specified number n of packets are received. (Note that the value of n is the combined sizes of all flows during each measurement period.) The average value of all counters will be $\frac{n}{m}$. We set the number of bits in each counter, denoted as b, to be $\log_2 \frac{n}{m} + 1$. Due to the additional bit, each counter can hold at least two times of the average before overflowing. If the allocated memory has M bits, the values of b and m can be determined from the following equations:

$$b \times m = M, \qquad \log_2 \frac{n}{m} + 1 = b. \tag{2–27}$$

Due to the randomized counter sharing design, roughly speaking, the packets are distributed in the counters at random. We observe in our experiments that the counter values approximately follow a Gaussian distribution with a mean of $\frac{n}{m}$. In this distribution, the fraction of counters that are more than four times of the mean is very small — less than 5.3% in all our experiments. Consequently, the impact of counter overflow in CSM or MLM is also very small for most flows. Though it is small, we will totally eliminate this impact in Section 2.10.4.

2.8 Flow Labels

The compact online data structure introduced in Section 2.4 only stores the flow size information. It does not store the flow labels. The labels are per-flow information, and it cannot be compressed in the same way we do for the flow sizes. In some applications, the flow labels are pre-known and do not have to be collected. For example, if an ISP wants to measure the traffic from its customers, it knows their IP addresses (which are the flow labels in this case). Similarly, if the system administrator of a large enterprise network needs the information about the traffic volumes of the hosts in the network, she has the hosts' addresses.

In case that the flow labels need to be collected and there is not enough SRAM to keep them, the labels have to be stored in DRAM. An efficient solution for label collection was proposed in [76]. A Bloom filter [7, 8] can be implemented in SRAM to encode the flow labels that have seen by the router during a measurement period, such that each label is only stored once in DRAM when it appears for the first time in the packet stream; storing each label once is the minimum overhead if the labels must be collected.

If we use three hash functions in the Bloom filter, each packet incurs three SRAM accesses in order to check whether the flow label carried the packet is already encoded in the Bloom filter. A recent work on one-memory-access Bloom filters [94] shows that three SRAM accesses per packet can be reduced to one. This overhead is further reduced if we only examine the UDF packets and the SYN packets (which carry the label information of TCP traffic). A recent study shows that UDF accounts for 20% of the Internet traffic [10] and the measurement of our campus traffic shows that SYN packets accounts for less than 10% of all TCP traffic. Therefore, the Bloom filter operation only needs to be carried out for less than 28% of all packets, which amortizes the overhead.

2.9 Experiments

We use experiments to evaluate our estimation methods, CSM (Counter Sum estimation Method) and MLM (Maximum Likelihood estimation Method), which are designed based on the randomized counter sharing scheme. We also compare our methods with CB (Counter Braids) [75] and MRSCBF (Multi-Resolution Space-Code Bloom Filters) [66]. Our evaluation is based on the performance metrics outlined in Section 2.1, including per-packet processing time, memory overhead, and estimation accuracy.

The experiments use a network traffic trace obtained from the main gateway of our campus. We perform experiments on various different types of flows, such as per-source flows, per-destination flows, per-source/destination flows, and TCP flows. They all lead to the same conclusions. Without losing generality, we choose TCP flows for presentation. The trace contains about 68 millions of TCP flows and 750 millions of packets. In each measurement period, 10 million packets are processed; it typically covers slightly more than 1 million flows.

2.9.1 Processing Time

The processing time is mainly determined by the number of memory accesses and the number of hash computations per packet. Table 2-1 presents the comparison. CSM or MLM performs two memory accesses and one hash computation for each packet. CB incurs three times of the overhead. It performs six memory accesses and three hash computations for each packet at the first counter level, and in the worst case makes six additional memory accesses and three additional hash computations at the second level. MRSCBF has nine filters. The ith filter uses k_i hash functions and encodes packets with a sampling probability p_i, where $k_1 = 3$, $k_2 = 4$, $k_i = 6$, $\forall i \in [3, 9]$, and $p_i = (\frac{1}{4})^{i-1}$, $\forall i \in [1, 9]$. When encoding a packet, the ith filter performs k_i hash computations and sets k_i bits. Hence, the total number of memory accesses (or hash computations) per packet for all filters is $\sum_{i=1}^{9}(p_i \cdot k_i) \approx 4.47$.

2.9.2 Memory Overhead and Estimation Accuracy

We study the estimation accuracies of CSM and MLM under different levels of memory availability. In each measurement period, 10M packets are processed, i.e., $n = 10$M, which translates into about 8 seconds for an OC-192 link (10+ Gbps) or about 2 seconds for an OC-768 link (40+ Gbps) with an average packet size of 1,000 bytes. The memory M allocated to this particular measurement function is varied from 2Mb (2×2^{20} bits) to 8Mb. The counter length b and the number of counters m are determined based on (2–27). The size of each storage vector is 50.

When $M = 2$Mb, the experimental results are presented in Fig. 2-2. The first plot from the left shows the estimation results by CSM for one measurement period; the results for other measurement periods are very similar. Each flow is represented by a point in the plot, whose x coordinate is the true flow size s and y coordinate is the estimated flow size $\hat{s}$. The equality line, $y = x$, is also shown for reference. An estimation is more accurate if the point is closer to the equality line.

The second plot presents the 95% confidence intervals for the estimations made by CSM. The width of each vertical bar shows the size of the confidence interval at a certain flow size (which is the x coordinate of the bar). The middle point of each bar shows the mean estimation for all flows of that size. Intuitively, the estimation is more accurate if the confidence interval is smaller and the middle point is closer to the equality line.

The third plot shows the estimation results by MLM, and the fourth plot shows the 95% confidence intervals for the estimations made by MLM. Clearly, MLM achieves better accuracy than CSM. The estimation accuracy shown in Fig. 2-2 is achieved with a memory of slightly less than 2 bits per flow,

We can improve the estimation accuracy of CSM or MLM by using more memory. We increase M to 4Mb and repeat the above experiments. The results are shown in Fig. 2-3. We then increase M to 8Mb and repeat the above experiments. The results

are shown in Fig. 2-4. The accuracy clearly improves as the confidence intervals shrink when M becomes larger.

We repeat the same experiments on CB, whose parameters are selected according to [75]. The results are presented in Fig. 2-5. The first plot shows that CB totally fails to produce any meaningful results when the available memory is too small: $M = 2$Mb, which translates into less than 2 bits per flow. In fact, its algorithm cannot converge, but instead produce oscillating results. We have to artificially stop the algorithm after a very long time. The second plot shows that CB works well when $M = 4$Mb. The algorithm still cannot converge by itself, even though it can produce very good results when we artificially stop it after a long time without observing any further improvement in the results. It can be seen that the results carry a small positive bias because most points are on one side of the equality line. The third plot shows that CB is able to return the exact sizes for most flows when the memory is $M = 8$Mb.

Combining the results in Table 2-1, we draw the following conclusion: (1) In practice, we should choose CSM/MLM if the requirement is to handle high measurement throughput (which means low per-packet processing time) or if the available memory is too small such that CB does not work, while relatively coarse estimation is acceptable. (2) We should choose CB if the processing time is less of a concern, sufficient memory is available, and the exact flow sizes are required.

We also run MRSCBF under different levels of memory availability. We begin with $M = 8$Mb. CSM or MLM works very well with this memory size (Fig. 2-4). The performance of MRSCBF is shown in the first plot of Fig. 2-6. There are some very large estimated sizes. To control the scale in the vertical axis, we artificially set any estimation beyond 2,800 to be 2,800. The results demonstrate that MRSCBF totally fails when $M = 8$Mb. The performance of MRSCBF improves when we increase the memory.

The results when $M = 40$Mb are shown in the second plot.[1] In the third plot, when we further increase M to 80Mb,[2] no obvious improvement is observed when comparing the second plot. A final note is that the original paper of MRSCBF uses log scale in their presentation. The third plot in Fig. 2-6 will appear as the fourth plot in log scale.

Clearly, the bitmap-based MRSCBF performs worse than CB, CSM or MLM. To measure flow sizes, counters are superior than bitmaps.

2.10 Extension of Estimation Range

We set the upper bound on the flow size that CSM and MLM can estimate in Section 2.9 to 2,500. However, in today's high-speed networks, the sizes of some flows are much larger than 2,500. In order to extend the estimation range to cover these large flows, we propose four approaches that increase the estimation upper bound, and present extensive experimental results to demonstrate their effectiveness. Since MLM generally performs better than CSE, we only discuss how to extend the estimation range of MLM. CSE can be easily enhanced by similar approaches.

According to Section 2.4, each flow is assigned a unique storage vector. A flow's storage vector consists of l counters and each counter has b bits. Therefore, the maximum number of packets that the storage vector can represent is $l \times (2^b - 1)$. If we increase b by one, the number of packets that the vector can represent will be doubled. Similarly, if we increase l by a certain factor, the number of packets that the vector can represent will be increased by the same factor. Based on these observations, we extend the estimation range of MLM by increasing the value of b and l, respectively. In addition,

[1] At the end of each measurement period, about half of the bits in the filters of MRSCBF are set to ones.

[2] At the end of each measurement period, less than half of the bits in the filters of MRSCBF are set to ones.

we add a sampling module to MLM and consider hybrid SRAM/DRAM implementation to extend the estimation range.

2.10.1 Increasing Counter Size b

Our first approach to extend the estimation range is to enlarge the counter size b. We repeat the same experiment on MLM presented in the third plot of Fig. 2-3 (Section 2.9.2), where $M = 4$Mb, $I = 50$, and $n = 10$M. This time, instead of computing b from (2–27), we vary its value from 6 to 9. The new experimental results are shown in Fig. 2-7. In the first plot, the maximum flow size that MLM can estimate is about 1,400 when $b = 6$. In the second plot, where $b = 7$, the maximum flow size is about 2,800, which is twice of the maximum flow size that the first plot can achieve. When b is set to 8, the third plot shows that the estimation range of MLM is further extended. The fourth plot shows that, when $b = 9$, the maximum flow size that MLM can estimate does not increase any more when comparing with the third plot, which we will explain shortly. The estimation accuracy of the above experiments is presented in Fig. 2-8, where the first plot shows the estimation bias and the second plot shows the standard deviation of the experimental results in Fig. 2-7. Generally speaking, both bias and standard deviation increase slightly when b increases.

Since flows share counters in MLM, the size information of one flow in a counter is the noise to other flows that share the same counter. When the amount of memory allocated to MLM is fixed ($M = 4$Mb in these experiments), a larger value for b will result in a smaller value for m, i.e., the total number of counters is reduced. Hence, each counter has to be shared by more flows, and the average number of packets stored in each counter will increase. That means heavier noise among flows, which degrades the estimation accuracy, as is demonstrated by Figure 2-8. Moreover, although a counter with a larger size b can keep track of a larger number of packets, since it also carries more noise, MLM has to substract more noise from the counter value during the estimation process. As a result, the estimation range cannot be extended indefinitely by

simply increasing b, which explains the fact that the maximum flow size that MLM can estimate does not increase when b reaches 9 in Figure 2-7.

2.10.2 Increasing Storage Vector Size *l*

Our second approach for extending the estimation range is to increase the storage vector size *l*. We repeat the experiments in the previous subsection for MLM with M =4Mb, $b = 7$, and $n = 10$M. We vary *l* from 50 to 1,000. Figure 2-9 presents the experimental results. The first plot shows that the maximum flow size that MLM can estimate is about 5,800 when $l = 50$. As we increase the value of *l*, MLM can estimate increasingly larger flow sizes. However, when *l* becomes too large, estimation accuracy will degrade, which is evident in the fourth plot. The reason is that each flow shares too many counters with others, which results in excessive noise in the counters and consequently introduce inaccuracy in the estimation process.

The estimation accuracy of the above experiments is presented in Fig. 2-10, where the first plot shows the estimation bias and the second plot shows the standard deviation of the experimental results in Fig. 2-9. Generally speaking, both bias and standard deviation increase slightly when *l* increases. Clearly, the value of *l* should not be chosen too large (such as $l = 1,000$) in order to prevent estimation accuracy to degrade significantly.

2.10.3 Employing Sampling Module

In our third approach, we add a sampling module to MLM to enlarge the estimation range. The sampling technique has been widely used in network measurement [13, 35, 36, 66, 128]. We show that it also works for MLM. Let p be the sampling probability. For each packet that the router receives in the data encoding phase, the router generates a random number r in a range $[0, N]$. If $r < p \times N$, the router processes the packet as we describe in Section 2.4. Otherwise, it ignores the packet without encoding it in the counter array. In the data decoding phase, the estimated flow size should be $\frac{\hat{s}}{p}$, where $\hat{s}$ is computed from (2–18). The estimation range is expanded by a factor of $\frac{1}{p}$.

We again repeat the experiments in the previous subsections for MLM with $M = 4$Mb, $l = 50$, and $n = 10$M. The value of b is computed from (2–27). This time, we introduce a sampling probability p and varies its value. Fig. 2-11 presents the experimental results of MLM with $p = 75\%, 50\%, 25\%$, and 2%, respectively. It demonstrates that when the sampling probability decreases, the estimation range increases. However, it comes with a penalty on estimation accuracy. Fig. 2-12 shows the estimation bias and standard deviation of the estimation results in Fig. 2-11. If the sampling probability is not decreased too small, e.g., when $p \geq 25\%$, the increase in bias and standard deviation is insignificant. However, if the sampling probability becomes too small such as 2%, the degradation in estimation accuracy also becomes noticeable.

2.10.4 Hybrid SRAM/DRAM Design

Can we extend the estimation range without any limitation and do so without any degradation in estimation accuracy? This will require a hybrid SRAM/DRAM design. In SRAM, we still choose the value of b based on (2–27). The limited size of each counter means that a counter may be overflowed during the data encoding phase even though the chance for this to happen is very small (Section 2.7). To totally eliminate the impact of counter overflow, we keep another array of counters in DRAM, each of which has a sufficient number of bits. The counters in DRAM are one-to-one mapped to the counters in SRAM. When a counter in SRAM is overflowed, it is reset to zero and the corresponding counter in DRAM is incremented by one. During offline data analysis, the counter values are set based on both SRAM and DRAM data. Because overflow happens only to a small fraction of SRAM counters and a DRAM access is made only after an overflowed SRAM counter is accessed 2^b times, the overall overhead of DRAM access is very small.

2.11 Summary

Per-flow traffic measurement provides real-world data for a variety of applications on accounting and billing, anomaly detection, and traffic engineering. Current online

data collection methods cannot meet the requirements of being both fast and compact. This work proposes a novel data encoding/decoding scheme, which mixes per-flow information randomly in a tight SRAM space for compactness. Its online operation only incurs a small overhead of one hash computation and one counter update per packet. Two offline statistical methods — the counter sum estimation and the maximum likelihood estimation — are used to extract per-flow sizes from the mixed data structures with good accuracy. Due to its fundamentally different design philosophy, the new measurement function is able to work in a tight space where exact measurement is no longer possible, and it does so with the minimal number of memory accesses per packet.

Table 2-1. Number of memory accesses and number of hash computations per packet

	memory accesses	hash computations	constant?
CSM	2	1	Y
MLM	2	1	Y
CB	≥ 6	≥ 3	N
MRSCBF	4.47	4.47	N

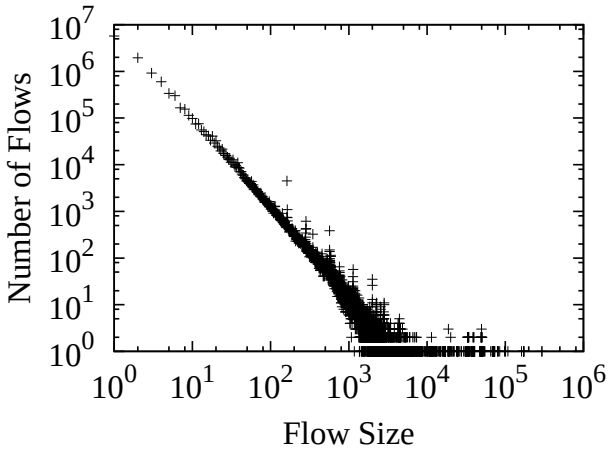

Figure 2-1. Traffic distribution: each point shows the number (y coordinate) of flows that have a certain size (x coordinate).

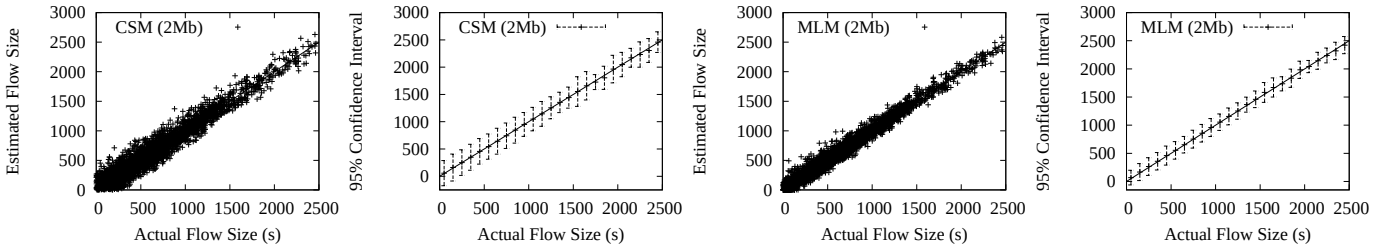

Figure 2-2. • *First Plot*: estimation results by CSE when $M = 2$Mb. Each flow is represented by a point in the plot, whose x coordinate is the true flow size s and y coordinate is the estimated flow size $\hat{s}$. The equality line, $y = x$, is also shown for reference. An estimation is more accurate if the point is closer to the equality line. • *Second Plot*: 95% confidence intervals for the estimations made by CSE when $M = 2$Mb. The width of each vertical bar shows the size of the confidence interval at a certain flow size (which is the x coordinate of the bar). The y coordinate of the middle point of each bar shows the mean estimation for all flows of that size. Intuitively, the estimation is more accurate if the confidence interval is smaller and the middle point is closer to the equality line. • *Third Plot*: estimation results by MLM when $M = 2$Mb. • *Fourth Plot*: 95% confidence intervals for the estimations made by MLM when $M = 2$Mb. In these experiments, $n = 10$M.

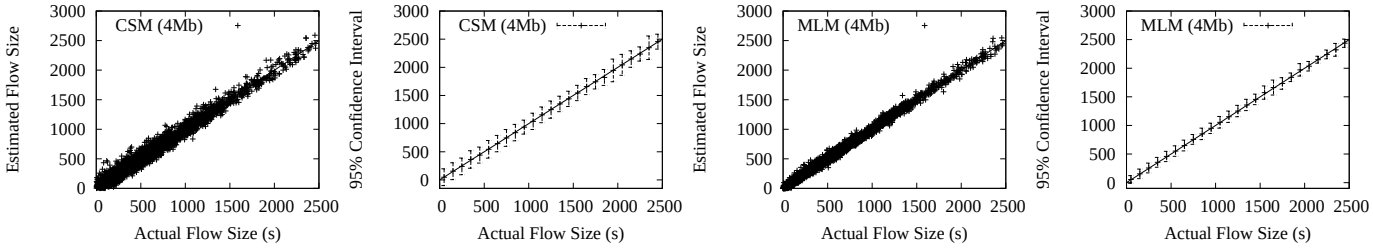

Figure 2-3. • *First Plot*: estimation results by CSM when $M = 4$Mb. • *Second Plot*: 95% confidence intervals for the estimations made by CSM when $M = 4$Mb. • *Third Plot*: estimation results by MLM when $M = 4$Mb. • *Fourth Plot*: 95% confidence intervals for the estimations made by MLM when $M = 4$Mb. See the caption of Fig. 2-2 for more explanation. In these experiments, $n = 10$M.

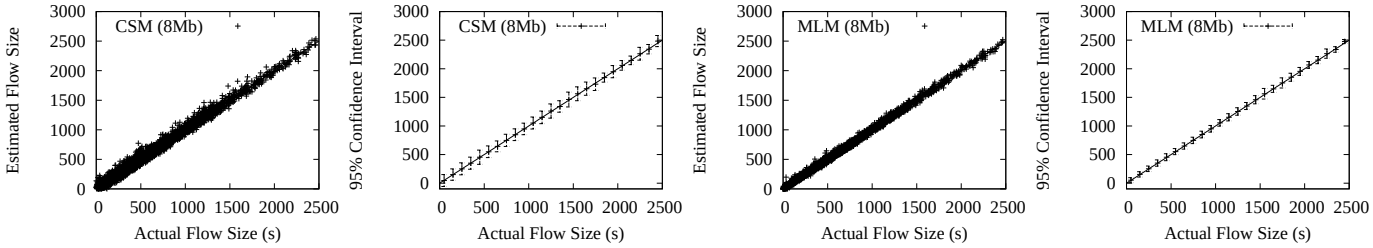

Figure 2-4. • *First Plot*: estimation results by CSM when $M = 8$Mb. • *Second Plot*: 95% confidence intervals for the estimations made by CSM when $M = 8$Mb. • *Third Plot*: estimation results by MLM when $M = 8$Mb. • *Fourth Plot*: 95% confidence intervals for the estimations made by MLM when $M = 8$Mb. See the caption of Fig. 2-2 for more explanation. In these experiments, $n = 10$M.

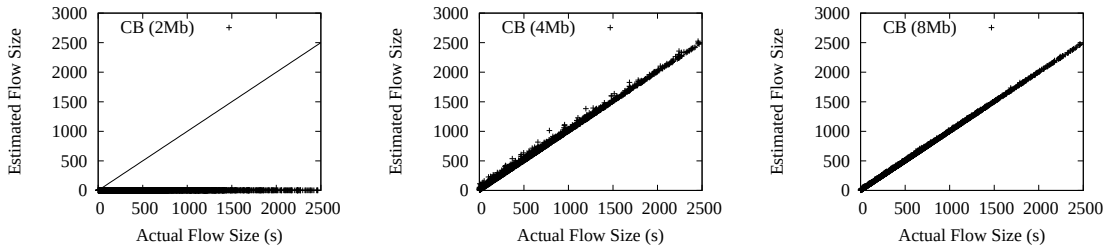

Figure 2-5. • *First Plot*: estimation results by CB when $M = 2$Mb. • *Second Plot*: estimation results by CB when $M = 4$Mb. • *Third Plot*: estimation results by CB when $M = 8$Mb.

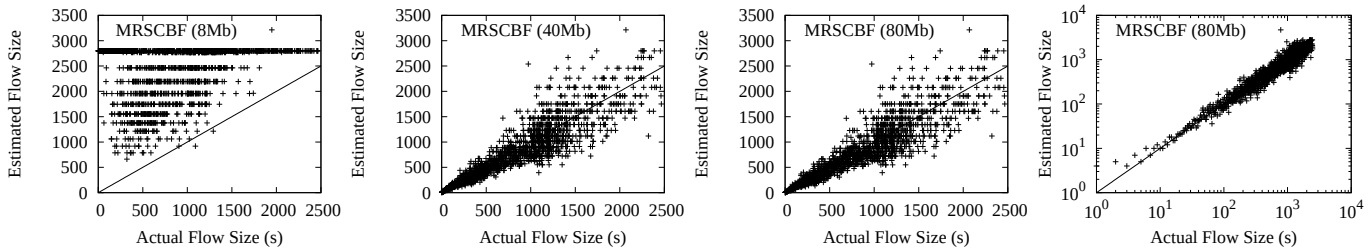

Figure 2-6. • *First Plot*: estimation results by MRSCBF when $M = 8$Mb. • *Second Plot*: estimation results by MRSCBF when $M = 40$Mb. • *Third Plot*: estimation results by MRSCBF when $M = 80$Mb. • *Fourth Plot*: estimation results in logarithmic scale by MRSCBF when $M = 80$Mb.

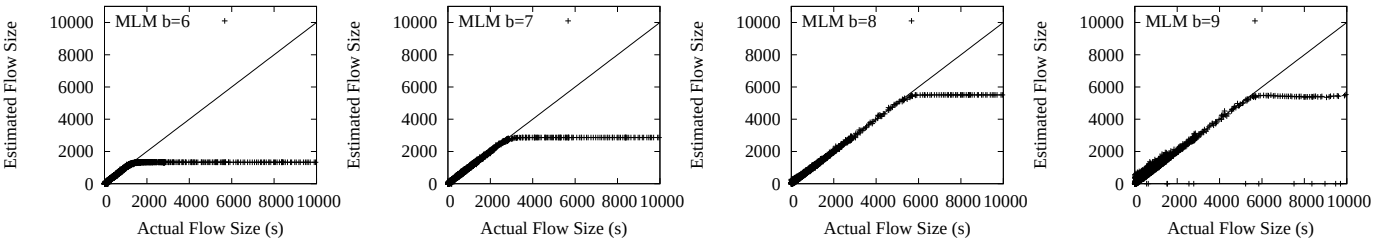

Figure 2-7. • *First Plot*: estimation results by MLM when $b = 6$. • *Second Plot*: estimation results by MLM when $b = 7$. • *Third Plot*: estimation results by MLM when $b = 8$. • *Fourth Plot*: estimation results by MLM when $b = 9$. In these experiments, $n = 10$M, $M = 4$Mb.

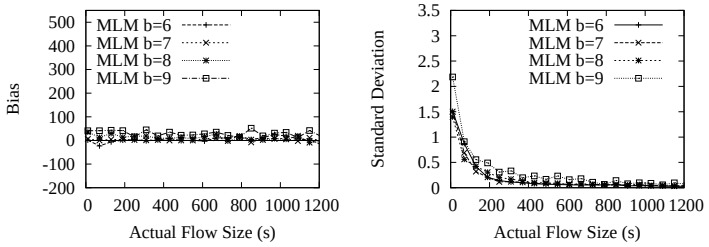

Figure 2-8. • *First Plot*: the estimation bias in the experimental results shown in Figure 2-7. It is measured as $E(\hat{s} - s)$ with respect to s. • *Second Plot*: the standard deviation of the experimental results in Figure 2-7. It is measured as $\frac{\sqrt{Var(\hat{s})}}{s}$.

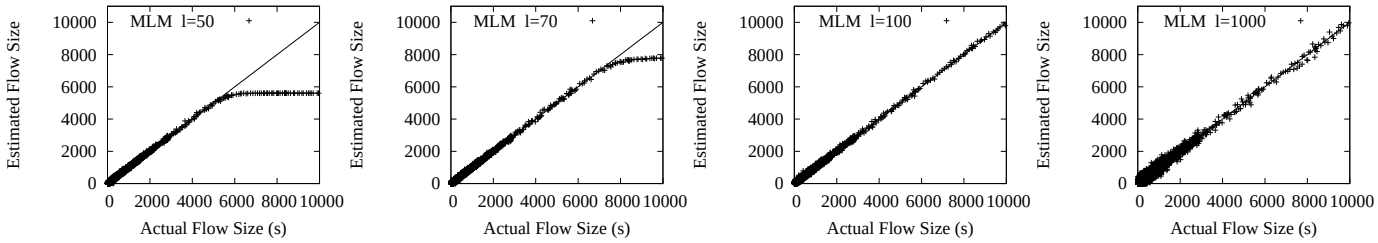

Figure 2-9. • *First Plot*: estimation results by MLM when $l = 50$. • *Second Plot*: estimation results by MLM when $l = 70$. • *Third Plot*: estimation results by MLM when $l = 100$. • *Fourth Plot*: estimation results by MLM when $l = 1000$. In these experiments, $n = 10$M, $M = 4$Mb.

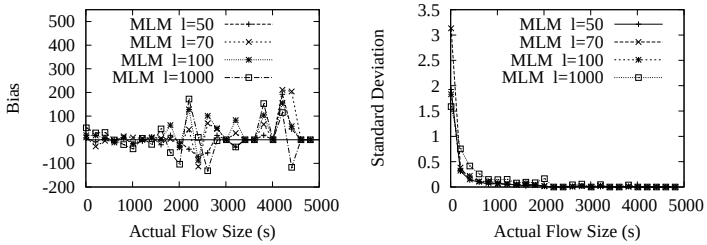

Figure 2-10. • *First Plot*: the estimation bias in the experimental results shown in Figure 2-9. It is measured as $E(\hat{s} - s)$ with respect to s. • *Second Plot*: the standard deviation of the experimental results in Figure 2-7. It is measured as $\frac{\sqrt{Var(\hat{s})}}{s}$.

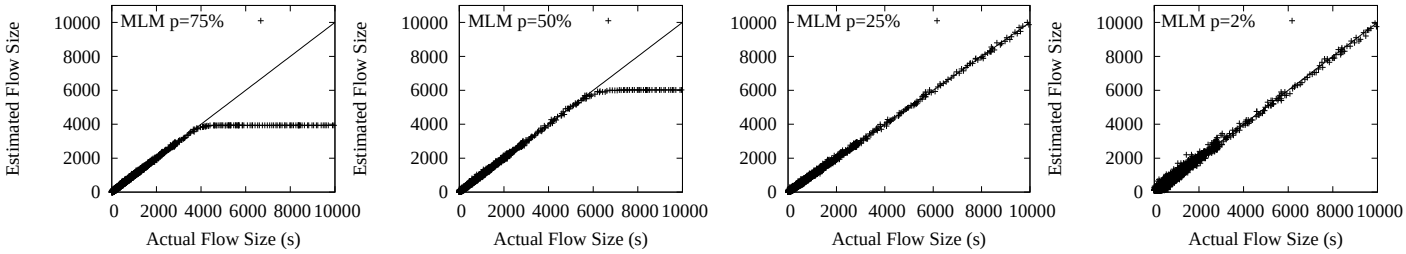

Figure 2-11. • *First Plot*: estimation results by MLM when $p = 75\%$. • *Second Plot*: estimation results by MLM when $p = 50\%$. • *Third Plot*: estimation results by MLM when $p = 25\%$. • *Fourth Plot*: estimation results by MLM when $p = 2\%$. In these experiments, $n = 10$M, $M = 4$Mb.

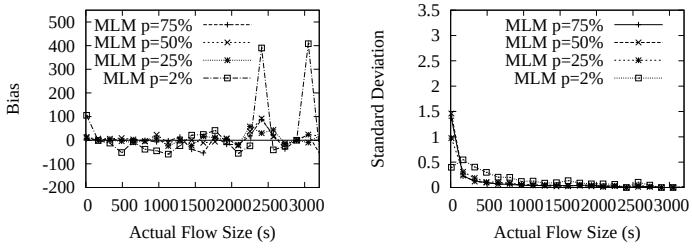

Figure 2-12. • *First Plot*: the estimation bias in the experimental results shown in Figure 2-11. It is measured as $E(\hat{s} - s)$ with respect to s. • *Second Plot*: the standard deviation of the experimental results in Figure 2-11. It is measured as $\frac{\sqrt{Var(\hat{s})}}{s}$.

CHAPTER 3
SCAN DETECTION IN HIGH-SPEED NETWORKS

This chapter attempts to fill a missing piece in the existing research landscape. Its goal is to optimally combine probabilistic sampling and bit sharing — the two most effective memory reduction methods — to fulfill quantitatively specified performance objectives. We have three contributions. First, we present a generalized scheme for scan detection based on bit sharing. It incorporates probabilistic sampling and enhances security through a private key. Second, as the main results in this chapter, we show analytically how to optimally combine probabilistic sampling and bit sharing. We derive the probability for the integrated sampling/bit-sharing scheme to miss reporting a scanner and the probability to mistakenly report a non-scanner. We then construct an iterative algorithm that solves a non-linear constrained optimization problem to obtain the optimal values for the sampling probability and other parameters such that the memory required to bound the above probabilities is minimized. Third, we perform experiments based on real traffic trace and demonstrate that, using the optimal parameters obtained from this work, we can reduce the memory consumption by three to twenty times when comparing with the best existing work. Remarkably, the number of bits required by our scheme is far smaller than the number of distinct sources in the traffic trace. On average, it takes much less than 1 bit per source to perform scan detection.

The rest of this chapter is organized as follows: Section 3.1 gives the problem definition. Section 3.2 describes the related work. Section 3.3 describes our generalized bit-sharing scheme. Section 3.4 presents the analytical results for optimal parameters. Section 3.5 presents the experimental results. Section 3.6 gives the summary.

3.1 Problem Statement

The number of distinct destination addresses that an external source has contacted is called the *spread* of the source. The problem of scan detection is to configure a

firewall or an intrusion detection system to report all external sources whose spreads exceed a certain threshold during a measurement period. We refer to these sources as *potential scanners* (or scanners for short).

If a firewall or an IDS keeps the exact count of distinct destinations that each source has contacted, it is able to report the scanners precisely. However, keeping track of per-source information consumes a large amount of resources. The limited SRAM may only allow us to estimate a rough count of distinct destinations that each source contacts [110, 114, 115, 129]. When precisely reporting scanners is infeasible, the function of scan detection must be defined in a probabilistic term.

We adopt the *probabilistic performance objective* from [110]. Let h and l be two positive integers, $h > l$. Let α and β be two probability values, $0 < \alpha < 1$ and $0 < \beta < 1$. The objective is to report any source whose spread is h or larger with a probability no less than α and report any source whose spread is l or smaller with a probability no more than β. Let k be the spread of an arbitrary source src. The objective can be expressed in terms of conditional probabilities:

$$\begin{aligned} &Prob\{\text{report } src \text{ as a scanner} \mid k \geq h\} \geq \alpha \\ &Prob\{\text{report } src \text{ as a scanner} \mid k \leq l\} \leq \beta \end{aligned} \tag{3–1}$$

We treat the report of a source whose spread is l or smaller as a *false positive*, and the non-report of a source whose spread is h or larger as a *false negative*. Hence, the above objective can also be stated as bounding the false positive ratio by β and the false negative ratio by $1 - \alpha$.

Our goal is to minimize the amount of SRAM that is needed for achieving the above objective.

The memory requirement for detecting aggressive scanners is likely to be small. For example, suppose an aggressive scanner makes 100 distinct contacts each second, whereas a normal host rarely makes 100 distinct contacts in a day. To detect such a

scanner, a firewall can set the measurement period to be a second. The number of contacts that pass the firewall in such a small period is likely to be small. Consequently, it does not need much memory to store them. However, the situation is totally different for stealthy scanners that make contacts at low rates. Consider a scanner that makes 500 distinct contacts a day. If the measurement period is a day, we are able to set it apart from the normal hosts. However, if the measurement period is a second, we will not detect this scanner because it makes less than 0.006 contact per second on average.

In order to detect different types of scanners, a firewall may execute multiple instances of a scan detection function simultaneously, each having a different measurement period. For aggressive scanners, a small period will be chosen so that they can be detected in real time. For stealthy scanners, a large period will be chosen. In the latter case, timely detection is of second priority because the scanners themselves operate slowly. But the memory requirement is of first priority due to the large number of contacts that are expected to pass through the firewall in a long measurement period. Reducing memory consumption is the focus of this study.

3.2 Related Work

Venkataraman et al. [110] use hash tables to store the addresses of the sampled contacts. Their main contribution is to derive the optimal sampling probability that achieves a classification objective with pre-specified upper bounds on false-positive ratio and false-negative ratio. However, because their algorithms store the contact addresses, it leaves great room for improvement. Even if Bloom filters are used, the room for improvement is still significant, as we have argued in Section 3.5.

Estan et al. [41] propose a variety of bitmap algorithms to store the contacts (or active flows in their context). It saves space because each destination address is stored as a bit. However, assigning one bitmap to each source is not cheap if the average number of contacts per source is small. In addition, an index structure is needed

to map a source to its bitmap. It is typically a hash table where each entry stores a source address and a pointer to the corresponding bitmap. Cao et al. [13] develop the thresholded bitmap algorithm based on the virtual bitmap algorithm presented in [41] for spread estimation. They use probabilistic sampling to reduce the information to be stored. Zhao et al. [129] share a set of bitmaps among all sources. The scheme assigns three pseudo-randomly selected bitmaps to each source. When the source contacts a destination, the destination is stored by setting one bit in each of the three bitmaps. Because the bitmaps are shared by others, the information stored for one source becomes noise for others. Yoon et al. [114, 115] observe that the noise introduced by sharing bitmaps cannot be appropriately removed if the number of bitmaps is not sufficiently large. By sharing bits instead of bitmaps, CSE considerably reduces the memory consumption.

Also related is the work by Bandi et al. [5] on the heavy distinct hitter problem, which is essentially the same as spreader classification. Their algorithm exploits TCAM (Ternary Content Addressable Memory), a special kind of memory found in NPUs (Network Processing Units). The emphasis of their work is on the processing time. A related branch of research is the detection of heavy-hitters [17, 29, 33, 35, 40, 48, 81, 123]. A heavy-hitter is a source that sends a lot of packets during a measurement period no matter whether the packets are sent to a few or many distinct destinations.

3.3 An Efficient Scan Detection Scheme

This section presents our efficient scan detection scheme (ESD).

3.3.1 Probabilistic Sampling

To save resources, a firewall (or IDS) samples the contacts made by external sources to internal destinations, and it only stores the sampled contacts. The firewall selects contacts for storage uniformly at random with a sampling probability p. The sampling procedure is simple: the firewall hashes the source/destination address pair of each packet that arrives at the external network interface into a number in a range

$[0, N)$. If the hash result is smaller than $p \times N$, the contact will be stored; otherwise, the contact will not be stored.

3.3.2 Bit-Sharing Storage

A bit array (also called *bitmap*) may be used to store all sampled contacts made by a source [41]. The bits are initially zeros. Each sampled contact is hashed to a bit in the bitmap, and the bit is set to one. At the end of the measurement period, we can estimate the number of contacts, i.e., the spread of the source, based on the number of zeros remaining in the bitmap. Using per-source bitmaps is not memory-efficient. On one hand, the size of each bitmap has to be large enough to ensure the accuracy in estimating the spread values of the scanners. On the other hand, the vast majority of normal sources have small spread values and their bitmaps are largely wasted because most bits remain zeros. To solve this problem, we want to put those wasted bits in good use by allowing bitmaps to share their bits.

To fully share the available bits, ESD stores contacts from different sources in a single bit array B. Let m be the number of bits in B. For an arbitrary source src, we use a hash function to pseudo-randomly select a number of bits from B to store the contacts made by src. The indices of the selected bits are $H(src \oplus R[0])$, $H(src \oplus R[1])$, ..., $H(src \oplus R[s-1])$, where $H(...)$ is a hash function whose range is $[0, m)$, R is an integer array, storing randomly chosen constants whose purpose is to arbitrarily alter the hash result, and s ($\ll m$) is a system parameter that specifies the number of bits to be selected. The above bits form a *logical bitmap* of source src, denoted as $LB(src)$.

Similarly, a logical bitmap can be constructed from B for any other source. Essentially, we embed the bitmaps of all possible sources in B. The bit-sharing relationship is dynamically determined on the fly as each new source src' whose contacts are sampled by the firewall will be allocated a logical bitmap $LB(src')$ from B.

At the beginning of a measurement period, all bits in B are reset to zeros. Consider an arbitrary contact $\langle src, dst \rangle$ that is sampled for storage, where src is the source

address and dst is the destination address. The firewall sets *a single bit* in B to one. Obviously, it must also be a bit in the logical bitmap $LB(src)$. The index of the bit to be set for this contact is given as follows:

$$H(src \oplus R[H(dst \oplus K) \bmod s]).$$

The second hash, $H(dst \oplus K)$, ensures that the bit is pseudo-randomly selected from $LB(src)$. The private key K is introduced to prevent the *hash collision attacks*. In such an attack, a scanner src finds a set of destination addresses, dst_1, dst_2, ..., that have the same hash value, $H(dst_1) = H(dst_2) = \ldots$ If it only contacts these destinations, the same bit in $LB(src)$ will be set, which allows the scanner to stay undetected. This type of attacks can be prevented if we use a cryptographic hash function such as MD5 or SHA1, which makes it difficult to find destination addresses that have the same hash value. However, if a weaker hash function is used for performance reason, then a private key becomes necessary. Without knowing the key, the scanners will not be able to predict which destination addresses produce the same hash value.

To store a contact, ESD only sets a single bit and performs two hash operations. This is more efficient than the methods that use hash tables [110] or have features similar to Bloom filters that require setting multiple bits for storing each contact [129].

3.3.3 Maximum Likelihood Estimation and Scanner Report

At the end of the measurement period, ESD will send the content of B to an offline data processing center. There, the logical bitmap of each source src is extracted and the estimated spread $\hat{k}$ of the source is computed. Only if $\hat{k}$ is greater than a threshold value T, ESD reports the source as a potential scanner. We will discuss how to keep track of the source addresses in Section 3.3.5, and explain how to determine the threshold T in Section 3.4. Below we derive the formula for $\hat{k}$.

Let k be the true spread of source src, and n be the number of distinct contacts made by all sources. Let V_m be the fraction of bits in B whose values are zeros at the

end of the measurement period, V_s be the fraction of bits in $LB(src)$ whose values are zeros, and U_s be the number of bits in $LB(src)$ whose values are zeros. Clearly, $V_s = \frac{U_s}{s}$. Depending on the context, V_m (or V_s, U_s) is used either as a random variable or an instance value of the random variable.

The probability for any contact to be sampled for storage is p. Consider an arbitrary bit b in $LB(src)$. A sampled contact made by src has a probability of $\frac{1}{s}$ to set b to '1', and a sampled contact made by any other source has a probability of $\frac{1}{m}$ to set b to '1'. Hence, the probability $q(k)$ for b to remain '0' at the end of the measurement period is

$$q(k) = (1 - \frac{p}{m})^{n-k}(1 - \frac{p}{s})^{k}. \tag{3–2}$$

Each bit in $LB(src)$ has a probability of $q(k)$ to remain '0'. The observed number of '0' bits in $LB(src)$ is U_s. The likelihood function for this observation to occur is given as follows:

$$L = q(k)^{U_s}(1 - q(k))^{s-U_s}. \tag{3–3}$$

In the standard process of maximum likelihood estimation, the unknown value k is technically treated as a variable in (3–3). We want to find an estimate $\hat{k}$ that maximizes the likelihood function. Namely,

$$\hat{k} = \arg\max_{k}\{L\}. \tag{3–4}$$

Since the maxima is not affected by monotone transformations, we use logarithm to turn the right side of (3–3) from product to summation:

$$\ln(L) = U_s \cdot \ln(q(k)) + (s - U_s) \cdot \ln(1 - q(k)).$$

From (3–2), the above equation can be written as

$$\begin{aligned}\ln(L) =& U_s((n-k)\ln(1-\frac{p}{m}) + k\ln(1-\frac{p}{s})) \\ &+ (s-U_s)\cdot\ln(1-(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k).\end{aligned}$$

To find the maxima, we differentiate both sides:

$$\frac{\partial \ln(L)}{\partial k} = \ln(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}) \cdot \frac{U_s - s(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k}{1-(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k}. \tag{3–5}$$

We then let the right side be zero. That is,

$$U_s = s(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k. \tag{3–6}$$

Taking logarithm on both sides, we have

$$\begin{aligned}\ln\frac{U_s}{s} &= n\ln(1-\frac{p}{m}) + k(\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})), \\ k &= \frac{\ln V_s - n\ln(1-\frac{p}{m})}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})}.\end{aligned} \tag{3–7}$$

where $V_s = \frac{U_s}{s}$. Suppose the number of sources (which equals to the number of logical bitmaps) is sufficiently large. Because every bit in every logical bitmap is randomly selected from B, in this sense, each of the n contacts has about the same probability $\frac{p}{m}$ of setting any bit in B. Hence, we have

$$E(V_m) = (1-\frac{p}{m})^n. \tag{3–8}$$

Applying (3–8) to (3–7), we have

$$k = \frac{\ln V_s - \ln E(V_m)}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})}. \tag{3–9}$$

Replacing $E(V_m)$ by the instance value V_m, we have the following estimation for k.

$$\hat{k} = \frac{\ln V_s - \ln V_m}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})}, \tag{3–10}$$

where V_s can be measured by counting the number of zeros in $LB(src)$, V_m can be measured by counting the number of zeros in B, and s, p and m are pre-set parameters of ESD (see the next section).

3.3.4 Variance of V_m

Let A_i be the event that the ith bit in B remains '0' at the end of the measurement period and 1_{A_i} be the corresponding indicator random variable. Let U_m be the random variable for the number of '0' bits in B. We first derive the probability for A_i to occur and the expected value of U_m. For an arbitrary bit in B, each distinct contact has a probability of $\frac{p}{m}$ to set the bit to one. All contacts are independent of each other when setting bits in B. Hence,

$$Prob\{A_i\} = (1 - \frac{p}{m})^n, \quad \forall i \in [0, s).$$

The probability for A_i and A_j, $\forall i, j \in [0, m), i \neq j$, to happen simultaneously is

$$Prob\{A_i \cap A_j\} = (1 - \frac{2p}{m})^n.$$

Since $V_m = \frac{U_m}{m}$ and $U_m = \sum_{i=1}^{m} 1_{A_i}$, we have

$$\begin{aligned} E(V_m^2) &= \frac{1}{m^2} E((\sum_{i=1}^{m} 1_{A_i})^2) \\ &= \frac{1}{m^2} E(\sum_{i=1}^{m} 1_{A_i}^2) + \frac{2}{m^2} E(\sum_{1 \le i < j \le m} 1_{A_i} 1_{A_j}) \\ &= \frac{1}{m}(1 - \frac{p}{m})^n + \frac{m-1}{m}(1 - \frac{2p}{m})^n. \end{aligned}$$

Based on (3–8) and the equation above, we have

$$\begin{aligned} Var(V_m) &= E(V_m^2) - E(V_m)^2 \\ &= \frac{1}{m}(1 - \frac{p}{m})^n + \frac{m-1}{m}(1 - \frac{2p}{m})^n - (1 - \frac{p}{m})^{2n} \\ &\simeq \frac{e^{-\frac{np}{m}}(1 - (1 + \frac{np^2}{m})e^{-\frac{np}{m}})}{m}. \end{aligned} \tag{3–11}$$

3.3.5 Source Addresses

ESD does not store the source address of *every* arrival packet. Instead, it stores a source address only when a contact sets a bit in B from '0' to '1'. Hence, the frequency of storing source addresses is much smaller than the frequency at which contacts are sampled for setting bits in B. First, numerous packets may be sent from a source to a destination in a TCP/UDP session. Only the first sampled packet may cause the source address to be stored because only the first packet sets a bit from '0' to '1' and the remaining packets will set the same bit (which is already '1'). Second, a source may send thousands or even millions of packets through a firewall, but the number of times its address will be stored is bounded by s (which is the number of bits in the source's logical bitmap). In summary, because the operation of storing source addresses is relatively infrequent, these addresses can be stored in the main memory.

3.4 Optimal System Parameters and Minimum Memory Requirement

In this section, we first develop the constraints that the system parameters must satisfy in order to achieve the probabilistic performance objective. Based on the constraints, we determine the optimal values for the size s of the logical bitmaps, the sampling probability p, and the threshold T. We also determine the minimum amount of memory m that should be allocated for ESD to achieve the performance objective. Recall that on-die SRAM may be shared by other functions.

3.4.1 Report Probability

Consider an arbitrary source *src* whose spread is k. Given a set of system parameters, m, s, p and T, we derive the probability for ESD to report *src* as a scanner,

i.e., $Prob\{\hat{k} \geq T\}$. From (3–10), we know that the following inequalities are equivalent.

$$\begin{aligned} \hat{k} &\geq T \\ \frac{\ln V_s - \ln V_m}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})} &\geq T \\ V_s &\leq V_m(\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T \end{aligned}$$

Let U_s be the random variable for the number of '0' bits in $LB(src)$. $U_s = s \cdot V_s$. The above inequality becomes

$$U_s \leq s \cdot V_m \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T. \tag{3–12}$$

For a set of parameters m, s, p and T, we define a constant

$$C = s \cdot V_m \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T,$$

where the instance value of V_m can be measured from B after the measurement period. Hence, the probability for ESD to report src is $Prob\{\hat{k} \geq T\} = Prob\{U_s \leq C\}$.

U_s follows the binomial distribution with parameters s and $q(k)$, where $q(k)$ in (3–2) is the probability for an arbitrary bit in $LB(src)$ to remain zero at the end of the measurement period. Hence, the probability of having exactly i zeros in $LB(src)$ is given by the following probability mass function:

$$Prob\{U_s = i\} = \binom{s}{i} \cdot q(k)^i \cdot (1-q(k))^{s-i}. \tag{3–13}$$

We must have

$$\begin{aligned} Prob\{\hat{k} \geq T\} &= Prob\{U_s \leq C\} \\ &= \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(k)^i \cdot (1-q(k))^{s-i}. \end{aligned} \tag{3–14}$$

3.4.2 Constraints for the System Parameters

We derive the constraints that the system parameters must satisfy in order to achieve the performance objective in (3–1). First, we give the variance of V_m, which is provided in (3–11).

$$Var(V_m) \simeq \frac{e^{-\frac{np}{m}}(1-(1+\frac{np^2}{m})e^{-\frac{np}{m}})}{m}. \tag{3–15}$$

It approaches to zero as m increases. In Figure 3-1, we plot the ratio of the standard deviation $Std(V_m) = \sqrt{Var(V_m)}$ to $E(V_m)$, which can be found in (3–8). The figure shows that $Std(V_m)/E(V_m)$ is very small when m is reasonably large. In this case, we can approximately treat V_m as a constant.

$$V_m \simeq E(V_m) \simeq (1-\frac{p}{m})^n. \tag{3–16}$$

The probabilistic performance objective can be stated as two requirements. First, the probability for ESD to report a source with $k \geq h$ must be at least α. That is, $Prob\{\hat{k} \geq T\} \geq \alpha, \forall k \geq h$. From (3–14), this requirement can be written as the following inequality:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(k)^i \cdot (1-q(k))^{s-i} \geq \alpha,$$

where $C = s \cdot V_m \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T \simeq s \cdot (1-\frac{p}{m})^n \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T$. The left side of the inequality is an increasing function in k. Hence, to satisfy the requirement in the worst case when $k = h$, the following constraint for the system parameters must be met:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1-q(h))^{s-i} \geq \alpha. \tag{3–17}$$

Second, the probability for ESD to report a source with $k \leq l$ must be no more than β. This requirement can be similarly converted into the following constraint:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1 - q(l))^{s-i} \leq \beta. \tag{3–18}$$

3.4.3 Optimal System Parameters

Our goal is to optimize the system parameters such that the memory requirement, m, is minimized under the constraints (3–17) and (3–18). The problem is formally defined as follows.

$$\begin{aligned} &\textit{Minimize} \quad m \\ &\textit{Subject to} \quad \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1 - q(h))^{s-i} \geq \alpha, \\ &\qquad \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1 - q(l))^{s-i} \leq \beta, \\ &\qquad C = s \cdot (1 - \frac{p}{m})^n \cdot (\frac{1 - \frac{p}{s}}{1 - \frac{p}{m}})^T. \end{aligned} \tag{3–19}$$

The parameters, h, l, α and β, are specified in the performance objective. The value of n is decided based on the history data in the past measurement periods. To be conservative, we take the the maximum number n^* of distinct contacts observed in a number of previous measurement periods. More specifically, (3–8) can be turned into a formula for estimating n in each previous period if we replace $E(V_m)$ with the instance value V_m.

$$\hat{n} = -\frac{m}{p} \ln V_m \tag{3–20}$$

We derive the relative bias and the relative standard deviation of the above estimation.

$$Bias(\frac{\hat{n}}{n}) = E(\frac{\hat{n}}{n}) - 1 \simeq \frac{e^{\frac{np}{m}} - \frac{np^2}{m} - 1}{2np} \tag{3–21}$$

$$Std(\frac{\hat{n}}{n}) = \frac{\sqrt{m}}{np}(e^{\frac{np}{m}} - \frac{np^2}{m} - 1)^{1/2} \tag{3–22}$$

They both approach to zero as m increases. Based on the largest $\hat{n}$ value observed in a certain number of past measurement periods, we can set the value of n^*.

To solve the constrained optimization problem (3–19), we need to determine the optimal values of the remaining three system parameters, s, p and T, such that m will be minimized. We consider the left side of (3–17) as a function $F_h(m, s, p, T)$, and the left side of (3–18) as $F_l(m, s, p, T)$. Namely,

$$F_h(m, s, p, T) = \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1 - q(h))^{s-i},$$
$$F_l(m, s, p, T) = \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1 - q(l))^{s-i}.$$

Both of them are *non-increasing functions in* T, according to the relation between C and T. In the following, we present an iterative numerical algorithm to solve the optimization problem. The algorithm consists of four procedures.

Algorithm 1 $Potential(m, s, p)$

INPUT: m, s, p and β
OUTPUT: The maximum value of $F_h(m, s, p, T)$ under the condition that $F_l(m, s, p, T) \leq \beta$

Pick a small integer T_1 such that $F_l(m, s, p, T_1) > \beta$ and a large integer T_2 such that $F_l(m, s, p, T_2) \leq \beta$;
while $T_2 - T_1 > 1$ **do**
  $\bar{T} = \lfloor (T_1 + T_2)/2 \rfloor$;
  if $F_l(m, s, p, \bar{T}) \leq \beta$ **then** $T_1 = \bar{T}$ **else** $T_2 = \bar{T}$;
end while
$T^* = \bar{T}$;
return $F_h(m, s, p, T^*)$;

First, we construct a procedure called $Potential(m, s, p)$, which takes a value of m, a value of s and a value of p as input and returns the maximum value of $F_h(m, s, p, T)$ under the condition that $F_l(m, s, p, T) \leq \beta$ is satisfied. Because $F_h(m, s, p, T)$ is a non-increasing function in T, we need to find the smallest value of T that satisfies $F_l(m, s, p, T) \leq \beta$. That can be done numerically through binary search: Pick

a small integer T_1 such that $F_l(m, s, p, T_1) \geq \beta$ and a large integer T_2 such that $F_l(m, s, p, T_2) \leq \beta$. We iteratively shrink the difference between them by resetting one of them to be the average $\frac{T_1+T_2}{2}$, while maintaining the inequalities, $F_l(m, s, p, T_1) \geq \beta$ and $F_l(m, s, p, T_2) \leq \beta$. The process stops when $T_1 = T_2$, which is denoted as T^*. The procedure $Potential(m, s, p)$ returns $F_h(m, s, p, T^*)$. The pseudo code is presented in Algorithm 3.4.3.

Essentially, what $Potential(m, s, p)$ returns is the maximum value of the left side in (3–17) under the condition that (3–18) is satisfied. The difference between $Potential(m, s, p)$ and α provides us with a quantitative indication on how conservative or aggressive we have chosen the value of m. If $Potential(m, s, p) - \alpha$ is positive, it means that the performance achieved by the current memory size is more than required. We shall reduce m. On the contrary, if $Potential(m, s, p) - \alpha$ is negative, we shall increase m.

Given the above semantics, when we determine the optimal values for p and s, our goal is certainly to maximize the return value of $Potential(m, s, p)$.

Second, given a value of m and a value of s, we construct a procedure $OptimalP(m, s)$ that determines the optimal value p^* such that $Potential(m, s, p^*)$ is maximized. When the values of m and s are fixed, $Potential(m, s, p)$ becomes a function of p. It is a curve as illustrated in Figure 3-2; see explanation under the caption (a) and ignore the arrows in the figure for now.

We use a binary search algorithm to find a near-optimal value of p. Let $p_1 = 0$ and $p_2 = 1$. Let δ be a small positive value (such as 0.001). Repeat the following operation: Let $\bar{p} = (p_1 + p_2)/2$. If $Potential(m, s, \bar{p}) < Potential(m, s, \bar{p} + \delta)$, set p_1 to be $\bar{p}$; otherwise, set p_2 to be $\bar{p}$. The above iterative operation stops when $p_2 - p_1 < \delta$. The procedure $OptimalP(m, s)$ returns $(p_1 + p_2)/2$, which is within $\pm\delta/2$ of the optimal. This difference can be made arbitrarily small when we decrease δ at the expense of increased computation overhead. We want to stress that it is one-time overhead (not

online overhead) to determine the system parameters before deployment. The operation of $OptimalP(m, s)$ is illustrated by the arrows in Figure 3-2; see explanation under the caption (b).

Third, given a value of m, we construct a procedure $OptimalS\ (m)$ that determines the optimal value s^* such that $Potential(\ m, s^*, OptimalP(m, s^*))$ is maximized. When the value of m is fixed, $Potential(m, s, OptimalP(m, s))$ becomes a function of s. It is a curve as illustrated in Figure 3-3. We can use a binary search algorithm similar to that of $OptimalP(m, s)$ to find s^*.

Fourth, we construct a procedure $OptimalM()$ that determines the minimum memory requirement m^* through binary search: Denote $Potential(m, OptimalS(m), OptimalP(m, OptimalS(m)))$ as $Potential(m, ...)$. Pick a small value m_1 such that $Potential(m_1, ...) \leq \alpha$, which means that the performance objective is not met — more specifically, according to the semantics of $Potential(...)$, the constraint (3–17) cannot be satisfied if the constraint (3–18) is satisfied. Pick a large value m_2 such that $Potential(m_2, ...) \geq \alpha$, which means that the performance objective is met. Repeat the following operation. Let $\bar{m} = \lfloor (m_1 + m_2)/2 \rfloor$. If $Potential(\bar{m}, ...) \leq \alpha$, set m_1 to be $\bar{m}$; otherwise, set m_2 to be $\bar{m}$. The above iterative operation terminates when $m_1 = m_2$, which is returned by the procedure $OptimalM()$.

In practice, a network administrator will first define the performance objective that is specified by α, β, h and l. He or she sets the value of n^* based on history data, and then sets $m = OptimalM()$, $s = OptimalS(m)$, $p = OptimalP(m, s)$ and T as the threshold value T^* before the last call to $Potential(m, s, p)$ is returned during the execution of $Optim\text{-}alM()$. After the firewall (or IDS) is configured with these parameters and begins to measure the network traffic, it also monitors the value of n^*. If the maximum number of distinct contacts in a measurement period changes significantly, the values of m, s, p and T will be recomputed.

3.5 Experiments

3.5.1 Experimental Setup

We evaluate the performance of ESD and compare it with the existing work, including the Two-level Filtering Algorithm (TFA) [110], the Thresholded Bitmap Algorithm (TBA) [13], and the Compact Spread Estimator (CSE) [114]. TFA uses two filters to reduce both the number of sources to be monitored and the number of contacts to be stored. It is designed to satisfy the probabilistic performance objective in (3–1). TBA is not designed for meeting the probabilistic performance objective. It cannot ensure that the false positive/false negative ratios are bounded. CSE is designed to estimate the spreads of the external sources in a very compact memory space. It can be used for scan detection by reporting the sources whose estimated spreads exceed a certain threshold. However, the design of CSE makes it unsuitable for meeting the objective in (3–1).

Online Streaming Module (OSM) [129] is another related work. We do not implement OSM in this study because Yoon et al. show that, given the same amount of memory, CSE estimates spread values more accurately than OSM [114]. Moreover, the operations of OSM share certain similarity with Bloom filters. To store each contact, it performs three hash functions and makes three memory accesses. In comparison, ESD performs two hash functions and makes one memory access.

The experiments use a real Internet traffic trace captured by Cisco's Netflow at the main gateway of our campus for a week. For example, in one day of the week, the traffic trace records 10,702,677 distinct contacts, 4,007,256 distinct source IP addresses and 56,167 distinct destination addresses. The average spread per source is 2.67, which means a source contacts 2.67 distinct destinations on average. Figure 3-4 shows the number of sources with respect to the source spread in log scale. The number of sources decreases exponentially as the spread value increases from 1 to 500. After that, there is zero, one or a few sources for each spread value.

We implement ESD, TFA, TBA and CSE, and execute them with the traffic trace as input. As part of the setup in each experiment, the values of h and l are given to specify what to report as scanners. For example, if $h = 500$ and $l = 0.7h$, the sources whose spreads are 500 or more should be reported, and the sources whose spreads are 350 or less should not be reported. In the experiments, the source of a contact is the IP address of the sender and the destination is the IP address of the receiver. The measurement period is one day. A long measurement period helps to separate low-rate scanners from normal hosts. The experimental results are the average over the week-long data.

One performance metric used in comparison is the amount of memory that is required for a scan detection scheme to meet a given probability performance objective. Remarkably, the number of bits required by ESD is far smaller than the number of distinct sources in the traffic trace. That is, ESD requires much less than 1 bit per source to perform scan detection. Other performance metrics include the false positive ratio and the false negative ratio, which will be explained further shortly.

3.5.2 Comparison in Terms of Memory Requirement

The first set of experiments compares ESD and TFA for the amount of memory that they need in order to satisfy a given probabilistic performance objective, which is specified by four parameters, α, β, h, and l. See Section 3.1 for the formal definition of the performance objective. We do not compare TBA and CSE here because they are not designed to meet this objective.

The memory required by ESD is determined based on the iterative algorithm in Section 3.4.3. The values of other parameters, s, T and p, are decided by the same algorithm. Using these parameters, we perform experiments on ESD with the traffic trace as input, and the experimental results confirm that the performance objective is indeed achieved for each day during the week. The amount of memory required by TFA

is determined experimentally based on the method in [110] together with the traffic trace. The parameters of TFA are chosen based on the original paper.

The memory requirements of ESD and TFA are presented in Tables 3-1-3-2 with respect to α, β, h and l. For $\alpha = 0.9$ and $\beta = 0.1$, Table 3-1 shows that TFA requires six to twenty-four times of the memory that ESD requires, depending on the values of h and l (which the system administrator will select based on the organization's security policy). For example, when $h = 500$ and $l = 0.5h$, ESD reduces the memory consumption by an order of magnitude when comparing with TFA.

To demonstrate the impact of probabilistic sampling, the table also includes the memory requirement of ESD when sampling is turned off (by setting $p = 1$). This version of ESD is denoted as ESD-1. Since p is set as a constant, the iterative algorithm in Section 3.4.3 needs to be slightly modified: The procedure $OptimalP(m, s)$ will always return 1, while other procedures remain the same. Table 3-1 shows that the memory saved by sampling is significant when h is large. For example, when $h = 5,000$ and $l = 0.3h$, ESD with sampling uses less than one thirteenth of the memory that is needed by ESD-1. However, when h becomes smaller or $\frac{l}{h}$ becomes larger, ESD has to choose a larger sampling probability in order to limit the error in spread estimation caused by sampling. Consequently, it has to store more contacts and thus require more memory. For instance, when $h = 500$ and $l = 0.5h$, ESD with sampling uses 55.6% of the memory that is needed by ESD-1.

Table 3-2 compares the memory requirements when $\alpha = 0.95$ and $\beta = 0.05$. It shows similar results: (1) ESD uses significantly less memory than TFA, and (2) the probabilistic sampling method in ESD is critical for memory saving especially when h is large or $\frac{l}{h}$ is small. The table also demonstrates that the memory requirement of either ESD or TFA increases when the performance objective becomes more stringent, i.e., α is set larger and β smaller.

TFA requires more memory because it stores the source and destination addresses of the contacts. In [128], the authors also indicate that Bloom Filters [7, 8] can be used to reduce the memory consumption. However, the paper does not give detailed design or parameter settings. Therefore, we cannot implement the Bloom-filter version of TFA. The paper claims that the memory requirement will be reduced by a factor of 2.5 when Bloom filters are used. Even when this factor is taken into account in Tables 3-1-3-2, memory saving by ESD will still be significant.

3.5.3 Comparison in Terms of False Positive Ratio and False Negative Ratio

The *false positive ratio* (FPR) is defined as the fraction of all non-scanners (whose spreads are no more than l) that are mistakenly reported as scanners. The *false negative ratio* (FNR) is the fraction of all scanners (whose spreads are no less than h) that are not reported by the system. In the previous subsection, we have shown that, given the bounds of FPR and FNR, it takes ESD much less memory to achieve the bounds than TFA. Since CSE and TBA are not designed for meeting a given set of bounds, we compare our ESD with them by a different set of experiments that measure and compare the FPR and FNR values under a fixed amount of SRAM.

Given a fixed memory size m, we use $OptimalS(m, s)$ in Section 3.4.3 to determine the value of s in ESD, use $OptimalP(m, s)$ to determine the value of p, and then set the threshold T as $\frac{h+l}{2}$. We perform experiments using the week-long traffic trace. For $m = 0.05MB, l = 0.5h$, the results are presented in Tables 3-3. We also perform the same experiments for CSE and TBA, and the results are presented in the table as well. The optimal parameters are chosen for each scheme based on the original papers.

When the available memory is very small, such as $0.05MB$ in Table 3-3, CSE has zero FNR but its FPR is 1.0, which means it reports all non-scanners. The reason is that, without probabilistic sampling, CSE stores information of too many contacts such that its data structure is fully saturated. In this case, the spread estimation method of CSE breaks down. TBA has a small FPR but its FNR is large. For example, when

$h = 500$, its FNR is 26%. Only ESD achieves small values for both FNR and FPR. For example, when $h = 500$, its FNR is 7.4% and its FPR is 5.0%. These values decrease quickly as h increases. When $h = 1,000$, they are 1.0% and 0.55%, respectively, while the FNR of TBA remains to be 26%.

3.6 Summary

Scan detection is one of the most important functions in intrusion detection systems. The recent research trend is to implement such a function in the tight SRAM space to catch up with the rapid advance in network speed. This work proposes an efficient scan detection scheme based on a new method called *dynamic bit sharing*, which optimally combines probabilistic sampling, bit-sharing storage, and maximum likelihood estimation. We demonstrate theoretically and experimentally that the new scheme is able to achieve a probabilistic performance objective with arbitrarily-set bounds on worst-case false positive/negative ratios. It does so in a very tight memory space where the number of bits available is much smaller than the number of external sources to be monitored.

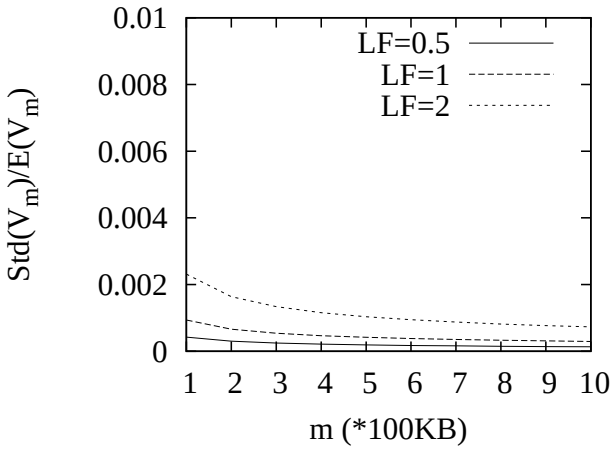

Figure 3-1. The relative standard deviation, $\frac{Std(V_m)}{E(V_m)}$, approaches to zero as m increases. The *load factor* (LF) is defined as $n \cdot p/m$, where $n \cdot p$ is the number of distinct contacts that are sampled by ESD for storage. In our experiments (reported in Section 3.5), when we use the system parameters determined by the algorithm proposed in this section, the load factor never exceeds 2.

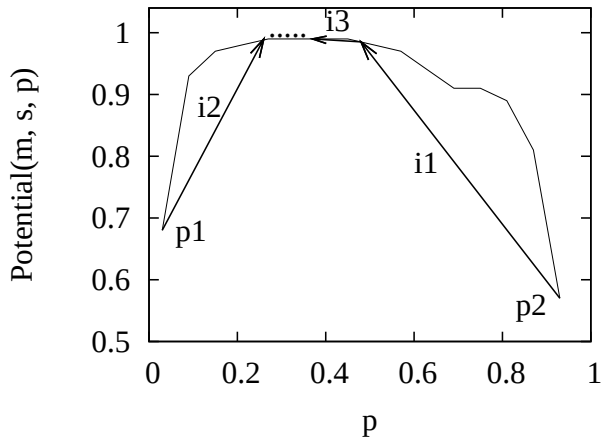

Figure 3-2. (A) The curve (without the arrows) shows the value of $Potential(m, s, p)$ with respect to p when $m = 0.45MB$ and $s = 150$. Its non-smooth appearance is due to $\lfloor C \rfloor$ in the formula of $F_h(m, s, p, T^*)$. $F_h(m, s, p, T^*)$ depends on the values of $\lfloor C \rfloor$ and $q(h)$, which are both functions of p. (B) The arrows illustrate the operation of $OptimalP(m, s)$. In the first iteration (arrow i_1), p_2 is set to be $(p_1 + p_2)/2$. In the second iteration (arrow i_2), p_1 is set to be $(p_1 + p_2)/2$. In the third iteration (arrow i_3), p_2 is set to be $(p_1 + p_2)/2$.

Table 3-1. Memory requirements (in MB) of ESD, TFA and ESD-1 (i.e. ESD with $p = 1$) when $\alpha = 0.9$ and $\beta = 0.1$.

	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
h	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1
500	0.09	2.02	0.33	0.19	2.53	0.43	0.30	3.61	0.54	0.97	6.12	1.01
1000	0.07	1.10	0.27	0.09	1.29	0.33	0.15	1.85	0.42	0.47	3.11	0.86
2000	0.03	0.55	0.24	0.05	0.71	0.29	0.08	1.02	0.42	0.25	1.62	0.86
3000	0.02	0.42	0.24	0.03	0.51	0.27	0.06	0.68	0.42	0.17	1.09	0.86
4000	0.01	0.32	0.21	0.03	0.38	0.27	0.03	0.52	0.42	0.13	0.83	0.86
5000	0.01	0.24	0.21	0.02	0.31	0.27	0.03	0.43	0.42	0.11	0.66	0.86

Table 3-2. Memory requirements (in MB) of ESD, TFA and ESD-1 (i.e. ESD with $p = 1$) when $\alpha = 0.95$ and $\beta = 0.05$.

	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
h	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1
500	0.12	2.41	0.38	0.22	3.27	0.48	0.48	4.59	0.68	1.56	8.03	1.60
1000	0.08	1.29	0.32	0.12	1.65	0.38	0.24	2.34	0.50	0.76	4.04	1.20
2000	0.03	0.69	0.26	0.08	0.87	0.32	0.13	1.21	0.47	0.38	2.12	1.20
3000	0.02	0.46	0.26	0.06	0.60	0.32	0.09	0.83	0.47	0.26	1.42	1.20
4000	0.02	0.37	0.23	0.04	0.45	0.32	0.06	0.63	0.47	0.20	1.08	1.20
5000	0.01	0.29	0.23	0.04	0.35	0.32	0.05	0.52	0.47	0.16	0.89	1.20

Table 3-3. False negative ratio and false positive ratio of ESD, CSE and TBA with $m = 0.05MB$.

	FNR			FPR		
h	ESD	CSE	TBA	ESD	CSE	TBA
500	7.4e-2	0	2.6e-1	5.0e-2	1	9.0e-6
1000	1.0e-2	0	2.6e-1	5.5e-3	1	9.0e-6
2000	4.2e-3	0	2.5e-1	2.0e-3	1	1.1e-5
3000	5.5e-3	0	2.5e-1	2.0e-3	1	1.0e-5
4000	0	0	2.4e-1	2.0e-3	1	7.0e-6
5000	0	0	2.4e-1	2.0e-3	1	7.0e-6

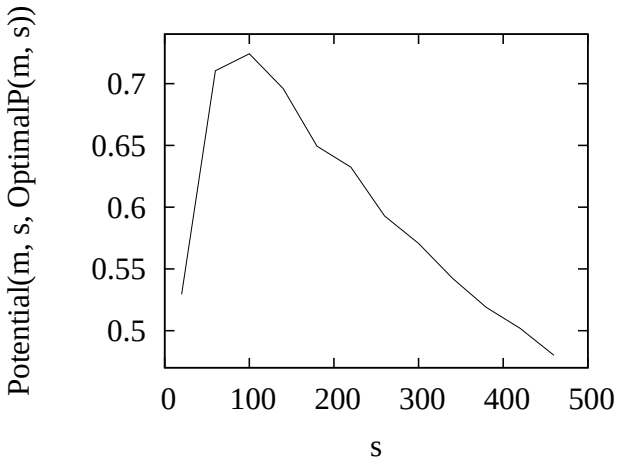

Figure 3-3. The value of $Potential(m, s, OptimalP(m, s))$ with respect to s when $m = 0.25MB$.

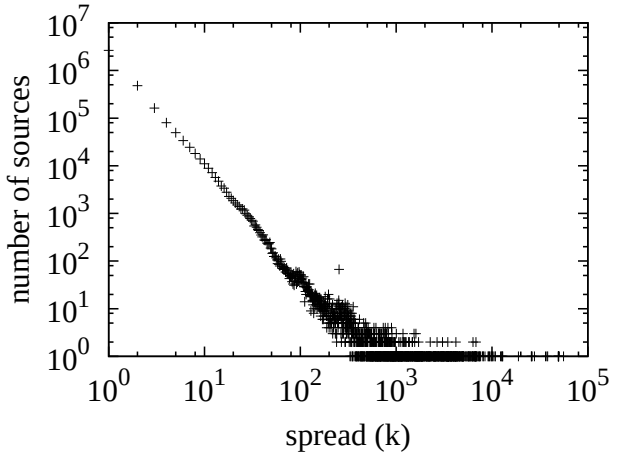

Figure 3-4. Traffic distribution: each point shows the number of sources having a certain spread value.

CHAPTER 4
ORIGIN-DESTINATION FLOW MEASUREMENT IN HIGH-SPEED NETWORKS

This chapter designs an efficient approach for origin-destination flow measurement in high-speed networks, where an origin-destination (OD) flow between two routers is the set of packets that pass both routers. The OD flow measurement has widely usage in many network management applications. We consider two performance challenges, measurement efficiency and accuracy. The former requires measurement functions to minimize per-packet processing overhead to keep up with today's high-speed network. The latter requires measurement functions to achieve accurate measurement results with small bias and standard deviation. We design a novel measurement method that employs a compact data structure for packet information storage and uses a new statistical inference approach for OD flow measurement. Both simulations and experiments are performed to demonstrate the effectiveness of our method.

The rest of this chapter is organized as follows: Section 4.1 gives the problem statement and performance metrics. Section 4.2 describes the related work. Section 4.3 presents our new origin-destination flow measurement method. Section 4.4 discusses the simulation results. Section 4.5 presents the experimental results. Section 4.6 gives the summary.

4.1 Problem Statement and Performance Metrics

4.1.1 Problem Statement

Let S be a subset of routers of interest in a network. The problem is to measure traffic volume between any pair of routers in S. We model an origin-destination (OD) flow as the set of packets traverse between two routers (the undirectional case) or traverse from one router to the other (the directional case). Our goal is to measure the size of each OD flow in terms of number of packets.

Consider the set of access routers on the perimeter of an ISP network. If each access router stores information about ingress packets (that enter the ISP network)

and egress packets (that leave the ISP network) in separate data structures, we can figure out the size of an directional OD flow by comparing the information in the ingress data structure of the origin router and the information in the egress data structure of the destination router. On the other hand, if each access router stores information of all arrival packets in the same data structure, we can figure out the size of an undirectional OD flow by comparing the information in the data structures of both routers. The measurement method proposed in this work can be applied to both cases even though our description uses the undirectional case for simplicity.

We consider two performance metrics, per-packet processing overhead and measurement accuracy, which are discussed below.

4.1.2 Per-packet Processing Overhead

The maximum packet throughput that an online measurement function can achieve is determined by the per-packet processing overhead of the function. In order to keep up with today's high-speed network, it is desirable to make the per-packet processing overhead as small as possible, especially when the SRAM and processing circuits are shared by other critical functions for routing, packet scheduling, traffic management and security purposes.

The per-packet processing overhead is mainly determined by the computational complexity and the number of memory accesses for each packet. When a router receives a packet, it needs to perform certain computations to determine the proper location for the information storage of that packet and at least one memory access for the storage operation. We will show that our OD flow measurement function is able to achieve extremely small per-packet processing overhead.

4.1.3 Measurement Accuracy

Let n_c be the OD flow size of an origin/destination router pair and $\hat{n}_c$ be the corresponding measured result. The event for n_c to fall into the interval $[\hat{n}_c \cdot (1 - \beta), \hat{n}_c \cdot (1 + \beta)]$ with probability at least α specifies the measurement accuracy of our

function, where α is a pre-determined accuracy parameter, e.g., 95%. A smaller value of β means better measurement results.

If the memory requirement and the processing speed for each packet are unlimited, we can achieve 100% measurement results. Otherwise, we have to compromise the measurement accuracy if the memory resource is not enough or the processing speed requirement is relatively stringent.

4.2 Related Work

The origin-destination (OD) flow measurement methods mainly fall into two categories. One is *intermediate-based* [74, 82, 88, 105, 119–121] and the other is *end-to-end-based* [11, 37, 42]. The intermediate based methods [119–121] employ statistical techniques to indirectly estimate the OD flows based on link load, network routing, and configuration data, which are widely available information. Zhang et al. [119] assume an underlying gravity model [82, 100] for OD flows and use edge link load data together with additional information on intermediate routers to analyze the model. After that, they introduce the tomographic method [12, 26] to determine the results that most fit with the obtained gravity model. The methods in [120, 121] extend the point-to-point measurement to point-to-multipoint measurement using a regularization based on entropy penalization. These intermediate-based methods share a common property that jeopardizes them from being widely applied: the estimation relies on traffic volumes, which are usually unknown information. As a result, these methods either cannot achieve high measurement accuracy or incurs severe computational cost.

Considine et al. [28] use the method of moments for OD packet counts, which extracts a traffic digest from the packet stream. As the study in [11] points out, when the noise-to-signal ratios are high, the performance of [28] will be degraded.

Cao, Chen and Bu [11] design a quasi-likelihood approach (QMLE) for OD flow measurement based on a continuous variant of the Flajolet-Martin sketches [43]. The approach maintains an array of buckets, whose initial values are all set to infinity at

the beginning of a measurement period, in each network node. When it receives a packet, the node performs two hash operations. The first one pseudo-randomly chooses a bucket i in the array for packet information storage. The second one generates an exponential random number v based on the packet, whose expected value is one. After the two hash operations, the node updates the bucket i by v. If the original value of i is larger than v, the node will set the value of bucket i to v. Otherwise, it will skip this packet. At the end of the measurement period, in order to estimate the OD flow size of two routers r_1 and r_2, QMLE derives the quasi-probability distribution of the packet information and employs the maximum likelihood estimation to compute the OD flow size based on the values of the two bucket arrays.

QMLE claims that it is able to achieve small per-packet update overhead and accurate measurement result with a compact memory requirement. However, for each packet, it needs to perform two hash operations and more than one memory access on average (It always needs one memory read and sometimes one memory write), while the optimal should be exactly one hash operation and one memory access per-packet. Moreover, it also has space to improve in terms of measurement accuracy. With the assignment of the same amount of memory resource, the proposed method in this study is able to achieve much more accurate measurement result, which will be demonstrated by simulations and experiments in Section 4.4 and 4.5, respectively.

Also related is to recover the missing values during traffic measurement by the technique of compressive sensing in [122], which proposes a spatio-temporal framework to exploit the presence of both global structure and local structure. Rincon et al. [98] provide a multi-resolution analysis to develop a general model for traffic matrices, which is based on the diffusion wavelet transform. They find that the model must be sparse and also demonstrate it by experimental results.

4.3 Origin-Destination Flow Measurement

We first describe two straightforward approaches and discuss their limitations. We then motivate the bitmap idea that we use in this study. Finally, we present our origin-destination flow measurement method (ODFM) in details.

4.3.1 Straightforward Approaches and Their Limitations

A straightforward approach is for each router to store the information of all packets that pass it. In this way, when we want to measure the OD flow size of two routers, we only need to compare the two sets of packet information and count how many packets the two sets have in common, i.e., the cardinality of the intersection of the two sets. Clearly, storing information of all packets is unrealistic since the number of packets passing a router is huge in high-speed networks and it imposes an extremely large memory requirement on the router.

In order to reduce the memory requirement, we can store the signatures of packets instead. The signature of a packet is a hash value of the packet with a fixed length. When the length of the signature is long enough, e.g., 160 bits if using SHA-1 [89], the chance of two packets having the same signatures is negligibly small. Therefore, we can count the number of identical signatures that stored in the two routers to obtain the OD flow size. This enhancement can reduce the memory requirement to some extent. However, it is still not memory efficient. Suppose there are $1M$ packets that pass a router during a measurement period. When the length of the signature is 160 bits long, a router needs $20MB$ ($1M \times 160/8$) memory to store the information of all signatures, which is still too much in practise. Using smaller signatures cannot solve the problem. For example, if we reduce the signature length to just 16 bits, the memory requirement is still $2MB$, far higher than the goal of this study, less than 1 bit per packet.

Another solution is for a router to maintain a counter, whose initial value is set to zero, for each of other routers in the network. In order to notify the current router which routers a packet has passed, it needs to carry the information of all previous routers

in its header. When a router receives a packet, it first checks the packet header and obtains the information of which routers this packet has passed. It then increases the corresponding counters by one. Finally, the router adds its own information into the packet before sending it out. At the end of the measurement period, in order to obtain the OD flow size of router r_1 and router r_2, we first check r_1 and find the counter that corresponds to r_2, which stores the number of packets that enter r_2 and exit from r_1. We then check r_2 and find the counter that corresponds to r_1, which stores the number of packets that enter r_1 and exit from r_2. The summation of the two counters is the OD flow size of r_1 and r_2. Although the computation for the OD flow size is very simple at the end of the measurement period, this approach has two main drawbacks during the packet processing period. First, the router needs to extract the information of all routers that stored in a packet and updates the corresponding counters, which slows down the packet processing speed and cannot keep up with the line speed in today's high-speed networks. Second, each packet has to carry the information of all routers it has passed, which requires the modification of the packet structure and incurs unnecessary storage overhead to the packet.

4.3.2 ODFM: Motivation and Overview

We design a bitmap based OD flow measurement method that is able to solve the problems that the above two approaches have. Instead of storing the signatures of packets, each router maintains a bit array with a fixed length and initially all bits in the array are set to zero. When the router receives a packet, it pseudo-randomly maps the packet to one bit of the array by a hash operation and sets the bit to one. At the end of the measurement period, we measure the OD flow size of two routers by comparing their bit arrays. Since a packet always uses the same hash function to choose a bit in the arrays for all routers and the size of each bit array is fixed within a measurement period, it will map to the same location in the bit arrays of any routers it has passed. Therefore, if a packet enters router r_1 and exits from r_2 or the other way

around, its corresponding bit in these two bit arrays must be both set to one. Based on this observation, we can take a bitwise AND operation of the two bit arrays and count the number of ones in the combined bit array to measure the OD flow.

Note that this approach may introduce the overestimation problem, which could lead to an inaccurate measurement result. Suppose two packets, called p_1 and p_2, map to a same location j by the hash function. While p_1 passes one router and p_2 passes the other. In this case, the jth bit of both bit arrays of the two routers will be set to one. When we compare the two bit arrays, we will falsely treat p_1 and p_2 as a same packet and overestimate the OD flow size. However, there is a nice property of our scheme: Because the bit for each packet is randomly picked in the bit array, the event for any two packets to choose the same bit in the array has an equal probability to happen. When the number of packets and the size of the bit array are large enough, this event occurs in the bit array uniformly at random and the overestimation problem can be removed through statistical analysis. This property enables us to design a compact yet accurate measurement method. Moreover, in our scheme, a router only needs to perform one hash operation and one memory accesses per packet, which is very efficient and feasible for high-speed networks.

4.3.3 ODFM: Storing the Packet Information

ODFM consists of two components: one for storing the packet information into routers, the other for measuring the OD flow of any two routers. This subsection presents the first component and the second one will be described in the next subsection.

At the beginning of the measurement period, each router maintains a bit array B with a fixed length m. Initially each bit in B is set to zero. The ith bit in the array is denoted as $B[i]$. When a router receives a packet p, it pseudo-randomly picks one bit in B by performing a hash operation $H(p)$ and set the bit to one, where $H(..)$ is a hash function whose output range is $[0..m-1]$. More specifically, to store the packet p, ODFM

performs the following assignment:

$$B[H(p)] := 1. \tag{4–1}$$

Actually a router does not have to perform the hash operation on all the content of a packet. In the network layer, a packet can be uniquely identified by its IP header, which stores the packet label information, i.e., source IP address and destination IP address and so on. For two packets that are fragments of some original, larger packet, although they share the same source/destination IP addresses and identification number, their fragmentation offset values are different. Therefore, a router only needs to perform the hash operation ($H(p)$) on the IP header of a packet, which can further reduce the hash computational complexity and improve the processing speed. This enhancement can be also applied to the two straightforward approaches in Section 4.3.1.

It is worth noting that a router only needs to perform one hash operation and sets one bit in its bit array per packet, which is very simple, efficient, and can be easily implemented in high-speed routers.

4.3.4 ODFM: Measuring the Size of Each OD Flow

At the end of the measurement period, all routers will report its bit array to a centralized server, e.g., the network management center, which performs the offline measurement. ODFM employs the maximum likelihood estimation (MLE) [15] to measure the OD flow of any two routers based on their bit arrays. Let S_1 and S_2 be the set of packets that pass the two routers r_1 and r_2. Let n_1 and n_2 be the cardinalities of S_1 and S_2, respectively, i.e., $n_1 = |S_1|$, $n_2 = |S_2|$, n_c be the number of common packets that r_1 and r_2 share, i.e., the OD flow size of the two routers, which is the value that we want to measure in this study. Figure 4-1 illustrates the relationship of n_1, n_2 and n_c. Obviously, we have $n_c = |S_1 \cap S_2|$. Let B_1 and B_2 be the two bit arrays of r_1 and r_2, U_1 and U_2 be the number of '0's in B_1 and B_2, respectively, V_1 and V_2 be the percentage of bits in B_1 and B_2 whose values are zero. Clearly, $V_1 = \frac{U_1}{m}$ and $V_2 = \frac{U_2}{m}$.

The measurement consists of two steps. In the first step, we compute the cardinality of S_1 (i.e., n_1) and the cardinality of S_2 (i.e., n_2) based on B_1 and B_2, respectively. In the second step, we take a bitwise AND operation of B_1 and B_2 to generate a new bit array, denoted as B_c, to compute the OD flow size n_c. Let U_c be the number of '0's in B_c, V_c be the percentage of bits in B_c whose values are zero. Clearly, $V_c = \frac{U_c}{m}$. We compute n_c based on B_c and the results obtained in previous step, i.e., the values of n_1 and n_2.

4.3.4.1 Measure n_1 and n_2

The number of packets that a router receives during a measurement period can be easily obtained by adding a counter whose initial value is set to zero. When it comes a new packet, the router simply increases the counter by one. In this way, we can obtain the exact values of n_1 and n_2, which we will use to measure n_c in the following subsection.

4.3.4.2 Measure n_c

After n_1 and n_2 are obtained, we take a bitwise AND operation of B_1 and B_2, denoted as B_c, to measure n_c. More specifically, we have

$$B_c[i] = B_1[i] \ \& \ B_2[i], \quad \forall i \in [0..m-1]. \tag{4–2}$$

For an arbitrary bit b in B_c, it is '0' if and only if the following two conditions are both satisfied. First, it is not chosen by any packet in $S_1 \cap S_2$. If b is chosen by a packet $p \in S_1 \cap S_2$, we know the corresponding bits in both B_1 and B_2 will be set to '1'. Therefore, b will be '1'. Second, it is either not chosen by any packet in $S_1 - S_2$ or not chosen by any packet $S_2 - S_1$. If it is chosen by both a packet $p_1 \in S_1 - S_2$ and a packet $p_2 \in S_2 - S_1$, the corresponding bits in both B_1 and B_2 will be also set to '1'. As a result, b will be '1'. For the first condition, a packet in $S_1 \cap S_2$ has probability $\frac{1}{m}$ to set b to '1', which means the probability for b not to be set by this packet is $1 - \frac{1}{m}$. As Figure 4-1 shows, $n_c = |S_1 \cap S_2|$. Therefore, the probability for b not to be set to '1' by any packet in $S_1 \cap S_2$ is $(1 - \frac{1}{m})^{n_c}$. Similarly, the probability for it not to be chosen by any packet in

$S_1 - S_2$ is $(1 - \frac{1}{m})^{n_1 - n_c}$ and the probability for it not to be chosen by any packet in $S_2 - S_1$ is $(1 - \frac{1}{m})^{n_2 - n_c}$. As a result, the probability $q(n_c)$ for b to remain '0' in B_c is

$$\begin{aligned} q(n_c) &= (1 - \frac{1}{m})^{n_c} \{1 - (1 - (1 - \frac{1}{m})^{n_1 - n_c}) \\ &\qquad \times (1 - (1 - \frac{1}{m})^{n_2 - n_c})\} \\ &= (1 - \frac{1}{m})^{n_1} + (1 - \frac{1}{m})^{n_2} - (1 - \frac{1}{m})^{n_1 + n_2 - n_c} \end{aligned} \tag{4–3}$$

Each bit in B_c has a probability $q(n_c)$ to be '0'. The observed number of '0' bits in B_c is U_c. Therefore, the likelihood function for this observation to occur is given as follows:

$$L = q(n_c)^{U_c} \times (1 - q(n_c))^{m - U_c} \tag{4–4}$$

Following the standard process of maximum likelihood estimation, we find an optimal value of n_c that can maximize the above likelihood function. Namely, we want to find

$$\hat{n}_c = \arg\max_{n_c} \{L\} \tag{4–5}$$

To find $\hat{n}_c$, we take a logarithm operation to both sides of (4–4).

$$\ln L = U_c \times \ln q(n_c) + (m - U_c) \times \ln(1 - q(n_c)) \tag{4–6}$$

We then differentiate the above equation:

$$\begin{aligned} \frac{d \ln L}{d n_c} &= (\frac{U_c}{q(n_c)} - \frac{m - U_c}{1 - q(n_c)}) \times q'(n_c) \\ &= (\frac{U_c}{q(n_c)} - \frac{m - U_c}{1 - q(n_c)}) \times \ln(1 - \frac{1}{m}) \\ &\qquad \times (1 - \frac{1}{m})^{n_1 + n_2 - n_c}, \end{aligned} \tag{4–7}$$

since according to (4–3), we have

$$\begin{aligned} q'(n_c) &= \frac{d q(n_c)}{d n_c} \\ &= \ln(1 - \frac{1}{m}) \times (1 - \frac{1}{m})^{n_1 + n_2 - n_c}. \end{aligned} \tag{4–8}$$

In order to compute $\hat{n}_c$, we set the right side of (4–7) to zero, i.e.

$$(\frac{U_c}{q(n_c)} - \frac{m - U_c}{1 - q(n_c)}) \times \ln(1 - \frac{1}{m}) \times (1 - \frac{1}{m})^{n_1 + n_2 - n_c} = 0 \tag{4–9}$$

Since neither of $\ln(1 - \frac{1}{m})$ and $(1 - \frac{1}{m})^{n_1+n_2-n_c}$ could be 0 when m is positive, we have

$$\frac{U_c}{q(n_c)} - \frac{m - U_c}{1 - q(n_c)} = 0. \tag{4–10}$$

Applying (4–3) to (4–10), we have

$$\begin{aligned}(1 - \frac{1}{m})^{n_1} + (1 - \frac{1}{m})^{n_2} - (1 - \frac{1}{m})^{n_1+n_2-n_c} &= \frac{U_c}{m} \\ &= V_c.\end{aligned} \tag{4–11}$$

In above equation, m, n_1, and n_2 are all known values, and V_c can also be computed when the packets information are recorded. As a result, we can measure n_c in the following formula:

$$n_c = n_1 + n_2 - \frac{\ln((1 - \frac{1}{m})^{n_1} + (1 - \frac{1}{m})^{n_2} - V_c)}{\ln(1 - \frac{1}{m})} \tag{4–12}$$

4.3.5 Measurement Accuracy

The previous subsection gives the measurement formula for n_c by MLE. We analyze the measurement accuracy of our method in this subsection. According to the standard theory of MLE [86], when the values of m, n_1, and n_2 are large enough, the measured OD flow size $\hat{n}_c$ approximately follows a normal distribution:

$$\hat{n}_c \sim Norm\left(n_c, \frac{1}{\mathcal{I}(\hat{n}_c)}\right), \tag{4–13}$$

where $\mathcal{I}(\hat{n}_c)$ is the fisher information [1] of L, which is defined as follows

$$\mathcal{I}(\hat{n}_c) = -E\left[\frac{d^2 \ln L}{dn_c^2}\right]. \tag{4–14}$$

According to (4–7), we compute the second-order derivative of $\ln L$

$$\begin{aligned}\frac{d^2 \ln L}{dn_c^2} = \ln(1-\frac{1}{m}) \times \Bigg[\Bigg(-\frac{U_c \cdot q'(n_c)}{q^2(n_c)} - \frac{(m-U_c)\cdot q'(n_c)}{(1-q(n_c))^2}\Bigg) \\ \times C - \left(\frac{U_c}{q(n_c)} - \frac{m-U_c}{1-q(n_c)}\right) \times C\Bigg],\end{aligned} \tag{4–15}$$

where $C = (1-\frac{1}{m})^{n_1+n_2-n_c}$ and $q'(n_c)$ is given in (4–8).

We use the probabilistic counting method [54] to compute the expected value of U_c. Let X_i be the event that the ith bit in B_c remains '0' at the end of the measurement period and 1_{X_i} be the corresponding indicator random variable. As the size of B_c is m, for an arbitrary bit b, it has probability $q(n_c)$ to remain '0'. U_c is the number of '0's in B_c, $U_c = \sum_{i=0}^{m-1} 1_{X_i}$. Hence,

$$E(U_c) = \sum_{i=0}^{m-1} E(1_{X_i}) = \sum_{i=0}^{m-1} q(n_c) = m \cdot q(n_c) \tag{4–16}$$

Therefore, we have

$$\begin{aligned}\mathcal{I}(\hat{n}_c) &= -E\left[\frac{d^2 \ln L}{dn_c^2}\right] \\ &= \ln(1-\frac{1}{m}) \times \left(\frac{m\cdot q'(n_c)}{q(n_c)} + \frac{m\cdot q'(n_c)}{1-q(n_c)}\right) \times C,\end{aligned} \tag{4–17}$$

as the expected value of $(\frac{U_c}{q(n_c)} - \frac{m-U_c}{1-q(n_c)})$ is 0.

[1] The fisher information [67] is a way of measuring the amount of information that an observable random variable x carries about an unknown parameter θ upon which the likelihood function of θ, $L(\theta) = f(x;\theta)$, depends.

According to (4–13), the variance of $\hat{n}_c$ is

$$Var(\hat{n}_c) = \frac{1}{\mathcal{I}(\hat{n}_c)} = \frac{1}{\ln(1-\frac{1}{m}) \times \left(\frac{m \cdot q'(n_c)}{q(n_c)} + \frac{m \cdot q'(n_c)}{1-q(n_c)}\right) \times C}. \quad (4–18)$$

and the confidence interval of our measurement is

$$\hat{n}_c \pm \frac{Z_\alpha}{\sqrt{\ln(1-\frac{1}{m}) \times \left(\frac{m \cdot q'(n_c)}{q(n_c)} + \frac{m \cdot q'(n_c)}{1-q(n_c)}\right) \times C}}, \quad (4–19)$$

where α is the confidence level parameter and Z_α is the α percentile for the standard Gaussian distribution [9]. For example, when $\alpha = 99\%$, $Z_\alpha = 2.58$.

4.4 Simulations

We first evaluate the performance of our method ODFM by simulations in this section. We will present experimental results based on real traffic trace in the next section. In both simulations and experiments, we compare ODFM with the most related work, QMLE [11]. For fair comparison, we assign the same amount of memory to ODFM and QMLE. We compare them in terms of online processing overhead and offline measurement accuracy.

Simulations are performed under system parameters, n_1, n_2, and n_c. For an origin-destination router pair, n_1 is the number of packets that one router receives during the measurement period, and n_2 is the number of packets that the othe router receives. Parameter n_c is the actual OD flow size. The amount of memory used is set to be 1MB.

In the first set of simulations, we let $n_1 = 6,000,000$, $n_2 = 6,000,000,300,000$, or $100,000$. We vary n_c from 100 to 50,000. We use ODFM and QMLE to measure the flow size, and compare it with n_c to see how accurate the measurement is.

In the second set of simulations, we model a more realistic scenario, where n_1, n_2 and n_c are randomly chosen. The values of n_1 and n_2 are randomly selected from the range of $[100,000, 10,000,000]$, and the value of n_c is randomly selected from $[100, 50,000]$ in each simulation run.

4.4.1 Processing Overhead

Per-packet processing overhead of a measurement method is mainly determined by the number of memory accesses and the number of hash operations for each packet. Table 4-1 shows the averaged results when $n_1 = 6,000,000$, $n_2 = 6,000,000$, and n_c varies from 100 to $50,000$. ODFM requires only 1 hash operation and 1 memory access (memory write) for each packet, which is the optimal. QMLE requires more per-packet processing overhead. It incurs 1.50 memory accesses and 2 hash operations on average. Furthermore, per-packet processing overhead of ODFM is constant, while QMLE requires variable per-packet processing overhead, which is undesirable in practice. Table 4-2 and Table 4-3 present similar results with $n_2 = 300,000$ and $100,000$ respectively. Table 4-4 shows the results when the values of n_1 and n_2 are randomly chosen in the range $[100,000, 10,000,000]$ and the value of n_c is randomly chosen in the range of $[100, 50,000]$.

4.4.2 Measurement Accuracy

Figures 4-2-4-4 present the measurement results of ODFM and QMLE. Each figure consists of four plots. Each point in the first plot (ODFM) or the second plot (QMLE) represents an OD flow. The x-axis is the actual flow size n_c, and the y-axis is the estimated value $\hat{n}_c$. We also show the equality line, $y = x$, for reference. Clearly, the closer a point is to the equality line, the better the estimation result is. The third plot shows the corresponding measured bias of the first two plots, which is $E(\hat{n}_c - n_c)$. The fourth plot shows the corresponding standard deviation of the first two plots, which is $\frac{\sqrt{Var(\hat{n}_c)}}{n_c}$. In order to clearly present the estimation results of the two methods, we divide the horizontal coordinate into 25 measurement bins of width 2,000, and numerically

measure the bias and standard deviation in each bin. The three figures present the following results.

As shown in the first plot of Figure 4-2, when the values of n_1 and n_2 are the same, ODFM has a small bias in its measurement, which is understandable because it is well known that the maximum likelihood estimation may produce small bias under certain parameter settings. The second plot shows that QMLE performs better and produces almost perfect results. However, this is only part of the story. When the values of n_1 and n_2 are different, as shown in Figure 4-3 where $n_1 = 6,000,000$ and $n_2 = 300,000$, ODFM performs nearly perfectly, while QMLE produces large bias. As the difference between n_1 and n_2 widens, the bias of QMLE becomes larger, whereas the performance of ODFM is actually improved, which is shown in Figure 4-4 where $n_1 = 6,000,000$ and $n_2 = 100,000$. Now the question is which case is closer to the reality, n_1 and n_2 having close values or diverse values? It is the latter, as we will show in the next section.

Figure 4-5 compares the performance of ODFM and QDFM when n_1 and n_2 are randomly picked in the range $[100,000, 10,000,000]$. Clearly, ODFM outperforms QMLE by a wide margin. The reason is that randomly-selected values of n_1 and n_2 tend to be very different than being close to each other.

4.5 Experiments

We further evaluate the performance of ODFM and QMLE by experiments in this section. The experimental dataset that we use is obtained from Abilene network (Internet2) [3], which is collected and shared by Yin Zhang [124]. The network consists of 12 routers that are located at different cities in US [1]. The dataset contains 24 weeks of Abilene traffic matrices from March 1st to September 10th, 2004. The resolution of the dataset is 5 minutes, which means there are $24 \times 7 \times 24 \times 12 = 48,384$ 5-min traffic matrices. In each 5-min traffic matrices, the traffic flows of the routers range from 0.5 Gigabytes to 20 Gigabytes. We set the duration of a measurement period to 5 minutes

and assume that the packet size is 1,500 bytes, which means the routers receive about $0.3M$ to $13M$ packets in one measurement period.

We allocate $1MB$ memory resource to each router and implement the two measurement methods based on the 24 weeks' traffic matrices. The experimental results are similar for those weeks. In this section, we only present the results for the first week.

4.5.1 Number of Packets for an Origin-Destination Pair

Before measuring the size of each OD flow, we first study the number of packets that the origin router and the destination router receive, which are denoted as n_1 and n_2 respectively. We randomly pick 100 OD pair in the traffic matrices and present the values of n_1 and n_2 in Figure 4-6. The x-axis is the index of the OD pair. Each index corresponds to an Origin-Destination pair, (n_1, n_2). The figure shows that the values of n_1 and n_2 are very different from each other in most cases. For example, for the tenth OD pair, $n_1 = 1,111,022$ and $n_2 = 17,795,961$. The ratio between n_1 and n_2 is about 0.06. As the previous section shows, ODFM is not able to work well in this situation. We will demonstrate it shortly.

4.5.2 Processing Overhead

Table 4-1 shows the averaged results of the per-packet processing overhead in terms of the number of memory accesses and the number of hash operations for each packet. Like previous section, ODFM requires only 1 hash operation and 1 memory access (memory write) for each packet. QMLE requires more per-packet processing overhead than ODFM. It incurs 1.17 memory accesses for each packet and 2 hash operations on average. Furthermore, ODFM requires constant per-packet processing overhead. While QMLE requires unpredicted per-packet processing overhead in terms of memory accesses.

4.5.3 Measurement Accuracy

Similar to Figure 4-2-4-5, Figure 4-7 has four plots. The first plot presents the estimation results of ODFM. The second plots presents the estimation results of QMLE. The third plot shows the corresponding estimation bias and the last plot shows the standard deviation. Clearly ODFM works far better than QMLE, which matches the simulation results in Figure 4-5. The reason is that in the traffic matrices, the origin router and the destination router are likely to receive different number of packets. And the performance of QMLE will degrade in that situation, while ODFM does not have this kind of problem.

4.6 Summary

This chapter proposes a new method for OD flow measurement which employs the bitmap data structure for packet information storage and uses statistical inference approach to compute the measurement results. Our method not only requires smaller per-packet processing overhead but also achieves much more accurate results, when comparing with the best existing approach. We implement both simulations and experiments to demonstrate the superior performance of our method.

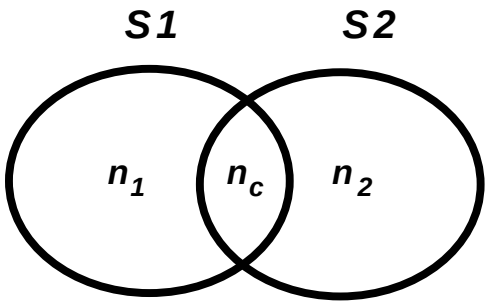

Figure 4-1. The relation between two routers r_1 and r_2

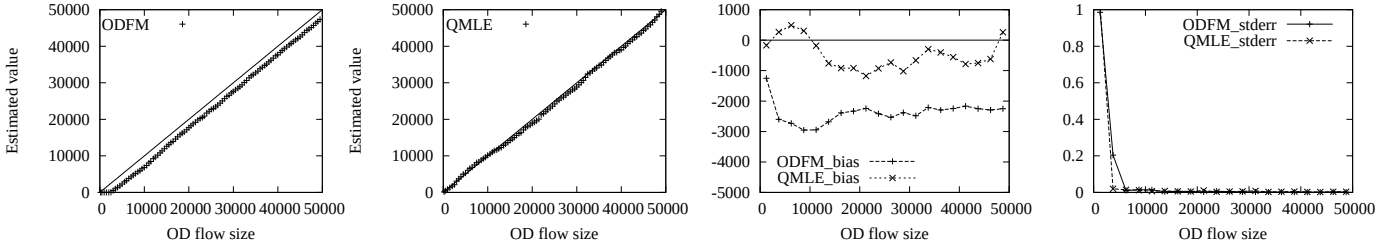

Figure 4-2. • *First Plot*: estimation results by ODFM when $n_1 = 6,000,000$ and $n_2 = 6,000,000$. • *Second Plot*: estimation results by QMLE when $n_1 = 6,000,000$ and $n_2 = 6,000,000$. • *Third Plot*: bias of ODFM and QMLE, which is the measured $E(\hat{n}_c - n_c)$ with respect to n_c. • *Fourth Plot*: standard deviation of ODFM and QMLE, which is the measured $\frac{\sqrt{Var(\hat{n}_c)}}{n_c}$.

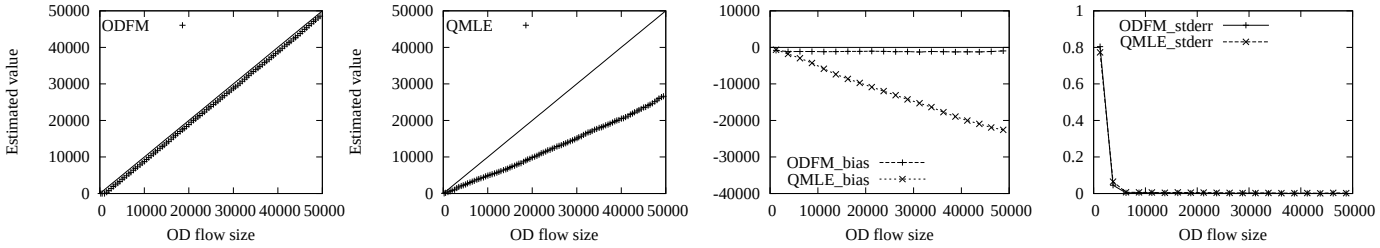

Figure 4-3. • *First Plot*: estimation results by ODFM when $n_1 = 6,000,000$ and $n_2 = 300,000$. • *Second Plot*: estimation results by QMLE when $n_1 = 6,000,000$ and $n_2 = 300,000$. • *Third Plot*: bias of ODFM and QMLE, which is the measured $E(\hat{n}_c - n_c)$ with respect to n_c. • *Fourth Plot*: standard deviation of ODFM and QMLE, which is the measured $\frac{\sqrt{Var(\hat{n}_c)}}{n_c}$.

Table 4-1. Number of memory accesses and number of hash operations per packet with $n_1 = 6,000,000$ and $n_2 = 6,000,000$

	memory accesses	hash operations	constant?
ODFM	1	1	Yes
QMLE	1.50	2	No

Table 4-2. Number of memory accesses and number of hash operations per packet with $n_1 = 6,000,000$ and $n_2 = 300,000$

	memory accesses	hash operations	constant?
ODFM	1	1	Yes
QMLE	1.56	2	No

Table 4-3. Number of memory accesses and number of hash operations per packet with $n_1 = 6,000,000$ and $n_2 = 100,000$

	memory accesses	hash operations	constant?
ODFM	1	1	Yes
QMLE	1.54	2	No

Table 4-4. Number of memory accesses and number of hash operations per packet with the values of n_1 and n_2 are randomly assigned between 100,000 and 10,000,000

	memory accesses	hash operations	constant?
ODFM	1	1	Yes
QMLE	1.22	2	No

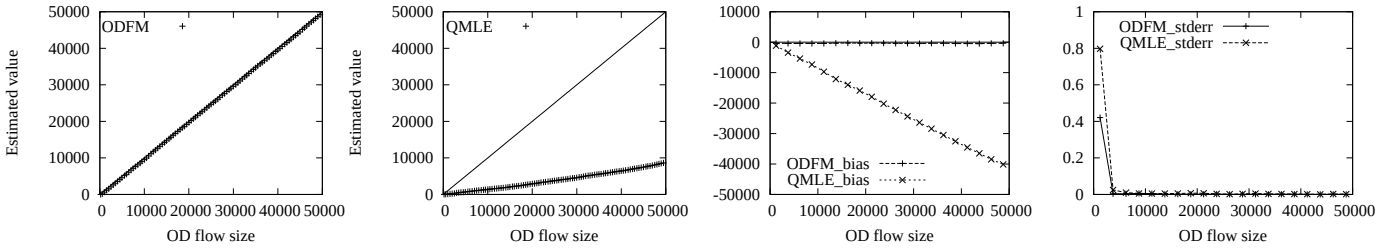

Figure 4-4. • *First Plot*: estimation results by ODFM when $n_1 = 6,000,000$ and $n_2 = 100,000$. • *Second Plot*: estimation results by QMLE when $n_1 = 6,000,000$ and $n_2 = 100,000$. • *Third Plot*: bias of ODFM and QMLE, which is the measured $E(\hat{n}_c - n_c)$ with respect to n_c. • *Fourth Plot*: standard deviation of ODFM and QMLE, which is the measured $\frac{\sqrt{Var(\hat{n}_c)}}{n_c}$.

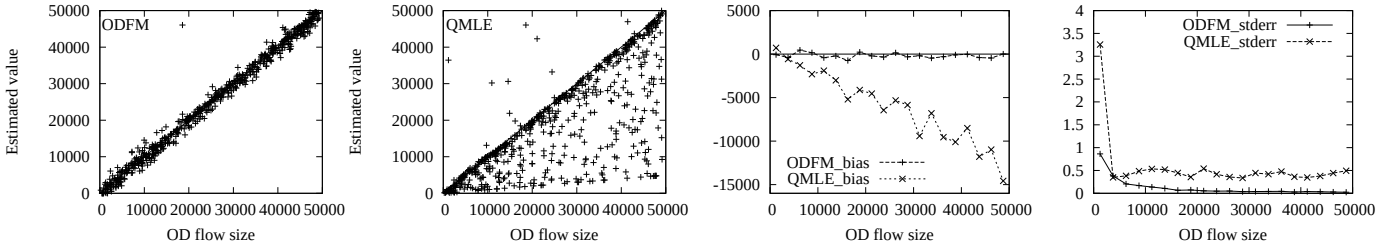

Figure 4-5. • *First Plot*: estimation results by ODFM when the values of n_1 and n_2 are randomly assigned between 100,000 and 10,000,000. • *Second Plot*: estimation results by QMLE when the values of n_1 and n_2 are randomly assigned between 100,000 and 10,000,000. • *Third Plot*: bias of ODFM and QMLE, which is the measured $E(\hat{n}_c - n_c)$ with respect to n_c. • *Fourth Plot*: standard deviation of ODFM and QMLE, which is the measured $\frac{\sqrt{Var(\hat{n}_c)}}{n_c}$.

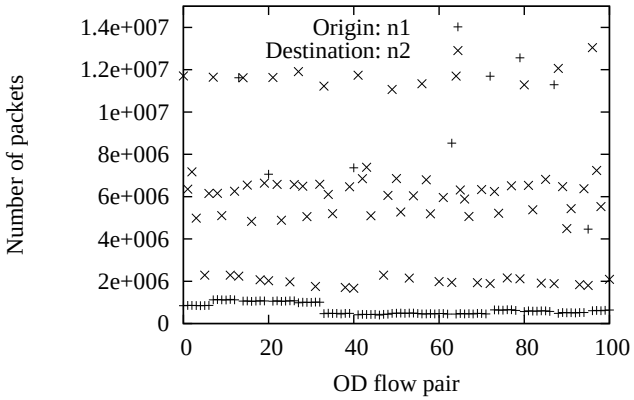

Figure 4-6. The number of packets for 100 OD pair.

Table 4-5. Number of memory accesses and number of hash operations per packet

	memory accesses	hash operations	constant?
ODFM	1	1	Yes
QMLE	1.17	2	No

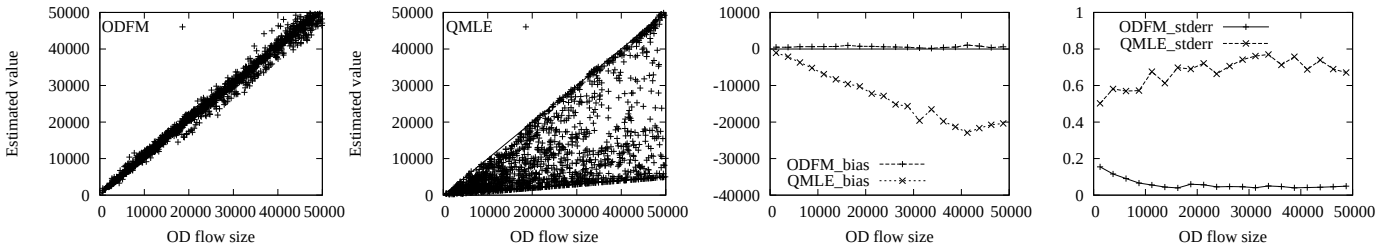

Figure 4-7. • *First Plot*: estimation results by ODFM when $n_1 = 1,000,000$ and $n_2 = 1,000,000$. • *Second Plot*: estimation results by QMLE when $n_1 = 1,000,000$ and $n_2 = 1,000,000$. • *Third Plot*: bias of ODFM and QMLE, which is the measured $E(\hat{n}_c - n_c)$ with respect to n_c. • *Fourth Plot*: standard deviation of ODFM and QMLE, which is the measured. $\frac{\sqrt{Var(\hat{n}_c)}}{n_c}$.

CHAPTER 5
SIZE ESTIMATION PROBLEM IN RFID SYSTEMS

RFID (radio-frequency identification) tags are becoming ubiquitously available in warehouse management, object tracking and inventory control. Researchers have been actively studying RFID systems as an emerging pervasive computing platform [73, 78, 79, 87, 116, 118], which helps create a multi-billion dollar market [31]. This chapter focuses on periodically and automatically estimating the number of RFID tags in a large deployment area. In a large RFID systems, active tags are likely to use due to their longer transmission distance. However, these battery-powered tags need to be recharged when they run out of energy. Recharging tens of thousands of tags is a laborious operation. Moreover, sometimes tagged products may be stacked up, which makes tags not easily accessible. To prolong the lifetime of tags and reduce the frequency of battery recharge, all functions that involve large-scale transmission by many tags should be energy-efficient. To the best of our knowledge, this work is the first to design energy-efficient protocols for the estimation problems in large-scale RFID systems that use active tags.

The rest of the study is organized as follows: Section 5.1 discusses the related work. Section 5.2 defines the problem to be solved and the system model. Sections 5.3 and 5.4 propose two energy-efficient algorithms for the RFID estimation problem. Section 5.5 evaluates the algorithms through simulations. Section 5.6 gives the summary.

5.1 Related Work

Most existing work focuses on how to efficiently read the tag IDs. Collision occurs when multiple tags transmit their IDs in the same time slot. Collision arbitration protocols mainly fall into two categories: the framed ALOHA-based protocols [16, 61, 94, 111, 117] and the tree-based protocols [25, 53, 72, 83, 85, 90]. In the former category, each polling request carries a frame length, and every tag individually chooses a slot in the

frame to transmit its ID. The process repeats until all tags successfully transmit their IDs to the RFID reader. In the latter category, a reader first sends out an ID prefix string. The tags whose ID matches the string will respond. If a collision happens, the reader will append a '0' or '1' to the prefix string and send out the new string. This process repeats until only one tag responds. Essentially the approach traverses a binary tree with the tag IDs being the leaf nodes.

Instead of identifying individual RFID tags, Floerkemeier [44, 45] studies the problem of estimating the cardinality of a tag set based on the number of empty slots. The proposed scheme employs a Bayesian probability estimation to achieve fast estimation. The scheme is similar to hash-based estimators [38, 113] and the difference is discussed in [64]. In Kodialam and Nandagopal's approach [63], information from tags are collected by a RFID reader in a series of time frames. Each frame consists of a number of slots, and the tags probabilistically respond in those slots. Using the probabilistic counting methods, the reader estimates the number of tags based on the number of empty slots or the number of collision slots in each frame. Their best estimator is called the Unified Probabilistic Estimator (UPE). A follow-up work by the same authors proposes the Enhanced Zero-Based Estimator (EZB) [64], which makes its estimation based on the number of empty slots. The focus of the above estimators is to reduce the time it takes a reader to complete the estimation process. Because their goal is not conserving energy for active tags, their design is not geared towards reducing the number of transmissions made by the tags.

The Lottery-Frame scheme (LoF) [93] by Qian et al. employs a geometric distribution-based scheme to determine which slot in a time frame each tag will respond. It significantly reduces the estimation time when comparing with UPE. However, every tag must respond in each of the time frames, resulting in large energy cost when active tags use their own power to transmit. The First Non-Empty slots Based algorithm

(FNEB) [49] uses the slot number of the first reply from tags in a frame to count RFID tags in both static and dynamic environments.

Also related is a novel security protocol proposed by Tan et al. to monitor the event of missing tags in the presence of dishonest RFID readers [108]. In order to prevent a dishonest reader from replaying previously collected information, they maintain a timer in the server and periodically update the system clock. Li et al. [68] design a series of efficient protocols that employ novel techniques to identify missing tags in large-scale RFID systems.

None of the above estimators are designed with energy conservation in mind. In the following, we will present our energy efficient estimators.

5.2 Problem Definition And System Model

5.2.1 RFID Estimation Problem

The problem is to design efficient algorithms to estimate the number of RFID tags in a deployment area without actually reading the ID of each tag. Let N be the actual number of tags and $\hat{N}$ be the estimate. The estimation accuracy is specified by a confidence interval with two parameters: a probability value α and an error bound β, both in the range of $(0, 1)$. The requirement is that the probability for $\frac{N}{\hat{N}}$ to fall in the interval $[1 - \beta, 1 + \beta]$ should be at least α, i.e.,

$$Prob\{(1 - \beta)\hat{N} \leq N \leq (1 + \beta)\hat{N}\} \geq \alpha.$$

Our goal is to reduce the energy overhead incurred to the tags during the estimation process that achieves the above accuracy. Prior works on the RFID estimation problem focus on time-efficiency, which is the amount of time a RFID reader spends in estimating the number of tags in the system. Our work focuses on energy-efficiency, which is the amount of energy the tags spend during estimation process.

5.2.2 Active Tags

The type of active RFID systems considered in this work is applicable to a large deployment area that is hundreds of feet or more across. Passive tags are beyond the scope of this work. If they were used, one would have to take the RFID reader and move around the whole area, collecting tag information once every few feet. Active tags allow a reader to collect information from one location.

Tagged goods (such as apparel) may stack in piles, and there may be obstacles, such as racks filled with merchandize, between a tag and the reader. We expect active tags are designed to transmit with significant power that is high enough to ensure reliable information delivery in such a demanding environment. Hence, energy cost due to the tags' transmissions is the main concern in our algorithm design; it increases at least in the square of the maximum distance to be covered by the RFID system. Energy consumption that powers a tag's circuit for computing and receiving information is not affected by long distance and obstacles. Our new estimators are designed for RFID systems where power consumption by tags is dominated by transmission events due to long distances that the systems need to cover. Energy consumed by the RFID reader is less of a concern. We assume the reader transmits at sufficiently high power.

5.2.3 Communication Protocol

We use the following communication protocol between a reader and tags. The reader first synchronizes the clocks of the tags and then performs a sequence of pollings. Clock synchronization only needs to happen at the beginning of the protocol execution. RFID systems operate in low-rate wireless channels. Our new estimators only take a few seconds to complete. Clock drift should not be a major issue in a low-rate channel within such a short period time.

In each polling, the reader sends out a request, which is followed by a slotted time frame during which the tags respond. The polling request from the reader carries a *contention probability* $0 < p \leq 1$ and a frame size f. Each tag will participate in the

current polling with probability p. If it decides to participate, it will pick a slot uniformly at random from the frame, and transmit a bit string (called *response*) in that slot. The format of the response depends on the application. If the tag decides to not participate, it will keep silent. In our solutions, p will be set in the order of $\frac{1}{N}$.

If we know a lower bound N_{min} of N, the contention probability can be implemented efficiently to conserve energy. For example, a company's inventory of certain goods may be in the thousands and never before reduced below a certain number, or the company has a policy on the minimum inventory, or the RFID estimation becomes unnecessary when the number of tags is below a threshold. In these cases, we will have a lower bound N_{min}, which can be much smaller than N. If we know such a value of N_{min}, we can implement a contention probability p without requiring all tags to participate in the contention process. Since only a small number of tags actually participate in contention, energy cost is reduced. The implementation is described as follows: At the beginning of a polling, each tag makes a probabilistic decision: It goes to a standby mode for the current polling with probability $1 - \frac{1}{N_{min}}$ and wakes up until the next polling starts, or it stays awake to receive the polling request with probability $\frac{1}{N_{min}}$ and then decides to respond with probability $min\{p \times N_{min}, 1\}$. For example, if $N = 10,000$ and $N_{min} = 1,000$, then only 10 tags stay awake in each polling. In Section 5.3.5, another energy-reduction method, called request-less pollings, will be proposed to eliminate most polling requests.

In the above communication protocol, the reader's request may include an optional prefix and only tags that satisfy the prefix will participate in the polling. For example, suppose all tags deployed in one section of a warehouse carry the 96-bit GEN2 IDs that begin with "000" in the Serial Number field. In order to estimate the number of tags in this section, the request carries a predicate testing whether the first three bits of a tag's Serial Number is "000".

5.2.4 Empty/Singleton/Collision Slots

A slot is said to be *empty* if no tag responds (transmits) in the slot. It is called a *singleton slot* if exactly one tag responds. It is a *collision slot* if more than one tag responds. A singleton or collision slot is also called a *non-empty slot*. The Philips I-Code system [101] requires a slot length of 10 bits in order to distinguish singleton slots from collision slots. On the contrary, one bit is enough if we only need to distinguish empty slots from non-empty slots — '0' means empty and '1' means non-empty. Hence, the response will be much shorter (or consume much less energy) if an algorithm only needs to know empty/non-empty slots, instead of all three types of slots as required by [63].

In order to prolong the lifetime of tags, there are two ways to reduce their energy consumption: reducing the size of each response and reducing the number of responses. We will design algorithms that require only the knowledge of empty/non-empty slots and employ statistical methods to minimize the amount of transmission needed from the tags.

5.3 Generalized Maximum Likelihood Estimation Algorithm

Our first estimator for the number of RFID tags is called the *generalized maximum likelihood estimation* (GMLE) algorithm. It fully utilizes the information from all pollings in order to minimize the number of pollings it needs to meet the accuracy requirement.

5.3.1 Overview

GMLE uses the polling protocol described in Section 5.2.3. The frame size f is fixed to be one slot. The RFID reader adjusts the contention probability for each polling. Let p_i be the contention probability of the ith polling. GMLE only records whether the sole slot in each polling is empty or non-empty. Based on this information, it refines the estimate $\hat{N}$ until the accuracy requirement is met. Let z_i be the slot state of the ith polling. When at least one tag responds, the slot is non-empty and $z_i = 1$. When no tag responds, it is empty and $z_i = 0$. The sequence of z_i, $i \geq 1$, forms the *response vector*.

At the ith polling, each tag has a probability p_i to transmit and, if any tag transmits, z_i will be one. Hence,

$$Prob\{z_i = 1\} = 1 - (1 - p_i)^N \approx 1 - e^{-Np_i}, \tag{5–1}$$

where N is the the actual number of tags.

If the contention probabilities of the pollings are picked too small, the response vector will contain mostly zeros. If the contention probabilities are picked too large, the response vector will contain mostly ones. Both cases do not provide sufficient statistical information for accurate estimation. As will be discussed shortly, our analysis shows that the optimal contention probability for minimizing the number of pollings is $p_i = 1.594/N$. The problem is that we do not know N (which is the quantity we want to estimate).

In order to determine p_i, GMLE consists of an *initialization phase* and an *iterative phase*. The former quickly produces a coarse estimation of N. The latter refines the contention probability and generates the estimation result.

5.3.2 Initialization Phase

We want to pick a small value for the initial contention probability p_1 at the first polling. The expected number of responding tags is Np_1. If p_1 is picked too large, a lot of tags will respond, which is wasteful because one response or many responses produce the same information — a non-empty slot. Suppose we know an upper bound N_{max} of N. This information is often available in practice. For example, we know N_{max} is 10,000 if the warehouse is designed to hold no more than 10,000 microwaves (each tagged with a RFID), or the company's inventory policy requires that in-store microwaves should not exceed 10,000, or the warehouse only has 10,000 RFID tags in use. N_{max} can be much bigger than N. We pick $p_1 = \frac{1}{N_{max}}$ such that the expected number of responding tags is no more than one. If $z_1 = 0$, we multiply the contention probability by a constant $C(> 1)$, i.e., $p_2 = p_1 \times C$ for the second polling. We continue multiplying the contention probability by C after each polling until a non-empty slot is observed. When that happens (say, at

the lth polling), we have a coarse estimation of N to be $1/p_l$. Then we move to the next phase. When C is relatively large, the initialization phase only takes a few pollings to complete due to the exponential increase of the contention probability.

5.3.3 Iterative Phase

This phase iteratively refines the estimation result after each polling, and terminates when the specified accuracy requirement is met. Let $\hat{N}_i$ be the estimated number of tags after the ith polling. To compute $\hat{N}_i$, the reader performs three tasks at the ith polling. First, it sets the contention probability as follows before sending out the polling request:

$$p_i = \frac{\omega}{\hat{N}_{i-1}}, \tag{5–2}$$

where $\hat{N}_{i-1}$ is the estimate after the previous polling and ω is a system parameter, which will be extensively analyzed in the next subsection. Second, based on the received z_i and the history information, the reader finds the new estimate of N that maximizes the following likelihood function:

$$L_i = \prod_{j=1}^{i} (1-p_j)^{N(1-z_j)} (1-(1-p_j)^N)^{z_j}, \tag{5–3}$$

where $(1-p_j)^{N(1-z_j)}(1-(1-p_j)^N)^{z_j}$ is the probability for the observed state z_j of the jth polling to occur. Namely, we want to find

$$\hat{N}_i = \arg\max_N \{L_i\}. \tag{5–4}$$

Third, after computing $\hat{N}_i$, the reader has to determine if the confidence interval of the new estimate meets the requirement. In the following, we show how the above tasks can be achieved.

5.3.3.1 Compute the value of $\hat{N}_i$

We compute the new estimate of N that maximizes (5–3). Since the maxima is not affected by monotone transformations, we use logarithm to turn the right side of the

equation from product to summation:

$$\ln(L_i) = \sum_{j=1}^{i} \left[N(1-z_j)\ln(1-p_j) + z_j \ln(1-(1-p_j)^N) \right].$$

To find the maxima, we differentiate both sides:

$$\frac{\partial \ln(L_i)}{\partial N} = \sum_{j=1}^{i} \left[(1-z_j)\ln(1-p_j) - z_j \frac{(1-p_j)^N \ln(1-p_j)}{1-(1-p_j)^N} \right]. \tag{5–5}$$

We then set the right side to zero and solve the equation for the new estimate $\hat{N}_i$. Note that the derivative is a monotone function of N, we can numerically obtain $\hat{N}_i$ through bisection search.

5.3.3.2 Termination Condition

Using the $\delta-$method [15], we show in Appendix A that, when i is large, $\hat{N}_i$ approximately follows the Gaussian distribution:

$$Norm\left(N, \frac{(1-(1-p_i)^N)}{i(1-p_i)^N \ln^2(1-p_i)} \right).$$

The variance of $\hat{N}_i$ is

$$Var(\hat{N}_i) \approx \frac{1-(1-p_i)^N}{i(1-p_i)^N \ln^2(1-p_i)}. \tag{5–6}$$

When N is large and p_i is small, we can approximate $(1-p_i)^N$ as e^{-Np_i} and $\ln(1-p_i)$ as p_i. The above variance becomes

$$Var(\hat{N}_i) \approx \frac{e^{Np_i}-1}{ip_i^2}. \tag{5–7}$$

Hence, the confidence interval of N is

$$\hat{N}_i \pm Z_\alpha \cdot \sqrt{\frac{e^{\hat{N}_i p_i}-1}{ip_i^2}}, \tag{5–8}$$

where Z_α is the α percentile for the standard Gaussian distribution. For example, when $\alpha = 95\%$, $Z_\alpha = 1.96$. Because N is undetermined, we use $\hat{N}_i$ as an approximation when computing the standard deviation in (5–8).

The termination condition for GMLE is therefore

$$Z_\alpha \cdot \sqrt{\frac{e^{\hat{N}_i p_i} - 1}{i p_i^2}} \leq \hat{N}_i \cdot \beta, \tag{5–9}$$

where β is the error bound. The above inequality can be rewritten as

$$\sqrt{i} \geq \frac{Z_\alpha \sqrt{e^{\hat{N}_i p_i} - 1}}{\hat{N}_i p_i \beta}. \tag{5–10}$$

When i is large, the estimation changes little from one polling to the next. Hence, $p_i = \omega / \hat{N}_{i-1} \approx \omega / \hat{N}_i$. We have

$$i \geq \frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega^2 \beta^2}. \tag{5–11}$$

Hence, if ω is determined, we can theoretically compute the approximate number of pollings that is required in order to meet the accuracy requirement. For example, if $\alpha = 95\%$, $\beta = 5\%$, and $\omega = 1.594$ (which is the optimal value to be given shortly), 2372 pollings will be required. Note that (5–11) is independent with the actual number of tags, N. Hence, our approach has perfect scalability.

Fig. 5-1 shows the simulation result of GMLE when $N = 10,000$, $\alpha = 95\%$, $\beta = 5\%$ and $\omega = 1.594$. The simulation setup can be found in Section 5.5. The middle curve is the estimated number of tags, $\hat{N}_i$, with respect to the number pollings. It converges to the true value N represented by the central straight line. The upper and lower curves represent the 95% confidence interval, which shrinks as the number of pollings increases.

5.3.4 Determine the value of ω

We demonstrate the impact of the value ω on two performance metrics: the *number of pollings* and the *number of tag responses* (i.e., the number of tag transmissions). The former measures the estimation time since each polling takes an equal amount of time for request/response exchange. The latter measures the energy cost because each response corresponds to one tag making one transmission in a slot.

5.3.4.1 Number of Pollings

According to (5–11), the number of pollings for meeting the accuracy requirement is $\frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega^2 \beta^2}$. To find its minimum value, we differentiate it with respect to ω and let the result be zero. Solving the equation, we have $\omega = 1.594$. Hence, the optimal value of p_i that minimizes the number of pollings is

$$p_i = \frac{1.594}{\hat{N}_{i-1}}. \tag{5–12}$$

5.3.4.2 Number of Responses

We count the total number of responses during the estimation process. After a small number of pollings, the estimation will closely approximate N (see Fig. 5-1). Hence, the expected number of responses for each polling is $Np_i \approx N_{i-1}p_i = \omega$. After $\frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega^2 \beta^2}$ pollings are made, the total number of responses is roughly

$$\frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega^2 \beta^2} \omega = \frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega \beta^2}. \tag{5–13}$$

Our simulation results in Section 5.5 demonstrate that the approximation in the above count is reasonably accurate. It is an increasing function with respect to ω, which means that a larger value of ω will lead to a larger number of responses. We give the intuition as follows: A larger ω means a larger contention probability and thus more collisions. Two or more responses in a collision slot produce the same amount of information as one response in a singleton slot (see further explanation in Section 5.3.6). In other words, in order to generate the necessary amount of information for meeting the accuracy requirement, more responses must be needed if there are more collisions.

5.3.4.3 Summary

In Fig. 5-2, we plot the number of pollings and the number of responses with respect to the value of ω. The number of pollings is minimized at $\omega = 1.594$. When ω is smaller than 1.594, its value controls the performance tradeoff between the two metrics. When we decrease ω, the energy cost (i.e., the number of responses) drops at the expenses of the estimation time (i.e., the number of pollings). Our further simulations in Section 5.5 show that even at $\omega = 1.594$, the energy cost of GMLE is far below those of the existing protocols.

5.3.5 Request-less Pollings

We observe that, after a number of pollings, the value of p_i will stay in a very small range and does not change much. It becomes unnecessary for the RFID reader to transmit it at each polling. Hence, we improve GMLE as follows: If the percentage change in p_i during a certain number M_1 of consecutive pollings is below a small threshold, the reader will broadcast a polling request, carrying the latest value of p_i, a flag indicating that it will no longer transmit polling requests for a certain number M_2 of slots, and the value of M_2. Without receiving further polling requests, the tags will respond with the same contention probability in the subsequent M_2 slots. This is called the *request-less pollings*. After M_2 slots, the reader will recalculate the contention probability, broadcast another polling request, carrying the new probability value, a flag, and M_2. This process repeats until the termination condition in (5–9) is met. With the threshold being 10%, $M_1 = 10$, and $M_2 = 50$, our simulation results show that the performance difference caused by request-less pollings is negligibly small even though the contention probability during request-less pollings may be slightly off the value set by (5–2). Request-less pollings can also be applied to the algorithm in the next section.

5.3.6 Information Loss due to Collision

GMLE has a frame size of one slot. It obtains only binary information at each polling. No matter how many tags respond, the information that the reader receives is

always the same, i.e., $z_i = 1$, which implies information loss when two or more tags decide to transmit at a polling. Let's compare two scenarios. In one scenario, only one tag responds at a polling. In the other, two tags respond. These two scenarios generate the same information but the energy cost of the second scenario is twice of the first. To address this issue, we design another algorithm that reduces the probability of collision and, moreover, compensate the impact of collision in its computation.

5.4 Enhanced Generalized Maximum Likelihood Estimation Algorithm

The *enhanced generalized maximum likelihood estimation* (EGMLE) algorithm is our second estimator for the number of RFID tags. It also utilizes history information from previous pollings and uses the maximum likelihood method to estimate the number of tags. However, instead of only obtaining binary information, it computes the number of responses in each polling. Because more information can be extracted, it is able to achieve much better energy efficiency than GMLE.

5.4.1 Overview

EGMLE uses the same polling protocol as GMLE does, except that its frame size f is larger than one in order to reduce the probability of collision. The result of the ith polling, x_i, is no longer a binary value. Instead, it is an estimate of the number of tags that respond during the polling.

EGMLE takes two steps to solve the collision problem. First, it increases the frame size f such that the tags that decide to respond at a polling are likely to respond at different slots in the frame. We pick values for p_i and f such that the collision probability is very small. Second, we compensate the remaining impact of collision in our computation.

EGMLE also consists of an *initialization phase* and an *iterative phase*. The initialization phase of EGMLE is the same as the initialization phase of GMLE, except that when the RFID reader obtains the first non-zero result x_l at the lth polling with a

contention probability p_l, it computes a coarse estimation of N as $\frac{x_l}{p_l}$. Then it moves to the next phase below.

5.4.2 Iterative Phase

This phase iteratively refines the estimation after each polling, and terminates when the specified accuracy requirement is met. The reader performs four tasks during the ith polling. First, it computes the contention probability before sending out the polling request.

$$p_i = \frac{\omega}{\hat{N}_{i-1}}, \tag{5–14}$$

where $\hat{N}_{i-1}$ is the estimate after the previous polling and ω is one by default. As we will show in the next subsection, performance tradeoff can be made by choosing other values for ω.

Second, the reader computes the number of responses x_i in the current frame.

Third, based on the received x_i and the history information, the reader computes the new estimate of N that maximizes the following likelihood function:

$$L_i = \prod_{j=l+1}^{i} \left[\frac{1}{\sqrt{2\pi N p_j (1-p_j)}} \cdot e^{-\frac{((1+\varepsilon)x_j - Np_j)^2}{2Np_j(1-p_j)}} \right], \tag{5–15}$$

where ε is introduced to compensate for collision and the iterative phase begins from the $(l+1)$th polling. The above formula and the value of ε will be derived shortly. The new estimate is

$$\hat{N}_i = \arg\max_{N} \{L_i\}. \tag{5–16}$$

Fourth, after computing $\hat{N}_i$, the reader determines if the estimate meets the accuracy requirement. In the following, we give the details of the above tasks.

5.4.2.1 Compute the number of responses

At the ith polling, the reader measures the number of non-empty slots in the frame, denoted as x_i, which is an integer in the range of $[0..f]$. Due to possible collision, the

actual number of responses, denoted as x_i^*, can be greater. Let $x_i^* = (1+\varepsilon)x_i$. The value of ε is determined below.

Since each tag independently decides to respond with probability p_i, x_i^* follows a binomial distribution, $Bino(N, p_i)$, i.e.,

$$Prob\{x_i^* = k\} = \binom{N}{k} p_i^k (1-p_i)^{N-k}. \tag{5–17}$$

Suppose ω takes the default value, 1. When i is large, N_{i-1} approximates N and thus $p_i \approx 1/N$. If N is sufficiently large, $Prob\{x_i^* = 2\} \approx 0.1839$, $Prob\{x_i^* = 3\} \approx 0.0613$, $Prob\{x_i^* = 4\} \approx 0.0153$, and the probability decreases exponentially with respect to k. $Prob\{x_i^* > 4\}$ is only about 0.0037.

Next, we compute the probability for collision to happen at the ith polling, which is denoted as $Prob_i\{collision\}$.

$$\begin{aligned} Prob_i\{collision\} &= \sum_{k=2}^{N} Prob_i\{collision | x_i^* = k\} \times Prob\{x_i^* = k\} \\ &= \sum_{k=2}^{f} (1 - \frac{P(f,k)}{f^k}) \times Prob\{x_i^* = k\} + \sum_{k=f+1}^{N} 1 \times Prob\{x_i^* = k\}, \end{aligned}$$

where $P(f,k) = \frac{f!}{(f-k)!}$ is the permutation function. Fig. 5-3 shows the collision probability $Prob_i\{collision\}$ with respect to f. It diminishes quickly as f increases. When $f = 10$ (which is what we use in the simulations), $Prob_i\{collision\}$ is just 0.046. With such a small probability, the chance for more than two tags involved in a collision or more than one collision at a polling is exceedingly small and thus ignored. Therefore, to approximate x_i^*, we multiply x_i by 1.046 to compensate the impact of collision. Namely, $\varepsilon = 0.046$.

5.4.2.2 Compute the value of $\hat{N}_i$

Recall that the iterative phase starts at the $(l+1)$th polling. After the ith polling, the reader has collected the values of x_j, $l < j \leq i$. By our previous analysis, we know

that $x_i^* = (1+\varepsilon)x_i$ and it follows a binomial distribution $Bino(N, p_j)$. When N is large enough, the binomial distribution can be closely approximated by a Gaussian distribution $Norm(\mu_j, \sigma_j)$ with parameters $\mu_j = Np_j$ and $\sigma_j = \sqrt{Np_j(1-p_j)}$. Namely,

$$x_j^* \approx (1+\varepsilon)x_j \sim Norm(Np_j, Np_j(1-p_j)). \tag{5–18}$$

Hence, the probability for the *measured number of responses*, $(1+\varepsilon)x_j$, to occur under this distribution is $\frac{1}{\sqrt{2\pi Np_j(1-p_j)}} \cdot exp[-\frac{((1+\varepsilon)x_j - Np_j)^2}{2Np_j(1-p_j)}]$. The likelihood function for all measured numbers of responses in the pollings, $(1+\varepsilon)x_j$, $l < j \leq i$, to occur is

$$L_i = \prod_{j=l+1}^{i} \left[\frac{1}{\sqrt{2\pi Np_j(1-p_j)}} \cdot e^{-\frac{((1+\varepsilon)x_j - Np_j)^2}{2Np_j(1-p_j)}} \right]. \tag{5–19}$$

Our goal is to find the value $\hat{N}_i$ that maximizes the likelihood function. We first take logarithm on both sides of (5–19).

$$\ln(L_i) = \sum_{j=l+1}^{i} \left[\ln \frac{1}{\sqrt{2\pi Np_j(1-p_j)}} - \frac{((1+\varepsilon)x_j - Np_j)^2}{2Np_j(1-p_j)} \right]. \tag{5–20}$$

We then differentiate both sides.

$$\begin{aligned} \frac{\partial \ln(L_i)}{\partial N} &= \sum_{j=l+1}^{i} \left[-\frac{1}{2N} + \frac{(1+\varepsilon)^2 x_j^2 - (Np_j)^2}{2N^2 p_j(1-p_j)} \right] \\ &= \sum_{j=l+1}^{i} \frac{(1+\varepsilon)^2 x_j^2 - (Np_j)^2}{2N^2 p_j(1-p_j)} - \frac{i-l}{2N}. \end{aligned} \tag{5–21}$$

Finally, we set the right side to be zero and numerically compute the value of $\hat{N}_i$.

5.4.2.3 Termination Condition

The *fisher information*[1] $\mathcal{I}(\hat{N}_i)$ of L_i is defined as follows

$$\mathcal{I}(\hat{N}_i) = -E\left[\frac{\partial^2 \ln(L_i)}{\partial N^2} \right]. \tag{5–22}$$

[1] The fisher information [67] is a way of measuring the amount of information that an observable random variable x carries about an unknown parameter θ upon which the likelihood function of θ, $L(\theta) = f(x;\theta)$, depends.

According to (5–21), we have

$$\mathcal{I}(\hat{N}_i) = E\Big[\sum_{j=l+1}^{i} \frac{(1+\varepsilon)^2 x_j^2}{N^3 p_j(1-p_j)} - \frac{i-l}{2N^2}\Big]$$

$$= \sum_{j=l+1}^{i} \frac{(Np_j)^2 + Np_j(1-p_j)}{N^3 p_j(1-p_j)} - \frac{i-l}{2N^2} \tag{5–23}$$

$$= \sum_{j=l+1}^{i} \frac{p_j}{N(1-p_j)} + \frac{i-l}{2N^2}. \tag{5–24}$$

Above, we have applied $E((1+\varepsilon)^2 x_j^2) = (Np_j)^2 + Np_j(1-p_j)$ in (5–23) because $(1+\varepsilon)x_j \sim Norm(Np_j, Np_j(1-p_j))$ and $E(x^2) = (E(x))^2 + Var(x)$.

Following the classical theory for MLE, when i is sufficiently large, the distribution of $\hat{N}_i$ is approximated by

$$Norm(N, \frac{1}{\mathcal{I}(\hat{N}_i)}). \tag{5–25}$$

Hence, the confidence interval is

$$\hat{N}_i \pm Z_\alpha \cdot \sqrt{\frac{1}{\mathcal{I}(\hat{N}_i)}}. \tag{5–26}$$

Note that we use $\hat{N}_i$ as an approximation for N in the computation when necessary since N is unknown. The termination condition for EGMLE to achieve the required accurary is

$$Z_\alpha \cdot \sqrt{\frac{1}{\mathcal{I}(\hat{N}_i)}} \leq \hat{N}_i \cdot \beta. \tag{5–27}$$

Fig. 5-4 shows the simulation result of EGMLE when $N = 10,000$, $\alpha = 95\%$, $\beta = 5\%$, and $\omega = 1$. The middle curve is the value of $\hat{N}_i$, which converges to the value of N represented by the central straight line. The upper and lower curves represent the 95% confidence interval, which shrinks as the number of pollings increases. The algorithm terminates after 1081 pollings.

5.4.3 Performance Tradeoff

According to (5–14), the contention probability is proportional to ω. We study how the value of ω controls the tradeoff between the estimation time and the energy

cost, which are measured by the number of pollings and the number of responses, respectively.

5.4.3.1 Number of Pollings

Since the MLE approach provides statistically consistent estimate, when i is large, (5–24) can be approximated as follows:

$$\begin{aligned}\mathcal{I}(\hat{N}_i) &= \sum_{j=l+1}^{i} \frac{p_j}{N(1-p_j)} + \frac{i-l}{2N^2} \\ &\approx \left(\frac{p_i}{N(1-p_i)} + \frac{1}{2N^2}\right) \cdot (i-l) \\ &\approx \frac{2Np_i+1}{2N^2} \cdot (i-l). \end{aligned} \tag{5–28}$$

where $p_i \ll 1$. According to (5–27), we have

$$\mathcal{I}(\hat{N}_i) \geq (\frac{Z_\alpha}{\hat{N}_i \cdot \beta})^2 \tag{5–29}$$

(5–28) and (5–29) give us the following inequality:

$$\begin{aligned}\frac{2Np_i+1}{2N^2} \cdot (i-l) &\geq (\frac{Z_\alpha}{\hat{N}_i \cdot \beta})^2, \\ i &\geq \frac{2Z_\alpha^2}{(2\omega+1)\beta^2}, \end{aligned} \tag{5–30}$$

where $\hat{N}_i \approx N$ and $l \ll i$. Hence, the number of pollings it takes to achieve the accuracy requirement is $\frac{2Z_\alpha^2}{(2\omega+1)\beta^2}$.

The solid line in Fig. 5-5 shows the number of pollings with respect to ω when $\alpha = 95\%$ and $\beta = 5\%$. It is a decreasing function in ω. The reason is that a larger ω results in more responses (and thus more information) in each polling. Consequently, a less number of pollings is needed to achieve a certain accuracy requirement.

5.4.3.2 Number of Responses

When i is large, the expected number of responses for each polling is $Np_i \approx N_{i-1}p_i = \omega$. After $\frac{2Z_\alpha^2}{(2\omega+1)\beta^2}$ pollings are made, the total number of responses is roughly

$$\frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega^2 \beta^2} \omega = \frac{Z_\alpha^2 \cdot (e^\omega - 1)}{\omega \beta^2}. \tag{5–31}$$

The dotted line in Fig. 5-5 shows the number of responses with respect to ω when $\alpha = 95\%$ and $\beta = 5\%$. It is an increasing function in ω, which means that a larger value of ω will lead to a larger number of responses.

5.4.3.3 Summary

Fig. 5-5 demonstrates the performance tradeoff under different values of ω. As we decrease ω, EGMLE achieves better energy efficiency by requiring a fewer number of responses, at the expense of time efficiency by requiring a larger number of pollings.

5.5 Simulations

We evaluate the performance of GMLE and EGMLE by simulations. In order to demonstrate the performance tradeoff between energy cost and estimation time, we choose two different contention probability parameters for each of the two algorithms. We use $\omega = 0.5$ and 1.594 for GMLE, i.e., $p_i = \frac{0.5}{\hat{N_{i-1}}}$ and $\frac{1.594}{\hat{N_{i-1}}}$. Note that 1.594 is the optimal value of ω for time efficiency in GMLE. We denote the corresponding variants of the algorithm as GMLE(0.5) and GMLE(1.594).

For EGMLE, Fig. 5-5 shows that the number of pollings and the number of responses are both monotonic functions with respect to ω, which means there is no optimal ω for either energy efficiency or time efficiency. We choose $\omega = 0.5$ and 1.0 for EGMLE, i.e., $p_i = \frac{0.5}{\hat{N_{i-1}}}$ and $\frac{1.0}{\hat{N_{i-1}}}$. The corresponding variants of the algorithm is denoted as EGMLE(0.5) and EGMLE(1.0). Section 5.4.2 shows how to compute the compensation parameter ε for EGMLE(1.0), which is 0.046. Following the same steps, we obtain $\varepsilon = 0.012$ for EGMLE(0.5). We compare the proposed algorithms with the state-of-the-art algorithms in the related work. They are the Unified Probabilistic Estimator (UPE) [63] and the Enhanced Zero-Based (EZB) estimator [64]. The original UPE, denoted as UPE-O, is very energy-inefficient because its contention probability

begins from 100% and thus all tags will respond. We modify it (denoted as UPE-M) to begin from a small initial contention probability $\frac{1}{N_{max}}$ and keep the remaining part of UPE-O. This section shows the performance of both UPE-O and UPE-M. We run each simulation 100 times and average the outcomes.

In the initialization phase of our algorithms, let $N_{max} = 1,000,000$ and $C = 2$. The frame size in EGMLE(0.5) and EGMLE(1.0) is 10 slots. The parameters for UPE and EZB are chosen based on the original papers whenever possible. All algorithms except for UPE need only to identify empty and non-empty slots. To set a non-empty slot apart from an empty slot, a tag only needs to respond with a short bit string (one bit) to make the channel busy. UPE has to identify empty, singleton and collision slots. To set a singleton slot apart from a collision slot, many more bits (10 used by UPE) are necessary [2]. For example, CRC may be used to detect collision.

The energy cost of an algorithm depends on (1) the number of responses that all tags transmit before the algorithm terminates and (2) the size of each response. We use 'S' to mean that the response is a short bit string (in the empty/non-empty case), and 'L' to mean a long bit string (in the empty/singleton/collision case).

We do not include the simulation results for LoF [93] because its energy cost is much higher than others. Its number of responses transmitted by the tags is kN, where k is the number of frames used in the estimation process.

5.5.1 Number of Responses

The first simulation studies the number of responses in each algorithm with respect to N, α and β. Table 5-1 shows the number of responses with respect to N when $\alpha = 90\%$ and $\beta = 9\%$. The proposed algorithms require fewer responses than UPE and EZB. As predicted, UPE-O is energy-inefficient; UPE-M works much better. The best algorithm is EGMLE(0.5), whose number of responses is about one fifth of what UPE-M requires and one ninetieth of what EZB requires when N is 20,000. Moreover, each response in UPE is much longer.

GMLE(0.5) has a smaller energy cost than GMLE(1.594). For example, $N = 10,000$, the ratio between the number of responses by GMLE(1.594) and that by GMLE(0.5) is 2.01, which is close to the theoretically-computed ratio of 1.90 in Fig. 5-2. Similarly, EGMLE(0.5) is more energy efficient than EGMLE(1.0). When $N = 10,000$, the ratio between the number of responses by GMLE(1.594) and that by GMLE(0.5) is 1.28, which is also close to the theoretical value of 1.34 in Fig. 5-5.

We vary α from 90% to 95% and to 99%, and vary β from 9% to 6% and to 3%. Tables 5-2 to 5-9 show similar comparison under different values of α and β values. In all cases, the number of responses increases when α increases or β decreases, and except for EZB, the number does not vary much with respect to N, meaning that all algorithms except for EZB achieve good scalability. The ratio between the numbers for different algorithms appears to be quite stable under different parameter settings.

5.5.2 Total Number of Bits Transmitted

The second simulation evaluates the energy cost of the algorithms. As mentioned before, one bit is enough to separate empty/non-empty slot. Hence, the response of GMLE, EGMLE and EZB is one bit long. A response in UPE-M is 10 bits long [63]. We compare the total number of bits transmitted by all tags before each algorithm terminates. We omit the results for UPE-O, which are much worse than the results of UPE-M. Fig. 5-6 shows the simulation results with respect to N when $\alpha = 90\%, \beta = 9\%, 6\%$ and 3%. For example, when $\alpha = 90\%$, $\beta = 3\%$, and $N = 20,000$, the ratio between the number of bits transmitted by UPE-M (EZB) and that by our best estimator EGMLE(0.5) is 45.32 (71.28). Fig. 5-7 and Fig. 5-8 show the comparison under different β values when $\alpha = 95\%$ and 99%, respectively. Their results are similar to Fig. 5-6. It should be noted that the number of bits transmitted is not an accurate measurement of the energy cost because it ignores the energy spent to power up the radio and synchronize with the reader. However, combining the number of bits and the number of

transmissions (in the previous subsection) still gives a good idea on how energy-efficient each algorithm is.

5.5.3 Estimation Time

The third simulation compares the time it takes for each algorithm to complete the estimation of N. Based on the specification of the Philips I-Code system [101], after the required waiting times (e.g., gap between transmissions) are included, it can be calculated that a RFID reader needs $0.4ms$ to detect an empty slot, $0.8ms$ to detect a collision or a singleton slot, and $1ms$ to broadcast a polling request. Hence, GMLE, EGMLE and EZB requires a slot length of $0.4ms$, while UPE-M requires a slot length of $0.8ms$. Recall that the contention probability takes the form of $\frac{\omega}{\hat{N}_i}$, where ω is a known constant. Thus the reader transmits $\hat{N}_i$ instead of the actual probability value in the polling requests. If we assume N_{max} is no more than a million, then 20 bits for $\hat{N}_i$ are sufficient. GMLE has a fixed frame size of one slot. EGMLE has a fixed frame size of 10 slots. EZB and UPE-M also have pre-determined frame sizes. Let $\alpha = 90\%, \beta = 9\%, 6\%$ and 3%. The three plots in Fig. 5-9 show the estimation times of the algorithms with respect to the number of tags in the deployment. The times grow very slowly as the number of tags increase, which suggests the algorithms all scale well. In the first plot of Fig. 5-9, UPE-M takes the least amount of time, only about 0.5 second, to estimate 20,000 tags, while the other algorithms take between 0.7 to 2.0 seconds. GMLE(1.594) takes less estimation time than GMLE(0.5) and the ratio is 0.61, which is consistent with the theoretical value of 0.58 in Fig. 5-2. Similarly, EGMLE(1.0) takes less time than EGMLE(0.5) and the ratio is 0.68, which is also consistent with the theoretical value of 0.67 in Fig. 5-5. Fig. 5-10 and Fig. 5-11 show similar simulation results when $\alpha = 95\%$ and 99%, respectively. Even though the new algorithms take longer to complete, their estimation time is still small. We believe the extra time needed can be well justified for the large energy saving.

There exists a performance tradeoff between GMLE and EGMLE. In the previous two subsections, we have examined energy cost in terms of number of responses and number of transmitted bits. EGMLE always performs better than GMLE. In this subsection, we compare estimation time of our two methods. GMLE performs better than EGMLE. Because the focus of this work is on energy efficiency, we regard EGMLE as our best estimator for energy saving.

5.6 Summary

This chapter proposes two probabilistic algorithms for estimating the number of RFID tags in a region. We believe the algorithms are the first of its kind that targets at prolonging the lifetime of the active RFIDs. Their energy cost is far less than the state-of-the-art algorithms in the related work. Moreover, we reveal a fundamental tradeoff between the energy cost and the estimation time. By tuning a system parameter, the algorithms can trade longer estimation time for less energy cost, or vice versa.

Appendix A: Distribution and Variance of $\hat{N}_i$

Let i be a large positive integer. Consider the sequence of Bernoulli random variables, Z_j, $1 \leq j \leq i$, whose success probability is $q = 1 - (1 - p_i)^N$. Let $\hat{q} = (\sum_{j=1}^{i} Z_j)/i$, which is the estimation of the success probability q. It is known that asymptotically $\hat{q}$ follows a normal distribution:

$$\hat{q} \sim Norm\left(q, \frac{q(1-q)}{i}\right). \tag{5–32}$$

Because the MLE approach provides statistically consistent estimate, when i is large, we can consider the contention probabilities in the later stage of the pooling process to be approximately a constant. In addition, the number of polling results before stabilization of the contention probability is limited, and their impact will diminish as i becomes large. That is, they can be ignored when the asymptotic property of $\hat{N}_i$ is considered. Hence, for the asymptotic property, we can let $p_j = p_i$, for $1 \leq j \leq i$, and

Eq. (5–5) becomes

$$\frac{\partial \ln(L_i)}{\partial N} = \ln(1-p_i)\left[(i - \sum_{j=1}^{i} Z_j) - \frac{(1-p_i)^N}{1-(1-p_i)^N}\sum_{j=1}^{i} Z_j\right]. \tag{5–33}$$

Therefore, the MLE $\hat{N}_i$ that solves $\frac{\partial \ln(L_i)}{\partial N} = 0$ satisfies

$$(1-p_i)^{\hat{N}_i} = 1 - (\sum_{j=1}^{i} Z_j)/i = 1 - \hat{q}. \tag{5–34}$$

Hence, from (5–32), $(1-p_i)^{\hat{N}}$ asymptotically follows the following normal distribution

$$Norm\left((1-p_i)^N, \frac{(1-(1-p_i)^N)(1-p_i)^N}{i}\right). \tag{5–35}$$

According to the δ-method [15], if a random variable X_i satisfies

$$X_i \xrightarrow{D} Norm(\theta, \frac{\sigma^2}{i}), \tag{5–36}$$

where θ and σ are finite constants and $\xrightarrow{D}$ means convergence in distribution, then we must have

$$g(X_i) \xrightarrow{D} Norm\left(g(\theta), \frac{\sigma^2 [g'(\theta)]^2}{i}\right), \tag{5–37}$$

for any function g such that $g'(\theta)$ exists and takes a non-zero value. Based on (5–36) and (5–37), taking the logarithm of (5–35), we have

$$\hat{N}_i \cdot \ln(1-p_i) \sim Norm\left(N\ln(1-p_i), \frac{(1-(1-p_i)^N)}{i(1-p_i)^N}\right). \tag{5–38}$$

$$\text{That is,} \quad \hat{N}_i \sim Norm\left(N, \frac{(1-(1-p_i)^N)}{i(1-p_i)^N \ln^2(1-p_i)}\right). \tag{5–39}$$

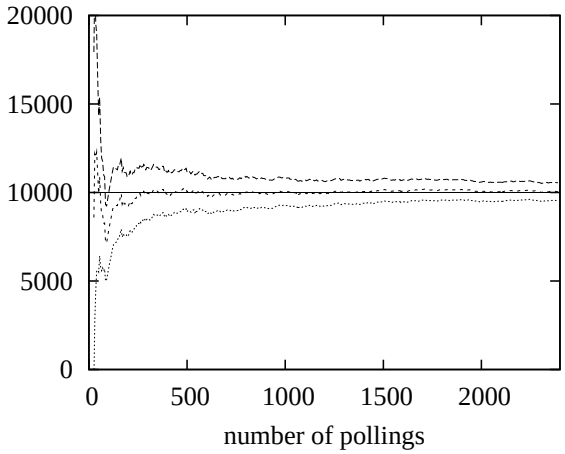

Figure 5-1. The middle curve shows the estimated number of tags with respect to the number of pollings. The upper and lower curves show the confidence interval. The straightline shows the true number of tags.

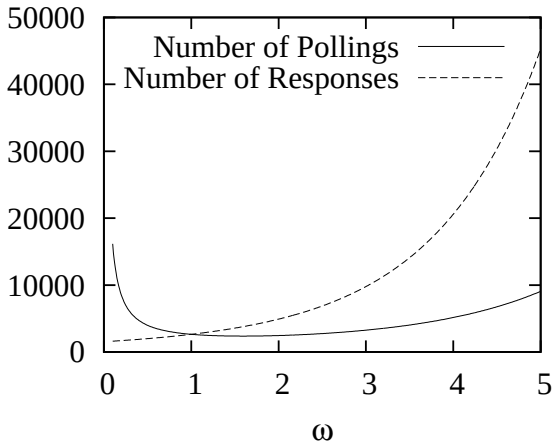

Figure 5-2. The solid line shows the number of pollings with respect to ω when $\alpha = 95\%$ and $\beta = 5\%$. The dotted line shows the number of responses with respect to ω for the same parameter settings.

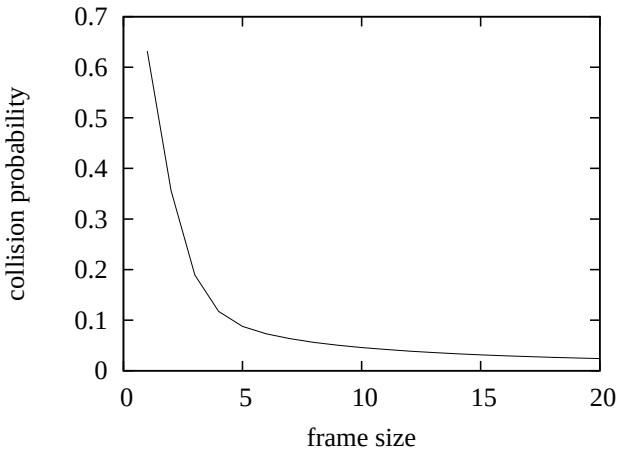

Figure 5-3. The collision probability with respect to the frame size f.

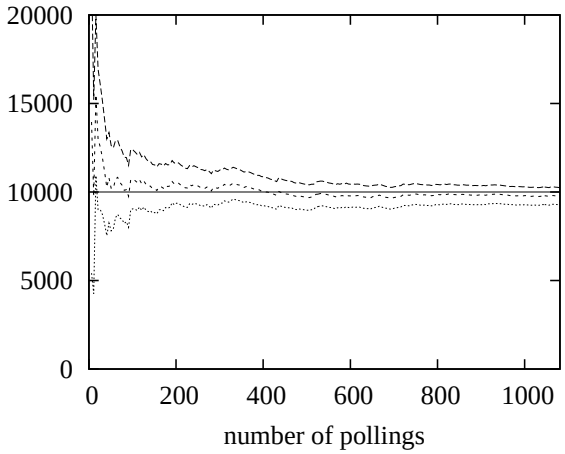

Figure 5-4. The middle curve shows the estimated number of tags with respect to the number of pollings. The upper and lower curves show the confidence interval. The straightline shows the true number of tags.

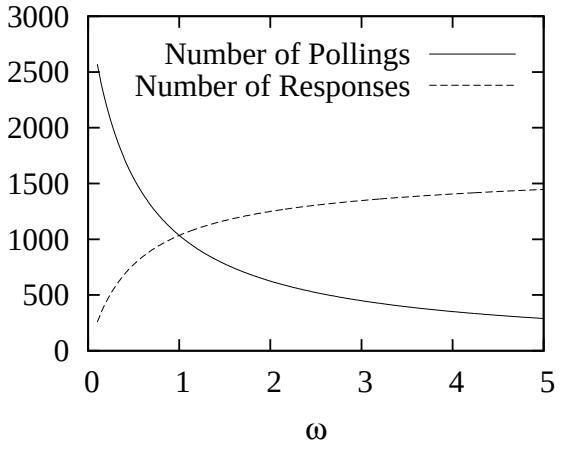

Figure 5-5. The solid line shows the number of pollings with respect to ω when $\alpha = 95\%$ and $\beta = 5\%$. The dotted line shows the number of responses with respect to ω for the same parameter settings.

Table 5-1. Number of Responses when $\alpha = 90\%, \beta = 9\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	432S	767 S	172 S	225 S	6345 L	709 L	4342 S
10000	414S	832 S	180 S	231 S	11986 L	899 L	8683 S
20000	402S	844 S	186 S	213 S	22895 L	977 L	17366 S

Table 5-2. Number of Responses when $\alpha = 90\%, \beta = 6\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	1041 S	1855 S	402 S	523 S	7144 L	1811 L	7236 S
10000	1153 S	1924 S	414 S	519 S	12645 L	1687 L	14472 S
20000	1015 S	1797 S	375 S	503 S	23808 L	1814 L	28944 S

Table 5-3. Number of Responses when $\alpha = 90\%, \beta = 3\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	3927S	7341 S	1499 S	2037 S	12664 L	6426 L	27497 S
10000	3760S	7339 S	1489 S	2059 S	18023 L	6581 L	54993 S
20000	3783S	7350 S	1543 S	2002 S	28708 L	6993 L	109987 S

Table 5-4. Number of Responses when $\alpha = 95\%, \beta = 9\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	603S	1112 S	258 S	330 S	6715 L	1073 L	4342 S
10000	669S	1120 S	247 S	304 S	12062 L	961 L	8683 S
20000	680S	1197 S	262 S	320 S	23345 L	1136 L	17366 S

Table 5-5. Number of Responses when $\alpha = 95\%, \beta = 6\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	1340 S	2515 S	581 S	736 S	7712 L	2598 L	10130 S
10000	1354 S	2511 S	596 S	736 S	13477 L	2318 L	20261 S
20000	1381 S	2630 S	555 S	749 S	24631 L	2510 L	40521 S

Table 5-6. Number of Responses when $\alpha = 95\%, \beta = 3\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	5687 S	10493 S	2181 S	2915 S	14678 L	8858 L	39074 S
10000	5673 S	10286 S	2267 S	2924 S	20845 L	9364 L	78148 S
20000	5588 S	10637 S	2217 S	2990 S	32339 L	9683 L	156297 S

Table 5-7. Number of Responses when $\alpha = 99\%, \beta = 9\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	1040 S	2162 S	427 S	453 S	7240 L	1726 L	7236 S
10000	1071 S	2135 S	416 S	529 S	12842 L	1906 L	14472 S
20000	1017 S	1916 S	439 S	573 S	23982 L	1819 L	28944 S

Table 5-8. Number of Responses when $\alpha = 99\%, \beta = 6\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	2527 S	4785 S	965 S	1269 S	9679 L	4311 L	17366 S
10000	2527 S	4637 S	973 S	1248 S	15336 L	4130 L	34733 S
20000	2440 S	4580 S	991 S	1293 S	26128 L	4044 L	69465 S

Table 5-9. Number of Responses when $\alpha = 99\%, \beta = 3\%$

N	Total number of responses						
	GMLE(0.5)	GMLE(1.594)	EGMLE(0.5)	EGMLE(1.0)	UPE-O	UPE-M	EZB
5000	9693 S	18690 S	3818 S	4993 S	21823 L	16705 L	65124 S
10000	9606 S	18223 S	3791 S	4998 S	27667 L	15882 L	130247 S
20000	9385 S	17735 S	3847 S	5027 S	38935 L	16471 L	260495 S

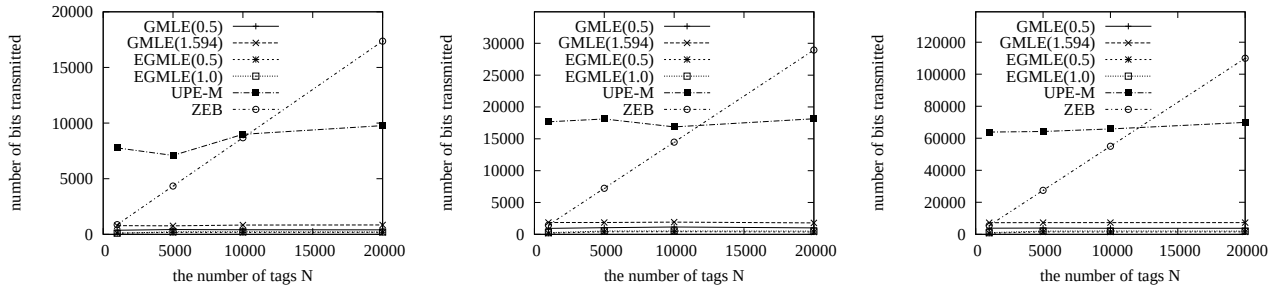

Figure 5-6. Numbers of bits transmitted when $\alpha = 90\%$, $\beta = 9\%, 6\%$ and 3%.

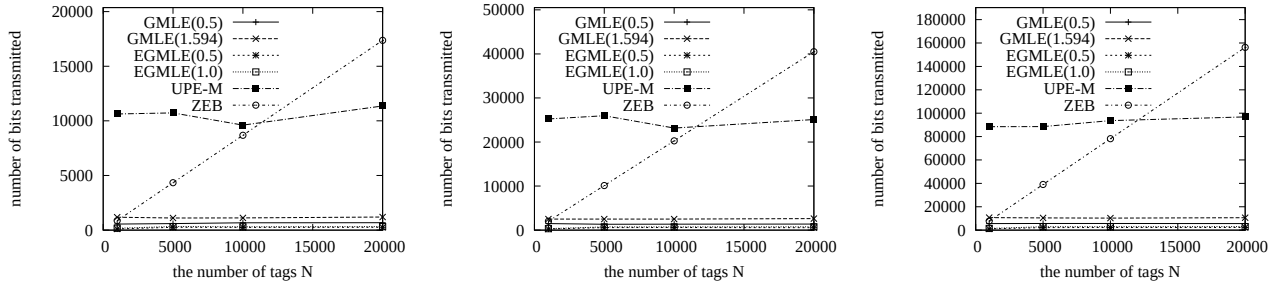

Figure 5-7. Numbers of bits transmitted when $\alpha = 95\%$, $\beta = 9\%, 6\%$ and 3%.

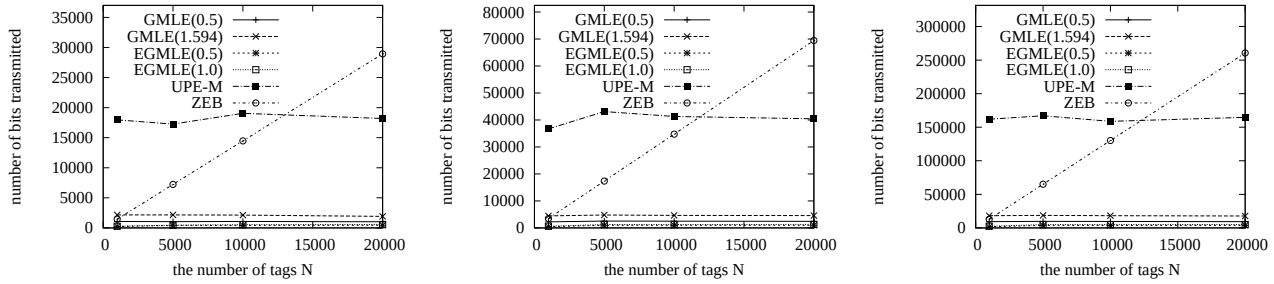

Figure 5-8. Numbers of bits transmitted when $\alpha = 99\%$, $\beta = 9\%, 6\%$ and 3%.

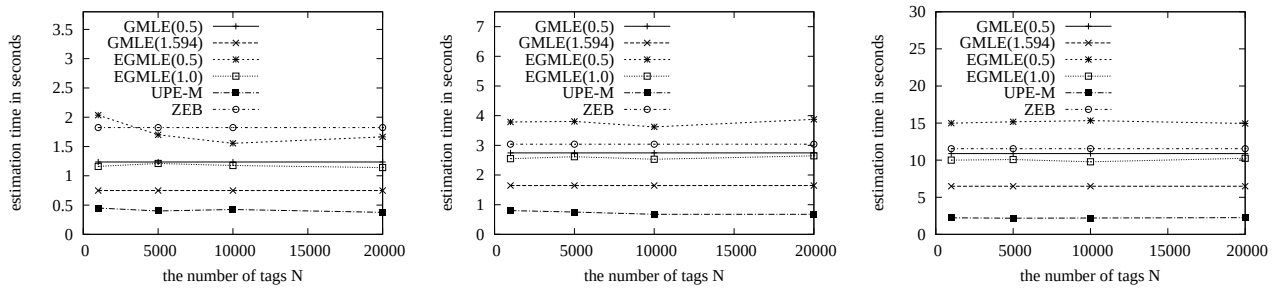

Figure 5-9. Estimation times of the algorithms when $\alpha = 90\%, \beta = 9\%, 6\%$ and 3%.

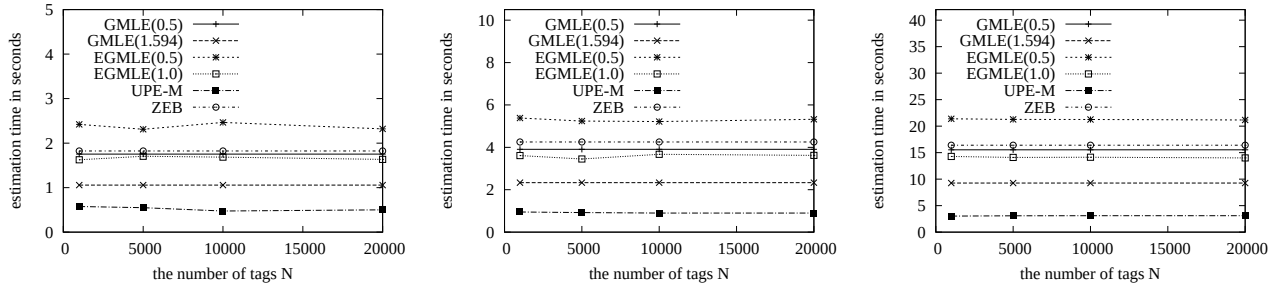

Figure 5-10. Estimation times of the algorithms when $\alpha = 95\%, \beta = 9\%, 6\%$ and 3%.

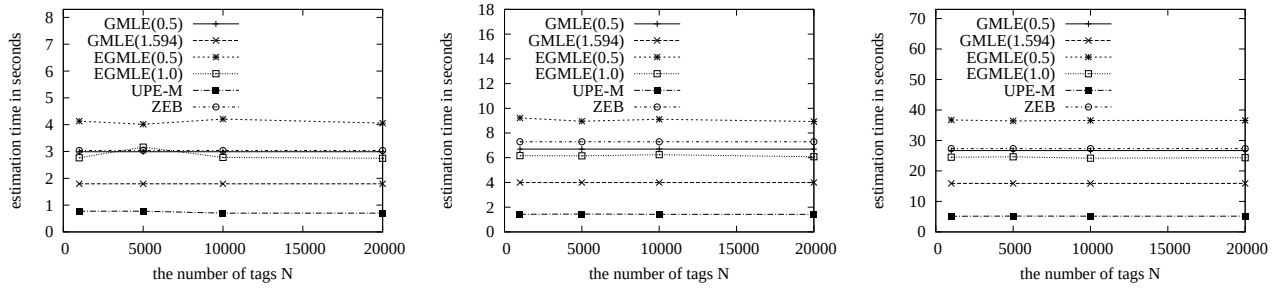

Figure 5-11. Estimation times of the algorithms when $\alpha = 99\%, \beta = 9\%, 6\%$ and 3%.

CHAPTER 6
CONCLUSIONS

In this dissertation, we first develop a fast and compact per-flow traffic measurement approach through randomized counter sharing. The approach employs a novel data encoding/decoding scheme, which mixes per-flow information randomly in a tight SRAM space for compactness.

We then focus on the scan detection problem in high-speed networks, which is another important research topic of online network measurement. We optimally combine probabilistic sampling, bit-sharing storage, and maximum likelihood estimation to achieve an efficient scan detection scheme.

Thirdly, we propose a new method for OD flow measurement which employs the bitmap data structure for packet information storage and uses statistical inference approach to compute the measurement results. Our method is able to achieve smaller per-packet processing overhead and much more accurate results, when comparing with the best existing approach.

Finally, we design two probabilistic algorithms for estimating the number of RFID tags in a region. We believe the algorithms are the first of its kind that targets at prolonging the lifetime of the active tags. Their energy cost is far less than the state-of-the-art algorithms in the related work. Moreover, we reveal a fundamental tradeoff between the energy cost and the estimation time. By tuning a system parameter, the algorithms can trade longer estimation time for less energy cost, or vice versa.

REFERENCES

[1] "Abilene Update." *http://www.internet2.edu/presentations/spring03/20030410-Abilene-Corbato.pdf* (2003).

[2] "EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz Version 1.0.9." *http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_0_9-standard-20050126.pdf* (2005).

[3] "Abilene Network." *http://en.wikipedia.org/wiki/Abilene_Network* (2011).

[4] Abramowitz, M. and Stegun, I. "Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables." *Dover Publications* (1964).

[5] Bandi, N., Agrawal, D., and Abbadi, A. "Fast Algorithms for Heavy Distinct Hitters using Associative Memories." *Proc. of IEEE International Conference on Distributed Computing Systems(ICDCS)* (2007).

[6] Basu, A., Buch, V., Vogels, W., and von Eicken, T. "U-Net: a user-level network interface for parallel and distributed computing ." *Proc. of ACM SOSP* (1995): 40–53.

[7] Bloom, B. H. "Space/Time Trade-offs in Hash Coding with Allowable Errors." *Communications of the ACM* 13 (1970).7: 422–426.

[8] Broder, A. and Mitzenmacher, M. "Network Applications of Bloom Filters: A Survey." *Internet Mathematics* 1 (2002).4: 485–509.

[9] Bryc, W. "The normal distribution: characterizations with applications." *Springer-Verlag* (1995).

[10] CAIDA. "Analyzing UDP Usage in Internet Traffic." *http://www.caida.org/research/traffic-analysis/tcpudpratio/* (2009).

[11] Cao, J., Chen, A., and Bu, T. "A Quasi-Likelihood Approach for Accurate Traffic Matrix Estimation in a High Speed Network." *Proc. of IEEE INFOCOM* (2008).

[12] Cao, J., Davis, D., Wiel, S. V., and Yu, B. "Time-varying network tomography." *J. Amer. Statist. Assoc* (2000).

[13] Cao, J., Jin, Y., Chen, A., Bu, T., and Zhang, Z. "Identifying High Cardinality Internet Hosts." *Proc. of IEEE INFOCOM* (2009).

[14] Casella, G. and Berger, R. "Statistical Inference." *Duxbury Press* (2001).

[15] Casella, G. and Berger, R. L. "Statistical Inference." *2nd edition, Duxbury Press* (2002).

[16] Cha, J. and Kim, J. “Novel Anti-collision Algorithms for Fast Object Identification in RFID System.” *Proc. IEEE ICPADS* (2005).

[17] Charikar, M., Chen, K., and Farach-Colton, M. “Finding Frequent Items in Data Streams.” *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP)* (2002).

[18] Chen, S., Deng, Y., Attie, P., and Sun, W. “Optimal Deadlock Detection in Distributed Systems based on Locally Constructed Wait-for Graphs.” (1996): 613–619.

[19] Chen, S., Fang, Y., and Xia, Y. “Lexicographic Maxmin Fairness for Data Collection in Wireless Sensor Networks.” *IEEE Transactions on Mobile Computing* 6 (2007).7: 762–776.

[20] Chen, S. and Nahrstedt, K. “Maxmin Fair Routing in Connection-oriented Networks.” *Proc. Euro-Parallel and Distributed Systems Conf* (1998): 163–168.

[21] Chen, S. and Shavitt, Y. “SoMR: A Scalable Distributed QoS Multicast Routing Protocol.” *Journal of Parallel and Distributed Computing* 68 (2008).2: 137–149.

[22] Chen, S., Song, M., and Sahni, S. “Two Techniques for Fast Computation of Constrained Shortest Paths.” *IEEE/ACM Transactions on Networking* 16 (2008).1: 105–115.

[23] Chen, S., Tang, Y., and Du, W. “Stateful DDoS Attacks and Targeted Filtering.” *Journal of network and computer applications* 30 (2007).3: 823–840.

[24] Cheswick, W. and Bellovin, S. “Firewalls and Internet Security: Repelling the Wily Hacker.” *Addison-Wesley* (1994).

[25] Choi, H., Cha, J., and Kim, J. “Fast Wireless Anti-collision Algorithm in Ubiquitous ID System.” *Proc. IEEE VTC* (2004).

[26] Coates, M., Hero, A., Nowak, R., and Yu, B. “Internet tomography.” *IEEE Signal Processing Magazine* (2002).

[27] Cohen, S. and Matias, Y. “Spectral Bloom Filters.” *Proc. of ACM SIGMOD* (2003).

[28] Considine, J., Li, F., Kollios, G., and Byers, J. “Approximate aggregation techniques for sensor databases.” *In The 20th International Conference on Data Engineering (ICDE)* (2004).

[29] Cormode, G. and Muthukrishnan, S. “Space Efficient Mining of Multigraph Streams.” *Proc. of ACM PODS* (2005).

[30] Cvetkovski, A. “An algorithm for approximate counting using limited memory resources.” *Proc. of ACM SIGMETRICS* (2007).

[31] Das, R. "Global RFID Market Tops $5.5 Billion." *http://www.convertingmagazine.com/article/CA 6653688.html* (2009).

[32] Deal, R. "Cisco Router Firewall Security." *Cisco Press, ISBN-10: 1-58705-175-3* (2004).

[33] Demaine, E., Lopez-Ortiz, A., and Ian-Munro, J. "Frequency Estimation of Internet Pacet Streams with Limited Space." *Proc. of Annual European Symposium on Algorithms (ESA)* (2002).

[34] Demaine, E., Lopez-Ortiz, A., and Munro, J. "Frequency Estimation of Internet Packet Streams with Limited Space." *Proc. of 10th ESA Annual European Symposium on Algorithms* (2002).

[35] Dimitropoulos, X., Hurley, P., and Kind, A. "Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy Hitters." *ACM SIGCOMM Computer Communication Review* 38 (2008).1: 7–16.

[36] Duffield, N., Lund, C., and Thorup, M. "Estimating Flow Distributions from Sampled Flow Statistics." *Proc. of ACM SIGCOMM* (2003).

[37] Duffield, N. G. and Grossglauser, M. " Trajectory sampling for direct traffic observation." *Proc. of ACM SIGCOMM* (2000).

[38] Durand, M. and Flajolet, P. "LogLog Counting of Large Cardinalities." *Proc. of European Symposium on Algorithms* (2003).

[39] Erramilli, V., Crovella, M., and Taft, N. "An independent-connection model for traffic matrices." *Proc. of Internet Measurement Conference (IMC)* (2006).

[40] Estan, C. and Varghese, G. "New Directions in Traffic Measurement and Accounting." *Proc. of ACM SIGCOMM* (2002).

[41] Estan, C., Varghese, G., and Fish, M. "Bitmap Algorithms for Counting Active Flows on High-Speed Links." *IEEE/ACM Transactions on Networking (TON)* 14 (2006).5: 925–937.

[42] Feldmann, A., Greenberg, A. G., Lund, C., Reingold, N., Rexford, J., and True, F. "Deriving traffic demands for operational IP networks: methodology and experience." *Proc. of ACM SIGCOMM* (2000).

[43] Flajolet, G. "Probabilistic counting." *Proc. of Symp. on Fundations of Computer Science (FOCS)* (1983).

[44] Floerkemeier, C. "Transmission Control Scheme for Fast RFID Object Idenfication." *Proc. of PerCom Workshops on Pervasive Wireless Networking* (2006).

[45] Floerkemeier, C. and Wille, M. "Comparison of Transmission Schemes for Frame ALOHA based RFID Protocols." *Proc. of SAINT Workshops on RFID and Extended Network Deployment of Technologies and Applications* (2006).

[46] Fortz, B. and Thorup, M. "Optimizing OSPF/IS-IS weights in a changing world." *IEEE JSAC Special Issue on Advances in Fundamentals of Network Management* (2002).

[47] Gardner, W. David. "Researchers Transmit Optical Data At 16.4 Tbps." *InformationWeek* (2008).

[48] Gibbons, P. and Matias, Y. "New Sampling-based Summary Statistics for Improving Approximate Query Answers." *Proc. of ACM SIGMOD* (1998).

[49] Han, H., Sheng, B., Tan, C., Li, Q., Mao, W., and Lu, S. "Counting RFID Tags Efficiently and Anonymously." *Proc. of IEEE Infocom* (2010).

[50] Hao, F., Kodialam, M., and Lakshman, T. V. "ACCEL-RATE: A Faster Mechanism for Memory Efficient Per-flow Traffic Estimation." *Proc. of ACM SIGMETRICS/Performance* (2004).

[51] Hermsmeyer, C., Song, H., Gemelli, R., and Bunse, S. "Towards 100G Packet Processing: Challenges and Technologies." *Bell Labs Technical Journal* 14 (2009).2: 57–80.

[52] Hollinger, R. and Davis, J. "National Retail Security Survey." *http://diogenesllc.com/NRSS_2001.pdf* (2001).

[53] Hush, D. and Wood, C. "Analysis of Tree Algorithm for RFID Arbitration." *Proc. IEEE ISIT* (1998).

[54] Hwang, K., Vander-Zanden, B., and Taylor, H. "A Linear-time Probabilistic Counting Algorithm for Database Applications." *ACM Transactions on Database Systems* 15 (1990).2.

[55] Jian, Y. and Chen, S. "Can CSMA/CA Networks Be Made Fair?" (2008): 235–246.

[56] Jian, Y., Chen, S., Zhang, Z., and Zhang, L. "A Novel Scheme for Protecting Receiver's Location Privacy in Wireless Sensor Networks." *IEEE Transactions on Wireless Communications* 7 (2008).10: 3769–3779.

[57] Jung, J., Paxson, V., Berger, A., and Balakrishnan, H. "Fast Portscan Detection Using Sequential Hypothesis Testing." *Proc. of IEEE Symposium on Security and Privacy* (2004).

[58] Kamiyama, N. and Mori, T. "Simple and Accurate Identification of High-rate Flows by Packet Sampling." *Proc. of IEEE INFOCOM* (2006).

[59] Kamvar, S., Schlosser, M., and Garcia-Molina, H. "The Eigentrust algorithm for reputation management in P2P networks." *Proc. of the World Wide Web Conference* (2003).

[60] Karp, R., Shenker, S., and Papadimitriou, C. "A Simple Algorithm for Finding Frequent Elements in Streams and Bags." *ACM Transactions on Database Systems* 28 (2003).1: 51–55.

[61] Klair, D., Chin, K., and Raad, R. "On the Energy Consumption of Pure and Slotted Aloha based RFID Anti-Collision Protocols." *Computer Communications* (2008).

[62] Kodialam, M., Lakshman, T. V., and Mohanty, S. "Runs bAsed Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation." *Proc. of INFOCOM* (2004).

[63] Kodialam, M. and Nandagopal, T. "Fast and Reliable Estimation Schemes in RFID Systems." *Proc. ACM MOBICOM, Los Angeles* (2006).

[64] Kodialam, M., Nandagopal, T., and Lau, W. "Anonymous Tracking using RFID tags." *Proc. IEEE INFOCOM* (2007).

[65] Kumar, A., Sung, M., Xu, J., and Wang, J. "Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution." *Proc. of ACM SIGMETRICS* (2004).

[66] Kumar, A., Xu, J., Wang, J., Spatschek, O., and Li, L. "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement." *Proc. of IEEE INFOCOM* (2004, A journal version was published in IEEE JSAC, 24(12):2327-2339, December 2006).

[67] Lehmann, E. and Casella, G. "Theory of Point Estimation." *Springer Press* (1998).

[68] Li, T., Chen, S., and Ling, Y. "Identifying the Missing Tags in a Large RFID System." *Proc. of ACM MobiHoc* (2010).

[69] ———. "Fast and Compact Per-Flow Traffic Measurement through Randomized Counter Sharing." *Proc. of IEEE INFOCOM* (2011).

[70] Li, T., Chen, S., Luo, W., and Zhang, M. "Scan Detection in High-Speed Networks Based on Optimal Dynamic Bit Sharing." *Proc. of IEEE INFOCOM* (2011).

[71] Li, T., Chen, S., and Qiao, Y. "Origin-Destination Flow Measurement in High-Speed Networks." *Proc. of IEEE INFOCOM, mini-conference* (2011).

[72] Li, T., W.Luo, Mo, Z., and Chen, S. "Privacy-preserving RFID Authentication based on Cryptographical Encoding." *Proc. of IEEE INFOCOM* (2012).

[73] Li, T., Wu, S., Chen, S., and Yang, M. "Energy Efficient Algorithms for the RFID Estimation Problem." *Proc. of IEEE INFOCOM* (2010).

[74] Liang, G. and Yu, B. “Maximum pseudo likelihood estimation in network tomography.” *IEEE Trans. Signal Processing* 51 (2003).2043C2053.

[75] Lu, Y., Montanari, A., Prabhakar, B., Dharmapurikar, S., and Kabbani, A. “Counter Braids: A Novel Counter Architecture for Per-Flow Measurement.” *Proc. of ACM SIGMETRICS* (2008).

[76] Lu, Y. and Prabhakar, B. “Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture.” *Proc. of IEEE INFOCOM* (2009).

[77] Lui, K.S., Nahrstedt, K., and Chen, S. “Hierarchical QoS Routing in Delay-bandwidth Sensitive Networks.” (2000): 579–588.

[78] Luo, W., Chen, S., Li, T., and Chen, S. “Efficient Missing Tag Detection in RFID Systems.” *Proc. of IEEE INFOCOM, mini-conference* (2011).

[79] Luo, W., Chen, S., Li, T., and Qiao, Y. “Probabilistic Missing-tag Detection and Energy-Time Tradeoff in Large-scale RFID Systems.” *Proc. of ACM MobiHoc* (2012).

[80] M. Roughan, M. Thorup and Zhang, Y. “Traffic engineering with estimated traffic matrices.” *Proc. of Internet Measurement Conference (IMC)* (2003).

[81] Manku, G. and Motwani, R. “Approximate Frequency Counts over Data Streams.” *Proc. of VLDB* (2002).

[82] Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., and Diot, C. “Traffic matrix estimation: Existing techniques and new directions.” *Proc. of ACM SIGCOMM* (2002).

[83] Myung, J. and Lee, W. “An adaptive memoryless tag anti-collision protocol for RFID networks.” *Proc. IEEE ICC* (2005).

[84] Nahrstedt, K. and Chen, S. “Coexistence of QoS and Best-effort Flows-routing and Scheduling.” (1998).

[85] Namboodiri, V. and Gao, L. “Energy-Aware Tag Anti-Collision Protocols for RFID Systems.” *Proc. of IEEE PerCom* (2007).

[86] Newey, W. and McFadden, D. “Large Sample Estimation and Hypothesis Testing.” *Dan. Handbook of Econometrics* 4 (1994): 2111–2245.

[87] Ni, L., Liu, Y., and Lau, Y. C. “Landmarc: Indoor Location Sensing using Active RFID.” *Proc. IEEE PerCom* (2003).

[88] Nucci, A., Cruz, R., Taft, N., and Diot, C. “Design of igp link weight changes for estimation of traffic matrices.” *Proc. of IEEE INFOCOM* (2004).

[89] of Standards, National Institute and Technology. “FIPS 180-1: Secure Hash Standard.” *http://csrc.nist.gov* (1995).

[90] Pan, L. and Wu, H. "Smart Trend-Traversal: A Low Delay and Energy Tag Arbitration Protocol for Large RFID Systems." *Proc. of IEEE Infocom* (2009).

[91] Pearson, M. "QDRTM-III: Next Generation SRAM for Networking." *http://www.qdrconsortium.org/presentation/QDR-III-SRAM.pdf* (2009).

[92] Plonka, D. "FlowScan: A Network Traffic Flow Reporting and Visualization Tool." *Proc. of USENIX LISA* (2000).

[93] Qian, C., Ngan, H., and Liu, Y. "Cardinality Estimation for Large-scale RFID Systems." *Proc. IEEE PerCom* (2008).

[94] Qiao, Y., Chen, S., Li, T., and Chen, S. "Energy-Efficient Polling Protocols in RFID Systems." *Proc. of ACM MobiHoc* (2011).

[95] Qiao, Y., Li, T., and Chen, S. "One Memory Access Bloom Filters and Their Generalization." *Proc. of IEEE INFOCOM* (2011).

[96] Ramabhadran, S. and Varghese, G. "Efficient Implementation of a Statistics Counter Architecture." *Proc. ACM SIGMETRICS* (2003).

[97] Ramakrishna, M., Fu, E., and Bahcekapili, E. "Efficient Hardware Hashing Functions for High Performance Computers." *IEEE Transactions on Computers* 46 (1997).12: 1378–1381.

[98] Rincon, D., Roughan, M., and Willinger, W. "Towards a Meaningful MRA of Traffic Matrices." *Proc. of ACM SIGCOMM IMC* (2008).

[99] Ringberg, H., Soule, A., Rexford, J., and Diot, C. "Sensitivity of PCA for traffic anomaly detection." *Proc. of ACM SIGMETRICS* (2007).

[100] Roughan, M., Greenberg, A., Kalmanek, C., Rumsewicz, M., Yates, J., and Zhang, Y. "Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning." *Porc. of ACM SIGCOMM Internet Measurement Workshop* (2002).

[101] Semiconductors, Philips. "I-CODE Smart Label RFID Tags." *http://www.nxp.com/acrobat_download/other/identification/SL092030.pdf* (2004).

[102] Shah, D., Iyer, S., Prabhakar, B., and McKeown, N. "Maintaining Statistics Counters in Router Line Cards." *Proc. of IEEE Micro* 22 (2002).1: 76–81.

[103] Shi, Y. and Hou, Y. Thomas. "Theoretical results on base station movement problem for sensor network." (2008).

[104] Song, H., Hao, F., Kodialam, M., and Lakshman, T. "IPv6 Lookups Using Distributed and Load Balanced Bloom Filters for 100Gbps Core Router Line Cards." *Proc. of INFOCOM* (2009).

[105] 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." *Proc. of ACM Sigmetrics* (2004).

[106] Staniford, S., Hoagland, J., and McAlerney, J. "Practical Automated Detection of Stealthy Portscans." *Journal of Computer Security* 10 (2002): 105 – 136.

[107] Stanojevic, R. "Small active counters." *Proc. of IEEE INFOCOM* (2007).

[108] Tan, Chiu C., Sheng, Bo, and Li, Qun. "How to Monitor for Missing RFID Tags." *Proc. IEEE ICDCS* (2008).

[109] Tang, Y., Chen, S., and Ling, Y. "State Aggregation of Large Network Domains." *Computer communications* 30 (2007).4: 873–885.

[110] Venkatataman, S., Song, D., Gibbons, P., and Blum, A. "New Streaming Algorithms for Fast Detection of Superspreaders." *Proc. of NDSS* (2005).

[111] Vogt, H. "Efficient Object Identification with Passive RFID Tags." *Proc. IEEE PerCom* (2002).

[112] Want, R. "An Introduction to RFID Technology." *Proc. IEEE PerCom* (2006).

[113] Whang, K., Vander-Zanden, B., and Taylor, H. "A Linear Time Probabilistic Counting Algorithm for Database Applications." *ACM Transactions on Database Systems* (1990).

[114] Yoon, M., Li, T., Chen, S., and Peir, J. "Fit a Spread Estimator in Small Memory." *Proc. of IEEE INFOCOM* (2009).

[115] ———. "Fit a Compact Spread Estimator in Small High-Speed Memory." *IEEE/ACM Transactions on Networking* 19 (2011).5.

[116] Zecca, G., Couderc, P., Banatre, M., and Beraldi, R. "Swarm Robot Synchronization Using RFID Tags." *Proc. of IEEE PERCOM* (2009).

[117] Zhai, J. and Wang, G. N. "An Anti-Collision Algorithm Using Two-functioned Estimation for RFID Tags." *Proc. ICCSA* (2005).

[118] Zhang, M., Li, T., Chen, S., and Li, B. "Using Analog Network Coding to Improve the RFID Reading Throughput." *Proc. of IEEE ICDCS* (2010).

[119] Zhang, Y., Roughan, M., Duffield, N., and Greenberg, A. "Fast accurate computation of large-scale ip traffic matrices from link loads." *Proc. of ACM SIGMETRICS* (2003).

[120] Zhang, Y., Roughan, M., Lund, C., and Donoho, D. "An informationtheoretic approach to traffic matrix estimation." *Proc. of ACM SIGCOMM* (2003).

[121] ———. “Estimating Point-to-Point and Point-to-Multipoint Traffic Matrices: An Information-Theoretic Approach.” *IEEE/ACM Transactions on Networking* 10 (2005).10.

[122] Zhang, Y., Roughan, M., Willinger, W., and Qiu, L. “Spatio-Temporal Compressive Sensing and Internet Traffic Matrices.” *Proc. of ACM SIGCOMM* (2009).

[123] Zhang, Y., Singh, S., Sen, S., Duffield, N., and Lund, C. “Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Application.” *Proc. of ACM SIGCOMM IMC* (2004).

[124] Zhang, Yin. “6 months of Abilene traffic matrices.” *http://www.cs.utexas.edu/ yzhang/research/AbileneTM/* (2004).

[125] Zhang, Z., Chen, S., Ling, Y., and Chow, R. “Capacity-aware Multicast Algorithms on Heterogeneous Overlay Networks.” *IEEE Transactions on Parallel and Distributed Systems* 17 (2006).2: 135–147.

[126] Zhang, Z., Chen, S., and Yoon, M. “MARCH: A Distributed Incentive Scheme for Peer-to-peer Networks.” (2007): 1091–1099.

[127] Zhao, H., Wang, H., Lin, B., and Xu, J. “Design and performance analysis of a DRAM-based statistics counter array architecture.” *Proc. of ACM/IEEE ANCS* (2009).

[128] Zhao, Q., Kumar, A., and Xu, J. “Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations.” *Proc. of USENIX/ACM Internet Measurement Conference* (2005).

[129] Zhao, Q., Xu, J., and Kumar, A. “Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation.” *IEEE Journal on Selected Areas in Communications (JASC)* 24 (2006).10: 1840–1852.

[130] Zhao, Q., Xu, J., and Liu, Z. “Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency.” *Proc. of ACM Sigmetrics/Performance* (2006).

BIOGRAPHICAL SKETCH

Tao Li was born in Fuyang, Anhui, China, in 1984. He received his B.E. degree in computer science and engineering from University of Science and Technology of China in 2007. After that, he joined the Department of Computer and Information Science and Engineering at the University of Florida to pursue his Ph.D. degree under the supervision of Dr. Shigang Chen. His research interests include Internet Traffic Measurement and RFID technologies.