STATISTICAL AND LEARNING ALGORITHMS FOR THE DESIGN, ANALYSIS, MEASUREMENT, AND MODELING OF NETWORKING AND SECURITY SYSTEMS

By

Muhammad Shahzad

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2015

ABSTRACT

STATISTICAL AND LEARNING ALGORITHMS FOR THE DESIGN, ANALYSIS, MEASUREMENT, AND MODELING OF NETWORKING AND SECURITY SYSTEMS

By

Muhammad Shahzad

The goal of this thesis is to develop statistical and learning algorithms for the design, analysis, measurement, and modeling of networking and security systems with specific focus on RFID systems, network performance metrics, user security, and software security. Next, I give a brief overview of these four areas of focus.

Radio frequency identification (RFID) systems are widely used in supply chain and inventory management. Existing RFID systems are primarily used to identify the RFID tags present in a tag population. While identifying individual tags is a useful operation, it is usually very time consuming and is not always desired or required. For example, if the objective is to determine whether any of the tags are missing (*e.g.*, to detect a theft), then first identifying all tags and then determining if any tags are missing is a very slow process. In this thesis, I present novel statistical algorithms to enable new applications in RFID systems, such as counting the number of tags in a population and detecting missing tags, while using existing infrastructure of RFID systems that is already deployed in industry.

With the growth in number and significance of the emerging applications that require extremely low latencies, network operators are facing increasing need to perform latency measurement on per-flow basis between any two observation points for network monitoring and troubleshooting. Per-flow latency measurement can be used reactively by network operators to perform tasks such as detecting and localizing delay spikes in a network, isolating offending flows that are responsible for causing delay bursts, and rerouting them through other paths. It can also be used proactively by network operators to monitor latencies between observation points for locating bottleneck links and replacing them with higher capacity links. In this thesis, I present a novel per-flow latency measurement scheme that requires no probe packets and time stamping.

With the rich functionalities and enhanced computing capabilities available on mobile computing devices with touch screens, users not only store sensitive information (such as credit card numbers) but also use privacy sensitive applications (such as online banking) on these devices, which make them hot targets for hackers and thieves. In this thesis, I present a gesture based user authentication scheme for the secure unlocking of touch screen devices. Unlike existing authentication schemes for touch screen devices, which use *what* user inputs as the authentication secret, our scheme authenticates users mainly based on *how* they input. Even if attackers see what gesture a user performs, they cannot reproduce the behavior of the user doing gestures through shoulder surfing or smudge attacks.

Software systems inherently contain vulnerabilities that have been exploited in the past resulting in significant revenue losses. The study of vulnerability life cycles can help in the development, deployment, and maintenance of software systems. It can also help in designing future security policies and conducting audits of past incidents. In this thesis, I present an exploratory measurement study of a large software vulnerability data set containing 46310 vulnerabilities disclosed since 1988 till 2011. Our exploratory analysis uncovers several statistically significant findings along several dimensions including phases in the life cycle of vulnerabilities and evolution of vulnerabilities over the years. These findings have important implications for software development and deployment.

To my parents and my beautiful wife,
for their support and encouragement.

ACKNOWLEDGEMENTS

Working towards a Ph.D. has been a deeply enriching and rewarding experience. Looking back, many people have helped shape my journey. I would like to extend them my thanks.

- First and foremost, my advisor, Prof. Alex X. Liu. My work would not have been possible without his constant guidance, his unwavering encouragement, his many insights, and his exceptional resourcefulness. And most importantly, his friendship. I have been very fortunate to have an advisor who has also been a close friend. For all of this, Alex, thank you.
- I would also like to thank the rest of my thesis committee Profs. Eric Torng, Guoliang Xing, and Subir Biswas for their encouragement and insightful comments during my qualifier and comprehensive exams.
- I would also like to thank Dr. Arjmand Samuel. I learnt a lot from him during my summer internships at Microsoft Research. My collaboration with him has been one of the most fruitful and fun engagements I have experienced.
- I would also like to thank Drs. Henrik Lundgren and Ioannis Pefkianakis. I really enjoyed working with them during my summer internship at Technicolor Research.
- Throughout my Ph.D., I was supported by various NSF research grants. Thanks NSF!
- I would also like to thank Michigan State University, and specifically Department of Computer Science and Engineering for providing me financial support to attend various conferences during my Ph.D.
- Many thanks to my colleagues in Systems and Security Lab at Michigan State University. In particular, I would like to thank Muhammad Zubair Shafiq, Momina Tabish, Kamran Ali, Jamal Afridi, Ann Wang, Ali Munir, Faraz Ahmed, and Fei Chen for numerous insightful discussions and collaborations on various projects.
- I must say that I owe my great time in Michigan State University to all of my fabulous friends. It is simply not feasible to list all of them here. I would like to thank them all for their friendship and support.

- I am also very thankful to Drs. Syed Ali Khayam and Muddassar Farooq, who advised me before my Pn.D. and encouraged me to pursue Ph.D.
- Finally, I do not know how I can thank my family enough: my wife and my parents, from whom I realized that kindness and devotion is endless, and my beautiful daughter who has been the best and persistent source of happiness and joy for me ever since she came into my world!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

In this thesis I present my work on measurement, modeling, design, and analysis of networking and security systems. For networking, I present my work on probabilistic network measurements in both wireless as well as wired networks. For wireless networks, I focus on the modeling, design, and analysis of probabilistic measurement schemes for radio frequency identification (RFID) systems. More specifically, I present my work on designing statistical algorithms for estimating the number of tags in a population of RFID tags, for optimizing the standardized RFID identification protocol, and for detecting missing tags from a population of RFID tags. The key distinction of my work compared to prior art is that my schemes are compliant with the EPCGlobal Class 1 Generation 2 (C1G2) RFID standard. It is critical for RFID schemes to be compliant with the C1G2 standard because the commercially available off-the-shelf RFID equipment follows the C1G2 standard. A scheme that does not comply with the C1G2 standard cannot be deployed on the existing installations of RFID systems because it requires custom hardware, which costs a lot. For wired networks, I focus on the modeling, design, and analysis of probabilistic schemes for measuring fundamental network performance metrics such as latency. More specifically, I present my work on designing statistical algorithms to measure latency of any given flow between any pair of observation points in a given network. The key distinction of my work compared to prior art is that my schemes do not use probe packets, which change the behavior of network traffic and thus skew the measurement results. For security, I present my work on the design of user security systems and the measurement of software security. For user security systems, I focus on designing learning algorithms for user authentication schemes for smart phones. For software security, I focus on characterizing trends in life cycles of software vulnerabilities by studying large vulnerability databases.

1.1 Contributions

This thesis takes an in-depth look at the following research problems.

1.1.1 RFID Estimation [103, 106]

We address the fundamental problem of estimating RFID tag population size, which is needed in many applications such as tag identification, warehouse monitoring, and privacy sensitive RFID systems. We propose a new scheme for estimating tag population size called *Average Run based Tag estimation* (ART). The technique is based on the average run-length of 1s in the bit string received using the standardized framed slotted Aloha protocol. ART is significantly faster than prior schemes. For example, given a required confidence interval of 0.1% and a required reliability of 99.9%, ART is consistently 7 times faster than the fastest existing schemes (UPE and EZB) for any tag population size. Furthermore, ART's estimation time is provably independent of the tag population sizes. ART works with multiple readers with overlapping regions and can estimate sizes of arbitrarily large tag populations. ART is easy to deploy because it neither requires modification to tags nor to the communication protocol between tags and readers. ART only needs to be implemented on readers as a software module.

1.1.2 RFID Identification [104, 108]

Identifying RFID tags in a given tag population is the most fundamental operation in RFID systems. While the Tree Walking (TW) protocol has become the industrial standard for identifying RFID tags, little is known about the mathematical nature of this protocol and only some ad-hoc heuristics exist for optimizing it. In this thesis, first, we analytically model the TW protocol, and then using that model, propose the Tree Hopping (TH) protocol that optimizes TW both theoretically and practically. The key novelty of TH is to formulate tag identification as an optimization problem and find the optimal solution that ensures the minimal average number of queries or identification time as per the requirement. With this solid theoretical underpinning, for different tag population sizes ranging from 100 to 100K tags, TH significantly outperforms the *best* prior tag identification protocols on the metrics of the total number of queries per tag, the total identification time per tag, and the average number of responses per tag by an average of 40%, 59%, and 67%, respectively, when tag IDs are non-uniformly distributed in the ID space, and of 50%, 10%, and 30%, respectively, when tag IDs are uniformly distributed.

1.1.3 RFID Missing Tags [109]

RFID systems have been deployed to detect missing products by affixing them with cheap passive RFID tags and monitoring them with RFID readers. Existing missing tag detection protocols require the tag population to contain only those tags whose IDs are already known to the reader. However, in reality, tag populations often contain tags with unknown IDs, called unexpected tags, and cause unexpected *false positives* i.e., due to them, missing tags are detected as present. We take the first step towards addressing the problem of detecting the missing tags from a population that contains unexpected tags. Our protocol, RUN, mitigates the adverse effects of unexpected false positives by executing multiple frames with different seeds. It minimizes the missing tag detection time by first estimating the number of unexpected tags and then using it along with the false positive probability to obtain optimal frame sizes and number of times Aloha frames should be executed. RUN works with multiple readers with overlapping regions. It is easy to deploy because it is implemented on readers as a software module and does not require modifications to tags or to the communication protocol between tags and readers. We implemented RUN along with four major missing tag detection protocols and the fastest tag ID collection protocol and compared them side-by-side. Our experimental results show that RUN always achieves the required reliability whereas the best existing protocol achieves a maximum reliability of only 67%.

1.1.4 Per-flow Latency Measurement [107]

With the growth in number and significance of the emerging applications that require extremely low latencies, network operators are facing increasing need to perform latency measurement on per-flow basis for network monitoring and troubleshooting. In this thesis, we propose COLATE, the first per-flow latency measurement scheme that requires no probe packets and time stamping. Given a set of observation points, COLATE records packet timing information at each point so that later for any two points, it can accurately estimate the average and standard deviation of the latencies experienced by the packets of any flow in passing the two points. The key idea is that when recording packet timing information, COLATE purposely allows noise to be introduced for minimizing storage space, and when querying the latency of a target flow, COLATE uses statistical techniques to denoise and obtain an accurate latency estimate. COLATE is designed to be efficiently implementable on network middleboxes. In terms of processing overhead, COLATE performs only one hash and one memory update per packet. In terms of storage space, COLATE uses less than 0.1 bit per packet, which means that, on a backbone link with about half a million packets per second, using a 256GB drive, COLATE can accumulate time stamps of packets traversing the link for over 1.5 years. We evaluated COLATE using three real traffic traces that include a backbone traffic trace, an enterprise network traffic trace, and a data center traffic trace. Results show that COLATE always achieves the required reliability for any given confidence interval.

1.1.5 User Security [110]

With the rich functionalities and enhanced computing capabilities available on mobile computing devices with touch screens, users not only store sensitive information (such as credit card numbers) but also use privacy sensitive applications (such as online banking) on these devices, which make them hot targets for hackers and thieves. To protect private information, such devices typically lock themselves after a few minutes of inactivity and prompt a password/PIN/pattern screen when reactivated. Passwords/PINs/patterns based schemes are inherently vulnerable to shoulder surfing attacks and smudge attacks. Furthermore, passwords/PINs/patterns are inconvenient for users to enter frequently. In this thesis, we propose GEAT, a gesture based user authentication scheme for the secure unlocking of touch screen devices. Unlike existing authentication schemes for touch screen devices, which use *what* user inputs as the authentication secret, GEAT authenticates users mainly based on *how* they input, using distinguishing features such as finger velocity, device acceleration, and stroke time. Even if attackers see what gesture a user performs, they cannot reproduce the behavior of the user doing gestures through shoulder surfing or smudge attacks. We implemented GEAT on Samsung Focus running Windows, collected 15009 gesture samples from 50 volunteers, and conducted real-world experiments to evaluate GEAT's performance. Experimental results show that our scheme achieves an average equal error rate of 0.5% with 3 gestures using only 25 training samples.

1.1.6 Software Security [111]

Software systems inherently contain vulnerabilities that have been exploited in the past resulting in significant revenue losses. The study of vulnerability life cycles can help in the development, deployment, and maintenance of software systems. It can also help in designing future security policies and conducting audits of past incidents. Furthermore, such an analysis can help customers to assess the security risks associated with software products of different vendors. In this thesis, we present an exploratory measurement study of a large software vulnerability data set containing 46310 vulnerabilities disclosed since 1988 till 2011. We investigate vulnerabilities along following seven dimensions: (1) phases in the life cycle of vulnerabilities, (2) evolution of vulnerabilities over the years, (3) functionality of vulnerabilities, (4) access requirement for exploitation of vulnerabilities, (5) risk level of vulnerabilities, (6) software vendors, and (7) software products. Our exploratory analysis uncovers several statistically significant findings that have important implications for software development and deployment.

1.2 Published Material

The chapters of this dissertation are based in part on the following publications.

- Muhammad Shahzad and Alex X. Liu. "Expecting the Unexpected: Fast and Reliable Detection of Missing RFID Tags in the Wild", IEEE **INFOCOM**, 2015.
- Muhammad Shahzad and Alex X. Liu. "Noise Can Help: Accurate and Efficient Per-flow Latency Measurement without Packet Probing and Time Stamping", ACM **SIGMETRICS**, 2014.
- Muhammad Shahzad, Alex X. Liu, and Arjmand Samuel. "Secure Unlocking of Mobile Touch Screen Devices by Simple Gestures – You can see it but you can not do it", ACM **MobiCom**, 2013.
- Muhammad Shahzad and Alex X. Liu. "Probabilistic Optimal Tree Hopping for RFID Identification", ACM **SIGMETRICS**, 2013.
- Muhammad Shahzad and Alex X. Liu. "Every Bit Counts - Fast and Scalable RFID Estimation", ACM **MobiCom**, 2012.
- Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. "A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles", **ICSE**, 2012.
- Muhammad Shahzad and Alex Liu. "Probabilistic Optimal Tree Hopping for RFID Identification", IEEE/ACM Transactions on Networking (**ToN**), 2014.

- Muhammad Shahzad and Alex Liu. “Fast and Accurate Estimation of RFID Tags”, IEEE/ACM Transactions on Networking (**ToN**), 2013.

2 RFID Estimation

2.1 Introduction

2.1.1 Motivation and Problem Statement

RFID systems are widely used in various applications such as object tracking [85], 3D positioning [123], indoor localization [86], supply chain management [67], inventory control, and access control [46, 84] because the cost of commercial RFID tags is negligible compared to the value of the products to which they are attached (*e.g.*, as low as 5 cents per tag [93]). An RFID system consists of tags and readers. A tag is a microchip with an antenna in a compact package that has limited computing power and communication range. There are two types of tags: (1) passive tags, which are powered up by harvesting the radio frequency energy from readers (as they do not have their own power sources) and have communication range often less than 20 feet; (2) active tags, which have their own power sources and have relatively longer communication range. A reader has a dedicated power source with significant computing power. It transmits a query to a set of tags and the tags respond over a shared wireless medium.

This chapter concerns the fundamental problem of estimating the size of a given tag population. This is needed in many applications such as tag identification, privacy sensitive RFID systems, and warehouse monitoring. In tag identification protocols, which read the ID stored in each tag, population size is estimated at the start to guide the identification process [105]. For example, for tag identification protocols that are based on the framed slotted

Aloha protocol (standardized in EPCGlobal Class-1 Generation-2 (C1G2) RFID standard [55] and implemented in commercial RFID systems), tag estimation is often used to calculate the optimal frame size. In privacy sensitive RFID systems, such as those used in parks for continuously monitoring the number of visitors in different areas of a park to plan the guided trips efficiently, readers may not have the permission to identify human individuals. In warehouses with RFID-based monitoring systems, managers often need a quick estimate of the number of products left in stock for various purposes such as the detection of employee theft. Note that although tag population size can be accurately measured by tag identification, the speed will be too slow.

We formally define the tag estimation problem as: *given a tag population of unknown size t, a confidence interval $\beta \in (0, 1]$, and a required reliability $\alpha \in [0, 1)$, a set of readers needs to collaboratively compute the estimated number of tags $\tilde{t}$ so that $P\left\{|\tilde{t} - t| \leq \beta t\right\} \geq \alpha$.* When the number of readers is one, we call this problem *single-reader estimation*; otherwise, we call this problem *multi-reader estimation*. A tag estimation scheme should satisfy the following three requirements:

1. *Reliability*: The actual reliability should always be greater than or equal to the required reliability. The reliability α given as input is called the *required reliability*. The reliability that an estimation scheme achieves is called its *actual reliability*.
2. *Scalability*: The estimation time needs to be scalable to large population sizes because in many applications, the number of passive tags can be very large due to their low cost, easy disposability, and powerless operation.
3. *Deployability*: The estimation scheme needs to be compliant with the C1G2 standard and should not require any changes to tags.

2.1.2 Proposed Approach

In this chapter, we propose a new scheme called *Average Run based Tag estimation* (*ART*), which satisfies all of the above three requirements. The communication protocol used by

ART is the standardized framed slotted Aloha protocol, in which a reader first broadcasts a value f to the tags in its vicinity where f represents the number of time slots present in a forthcoming frame. Then each tag randomly picks a time slot in the frame and replies during that slot. Thus, the reader gets a binary sequence of 0s and 1s by representing a slot with no tag replies as 0 and a slot with one or more tag replies as 1. The key idea of ART is to estimate tag population size based on the average run size of 1s in the binary sequence. We show that the average run size of 1s in a frame monotonously increases with the increase in the size of tag population. Thus, average run size of 1s is an indicator of tag population size.

2.1.3 Advantages of ART over Prior Art

ART is advantageous in terms of speed and deployability. For speed, ART is faster than all prior schemes. For example, given a confidence interval of 0.1% and the required reliability of 99.9%, ART is consistently 7 times faster than the fastest existing schemes (*i.e.*, UPE [61] and EZB [62]) for any tag population size. The reason behind ART being faster than prior schemes is that the new estimator that we propose in this chapter, namely the average run size of 1s, has significantly smaller variance compared to the estimators used in prior schemes (such as the total number of 0s [61, 62] and the location of the first 1 in the binary sequence [53]), as we analytically show in Section 2.7.3. An estimator with small variance is faster because the Aloha frames need to be repeated fewer times to achieve the required reliability. Furthermore, the estimation time of ART is provably independent of tag population sizes. In contrast, as tag volume increases, the estimation time of some prior schemes (*e.g.*, FNEB [53]) increases.

For deployability, ART neither requires modification to the tags nor to the communication protocol between tags and readers. ART only needs to be implemented on the reader side as a software module without any hardware modifications. ART also does not demand any unpractical system parameters beyond the C1G2 standard. In contrast, some prior schemes require modification to tags and some demand unrealistic system parameters. For example, the scheme in [90] requires each tag to store thousands of hash functions, which is not practical to implement on passive tags and is not compliant with the C1G2 standard. As another example, the scheme in [53] uses increasingly large frame sizes as population size increases (*e.g.*, the frame size required by the scheme in [53] is greater than half of tag population size), which soon exceeds the maximum limit allowed by the C1G2 Standard.

2.2 Related Work

The first tag estimation scheme, called Unified Probabilistic Estimator (UPE), was proposed by Kodialam and Nandagopal in 2006 [61]. UPE uses the framed slotted Aloha protocol and makes estimation based on either the number of empty slots or that of collision slots in a frame. Besides this estimator having larger variance than ART, UPE requires the differentiation among empty, single, and collision slots, which takes significantly more time than differentiating between empty and non-empty slots. According to C1G2, a reader requires 300μs to detect an empty slot, 1500μs to detect a collision, and 3000μs to complete a successful read. In [62], Kodialam *et al.* proposed an improved framed slotted Aloha protocol based estimation scheme called Enhanced Zero Based (EZB) estimator, which performs estimation based on the total number of 0s in a frame. While UPE estimates population size in each round and averages the estimated sizes when all rounds are finished, EZB only records the total number of 0s in each frame and at the end of all rounds, EZB first averages the recorded values and then uses it to do estimation.

In [90], Qian *et al.* proposed an estimation scheme called Lottery Frame (LoF). Compared to UPE and EZB, LoF is faster; but, it is impractical to implement as it requires each tag to store a large number (*i.e.*, the number of bits in a tag ID times the number of frames, which can be in the scale of thousands) of unique hash functions. LoF needs to modify both tags and the communication protocol between readers and tags, which makes it non-compliant with C1G2. Han *et al.* proposed a tag estimation scheme called First Non Empty Based (FNEB) estimator, which is based on the size of the first run of 0s in a frame [53]. FNEB is based on an assumption that frame size can be arbitrarily large, which does not hold in practice. Li *et al.* proposed an estimation scheme called Maximum Likelihood Estimator (MLE) for active tags with the goal of minimizing power consumption of active tags [72]. In [101], Shah and Wong proposed a multi-reader tag estimation scheme which is based on an unrealistic assumption that any tag covered by multiple readers only replies to one reader. In [127], Zanella proposed Collision Set Estimator (CSE) that utilizes maximum likelihood estimation to estimate the number of tags in a population. CSE does not take accuracy requirements (α and β) as input and, therefore, can not achieve any arbitrary required reliability.

2.3 ART — Scheme Overview

2.3.1 Communication Protocol Overview

ART uses the framed slotted Aloha protocol specified in C1G2 as its MAC layer communication protocol. In this protocol, the reader first tells tags the frame size f and a random seed number R. Later in the chapter, we will see how a simple use of seed number R will make it straightforward to extend our estimation scheme to use multiple readers with overlapping regions. Each tag within the transmission range of the reader then uses f, R, and its ID to select a slot in the frame by evaluating a hash function $h(f, R, ID)$ whose result is in $[1, f]$ following a uniform distribution. Each tag has a counter initialized with the slot number it chose to reply. After each slot, the reader first transmits an end of slot signal and then each tag decrements its counter by one. In any given slot, all the tags whose counters are equal to 1 respond to the reader. In essence, each tag picks a random slot from 1 to f following a uniform distribution. If no tag replies in a slot, it is called an *empty slot*; if exactly one tag replies, it is called a *singleton slot*; and if two or more tags reply, it is called a *collision slot*.

2.3.2 Estimation Scheme Overview

At the end of a frame, the reader obtains a sequence of 0s and 1s by representing an empty slot with 0 and a singleton or collision slot with 1. In this binary sequence, a *run* is a subsequence where all bits in this subsequence are 0s (or 1s) but the bits before and after the subsequence are 1s (or 0s), if they exist. For example, 011100 has 3 runs: 0, 111, and 00.

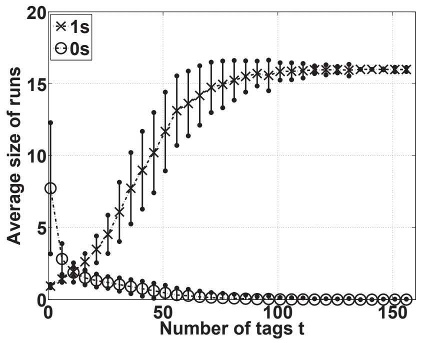


Figure 2.1 Average run size of 0s and 1s vs. tag population size t. ($f = 16$)

ART uses the average run size of 1s to estimate tag population size. The intuition is that as tag population size increases, the average run size of 1s increases (and that of 0s decreases). We illustrate this intuition using the simulation results in Figure 2.1, which shows that the average run size of 1s increases as tag population size increases from 0 to 160. The markers in this figure are the average of 100 runs. The lines above and below each marker show the standard deviation of the experiments. This figure shows that given a tag population size and a frame size, there is a distinct expected value of the average run size of 1s. The expected value of the average run size of 1s is a monotonic function of the number of tags, which means that a unique inverse of this function exists. Thus, given the observed average run size of 1s, using the inverse function, we can get the estimated value $\tilde{t}$ of tag population size t. Similar to other tag estimation schemes, ART also uses multiple frames obtained from multiple rounds of the framed slotted Aloha protocol to reduce its estimation variance and therefore increase its estimation reliability. Using different seed values for different frames, in each frame, the same tag will choose a different slot to respond.

To scale to large tag population sizes, ART uses a persistence probability p by which a tag decides whether it should reply to the reader in a given frame. The persistence probability was first introduced in [61]. To avoid making any modification to tags, this probability is implemented by "virtually" extending frame size $1/p$ times, *i.e.*, the reader announces a frame size of f/p but terminates the frame after the first f slots. According to C1G2, the reader can terminate a frame at any point. By adjusting p, ART is able to estimate tag populations of large sizes.

2.3.3 Formal Development: Overview and Assumptions

To formally develop an estimator, we first need to derive the equation for the expected value of average run size of 1s as a function of frame size f, tag population size t, and persistence probability p. We then use the inverse of this function to get the estimated value $\tilde{t}$ from the observed value of the average run size of 1s. To achieve the required reliability in minimum estimation time, we optimize f, p, and the number of rounds n so that the total number of slots $(f + l) \times n$ is minimized while satisfying $P\{|\tilde{t} - t| \leq \beta t\} \geq \alpha$. Here l is a constant that represents the C1G2 specified mandatory time delay in terms of number of empty slots between the end of a frame and the start of next frame. Typically, this delay is about $1ms$ (*i.e.*, $l \approx 3.33$ empty slots) [55, 100].

To make the formal development tractable, we assume that instead of picking a single slot to reply at the start of frame of size f, a tag independently decides to reply in each slot of the frame with probability $1/f$ regardless of its decision about previous or forthcoming slots. Vogt first used this assumption for the analysis of framed slotted Aloha protocol for RFID and justified its use by recognizing that this problem belongs to a class of problems known as "occupancy problems", which deals with the allocation of balls to urns [121]. Ever since, the use of this assumption has been a norm in the formal analysis of all Aloha based RFID protocols [121, 37, 129, 61, 62, 90, 53, 72, 101, 102].

The implication of this assumption is that when a tag independently chooses a slot to reply, it can end up choosing more than one slots in the same frame or even not choosing any at all, which is not in accordance with C1G2 standard that requires a tag to pick exactly one slot in a frame. However, even with the independence assumption, the expected number of slots that a tag chooses in a frame is still one. As we draw our estimate from a large number of frames to achieve required reliability, we can expect to observe this expected number. Therefore, the analysis with the assumption of independence is asymptotically the same as that without the independence assumption. Bordenave *et al.* further explained in detail why this independence assumption in analyzing Aloha based protocols provides results just as accurate as if all the analysis was done without this assumption [33]. Note that this independence assumption is made only to make the formal development tractable. In all the simulations we have presented in this chapter, a tag chooses exactly one slot at the start of frame.

2.4 ART — Estimation Algorithm

Next, we first focus on the single-reader version of ART. In Section 2.6.2, we will present a method to extend ART to handle multiple-readers with overlapping regions.

For ART, in each round of the Aloha protocol, we calculate the average run size of b. For example, the average run size of 1 in frame 01110011 (which has two runs of 1, *i.e.*, 111 and 11) is $(3+2)/2 = 2.5$. After n rounds, we obtain n average run sizes of b and then calculate the average of these n values. This final value is then substituted for the expected value of the average run size of b in a frame to estimate the tag population size.

The probability that a slot in a frame is b, where $b = 0$ or 1, can be calculated using Lemma 1.

Lemma 1. *Let t be the actual tag population size, f be the frame size, p be the persistence probability (i.e., the probability that a tag participates in a frame), and q_b be the probability* *that a slot in a frame is b. Thus:*

$$q_b = \begin{cases} (1-\frac{p}{f})^t & \text{if } b=0 \\ 1-(1-\frac{p}{f})^t & \text{if } b=1 \end{cases} \tag{2.1}$$

Proof. The probability that a tag chooses a given slot in a frame is p/f. The probability that it does not choose that slot is $1-\frac{p}{f}$. The probability that none of the tags choose that slot is $(1-\frac{p}{f})^t$, which is the value of q_0. As the tags choose the slots independently, q_b is the same for each slot of the frame. The probability that a slot is chosen by at least one tag is $1-q_0$, which is the value of q_1. $\square$

Let X_b be the random variable representing the average run size of b in a frame. Next, we calculate the expectation and variance of X_b. The expectation of X_b will be used to estimate the tag population size and the variance of X_b will be used to calculate the values of p, n, and f that will ensure that the actual reliability is greater than the required reliability and the estimation time is minimium. Let Y_b be the random variable representing the number of times b occurs in a frame and R_b be the random variable representing the number of runs of b in a frame. By definition, $X_b = \frac{Y_b}{R_b}$ holds for any frame. Next, we first calculate $E[Y_b]$, $\text{Var}(Y_b)$, $E[R_b]$, $\text{Var}(R_b)$, and $\text{Cov}(Y_b, R_b)$ in Lemmas 2 and 3. Then, we use them to calculate $E[X_b]$ and $\text{Var}(X_b)$ in Theorem 4. Using Equation (2.12) in Theorem 4, replacing $E[X_b]$ by the observed average run size of b from n frames, we obtain an equation with only one unknown t. Finally, we use Brent's method to obtain the numerical solution of this equation. The result is the estimated tag population size $\tilde{t}$. Since ART uses X_b to estimate the tag population size, we call X_b the *estimator* of ART.

Lemma 2. *Let Y_b be the random variable representing the number of times b occurs in a frame and R_b be the random variable representing the number of runs of b in a frame. Given tag population size t, frame size f, and persistence probability p, we have:*

$$E[Y_b] = fq_b \tag{2.2}$$

$$\mathrm{Var}(Y_b) = fq_b(1 - q_b) \tag{2.3}$$

$$E[R_b] = q_b\big(q_b + f(1 - q_b)\big) \tag{2.4}$$

$$\mathrm{Var}(R_b) = f(q_b - 4q_b^2 + 6q_b^3 - 3q_b^4) + (3q_b^2 - 8q_b^3 + 5q_b^4) \tag{2.5}$$

Proof. Each slot i of frame f has probability q_b of being b. Therefore, $Y_b \sim \mathrm{Binom}(f, q_b)$. Using general formula for expectation and variance of a binomial random variable, $E[Y_b]$ and $\mathrm{Var}(Y_b)$ are given by Equations (2.2) and (2.3).

Let $\gamma_1, \gamma_2, \ldots, \gamma_f$ represent the sequence of binary random variables representing the value of each slot in a frame of size f. Since each tag randomly and independently picks a slot in the frame, all γ_i are identically distributed. Furthermore, $P\{\gamma_i = b\} = q_b$. Let $\overline{b} = 1 - b$ and let I_i be the indicator random variable whose value is 1 if a run of b begins at γ_i.

$$I_i = \begin{cases} 1 & \text{if } (\gamma_i = b, i = 1) \vee (\gamma_i = b \wedge \gamma_{i-1} = \overline{b}, i > 1) \\ 0 & \text{otherwise} \end{cases}$$

Thus, $R_b = \sum_{i=1}^{f} I_i$. Because

$$E[I_i] = \begin{cases} P\{\gamma_i = b\} = q_b & \text{if } i = 1 \\ P\{\gamma_{i-1} = \overline{b}, \gamma_i = b\} = q_b(1 - q_b) & \text{if } i > 1 \end{cases}$$

we get

$$E[R_b] = \sum_{i=1}^{f} E[I_i] = q_b + \sum_{i=2}^{f} q_b(1 - q_b) = q_b\big(q_b + f(1 - q_b)\big)$$

As R_b is sum of f random variables, some of which are correlated, we use the general expression for variance of sum of correlated random variables to obtain the variance of R_b.

$$\mathrm{Var}(R_b) = \mathrm{Var}(\sum_{i=1}^{f} I_i) = \sum_{i=1}^{f} \mathrm{Var}(I_i) + 2\sum_{j=2}^{f}\sum_{\forall i<j} \mathrm{Cov}(I_i, I_j)$$

Here we used the fact that the frame size is always greater than 1 during the estimation process whenever the information about runs is used. As $I_i \sim \mathrm{Bernoulli}(q_b)$, its variance is that of a bernoulli random variable given by

$$\mathrm{Var}(I_i) = E[I_i](1 - E[I_i]) \tag{2.6}$$

Note that I_i and I_j are dependent on each other if and only if $i = j-1$ because I_{j-1} and I_j can not both be 1 in the same frame. Other than that, $\forall i < j-1$, I_i and I_j are independent. Thus,

$$\text{Cov}(I_i, I_j) = \begin{cases} 0 & \text{if } i < j-1 \\ -E[I_i]E[I_j] = -E[I_i]q_b(1-q_b) & \\ & \text{if } i = j-1 \end{cases}$$

Hence we have:

$$\text{Var}(R_b) = \text{Var}(I_1) + \sum_{j=2}^{f} \text{Var}(I_j) + 2\text{Cov}(I_1, I_2) + 2\sum_{j=3}^{f} \text{Cov}(I_{j-1}, I_j)$$
$$= q_b(1-q_b) + (f-1)q_b(1-q_b)\{1 - q_b(1-q_b)\} - 2q_b^2(1-q_b) - 2(f-2)q_b^2(1-q_b)^2$$
$$= f(q_b - 4q_b^2 + 6q_b^3 - 3q_b^4) + (3q_b^2 - 8q_b^3 + 5q_b^4) \qquad \square$$

Lemma 3. *Given tag population size t, frame size f, and persistence probability p, we have:*

$$\text{Cov}(Y_b, R_b) = \sum_{y=0}^{f} \sum_{r=0}^{\lceil \frac{f}{2} \rceil} yrq_b^y(1-q_b)^{f-y}.\xi\{f, y, r\} - E[Y_b]E[R_b] \tag{2.7}$$

where

$$\xi\{f, y, r\} = \begin{cases} \binom{y-1}{r-1}\left[\binom{f-y-1}{r-2} + 2\binom{f-y-1}{r-1} + \binom{f-y-1}{r}\right] \\ \text{if } r > 1 \wedge 0 < y < f \wedge r \le y \wedge r \le f-y-1 \\ \\ \binom{y-1}{r-1}\left[2\binom{f-y-1}{r-1} + \binom{f-y-1}{r}\right] \\ \text{if } r = 1 \wedge 0 < y < f \wedge r \le y \wedge r \le f-y-1 \\ \\ 1 \quad \text{if } r = 1 \wedge y = f \\ \\ 1 \quad \text{if } r = 0 \wedge y = 0 \\ \\ 0 \quad \text{otherwise} \end{cases}$$

Proof. By definition, we have

$$\text{Cov}(Y_b, R_b) = \sum_{y=0}^{f} \sum_{r=0}^{f} yrP\{Y_b = y, R_b = r\} - E[Y_b]E[R_b] \tag{2.8}$$

Here $P\{Y_b = y, R_b = r\}$ represents the probability that exactly y out of f slots in the frame are b and at the same time the number of runs of b is r. This probability is difficult to evaluate directly, but conditioning on Y_b simplifies the task.

$$P\{Y_b = y, R_b = r\} = P\{R_b = r | Y_b = y\} \times P\{Y_b = y\} \tag{2.9}$$

As $Y_b \sim \text{Binom}(f, q_b)$, we have:

$$P\{Y_b = y\} = \binom{f}{y} q_b^y (1 - q_b)^{f-y} \tag{2.10}$$

Now we calculate $P\{R_b = r | Y_b = y\}$ *i.e.*, the probability of having r runs of b in a frame of size f given that y out of f slots are b. As tags choose the slots independently, each occurrence with r runs having y slots of b is equally likely. Therefore, we determine the total number of ways, denoted by $\xi\{f, y, r\}$, in which y occurrences of b and $f - y$ occurrences of $\overline{b}$ can be arranged such that the number of runs of b is r. We treat this as an ordered partition problem. First, we separate all the y occurrences of b from the frame and make r partitions of these y occurrences. Then, we create appropriate number of partitions of $f - y$ occurrences of $\overline{b}$ such that between consecutive partitions of b, the partitions of $\overline{b}$ can be *interleaved*. For r partitions of b, there are 4 possible partitions of $\overline{b}$.

1. The frame starts with b and ends with b, implying that there are $r - 1$ partitions of $\overline{b}$, each interleaved between adjacent partitions of b.
2. The frame starts with b and ends with $\overline{b}$, implying that there are r partitions of $\overline{b}$.
3. The frame starts with $\overline{b}$ and ends with b, implying that there are r partitions of $\overline{b}$.
4. The frame starts with $\overline{b}$ and ends with $\overline{b}$, implying that there are $r + 1$ partitions of $\overline{b}$.

We can make r partitions of y occurrences of b in $\binom{y-1}{r-1}$ ways and r partitions of $f - y$ occurrences of $\overline{b}$ in $\binom{f-y-1}{r-1}$ ways. Similarly, we can make $r+1$ partitions of $f-y$ occurrences of $\overline{b}$ in $\binom{f-y-1}{r}$ ways and $r - 1$ partitions of of $f - y$ occurrences of $\overline{b}$ in $\binom{f-y-1}{r-2}$ ways. The equation of $\xi\{f, y, r\}$ in the lemma statement follows from this discussion. The total number of ways in which y zeros can be arranged among f slots is $\binom{f}{y}$. Thus, we get

$$P\{R_b = r | Y_b = y\} = \frac{\xi\{f, y, r\}}{\binom{f}{y}} \tag{2.11}$$

Substituting values from Eqs. (2.10) and (2.11) in (2.9) and (2.8) gives Eq. (2.7). □

Theorem 4. *Given tag population size t, frame size f, and persistence probability p, we have:*

$$E[X_b] = \frac{E[Y_b]}{E[R_b]} - \frac{\text{Cov}(Y_b, R_b)}{E^2[R_b]} + \frac{E[Y_b]}{E^3[R_b]}\text{Var}(R_b) \tag{2.12}$$

$$\text{Var}(X_b) = \frac{\text{Var}(Y_b)}{E^2[R_b]} - \frac{2E[Y_b]}{E^3[R_b]}\text{Cov}(Y_b, R_b) + \frac{E^2[Y_b]}{E^4[R_b]}\text{Var}(R_b) \tag{2.13}$$

Proof. Let $g(Y_b, R_b) = X_b = \frac{Y_b}{R_b}$. The Taylor series expansion of g around (θ_1, θ_2) is:

$$g(Y_b, R_b) = \sum_{j=0}^{\infty} \left\{ \frac{1}{j!} \left[(Y_b - \theta_1)\frac{\partial}{\partial Y_b'} + (R_b - \theta_2)\frac{\partial}{\partial R_b'} \right]^j \times g(Y_b', R_b') \right\}_{\substack{Y_b'=\theta_1 \\ R_b'=\theta_2}}$$

According to Bienaymé-Chebyshev inequality, we have $\theta_1 = E[Y_b]$ and $\theta_2 = E[R_b]$. Therefore, we get the following expansion of the Taylor series of $g(Y_b, R_b)$:

$$\begin{aligned} g(Y_b, R_b) = g(\theta_1, \theta_2) &+ \left[(Y_b - \theta_1)\frac{\partial g}{\partial Y_b} + (R_b - \theta_2)\frac{\partial g}{\partial R_b} \right] \\ &+ \frac{1}{2}\left[(Y_b - \theta_1)^2 \frac{\partial^2 g}{\partial Y_b^2} + 2(Y_b - \theta_1)(R_b - \theta_2)\frac{\partial^2 g}{\partial Y_b \partial R_b} + (R_b - \theta_2)^2 \frac{\partial^2 g}{\partial R_b^2} \right] + O(j^{-1}) \end{aligned}$$

Taking the expectation of both sides, we get

$$E[g(Y_b, R_b)] \approx \frac{1}{2}\left[\text{Var}(Y_b)\frac{\partial^2 g}{\partial Y_b^2} + 2\text{Cov}(Y_b, R_b)\frac{\partial^2 g}{\partial Y_b \partial R_b} + \text{Var}(R_b)\frac{\partial^2 g}{\partial R_b^2} \right] + g(\theta_1, \theta_2) \tag{2.14}$$

Evaluating the partial derivatives of g as required in Equation (2.14), we get

$$\frac{\partial^2 g(Y_b, R_b)}{\partial Y_b^2}\Big|_{\substack{Y_b=\theta_1 \\ R_b=\theta_2}} = 0, \quad \frac{\partial^2 g(Y_b, R_b)}{\partial Y_b \partial R_b}\Big|_{\substack{Y_b=\theta_1 \\ R_b=\theta_2}} = -\frac{1}{\theta_2^2}, \quad \frac{\partial^2 g(Y_b, R_b)}{\partial R_b^2}\Big|_{\substack{Y_b=\theta_1 \\ R_b=\theta_2}} = 2\frac{\theta_1}{\theta_1^3}$$

Putting these values in Equation (2.14) and using $\theta_1 = E[Y_b]$ and $\theta_2 = E[R_b]$, we get Equation (2.12). The variance can be calculated as follows:

$$\text{Var}\big(g(Y_b, R_b)\big) = E\left[\left\{g(Y_b, R_b) - E[g(Y_b, R_b)]\right\}^2\right] \tag{2.15}$$

Considering that $E[g(Y_b, R_b)]$ is being squared in the expression above, we use first order Taylor series expansion to get the value of $E[g(Y_b, R_b)]$ and substitute it in Equation (2.15).

$$\begin{aligned} E[g(Y_b, R_b)] &= E\Big[(Y_b - \theta_1)\frac{\partial g}{\partial Y_b} + (R_b - \theta_2)\frac{\partial g}{\partial R_b}\Big] + g(\theta_1, \theta_2) + O(j^{-1}) \\ &= \Big[(0)\frac{\partial g}{\partial Y_b} + (0)\frac{\partial g}{\partial R_b}\Big] + g(\theta_1, \theta_2) + O(j^{-1}) \approx g(\theta_1, \theta_2) \end{aligned}$$

Substituting the value of $E[g(Y_b, R_b)]$ and using the first order Taylor series expansion of $g(Y_b, R_b)$ in (2.15), we get

$$\begin{aligned} \operatorname{Var}\big(g(Y_b, R_b)\big) &= E\big[\big\{(Y_b - \theta_1)\frac{\partial g}{\partial Y_b} + (R_b - \theta_2)\frac{\partial g}{\partial R_b}\big\}^2\big] + O(j^{-1}) \\ &\approx \operatorname{Var}(Y_b)(\frac{\partial g}{\partial Y_b})^2 + 2\operatorname{Cov}(Y_b, R_b)\frac{\partial g}{\partial Y_b}\frac{\partial g}{\partial R_b} + \operatorname{Var}(R_b)(\frac{\partial g}{\partial R_b})^2 \end{aligned} \tag{2.16}$$

Evaluating the partial derivatives of g as required in the equation above, we get

$$\frac{\partial g(Y_b, R_b)}{\partial Y_b}\Big|_{\substack{Y_b=\theta_1 \\ R_b=\theta_2}} = \frac{1}{\theta_2}, \quad \frac{\partial g(Y_b, R_b)}{\partial R_b}\Big|_{\substack{Y_b=\theta_1 \\ R_b=\theta_2}} = -\frac{\theta_1}{\theta_2^2}$$

Putting these values in Equation (2.16) and using $\theta_1 = E[Y_b]$ and $\theta_2 = E[R_b]$, we get Equation (2.13). □

Figures 2.2 and 2.3 show the expectation and variance of X_1 calculated using Equations (2.12) and (2.13), respectively, with $f = 16$ and $p = 1$. The dots in these figures represent the corresponding values obtained through 100 repetitions of simulation for each tag population size. These figures show that the values given by Equations (2.12) and (2.13) track the simulation results very well, which serves as an experimental proof that the assumption "instead of picking a single slot to reply at the start of frame of size f, a tag independently decides to reply in each slot of the frame with probability $1/f$ regardless of its decision about previous or forthcoming slots" practically holds.

2.5 ART — Parameter Tuning

To minimize estimation time while achieving required reliability, next, we obtain values of persistence probability p, number of rounds n, and frame size f. As we have three unknowns,

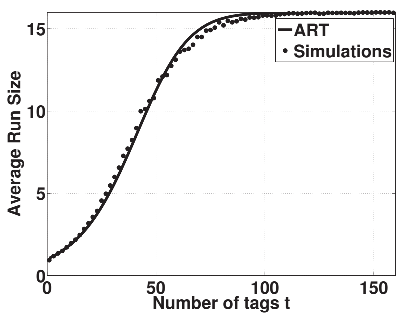


Figure 2.2 Expectation of ART estimator

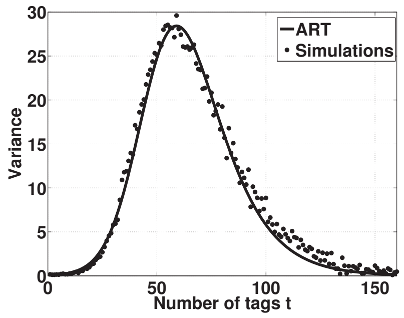


Figure 2.3 Variances of ART estimator

we require three equations that can be solved simultaneously. We derive these three equations using following three conditions: (1) the confidence interval should be symmetric around t *i.e.*, $|\tilde{t} - t| \leq \beta t$, (2) actual reliability is greater than or equal to the required reliability *i.e.*, $P\left\{|\tilde{t} - t| \leq \beta t\right\} \geq \alpha$, and (3) estimation time is minimized. We use the first condition to calculate p, the second condition to calculate n, and the last condition to calculate f.

Although both X_0 and X_1 can be used to estimate the tag population size, we choose X_1 for ART because the tag population size estimation calculated from X_1 has smaller variance compared to X_0 as we show in Section 2.7.3. It is worth noting that X_0 and X_1 are not equivalent estimators. The average run size of 0s cannot be inferred from the average run size of 1s, and vice versa. For example, 1100011 and 1100110 have the same average run size of 1s, but they have different average run size of 0s. Fundamentally, X_0 and X_1 are not equivalent estimators because for any slot, the probability of it being 0 and that of it being 1 are different.

2.5.1 Persistence Probability p

We express confidence interval requirement $|\tilde{t} - t| \leq \beta t$ as

$$(1 - \beta)t \leq \tilde{t} \leq (1 + \beta)t \tag{2.17}$$

Recall from Lemma 1 that we use q_1 to denote the probability that a slot in a frame is 1 when the number of tags in the population are t and the persistence probability is p. Let q_1^+ and q_1^- denote the probabilities that a slot in a frame is 1 when the number of tags in the population are $(1+\beta)t$ and $(1-\beta)t$, respectively, and the persistence probability is p. Let $\tilde{q_1}$ represent the estimate of q_1. Therefore, we have

$$q_1^+ = 1 - (1 - \frac{p}{f})^{(1+\beta)t} \Rightarrow (1+\beta)t = \frac{\ln\left\{1 - q_1^+\right\}}{\ln\left\{1 - \frac{p}{f}\right\}} \tag{2.18}$$

$$q_1^- = 1 - (1 - \frac{p}{f})^{(1-\beta)t} \Rightarrow (1-\beta)t = \frac{\ln\left\{1 - q_1^-\right\}}{\ln\left\{1 - \frac{p}{f}\right\}} \tag{2.19}$$

$$\tilde{q_1} = 1 - (1 - \frac{p}{f})^{\tilde{t}} \quad \Rightarrow \quad \tilde{t} = \frac{\ln\left\{1 - \tilde{q_1}\right\}}{\ln\left\{1 - \frac{p}{f}\right\}} \tag{2.20}$$

Substituting values of $(1+\beta)t$, $(1-\beta)t$, and $\tilde{t}$ from Equations (2.18), (2.19), and (2.20), respectively, into Expression (2.17), we get

$$\frac{\ln\left\{1 - q_1^-\right\}}{\ln\left\{1 - \frac{p}{f}\right\}} \le \frac{\ln\left\{1 - \tilde{q_1}\right\}}{\ln\left\{1 - \frac{p}{f}\right\}} \le \frac{\ln\left\{1 - q_1^+\right\}}{\ln\left\{1 - \frac{p}{f}\right\}}$$

As $\ln\left\{1 - \frac{p}{f}\right\} < 0$, thus,

$$\ln\left\{1 - q_1^+\right\} \le \ln\left\{1 - \tilde{q_1}\right\} \le \ln\left\{1 - q_1^-\right\}$$

Exponentiating and rearranging, the confidence interval requirement becomes

$$q_1^- \le \tilde{q_1} \le q_1^+$$

As $E[X_1]$ and $\text{Var}(X_1)$ are functions of q_1, denoting $E[X_1]$ by $\mu\left\{q_1\right\}$, $\text{Var}(X_1)$ by $\sigma^2\left\{q_1\right\}$, and the observed average value of X_1 from the n frames by $\tilde{X_1}$, we have $\tilde{q_1} = \mu^{-1}\{\tilde{X_1}\}$. Using $\mu^{-1}\{\tilde{X_1}\}$ to substitute $\tilde{q_1}$ in the expression above, we get

$$q_1^- \le \mu^{-1}\{\tilde{X_1}\} \le q_1^+ \Rightarrow \mu\left\{q_1^-\right\} \le \tilde{X_1} \le \mu\left\{q_1^+\right\}$$

Based on the fact that the variance of a random variable is reduced by n times if the same experiment is repeated n times, by running n rounds and getting n frames, the variance of X_1

becomes $\frac{\sigma^2\{q_1\}}{n}$ and the standard deviation of X_1 becomes $\frac{\sigma\{q_1\}}{\sqrt{n}}$. Let Z denote $\frac{\tilde{X_1}-\mu\{q_1\}}{\sigma\{q_1\}/\sqrt{n}}$. Thus, the expression above becomes

$$\frac{\mu\left\{q_1^-\right\}-\mu\left\{q_1\right\}}{\frac{\sigma\{q_1\}}{\sqrt{n}}} \leq Z \leq \frac{\mu\left\{q_1^+\right\}-\mu\left\{q_1\right\}}{\frac{\sigma\{q_1\}}{\sqrt{n}}} \tag{2.21}$$

By the central limit theorem, Z approximates a standard normal random variable. The area under the standard normal curve gives the success probability, which is the required reliability in our context. For the confidence interval to be symmetric on both the upper and lower sides of the population size as per the first of the three conditions, the absolute value of the upper and lower limits of Z should be equal. Let k represent the absolute value of these upper and lower limits. Thus, we can represent Z as follows:

$$-k \leq Z \leq k \tag{2.22}$$

From Expressions (5.19) and (5.20), we get

$$\frac{\mu\left\{q_1^-\right\}-\mu\left\{q_1\right\}}{\frac{\sigma\{q_1\}}{\sqrt{n}}} = -k, \qquad \frac{\mu\left\{q_1^+\right\}-\mu\left\{q_1\right\}}{\frac{\sigma\{q_1\}}{\sqrt{n}}} = k \tag{2.23}$$

As absolute values of the right hand sides (R.H.S.) of both equations above are k, we get

$$2\mu\left\{q_1\right\}-\mu\left\{q_1^+\right\}-\mu\left\{q_1^-\right\}=0 \tag{2.24}$$

The equation above gives the condition that needs to be satisfied to make the confidence interval symmetric around the tag population size. Figure 2.4 plots the value of left hand side (L.H.S) of this equation as a function of p for three different values of β. We can see that it is a well behaved function of p and thus, there exists a unique value of p that makes it equal to zero. Furthermore, we also observe that all the curves cross the zero line at the same point which gives us a hint that the solution to the equation above is independent of β. Next we solve this equation.

Applying the first order Taylor series expansion on $\mu\left\{q_1\right\}$, we get $\mu\left\{q_1\right\} = E[Y_1]/E[R_1]$. Using the expressions of $E[Y_1]$ and $E[R_1]$ from Equations (2.2) and (2.4) respectively, we

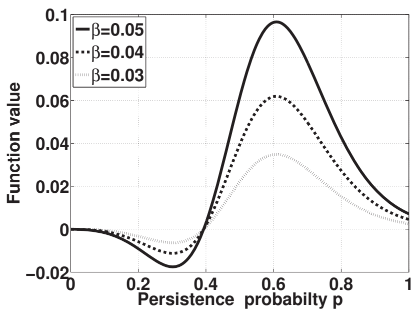


Figure 2.4 Equation (2.24) as a function of p

can express $\mu\left\{q_1\right\}$, $\mu\left\{q_1^+\right\}$, and $\mu\left\{q_1^-\right\}$ as follows:

$$\begin{aligned}\mu\left\{q_1\right\} &= \frac{fq_1}{q_1\big(q_1+f(1-q_1)\big)}\\ \mu\left\{q_1^+\right\} &= \frac{fq_1^+}{q_1^+\big(q_1^++f(1-q_1^+)\big)}\\ \mu\left\{q_1^-\right\} &= \frac{fq_1^-}{q_1^-\big(q_1^-+f(1-q_1^-)\big)}\end{aligned}$$

Substituting these expressions in Equation (2.24), we get

$$\frac{2}{q_1+f(1-q_1)}-\frac{1}{q_1^++f(1-q_1^+)}-\frac{1}{q_1^-+f(1-q_1^-)}=0$$

Substituting the value of q_1, q_1^+, and q_1^- from Equations (2.1), (2.18), and (2.19) respectively, into the equation above, and to simplify the presentation, using $\eta=(1-\frac{p}{f})^t$, we get

$$\frac{2}{1-\eta+f\eta}-\frac{1}{1-\eta^{1+\beta}+f\eta^{1+\beta}}-\frac{1}{1-\eta^{1-\beta}+f\eta^{1-\beta}}=0$$

Next, we do algebraic simplification of the expression above.

$$\begin{aligned}-\big(1-\eta+f\eta\big)\Big\{1-\eta^{1+\beta}+f\eta^{1+\beta}+1-\eta^{1-\beta}+f\eta^{1-\beta}\Big\}&\\ +2\big(1-\eta^{1+\beta}+f\eta^{1+\beta}\big)\big(1-\eta^{1-\beta}+f\eta^{1-\beta}\big)&=0\end{aligned}$$

Dividing the equation above by $\eta^{1-\beta}$, we get

$$\begin{aligned}-\big(1-\eta+f\eta\big)\Big\{2\eta^{\beta-1}-\eta^{2\beta}+f\eta^{2\beta}-1+f\Big\}&\\ +2\big(1-\eta^{1+\beta}+f\eta^{1+\beta}\big)\big(\eta^{\beta-1}-1+f\big)&=0\end{aligned}$$

Simplifying the equation above, we get

$$\begin{aligned}
&(f-1)+\eta^{2\beta}\Big(-1+f+2f\eta-\eta-f^2\eta\Big)\\
&\quad+2\eta^{\beta}\Big(\eta(f-1)^2+1-f\Big)-\eta\Big(1-2f+f^2\Big)=0\\
\Rightarrow\ &(f-1)+\eta^{2\beta}\Big((f-1)-\eta(f-1)^2\Big)\\
&\quad+2\eta^{\beta}\Big(\eta(f-1)^2-(f-1)\Big)-\eta(f-1)^2=0
\end{aligned}$$

Dividing the equation above by $f-1$ and simplifying, we get

$$\Rightarrow \big(1-\eta(f-1)\big)(1-\eta^{\beta})^2=0$$

In the equation above, either $1-\eta(f-1)=0$ and/or $1-\eta^{\beta}=0$. The value of $1-\eta^{\beta}$ equals zero only when $\beta=0$, but we know from our problem statement that $\beta\in(0,1]$ *i.e.*, $\beta\neq 0$. Therefore, $1-\eta(f-1)=0$. Putting back $\eta=(1-\frac{p}{f})^t$ and solving $1-(f-1)(1-\frac{p}{f})^t=0$ for p, we get

$$p=f\Big\{1-\Big(\frac{1}{f-1}\Big)^{\frac{1}{t}}\Big\} \tag{2.25}$$

Note that this equation does not involve β, which shows that indeed the solution to Equation (2.24) is independent of β as we had intuitively inferred from Figure 2.4.

Equation (2.25) is first of the three equations that we will solve simultaneously. This equation requires the value of actual tag population size t which we do not know. Fortunately, we can calculate an upper bound, t_m, on the actual tag population size and use that in Equation (2.25) instead of t. We will describe a method to obtain t_m in Section 2.5.4, and also determine how close t_m has to be to t to ensure that ART achieves the required reliability.

2.5.2 Number of Rounds n

Using the persistence probability calculated in Equation (2.25), the two equations in (2.23) hold. From them, we get

$$\left(\frac{k\sigma\left\{q_1\right\}}{\mu\left\{q_1^+\right\}-\mu\left\{q_1\right\}}\right)^2=n=\left(\frac{-k\sigma\left\{q_1\right\}}{\mu\left\{q_1^-\right\}-\mu\left\{q_1\right\}}\right)^2 \tag{2.26}$$

Let Φ be the cumulative distribution function of a standard normal distribution and $\operatorname{erf}\{.\}$ be the standard error function, we get

$$P\left\{-k\leq Z\leq k\right\}=\Phi(k)-\Phi(-k)=\operatorname{erf}\left\{\frac{k}{\sqrt{2}}\right\} \tag{2.27}$$

$P\left\{-k\leq Z\leq k\right\}$ gives the success probability in terms of the area under the standard normal curve between $-k$ and $+k$. As per the second of the three conditions, this area should be at least equal to the required reliability α *i.e.*,

$$P\left\{-k\leq Z\leq k\right\}=\alpha \tag{2.28}$$

From Equations (5.21) and (2.28), we get

$$k=\sqrt{2}\operatorname{erf}^{-1}\left\{\alpha\right\} \tag{2.29}$$

From Equations (2.26) and (2.29), we get

$$\left(\frac{\sqrt{2}\operatorname{erf}^{-1}\{\alpha\}\times\sigma\left\{q_1\right\}}{\mu\left\{q_1^+\right\}-\mu\left\{q_1\right\}}\right)^2=n=\left(\frac{-\sqrt{2}\operatorname{erf}^{-1}\{\alpha\}\times\sigma\left\{q_1\right\}}{\mu\left\{q_1^-\right\}-\mu\left\{q_1\right\}}\right)^2 \tag{2.30}$$

Equation (2.30) is second of the three equations that we will solve simultaneously.

2.5.3 Optimal Frame Size f

As per the third of the three conditions, total estimation time should be minimum. The total estimation time is directly proportional to total number of slots, $(f+l)\times n$, which is

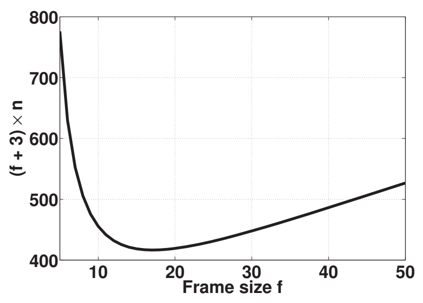


Figure 2.5 Total estimation time vs. frame size

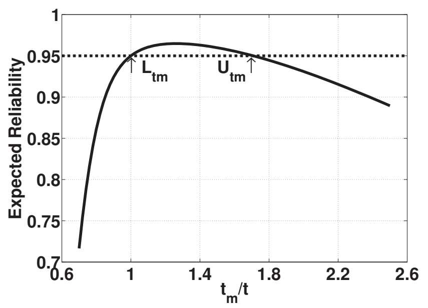


Figure 2.6 Expected value of actual reliability vs. $\frac{t_m}{t}$

a convex function of f as seen from Figure 2.5. This means that an optimal frame size f_{op} exists and can be obtained by differentiating $(f+l)\times n$ with respect to f as shown below:

$$\frac{d}{df}\Big\{(f+l)\times n\Big\} = 0 \tag{2.31}$$

Equation (2.31) is third of the three equations that we will solve simultaneously.

Required reliability α and confidence interval β are given constants and t_m is calculated using method proposed in the next Section 2.5.4. Thus, p, q_1, q_1^+, and q_1^- are all functions of f. Consequently, n is a function of f and, therefore, $(f+l)\times n$ is also a function of f with only one unknown, *i.e.*, f. The numerical solution of Equation (2.31) gives the optimal value of frame size, represented by f_{op}.

To numerically solve Equation (2.31), we substitute the value of n from Equation (2.30) in Equation (2.31). As both expressions for n given in Equation (2.30) have same values when p is calculated using Equation (2.25), either of them can be used to calculate n. Substituting n in Equation (2.31) by the L.H.S of the expression for n in Equation (2.30), we get

$$\Big[\mu\{q_1^+\}-\mu\{q_1\}\Big]\Big[\sigma\{q_1\}+2(f+l)\frac{\partial\sigma\{q_1\}}{\partial f}\Big]-2(f+l)\sigma\{q_1\}\Big[\frac{\partial\mu\{q_1^+\}}{\partial f}-\frac{\partial\mu\{q_1\}}{\partial f}\Big] = 0 \tag{2.32}$$

where $\frac{\partial\mu\{.\}}{\partial f}$ and $\frac{\partial\sigma\{.\}}{\partial f}$ are obtained through the differentiation of expressions for $E[X_b]$ and $\text{Var}(X_b)$ in Equations (2.12) and (2.13), respectively. We solve Equation (2.32) numerically to obtain f_{op}.

2.5.3.1 Summary of steps to calculate p, n, and f_{op}

First, we calculate the value of t_m, as explained in the next Section 2.5.4. Second, we numerically solve Equation (2.32) to obtain f_{op}. Third, we put this value of f_{op} along with t_m in Equation (2.25) to obtain the value of p. Last, we put the resulting value of p along with f_{op} in Equation (2.30) and obtain the value of n. Note that although Equation (2.25) does not involve α and β, p still depends on them because it is a function of f and the optimal value of f depends on α and β.

Table 2.1 shows the values of p, n, and f_{op} for different accuracy requirements and tag population sizes calculated using the steps described above. We observe from this table that for a given tag population size, as the value of α increases and/or β decreases, the value of n increases to fulfill the more stringent accuracy requirements. We also observe from this table that for a given (α, β) pair, the values of f_{op} and n are the same for all tag population sizes, which shows that total number of slots, $(f_{\text{op}} + l) \times n$, depends only on the accuracy requirements and is independent of tag population size. We will formally prove the independence of estimation time from tag population size in Section 2.7.1. We further observe that as the tag population size increases, the value of p decreases to reduce the number of tags participating in a frame to keep the value of f_{op} and n independent of tag population size.

Table 2.1 Values of f_{op}, n, and p for different values of α, β and tag population size

Accuracy Requirement	**Tag Population Size**								
	10^2			10^4			10^6		
	f_{op}	n	p	f_{op}	n	p	f_{op}	n	p
$\alpha = 60.0\%, \beta = 40.0\%$	12	1.00E+00	2.84E-01	12	1.00E+00	2.88E-03	12	1.00E+00	2.88E-05
$\alpha = 70.0\%, \beta = 30.0\%$	14	2.00E+00	3.55E-01	14	2.00E+00	3.59E-03	14	2.00E+00	3.59E-05
$\alpha = 80.0\%, \beta = 20.0\%$	15	4.00E+00	3.91E-01	15	4.00E+00	3.96E-03	15	4.00E+00	3.96E-05
$\alpha = 90.0\%, \beta = 10.0\%$	15	2.50E+01	3.91E-01	15	2.50E+01	3.96E-03	15	2.50E+01	3.96E-05
$\alpha = 95.0\%, \beta = 5.00\%$	15	1.43E+02	3.91E-01	15	1.43E+02	3.96E-03	15	1.43E+02	3.96E-05
$\alpha = 99.0\%, \beta = 1.00\%$	15	6.24E+03	3.91E-01	15	6.24E+03	3.96E-03	15	6.24E+03	3.96E-05
$\alpha = 99.9\%, \beta = 0.10\%$	15	1.02E+06	3.91E-01	15	1.02E+06	3.96E-03	15	1.02E+06	3.96E-05

2.5.4 Obtaining Population Upper Bound $\mathbf{t_m}$

So far we have assumed the knowledge of an upper bound t_m on tag population size t. We now present a fast scheme to obtain t_m based on Flajolet and Martin's probabilistic counting algorithm [47]. Before calculating system parameters p, n, and f_{op}, the reader uses this scheme to obtain t_m. In this scheme, the reader keeps issuing single-slot frames, where the persistence probability p follows a geometric distribution starting from $p = 1$ (*i.e.*, $p = \frac{1}{2^{i-1}}$ in the i^{th} frame), until the reader gets an empty slot. Suppose the empty slot occurred in the i^{th} frame, then $t_m = 1.2897 \times 2^{i-2}$ is an upper bound on t [90, 47]. According to [47], t_m asymptotically approaches t when instead of using a single value of the first empty slot from one experiment, we use average of values of the first empty slot from a large number of experiment.

Next, we determine how close the upper bound t_m has to be to the actual tag population size to ensure that ART achieves the required reliability and examine whether t_m obtained using $t_m = 1.2897 \times 2^{i-2}$ lies close enough to t. We derive an expression to calculate the expected value of actual reliability, denoted by $\tilde{\alpha}$, as a function of t_m given that the required reliability α, confidence interval β, and the actual tag population size t are known.

Equation (2.30) is obtained using the condition that actual reliability should be greater than or equal to the required reliability. Therefore, we use this equation to derive an expression for expected value of actual reliability. In Equation (2.30) , we calculate n using q_1, q_1^+, and q_1^-, which are obtained from Equations (2.1), (2.18) and (2.19), respectively, by putting $t = t_m$, $f = f_{\text{op}}$, and $p = f_{\text{op}}\Big\{1 - \Big(\frac{1}{f_{\text{op}}-1}\Big)^{\frac{1}{t_m}}\Big\}$. This gives us:

$$q_1 = 1 - \Big(\frac{1}{f_{\text{op}} - 1}\Big),\ q_1^{\pm} = 1 - \Big(\frac{1}{f_{\text{op}} - 1}\Big)^{1\pm\beta} \tag{2.33}$$

As the number of tags in the population are t and not t_m, when the reader executes the frames, the actual values of q_1, q_1^+, and q_1^- represented by $\hat{q_1}$, $\hat{q_1}^+$, and $\hat{q_1}^-$, respectively, follow the equations below.

$$\hat{q_1} = 1 - \Big(\frac{1}{f_{\text{op}} - 1}\Big)^{\frac{t}{t_m}},\ \hat{q_1}^{\pm} = 1 - \Big(\frac{1}{f_{\text{op}} - 1}\Big)^{\frac{t}{t_m}(1\pm\beta)} \tag{2.34}$$

Let $\tilde{\alpha}$ represent the expected value of actual reliability in n rounds when the population contains t tags and the calculated upper bound is t_m, then the following equality holds.

$$\left(\frac{\sqrt{2}\,\mathrm{erf}^{-1}\{\tilde{\alpha}\}\times\sigma\{\hat{q_1}\}}{\mu\{\hat{q_1}^{+}\}-\mu\{\hat{q_1}\}}\right)^2 = n = \left(\frac{-\sqrt{2}\,\mathrm{erf}^{-1}\{\tilde{\alpha}\}\times\sigma\{\hat{q_1}\}}{\mu\{\hat{q_1}^{-}\}-\mu\{\hat{q_1}\}}\right)^2$$

Substituting value of n from Eq. (2.30) into the equation above and solving for $\tilde{\alpha}$, we get

$$\begin{aligned}\tilde{\alpha} &= \mathrm{erf}\Big\{\mathrm{erf}^{-1}\{\alpha\}\times\frac{\sigma\{q_1\}}{\sigma\{\hat{q_1}\}}\times\frac{\mu\{\hat{q_1}^{+}\}-\mu\{\hat{q_1}\}}{\mu\{q_1^{+}\}-\mu\{q_1\}}\Big\}\\ &= \mathrm{erf}\Big\{\mathrm{erf}^{-1}\{\alpha\}\times\frac{\sigma\{q_1\}}{\sigma\{\hat{q_1}\}}\times\frac{\mu\{\hat{q_1}^{-}\}-\mu\{\hat{q_1}\}}{\mu\{q_1^{-}\}-\mu\{q_1\}}\Big\}\end{aligned} \tag{2.35}$$

The expected actual reliability $\tilde{\alpha}$ is a convex function of $\frac{t_m}{t}$ and is equal to α for two values of $\frac{t_m}{t}$ represented by L_{tm} and U_{tm}. Figure 2.6 plots the expected value of actual reliability $\tilde{\alpha}$ as a function of $\frac{t_m}{t}$ using Equation (2.35) with $\alpha = 95\%$ and $\beta = 5\%$. The dashed horizontal line in the figure marks the required reliability $\alpha = 95\%$. The actual reliability will be greater than or equal to the required reliability as long as the value of $\frac{t_m}{t}$ satisfies the following condition:

$$L_{tm} \leq \frac{t_m}{t} \leq U_{tm} \tag{2.36}$$

The values of L_{tm} and U_{tm} can be obtained by using $\tilde{\alpha} = \alpha$ in Equation (2.35) and solving it for t_m and dividing it by the tag population size t. This results in two values of $\frac{t_m}{t}$ because $\tilde{\alpha}$ is a convex function of $\frac{t_m}{t}$ and its maxima is greater than α. The value of L_{tm} is always equal to 1 and the value of U_{tm} is calculated by the numerical solution of Equation (2.35) using $\tilde{\alpha} = \alpha$.

The value of U_{tm} depends on the required reliability α and confidence interval β. Table 2.2 tabulates the values of U_{tm} for different population sizes and accuracy requirements. We observe from Table 2.2 that the value of U_{tm} is independent of tag population size. This is because U_{tm} depends on q_1 for a given α and β (according to Equation 2.35) and q_1 is independent of tag population size as we will discuss in Section 2.7.1. We also observe that U_{tm} decreases with increasing accuracy requirements. This makes intuitive sense because the higher the required accuracy, the lesser the error in the upper bound t_m that can be tolerated.

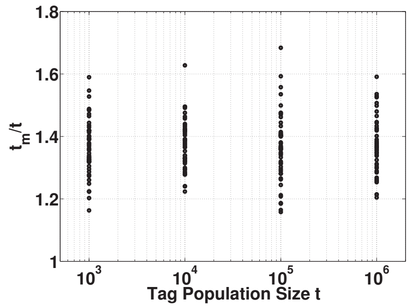


Figure 2.7 Experimentally observed values of ratio $\frac{t_m}{t}$

We see from Table 2.2 that even for very high accuracy requirements of $\alpha = 99.99\%$ and $\beta = 0.01\%$, the value of t_m calculated as $t_m = 1.2897 \times 2^{i-2}$ can be up to $1.64 \times t$.

Table 2.2 U_{tm} for different population sizes and accuracy requirements

Accuracy Requirement	**Tag Population Size**			
	10^3	10^4	10^5	10^6
$\alpha = 90.00\%, \beta = 10.0\%$	1.83	1.83	1.83	1.83
$\alpha = 95.00\%, \beta = 5.00\%$	1.71	1.71	1.71	1.71
$\alpha = 99.00\%, \beta = 1.00\%$	1.66	1.66	1.66	1.66
$\alpha = 99.90\%, \beta = 0.10\%$	1.64	1.64	1.64	1.64
$\alpha = 99.99\%, \beta = 0.01\%$	1.64	1.64	1.64	1.64

From simulations, we have observed that the value of t_m calculated as $t_m = 1.2897 \times 2^{i-2}$ always lies within t and $1.64 \times t$. This is seen in Figure 2.7, where we plot the observed values of $\frac{t_m}{t}$ obtained through 100 runs of simulations using $t_m = 1.2897 \times 2^{i-2}$ for different values of tag population size. Within each simulation run, we obtained 10 values of i, averaged them, and replaced i with that average in the equation $t_m = 1.2897 \times 2^{i-2}$ to obtain $\frac{t_m}{t}$.

2.6 ART — Practical Considerations

In this section, we describe how ART estimates sizes of arbitrarily large tag populations. We also present the method that ART employs to enable the use of multiple RFID readers for estimating the size of a given RFID tag population.

2.6.1 Unbounded Tag Population Size

For a given value of frame size f, Theorem 5 calculates the upper bound t_M on the number of tags that ART can estimate. This upper bound exists because for tag population sizes larger than t_M, the system parameters take on values that can not be implemented practically. After Theorem 5, we describe how we extend ART to estimate sizes of arbitrarily large populations.

Theorem 5. *For a given frame size $f > 1$, the maximum number of tags t_M that ART can estimate is:*

$$t_M = -\frac{\ln\{f-1\}}{\ln\left\{1-\frac{1}{2^{15}}\right\}} \tag{2.37}$$

Proof. In theory, we can increase the estimation scope of ART to any population size by decreasing the value of p according to Equation (2.25). In practice, however, f/p has a minimum value of $2^{15}-1$. Recall that in ART, the reader announces a virtual frame size of f/p (although terminates the frame after the first f slots) and each tag uses the result of a hash function h to select a slot in the range $[1, f/p]$. The number of bits to store the result of the hash function is specified to be 15 in the C1G2 standard. Thus, the maximum value of f/p can be $2^{15}-1$, *i.e.*

$$p > \frac{f}{2^{15}}$$

Substituting the value of p from Equation (2.25) into the equation above, we get

$$f\left\{1-\left(\frac{1}{f-1}\right)^{\frac{1}{t}}\right\} > \frac{f}{2^{15}}$$

Rearranging the expression above and solving for t, we get

$$t < -\frac{\ln\{f-1\}}{\ln\left\{1-\frac{1}{2^{15}}\right\}} = t_M \qquad \square$$

As an example, with $f = 15$, t_M is just 86,475. Practically, ART achieves required reliability only for tag populations smaller than t_M. If population size is larger than t_M, ART requires $p \leq \frac{f}{2^{15}}$, which is practically not possible with C1G2 RFID tags. This limitation exists with all the existing estimation schemes but has never been addressed before.

Next, we present a strategy to estimate the sizes of arbitrarily large tag populations. The key idea is to first divide the entire population into smaller sub-populations of roughly equal sizes and then estimate the size of each sub-population independently. At the end, adding the estimated sizes of all sub-populations gives the estimate of number of tags in the entire population. The size of any sub-population should not require $\frac{f}{p} \geq 2^{15}$.

Next, we first calculate the number of sub-populations that ART should divide a given tag population into and then present a strategy to perform this division virtually (*i.e.*, requiring no manual division of tags). Maximum number of tags that a sub-population can have is given by Equation (2.37). Therefore, the minimum number of sub-populations that the entire tag population should be divided into is $\frac{t_m}{t_M}$, where t_m is calculated as explained in Section 2.5.4.

To divide the tag population into sub-populations, we use the SELECT command standardized in the C1G2 standard. The ID of a tag is stored in its memory at a specific memory address. The tag can retrieve any bits stored in its memory by specifying an appropriate address range. Using the SELECT command, a reader can broadcast an address range and a bit mask that specifies which tags should participate in an Aloha frame. Each tag compares the bit mask with the bits in the specified address range in its memory and participates in the frame only if the bit mask matches the specified bits in its memory. To divide the whole population into sub-populations of roughly equal sizes, we leverage the fact that in large populations, the expected number of tags whose IDs have the least significant bit (LSB) of 0 is approximately the same as the expected number of tags whose IDs have the LSB of 1. Similarly, the expected number of tags whose IDs have the two LSBs of 00 is approximately the same as the expected number of tags whose IDs have the two LSBs of 01, 10, or 11, and so on. Therefore, a reader can divide the tag population into 2^z groups of roughly equal sizes by specifying appropriate masks for the address range corresponding to the z LSBs of tag IDs. The value of z is given by $\lceil \log_2 \{\frac{t_m}{t_M}\} \rceil$.

To summarize, a reader first obtains the value of upper bound t_m. Second, it calculates the value of n and f_{op}. Third, it calculates the value of t_M using Equation (2.37). Fourth, it calculates $z = \lceil \log_2 \{\frac{t_m}{t_{\max}}\} \rceil$. Fifth, it executes 2^z independent estimation rounds for required reliability α and confidence interval β, where in each round it uses SELECT command with a unique z bit mask for the z LSBs of the tag IDs. In each independent estimation round, it uses $p = f_{\text{op}}\Big\{1 - \Big(\frac{1}{f_{\text{op}}-1}\Big)^{\frac{1}{t_m/2^z}}\Big\}$. Finally, it adds up all 2^z estimates to obtain the estimate of total number of tags in the population.

2.6.2 ART with Multiple Readers

We next discuss how to obtain t_m and $\tilde{t}$ using multiple readers with overlapping coverage. To obtain t_m using multiple readers, we can let each reader obtain the t_m value on its own and then sum them up as the final overall t_m because of two reasons. First, our requirement on t_m is only a rough upper bound with an error tolerance of over $1.64 \times t$. Second, deployment of multiple readers in practice often requires site surveys to ensure minimal overlapping between readers.

To obtain $\tilde{t}$ using multiple readers, we adapt the approach proposed by Kodialam *et al.* in [62], which uses a central controller for all readers. ART parameters β, α, t_m, p, n, and f_{op} have the same value across all readers. When a reader transmits seed R_i in its i^{th} frame, it does not generate R_i on its own, rather it uses the i^{th} seed R_i issued by the central controller. That is, each reader generates the same sequence of n seeds. In the i^{th} frames from different readers, because all readers use the same seed R_i, the slot number that a given tag chooses is the same (*i.e.*, $h(f, R_i, ID)$) in the frame of each reader covering this tag. Once a reader has completed its frame, it sends the frame to the central controller. The controller applies the logical OR on all the i^{th} frames from all readers, and gets a single i^{th} frame as if using a single reader. ART uses the n frames computed by logical OR to estimate the population size.

2.7 ART — Analysis

In this section, first we prove that the estimation time of ART is independent of the tag population size. Second, we briefly discuss the computational complexity of ART. Last, we perform an analytical comparison of ART with existing schemes to mathematically justify the faster speed of ART compared to existing schemes.

2.7.1 Independence of Estimation Time from Tag Population Size

There are three inputs to ART: confidence interval β, required reliability α, and a population of t tags where t is unknown. The total number of slots of ART, $(f_{\text{op}} + l) \times n$, actually does not depend on t. Intuitively, the larger t is, the smaller p is according to Equation (2.25). Although t plays an important role in computing p, n, and f individually, in formula $(f_{\text{op}} + l) \times n$ the impact of t eventually gets canceled out. Next, we prove this independence.

From Equation (2.30), we observe that the value of n depends on α, β, μ, σ and from Equation (2.32), we observe that the value of f_{op} depends upon β, μ, σ. Thus, the total number of slots $(f_{\text{op}} + l) \times n$ depends on α, β, μ, σ. The values of α and β are given constants and μ and σ are functions of q_1, as seen from Equations (2.12) and (2.13). To prove that $(f_{\text{op}} + l) \times n$ is independent of t, we have to prove that q_1 is independent of t. From Equation (2.1), we have $q_1 = 1 - (1 - \frac{p}{f})^t$. As we do not know the value of t, rather we know t_m, we use $q_1 = 1 - (1 - \frac{p}{f})^{t_m}$. Substituting the value of p using $t = t_m$ from Equation (2.25) into this expression of q_1, we get

$$q_1 = 1 - \left(1 - \frac{1}{f} \times f\left\{1 - \left(\frac{1}{f-1}\right)^{\frac{1}{t_m}}\right\}\right)^{t_m} = \frac{f-2}{f-1} \tag{2.38}$$

Thus the value of q_1 that we use to calculate μ and σ and consequently f_{op} and n is independent of tag population size t or the upper bound on tag population size t_m. Therefore, f_{op} and n depend only on α and β regardless of the value of t or t_m. The upper bound on tag population size t_m only affects the value of p. For ART to achieve the required reliability, this upper bound has to satisfy the condition $L_{tm} \leq t_m \leq U_{tm}$. If $t_m > t \times U_{tm}$, the required reliability will not be achieved because the value of p will become so small that enough number of tags will not participate in the frames. Regardless, the value of $(f_{\text{op}}+l)\times n$ stays the same. We have seen from Figure 2.7 that for all practical purposes, the value of t_m satisfies the requirement $L_{tm} \leq t_m \leq U_{tm}$ when calculated using the method proposed in Section 2.5.4.

2.7.2 Computational Complexity

The two most computationally intensive tasks in ART are the numerical solutions of Equation (2.12) to obtain the estimate $\tilde{t}$ and of Equation (2.32) to calculate f_{op}. Fortunately, these two equations need to be solved numerically only once during the estimation process: Equation (2.32) before executing the frames and Equation (2.12) after executing the frames. Consequently, the runtime complexity of ART is no larger than that of a standardized Aloha protocol. Almost all existing schemes involve numerical solutions of equations to obtain the estimate $\tilde{t}$. Therefore, the off-line computational complexity of ART is comparable to those of existing estimation schemes.

2.7.3 Analytical Comparison of Estimators

Next, we show that the ART estimator, namely the average run size of 1s, has less variance than many other framed slotted Aloha based estimators, namely (1) the size of the first run of 0s (used by FNEB [53]), (2) the average run size of 0s, (3) the total number of 0s (used by UPE [61] and EZB [62]), (4) the total number of 1s, (5) the total number of runs of 0s, and (6) the total number of runs of 1s. Higher the variance of an estimator, more number of rounds n are needed to improve reliability, and more rounds means larger estimation time.

Figure 2.8 shows the analytical plots of the variances of the ART estimator and the above six estimators with frame size $f = 16$ versus tag population sizes. This figure shows that *the variance of ART estimator is significantly lower than all other estimators.* Runs of 1s

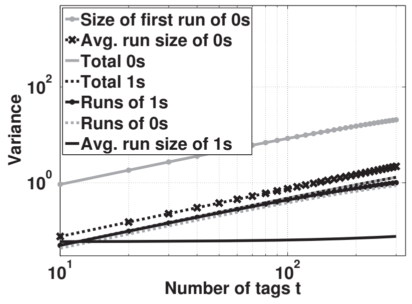


Figure 2.8 Variance of different estimators versus RFID tag population size

and runs of 0s have smaller variance compared to ART for very small tag population sizes. This observation, however, is insignificant because both these quantities are non-monotonic functions of tag population size and therefore, cannot be used alone for estimation. The variances of these estimators are calculated as follows. The variance of the total number of 0s and 1s is calculated using Equation (2.3). The variance of the size of the first run is calculated using Equation (3) in [102] by setting $i = 1$. The variance of the number of runs of 0s and that of 1s is calculated using Equation (2.5). We emphasize that plots in Figure 2.8 are not based on experimental results, instead, they are based on analytical formulas.

2.8 Performance Evaluation

We numerically evaluated in Matlab our ART scheme as well as four prior RFID estimation schemes: UPE [61], EZB[62], FNEB [53], and MLE [72]. We did not evaluate LoF [90] because it is non-compliant with C1G2 and CSE [127] because it does not take accuracy requirements as input. The estimation times for ART reported in this section include the time required to obtain the value of t_m. To ensure compliance with the C1G2 standard, in all our simulations, each tag picks up exactly one slot at the start of frame as soon as the reader broadcasts the frame size.

Next, we first conduct a side-by-side comparison on estimation time between ART and the four prior schemes. Then, we conduct experiments to show that ART indeed achieves the required reliability.

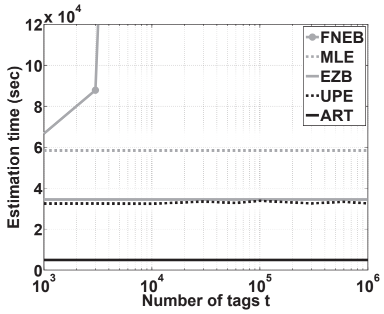
2.8.1 Estimation Time

The results in Figures 2.9, 2.10, and 2.11 show that *the estimation time of ART is significantly smaller than all prior schemes.* Note that in Figures 2.10 and 2.11, the plots for FNEB are out of the range of the vertical axes, and the plots of UPE and EZB are almost overlapping.

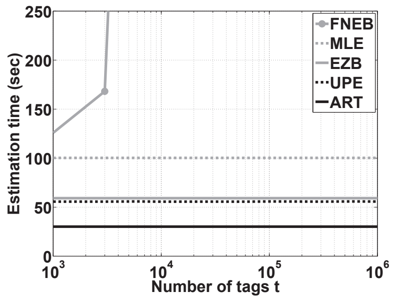
We make three main observations from Figures 2.9 (a), (b), and (c), which show the estimation time needed by each scheme with population sizes of up to one million tags for different configurations of α and confidence interval β. First, we observe that ART is faster than all four prior schemes in all these configurations. For $\alpha = 99.9\%$ and $\beta = 0.1\%$, ART is 7 times faster than the fastest prior estimation schemes, which are UPE [61] and EZB [62]. For $\alpha = 99\%$ and $\beta = 1\%$, ART is 1.96 times faster than UPE and EZB. For $\alpha = 95\%$ and $\beta = 5\%$, ART is 1.68 times faster than UPE and EZB. Second, we observe that ART, UPE, EZB, and MLE perform estimation in constant time, which attributes to the use of persistence probabilities. Third, we observe that FNEB, whose estimator is the size of the first run of 0s, is the slowest. This concurs with our analytical analysis in Figure 2.8, where we show that FNEB has the largest variance. The larger the variance of an estimator, the more the rounds of execution needed to achieve the required reliability, and the longer the estimation time.

We make three main observations from Figures 2.10 (a), (b), and (c), which show the estimation time of each scheme for 5000 tags with the required reliability α varying from 90% to 99.9% for different configurations of confidence interval β. First, we observe that ART is faster than all four prior estimation schemes in all these configurations. Second, the difference between the estimation time of ART and those of prior schemes increases as the required reliability increases. For example, for $\beta = 5\%$ and $\alpha = 95\%$, ART is 1.68 times faster than UPE and EZB while for $\beta = 0.1\%$ and $\alpha = 99.9\%$, it is 7 times faster. This shows that ART becomes more and more advantageous over existing schemes when the required reliability increases. Third, for all schemes, the estimation time increases as the required reliability increases because more number of rounds are needed to achieve the required reliability. We further observe that ART's estimation time increases at the lowest rate as the required reliability increases because its estimator has the smallest variance.

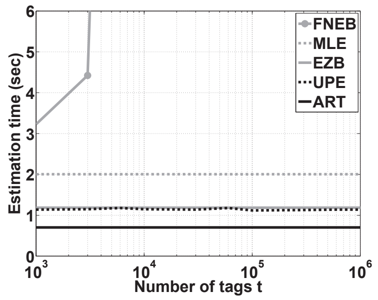
We make three main observations from Figures 2.11 (a), (b), and (c), which show the estimation time of each scheme for 5000 tags with the confidence interval β varying from 0.1% to 10% for different configurations of α. First, we observe that ART is faster than all estimation schemes in all these configurations. Second, for all schemes, the estimation time decreases as the confidence interval increases because lesser number of rounds are needed to achieve the required reliability.



(a) $\alpha = 99.9\%, \beta = 0.1\%$

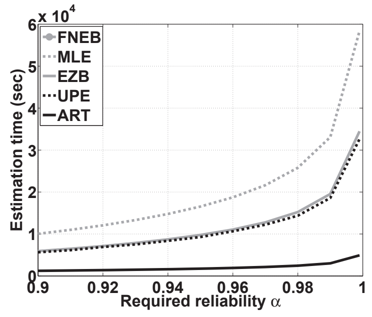


(b) $\alpha = 99\%, \beta = 1\%$

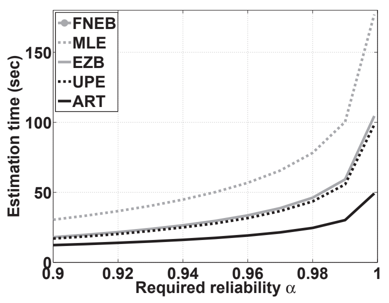


(c) $\alpha = 95\%, \beta = 5\%$

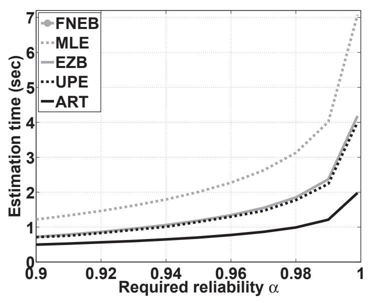
Figure 2.9 Estimation time vs. tag population size of ART and existing schemes



(a) $\beta = 0.1\%$

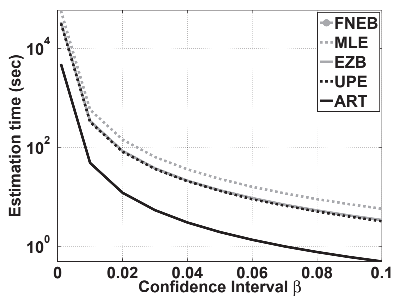


(b) $\beta = 1\%$

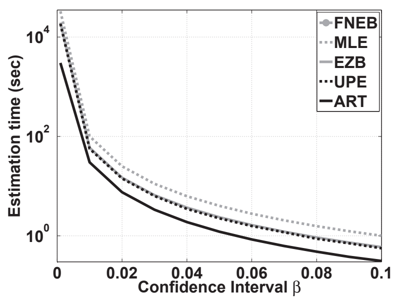


(c) $\beta = 5\%$

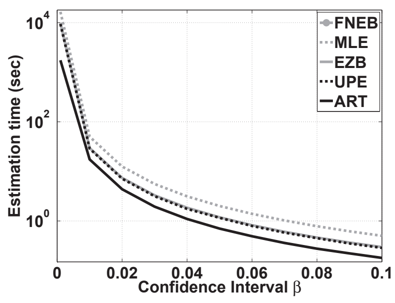
Figure 2.10 Estimation time vs. required reliability for ART and existing schemes



(a) $\alpha = 99.9\%$

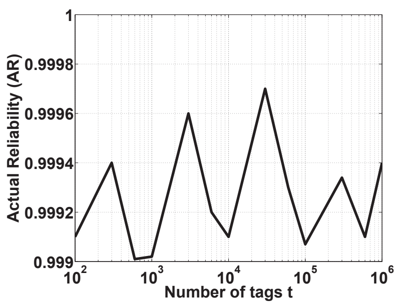


(b) $\alpha = 99\%$

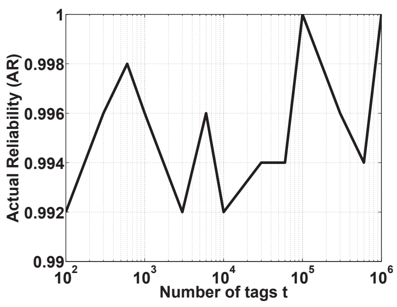


(c) $\alpha = 95\%$

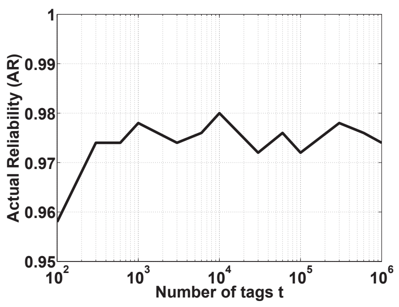
Figure 2.11 Estimation time vs. confidence interval for ART and existing schemes



(a) $\alpha = 99.9\%, \beta = 0.1\%$



(b) $\alpha = 99\%, \beta = 1\%$



(c) $\alpha = 95\%, \beta = 5\%$

Figure 2.12 Actual reliability achieved by ART for three different requirements

2.8.2 Actual Reliability

The subfigures in Figure 2.12 show the actual reliability of ART versus the number of tags for different configurations of required reliability α and confidence interval β. We observe that *ART always achieves the required reliability.* These figures show several ups and downs in the plotted values. These ups and downs are not because of any noise, rather we see them because of the magnification level of vertical axis in these figures.

2.9 Conclusion

The key technical novelty of this chapter is in proposing the new estimator, the average run size of 1s, for estimating RFID tag population size of arbitrarily large sizes. Using analytical plots, we show that our estimator has much smaller variance compared to other estimators including those used in prior work. It is this smaller variance that makes our scheme faster than the previous ones. The key technical depth of this chapter is in the mathematical development of the estimation theory using this estimator. ART can estimate arbitrarily large tag populations with arbitrarily high accuracy. It works with single as well as multiple readers. Our experimental results show that ART is significantly faster than all prior RFID estimation schemes. We have shown, both theoretically and experimentally, that the estimation time of ART is independent of the tag population size.

3 RFID Identification

3.1 Introduction

3.1.1 Background and Problem Statement

As the cost of commercial RFID tags, which is as low as 5 cents per tag [93], has become negligible compared to the prices of the products to which they are attached, RFID systems are being increasingly used in various applications such as supply chain management [67], indoor localization [86], 3D positioning [123], object tracking [85], inventory control, electronic toll collection, and access control [46, 84]. For example, Walmart has started to use RFID tags to track jeans and underwear for better inventory control. Large warehouses, such as those of Amazon with sizes up to 1 million ft^2 [22], or distribution centers with sizes up to 3 million ft^2 [1], contain hundreds of thousands of items. RFID systems can make the inventory management and tracking in these large warehouses and distribution centers much easier and error free. An RFID system consists of tags and readers. A tag is a microchip combined with an antenna in a compact package that has limited computing power and communication range. There are two types of tags: (1) passive tags, which do not have their own power source, are powered up by harvesting the radio frequency energy from readers, and have communication ranges often less than 20 feet; (2) active tags, which come with their own power sources and have relatively longer communication ranges. A reader has a dedicated power source with significant computing power. RFID systems mostly work in a query-response fashion where a reader transmits queries to a set of tags and the tags respond with their IDs over a shared wireless medium.

This chapter addresses the fundamental RFID *tag identification* problem, namely reading all IDs of a given set of tags, which is needed in almost all RFID systems. Because tags respond over a shared wireless medium, tag identification protocols are also called *collision arbitration*, *tag singulation*, or *tag anti-collision* protocols. Tag identification protocols need to be scalable as the number of tags that need to be identified could be as large as tens of thousands with the increasing adoption of RFID tags. An RFID system with a large number of tags may require multiple readers with overlapping regions. In this chapter, we first focus on the *single reader* version of the tag identification problem and then extend our solution to the *multiple reader* problem.

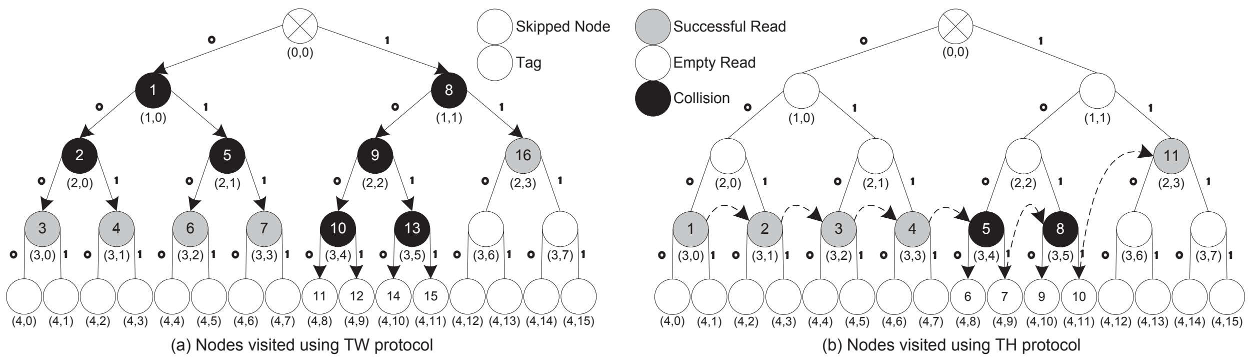

Skipped Node
Tag
Successful Read
Empty Read
Collision
(a) Nodes visited using TW protocol
(b) Nodes visited using TH protocol

Figure 3.1 Identifying a population of 9 tags using TW and TH.

3.1.2 Summary and Limitations of Prior Art

The industrial standard, EPCGlobal Class 1 Generation 2 (C1G2) RFID [55], adopted two tag identification protocols, namely framed slotted Aloha and Tree Walking (TW). In framed slotted Aloha, a reader first broadcasts a value f to the tags in its vicinity where f represents the number of time slots present in a forthcoming frame. Then each tag whose inventory bit is 0 randomly picks a time slot in the frame and replies during that slot. Each C1G2 compliant tag has an inventory bit, which is initialized to be 0. In any slot, if exactly one tag responds, the reader successfully gets the ID of that tag and issues a command to the tag to change its inventory bit to 1. The key limitation of framed slotted Aloha is that it can not identify large tag populations due to the finite possible size of f. Qian *et al.* have shown that framed slotted Aloha is most efficient when f is equal to the number of tags [89]. Therefore, although theoretically any arbitrarily large tag population can be identified by indefinitely increasing the frame size, practically this is infeasible because during the entire identification process, Aloha based protocols require all tags, including those that have been identified, to stay powered up and listen to all the messages from the reader in order to maintain the value of the inventory bit. This results in high instability because any intermittent loss of power at a tag will set its inventory bit back to 0, leading the tag to contend in the subsequent frame. The instability of Aloha based protocols has formally been proven by Rosenkrantz and Towsley in [94].

TW is a fundamental multiple access protocol, which was first invented by U.S. Army for testing soldiers for syphilis during World War II [44]. TW was proposed as an RFID tag identification protocol by Law *et al.* in [66]. In TW, a reader first queries 0 and all the tags whose IDs start with 0 respond. If result of the query is a successful read (*i.e.*, exactly one tag responds) or an empty read (*i.e.*, no tag responds), the reader queries 1 and all the tags whose IDs start with 1 respond. If the result of the query is a collision, the reader generates two new query strings by appending a 0 and a 1 at the *end* of the previous query string and queries the tags with these new query strings. All the tags whose IDs start with the new query string respond. This process continues until all the tags have been identified. This identification process is essentially a partial Depth First Traversal (DFT) on the complete binary tree over the tag ID space, and the actual traversal forms a binary tree where the leaf nodes represent successful or empty reads and the internal nodes represent collisions. Nodes on level l correspond to l^{th} most significant bit of the tag IDs. Figure 3.1(a) shows the tree walking process for identifying 9 tags over a tag ID space of size 2^4. Here a `successful read` node is one that an identification protocol visits and there is exactly one tag in the subtree rooted at this node, an `empty read` node is one that an identification protocol visits and there is no tag in the subtree rooted at this node, and a `collision` node is one that an identification protocol visits and there are more than one tags in the subtree rooted at this node. The key limitation of TW based protocols is that they visit a large number of collision nodes in the binary tree, which makes the identification process slow. Although several heuristics have been proposed to reduce the number of visits to collision nodes [87, 83], all these heuristics based methods are not guaranteed to minimize such futile visits. Prior Aloha-TW hybrid protocols also have this limitation.

3.1.3 System Model

As most commercially available tags and readers already comply with the C1G2 standard, we do not assume changes to either tags or their physical protocol. We assume that readers can be reprogrammed to adopt new tag identification software. For reliable tag identification, we are given the probability of successful query-response communication between the reader and a tag.

3.1.4 Proposed Approach

To address the fundamental limitations that lie in the heuristic nature of prior TW based protocols, we propose a new approach to tag identification called Tree Hopping (TH). The key novel idea of TH is to formulate the tag identification problem as an optimization problem and find the optimal solution that ensures either minimal expected number of queries (*i.e.*, nodes visited on the binary tree) or minimal expected identification time, as per the requirement. In TH, we first quickly estimate the tag population size. Second, based on the estimated tag population size, we calculate the optimal level to start tree traversal so that the expected number of queries or expected identification time is minimal, *hop* directly to the left most node on that level, and then perform DFT on the subtree rooted at that node. Third, after that subtree is traversed, we re-estimate the size of remaining unidentified tag population, re-calculate the new optimal level, hop directly to the new optimal node, and perform DFT on the subtree rooted at that node. Hopping to optimal nodes in this manner skips a large number of collision nodes. This process continues until all the tags have been identified. Figure 3.1(b) shows the nodes traversed by TH for the same population of 9 tags as in Figure 3.1(a). Here a `skipped node` is one that TW visits but TH does not. We can see that TH traverses 11 nodes to identify these 9 tags. In comparison, TW traverses 16 nodes as shown in Figure 3.1(a). This difference scales significantly as tag population size increases.

3.1.4.1 Population Size Estimation

TH first uses a framed slotted Aloha based method to quickly estimate the tag population size. For this, TH requires each tag to respond to the reader with a probability q. As C1G2 compliant tags do not support this probabilistic responding, we implement this by "virtually" extending the frame size $\frac{1}{q}$ times. To estimate the tag population size, the reader announces a frame size of $\frac{1}{q}$ but terminates it after the first slot. To terminate a frame, the reader issues a SELECT command, specified in the C1G2 standard, with its *position*, *target*, and *action* parameters set to 0. This command "resets" all tags and they go into a state where they expect a new frame to start. For further details on frame termination, see Section 6 of [55]. The reader issues several single-slot frames while reducing q with a geometric distribution (*i.e.*, $q = \frac{1}{2^{i-1}}$ in i^{th} frame) until the reader gets an empty slot. Suppose the empty slot occurred in the ith frame, TH estimates the tag population size to be $1.2897 \times 2^{i-2}$ based on Flajolet and Martin's algorithm used in databases [47, 90].

3.1.4.2 Finding Optimal Level

To determine the optimal level γ_{op} that TH directly hops to, we first calculate the expected number of nodes that TH will visit or expected identification time that TH will take if it starts DFTs from nodes on any given level γ. Let b be the number of bits in each tag ID (which is 64 for C1G2 compliant tags), then, we have $1 \leq \gamma \leq b$. If γ is small, more collision nodes will be visited while if it is large, more empty read nodes will be visited. Our objective is to calculate an optimal level γ_{op} that will, depending on the requirement, result in either the smallest number of nodes visited or the smallest identification time. To find γ_{op} for minimizing number of queries, we first derive the expression for calculating the expected number of nodes visited by TH if TH directly hops to level γ. Then we calculate the value of γ which minimizes this expression. This value of γ is the value of optimal level γ_{op}. We present the technical details of finding γ_{op} in Section 3.3. In Section 3.4, we derive the expression for calculating the expected identification time of TH if TH directly hops to level γ. We use this expression to calculate γ_{op} when we need to minimize the identification time instead of number of queries.

3.1.4.3 Population Size Re-estimation

If the tags that we want to identify are uniformly distributed in the ID space $[0, 2^b - 1]$, then performing DFTs from each node on level γ_{op} will result in minimum number of nodes visited. However, in reality, the tags may not be uniformly distributed. In such cases, each time when the DFT of a subtree is finished, TH needs to re-estimate the total tag population size to find the next optimal level and the hoping destination node. TH performs the re-estimation as follows. Let z be the first tag population size estimated using the Aloha based method, x be the number of tags that have been identified, and s be the size of the tag ID space covered by the nodes visited. Naturally, $z - x$ is an estimate of the remaining tag population size; however, we cannot use this estimate to calculate the next optimal level because the remaining leftover ID space may not form a complete binary tree. Instead, based on the node density in the remaining ID space, TH extrapolates the total tag population size to be $\frac{z-x}{2^b-s} \times 2^b$ and uses it to find the next hopping destination node. Note that if tags are uniformly distributed, we have $\frac{z-x}{2^b-s} \times 2^b = z$.

3.1.4.4 Finding Hopping Destination

Each time after a DFT is done and the new optimal level is recalculated, TH needs to find the next node to hop to, which may not be the leftmost node on the optimal level. Consider the example shown in Figure 3.1(b). Assuming a uniform distribution, the optimal level to start the DFT is 3. In this chapter, we use (l, p) to denote the p^{th} node on level l. TH performs DFTs on the subtrees of nodes $(3, 0)$ to $(3, 5)$ and identifies 8 out of 9 tags. Based on the number of remaining tags after the last DFT, which is 1, the optimal level for the next hop is changed from 3 to 1. However, if TH starts the DFT from the leftmost node on level 1, which is $(1, 0)$, it will result in identifying all tags in its subtree again which is wasteful. Similarly, if TH starts the DFT from the second leftmost node on level 1, which is $(1, 1)$, it will visit the subtree of $(2, 2)$, which is wasteful as all the tags in the subtree of $(2, 2)$ have already been identified. Similarly, if there had been a third leftmost node on the new optimal level and if TH starts the DFT from that third left most node, it will not visit the subtree of $(2, 3)$, resulting in tag $(4, 13)$ not being identified. To avoid both scenarios, *i.e.*, some subtrees being traversed multiple times and some subtrees with tags not being traversed, after the optimal level is recalculated, TH hops to the root of the largest subtree that can contain the next tag to be identified but does not contain any previously identified tag. The level at which this root is located can not be smaller than the new optimal level. For the example in Figure 3.1(b), after the subtree rooted at node $(2, 2)$ has been traversed, the recalculated optimal level is 1 and the next node that TH hops to is $(2, 3)$.

Our experimental results in Figure 3.2 show that when the tags are not uniformly distributed in the ID space, our technique of dynamically adjusting γ_{op} according to the leftover population size significantly reduces the total number of queries and the average number of responses per tag. The two curves "TH w re-estimation-Seq" and "TH w/o re-estimation-Seq" show the total number of queries needed, respectively, with and without the dynamic adjustment of γ_{op} for non-uniformly distributed tag IDs. For example, for 10K tags, this dynamic level adjustment reduces the total number of queries by 31.5%. Our experimental results in Figure 3.2 also show that when the tags are uniformly distributed in the ID space, there is no need to dynamically adjust γ_{op}. The two curves "TH w re-estimation-Uni" and "TH w/o re-estimation-Uni" show the total number of queries needed, respectively, with and without the dynamic adjustment for uniformly distributed tag IDs. These two curves are similar because for uniformly distributed tag IDs, γ_{op} does not usually change after each DFT and thus the benefit of dynamically adjusting γ_{op} is relatively small. Our experimental results in Figure 3.2 further show that the performance of TH on non-uniformly distributed populations is asymptotically the same as its performance on uniformly distributed populations when it uses the technique of dynamically adjusting γ_{op} according to the leftover population size. The curve "TH w re-estimation-Seq" approaches the curves "TH w re-estimation-Uni" and "TH w/o re-estimation-Uni" as the tag population size increases.

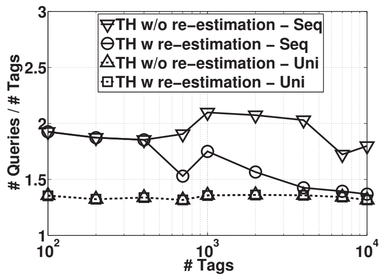


Figure 3.2 Impact of dynamic adjustment of γ_{op} on different types of populations.

3.1.4.5 Population Distribution Conversion

Although dynamically adjusting γ_{op} for a non-uniformly distributed population reduces the number of queries, the number of queries is still not as low as it would have been had the population been uniformly distributed. Furthermore, the extent of reduction depends on the distribution and size of the population. In Section 3.5.1, we present a simple technique that TH uses to virtually convert almost any non-uniformly distributed population into near-uniformly distributed population. The key idea is that instead of comparing the query strings, transmitted by the reader, with the starting bits of the tag ID, each tag compares the query string with the ending bits of its ID. The resulting binary tree has all the tags near-uniformly distributed in the ID space. We will show that this can be implemented without any modifications to the physical communication protocol and the tags. This technique, combined with the dynamic level adjustment, enables TH to identify any non-uniformly distributed population in almost the same number of queries or time as for uniformly distributed population of the same size. In what follows, we first assume that tags compare the query string with the starting bits of its ID, as in TW protocol, until Section 3.5.1 where we explain this technique in detail.

3.2 Related Work

We review existing identification protocols, which can be classified as nondeterministic, deterministic, or hybrid.

3.2.1 Nondeterministic Identification Protocols

Existing such protocols are either based on framed slotted Aloha [129] or Binary Splitting (BS) [35]. As we discussed above, Aloha based protocols only work for small tag populations. In BS [35], the identification process starts with the reader asking the tags to respond. If more than one tags respond, BS divides and subdivides the population into smaller groups until each group has only one or no tag. This process of random subdivision incurs a lot of collisions. Furthermore, BS requires the tags to perform operations that are not supported by the C1G2 standard. ABS is a BS based protocol that is designed for continuous identification of tags [82].

3.2.2 Deterministic Identification Protocols

There are 3 such protocols: (1) the basic TW protocol [66], (2) the Adaptive Tree Walking (ATW) protocol [115], and (3) the TW-based Smart Trend Traversal (STT) protocol [87].

ATW is an optimized version of TW that always starts DFTs from the level of $\log z$, where z is the size of tag population. This is the traditional wisdom for optimizing TW. The key limitation of ATW is that it is optimal only when all tag IDs are evenly spaced in the ID space; however, this is often not true in real-world applications. In contrast, during the identification process, our TH protocol adaptively chooses the optimal level to hop to based on distribution of IDs. STT improves TW using some ad-hoc heuristics to select prefixes for next queries based upon the type of response to previous queries. It assumes that the number of tags identified in the past k queries is the same as the number of tags that will be identified in the next k queries. This may not be true in reality.

3.2.3 Hybrid Identification Protocols

Hybrid protocols combine features from nondeterministic and deterministic protocols. There are two major such protocols: Multi slotted scheme with Assigned Slots (MAS) [83] and Adaptively Splitting-based Arbitration Protocol (ASAP) [89]. MAS is a TW-based protocol in which each tag that matches the reader's query picks up one of the f time slots to respond. For large populations, due to the finite practical size of f, for queries corresponding to higher levels in the binary tree, the response in each of the f slots is most likely a collision, which increases the identification time. ASAP divides and subdivides the tag population until the size of each subset is below a certain threshold and then applies Aloha on each subset. For this, ASAP requires tags to pick slots using a geometric distribution, which makes it incompliant with the C1G2 standard. Furthermore, subdividing the population before identification is in itself very time consuming.

3.3 Optimal Tree Hopping

After quick population size estimation using Flajolet and Martin's algorithm [47], TH needs to find the optimal level to hop to. First, we derive an expression to calculate the expected number of queries (*i.e.*, the number of nodes that TH will visit) if it starts DFTs from the nodes on level γ, assuming that tags are uniformly distributed in the ID space. The expression to calculate the expected identification time will be derived in Section 3.4. Second, as the derived expression is too complex to calculate the optimal value of γ that minimizes the expected number of queries by simply differentiating the expression with respect to γ, we present a numerical method to calculate the optimal level γ_{op}. If tags are not uniformly distributed, each time when the DFT on a node is completed, as stated in Section 6.1.2, TH re-estimates the total population size based on the initial estimate and the number of tags that have been identified, re-calculates the new optimal level, and finds the hopping destination node.

3.3.1 Average Number of Queries

Let random variable Q denote the total number of nodes that TH visits to identify all tags. Note that each node visit corresponds to one reader query. We next calculate $E[Q]$. Let $I(l,p)$ be an indicator random variable whose value is 1 if and only if node (l,p) is visited. Thus, Q is the sum of $I(l,p)$ for all l and all p.

$$Q = \sum_{l=1}^{b} \sum_{p=0}^{2^l-1} I(l,p) \tag{3.1}$$

Let $P\{(l,p)\}$ be the probability that TH visits node (l,p). Thus, $E[Q]$ can be expressed as follows:

$$E[Q] = \sum_{l=1}^{b} \sum_{p=0}^{2^l-1} P\{(l,p)\} \tag{3.2}$$

Next, we focus on expressing $P\{(l,p)\}$ using variable γ, where γ denotes the level that TH hops to. Recall that TH skips all nodes on levels from 1 to $\gamma-1$ and performs DFT on each of the 2^γ nodes on level γ, where $1 \le \gamma \le b$. Note that the root node of the whole binary tree is always meaningless to visit as it corresponds to a query of length 0. Here $P\{(l,p)\}$ is calculated differently depending on whether node (l,p) is the left child of its parent or the right. Let $P_l\{(l,p)\}$ and $P_r\{(l,p)\}$ denote the probability of visiting (l,p) when (l,p) is the left and right child of its parent, respectively. If the estimated total number of tags z is zero, then $P_l\{(l,p)\} = P_r\{(l,p)\} = 0$ for all l and p. Below we assume $z > 0$. As TH skips all nodes from levels 1 to $\gamma - 1$, we have

$$P_l\{(l,p)\} = P_r\{(l,p)\} = 0 \textbf{ if } 1 \leq l < \gamma \tag{3.3}$$

As TH performs DFT from each node on level γ, it visits each node on this level. Thus, we have

$$P_l\{(l,p)\} = P_r\{(l,p)\} = 1 \textbf{ if } l = \gamma \tag{3.4}$$

For each remaining level $\gamma < l \leq b$, when (l,p) is the left child of its parent, $P_l\{(l,p)\}$ is equal to the probability that the parent of (l,p) is a collision node. When (l,p) is the right child of its parent, if the parent is a collision node and $(l,p-1)$ is an empty read node, then (l,p) will also be a collision node. Thus, instead of visiting (l,p), TH should directly hop to the left child of (l,p). Therefore, $P_r\{(l,p)\}$ is equal to the probability that the parent of (l,p) is a collision node and $(l,p-1)$ is not an empty read node.

Let k denote the number of tags *covered* by the parent of node (l,p) (*i.e.*, the number of tags that are in the subtree rooted at the parent of (l,p)). Let $m = 2^{b-l+1}$ denote the maximum number of tags that the parent of (l,p) can cover and $n = 2^b$ denote the maximum number of tags that can be accommodated in the whole ID space. The probability that the parent of (l,p) covers k of z tags follows a hypergeometric distribution:

$$P\{\#\text{tags} = k\} = \frac{\binom{m}{k}\binom{n-m}{z-k}}{\binom{n}{z}} \tag{3.5}$$

Let P_e be the probability that the parent of (l,p) is an empty read. Thus,

$$P_e = P\{\#\text{tags} = 0\} = \frac{\binom{n-m}{z}}{\binom{n}{z}} \tag{3.6}$$

Let P_s be the probability that the parent of (l,p) is a successful read. Thus,

$$P_s = P\{\#\text{tags} = 1\} = \frac{m\binom{n-m}{z-1}}{\binom{n}{z}} \tag{3.7}$$

Let P_c be the probability that the parent of (l, p) is a collision node. Thus,

$$P_c = 1 - (P_e + P_s) = 1 - \frac{\binom{n-m}{z}}{\binom{n}{z}} - \frac{m\binom{n-m}{z}}{\binom{n}{z}} \tag{3.8}$$

Next we calculate $P_l\{(l,p)\}$ and $P_r\{(l,p)\}$ for $\gamma < l \leq b$ for the following three cases: $n - m < z - 1$, $n - m = z - 1$, and $n - m > z - 1$. Note that $n - m$ is the size of the ID space that is not covered by the parent of (l, p), and $z - k$ is the remaining number of tags that are not covered by the parent of (l, p). Thus, $z - k \leq n - m$.

Case 1 $n - m < z - 1$. In this case, $z - k \leq n - m < z - 1$, which means $k \geq 2$. Thus, as the parent of (l, p) covers at least two tags, it must be a collision node, *i.e.*$P_c = 1$. Thus, if (l, p) is the left child of its parent, TH for sure visits it:

$$P_l\{(l,p)\} = 1 \tag{3.9}$$

If (l, p) is the right child of its parent, TH visits it if and only if node $(l, p-1)$, which is the left sibling of (l, p), is not an empty read. If $(l, p-1)$ is an empty read, as its parent is a collision node, (l, p) must also be a collision node, which means that TH will directly visit the left child of (l, p) instead of (l, p). The size of the ID space covered by $(l, p-1)$ is $\frac{m}{2}$. If $n - \frac{m}{2} \leq z - 1$, then node $(l, p-1)$ covers at least one tag, which means that $(l, p-1)$ is not an empty read and TH for sure visits (l, p), *i.e.*, $P_r\{(l,p)\} = 1$. If $n - \frac{m}{2} > z - 1$, then the probability that TH visits (l, p) is equal to the probability that $(l, p-1)$ is not an empty read, which is $1 - \binom{n-\frac{m}{2}}{z}/\binom{n}{z}$ based on Equation (3.6). Finally, we have

$$P_r\{(l,p)\} = \begin{cases} 1 - \frac{\binom{n-\frac{m}{2}}{z}}{\binom{n}{z}} & \textbf{if } n - \frac{m}{2} > z - 1 \\ 1 & \textbf{if } n - \frac{m}{2} \leq z - 1 \end{cases} \tag{3.10}$$

Case 2 $n - m = z - 1$. In this case, $z - k \leq n - m = z - 1$, which means $k \geq 1$. As the parent of (l, p) covers $k \geq 1$ tags, the probability of the parent of (l, p) being an empty read is 0 and the probability of the parent of (l, p) being a successful read is $m\binom{n-m}{z-1}/\binom{n}{z} =$

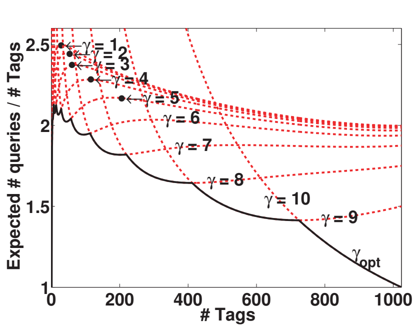


Figure 3.3 Norm. $E[Q]$ vs. pop. size $\forall\gamma$

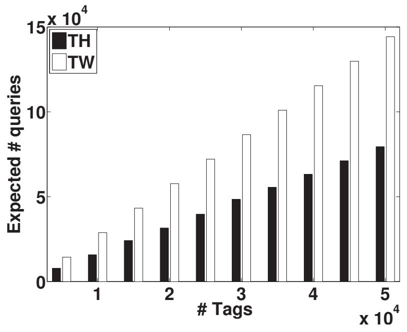


Figure 3.4 $E[Q]$: TH vs. TW

$m\binom{z-1}{z-1}/\binom{n}{z} = m/\binom{n}{z}$ based on Equation (3.7). If (l, p) is the left child of its parent, then TH visits it if and only if the parent of (l, p) is a collision node. Thus, the probability of visiting (l, p) is equal to the probability of the parent of (l, p) being a collision node, which is equal to $1 - P_e - P_s$. Thus, we have

$$P_l\{(l,p)\} = 1 - P_e - P_s = 1 - \frac{m}{\binom{n}{z}} \tag{3.11}$$

If (l, p) is the right child of its parent, then TH visits it if and only if both the parent of (l, p) is a collision node and $(l, p-1)$ is not an empty read. The probability that the parent of (l, p) is a collision node is $1 - m/\binom{n}{z}$ as calculated above. Given that the parent of (l, p) is a collision node, the probability that $(l, p-1)$ is an empty read is $\left(\binom{n-\frac{m}{2}}{z} - \frac{m}{2}\right)/\left(\binom{n}{z} - m\right)$.

$$P_r\{(l,p)\} = \left[1 - \frac{m}{\binom{n}{z}}\right].\left[1 - \frac{\binom{n-\frac{m}{2}}{z} - \frac{m}{2}}{\binom{n}{z} - m}\right] \tag{3.12}$$

Case 3 $n - m > z - 1$. In this case, $k \geq 0$. Similar to the calculations above, as per Equations (3.6) and (3.7), we have:

$$P_l\{(l,p)\} = 1 - P_e - P_s = 1 - \frac{\binom{n-m}{z} + m\binom{n-m}{z-1}}{\binom{n}{z}} \tag{3.13}$$

$$P_r\{(l,p)\} = \left[1 - \frac{\binom{n-m}{z} + m\binom{n-m}{z-1}}{\binom{n}{z}}\right] \times \left[1 - \frac{\binom{n-\frac{m}{2}}{z} - \left\{\binom{n-m}{z} + \frac{m}{2}\binom{n-m}{z-1}\right\}}{\binom{n}{z} - \left\{\binom{n-m}{z} + m\binom{n-m}{z-1}\right\}}\right] \tag{3.14}$$

Finally, Equations (3.3) through (3.14) completely define the probabilities $P_l\{(l,p)\}$ and

$P_r\{(l,p)\}$. Note that as tags are uniformly distributed, the probability of visiting node (l,p) is independent of the horizontal position p.

The expected number of queries can now be calculated using Theorem 6.

Theorem 6. *For a population of z tags uniformly distributed in the ID space, where each tag has an ID of b bits, if TH hops to level γ to perform DFT from each node on this level, the expected number of queries for identifying all z tags is:*

$$E[Q] = 2^{\gamma} + \sum_{l=\gamma+1}^{b} 2^{l-1}[P_l\{(l,p)\} + P_r\{(l,p)\}] \tag{3.15}$$

Proof. First, on level γ, all the 2^{γ} nodes are visited by TH. Second, on any level l where $\gamma + 1 \leq l \leq b$, the probabilities of left and right nodes being visited are $P_l\{(l,p)\}$ and $P_r\{(l,p)\}$ respectively. As there are 2^{l-1} pairs of left and right nodes on level l, the expected number of nodes visited by TH on level l is $2^{l-1}[P_l\{(l,p)\} + P_r\{(l,p)\}]$. □

When $\gamma = 1$, Equation (3.15) is also the analytical model for calculating expected number of queries of TW protocol.

3.3.2 Calculating Optimal Hopping Level

Equation (3.15) shows that $E[Q]$ is a function of γ as $n = 2^b$, $m = 2^{b-l+1}$, and b is given. For any given z, we want to find the optimal level $\gamma = \gamma_{\text{op}}$ so that $E[Q]$ is minimal. The conventional approach to finding the optimal variable value that minimizes a given function is to differentiate the function with respect to that variable, equate the resulting expression to zero, and solve the equation to obtain the optimal variable value. However, it is very difficult, if not impossible, to use this approach to find the optimal level because Equation (3.15) for calculating $E[Q]$ is too complex.

Next, we present a numerical method to find the optimal level. First, we define *normalized* $E[Q]$ as the ratio of $E[Q]$ to tag population size. Figure 3.3 shows the plots of normalized $E[Q]$ vs. the number of tags for different γ values ranging from 1 to b (here we used $b = 10$

for illustration). From this figure, we observe that for any tag population size, there is a unique optimal value of γ. For example, for a population of 600 tags, $\gamma_{\text{op}} = 9$. Second, we define *crossover points* as follows: *for a given ID length b, the crossover points are the tag population sizes $c_0 = 0, c_1, c_2, \cdots, c_{b+1} = 2^b$ such that for any tag population size in $[c_i, c_{i+1})$ $(0 \leq i \leq b)$, $\gamma_{op} = i$.* These crossover points are essentially the x-coordinates of the intersection points of the normalized $E[Q]$ curves of consecutive values of γ in Figure 3.3. Thus, the value of c_i can be obtained by putting $z = c_i$ and numerically solving $E[Q, \gamma = i - 1] = E[Q, \gamma = i]$ for c_i using the bisection method. Once c_i is calculated for each $1 \leq i \leq b$, γ_{op} for a given z can be obtained by simply identifying the unique interval $[c_i, c_{i+1})$ in which z lies and then using $\gamma_{\text{op}} = i$. The solid line in Figure 3.3 is plotted using the values of γ_{op} obtained using the proposed strategy. As values of c_i only depend on b, it is a one time cost to calculate them.

We next conduct an analytical comparison between the expected number of queries for TH and that for TW. Figure 3.4 shows the expected number of queries for TH, which is calculated using Equation (3.15) using $\gamma = \gamma_{\text{op}}$, and that for TW, which is calculated using Equation (3.15) using $\gamma = 1$, for 64 bit tag IDs. We observe that TH significantly outperforms TW for the expected number of queries. For example, for a population of 10K tags, the expected number of queries for TH is only 54% of that for TW. We will present detailed experimental comparison between TH and other protocols in Section 6.9.

3.3.3 Maximum Number of Queries

Although the primary goal of our TH protocol is to minimize the average number of queries, next, we analyze the maximum number of queries of TH and analytically show that it is still smaller than that of TW. The maximum number of queries that TH may need to identify z tags with b-bit IDs is shown in Theorem 7.

Theorem 7. *Let V denote the number of queries that TH may need to identify a population* *of $z \geq 2$ tags with b-bit IDs using $\gamma = \gamma_{op}$. We have*

$$V \leq z(b - \gamma_{\text{op}} + 1) - 2^{\gamma_{\text{op}}} + 2\theta_0 - \theta_1(b - \gamma_{\text{op}} - 1) \tag{3.16}$$

where

$$\begin{aligned} \theta_0 &= 2^{\gamma_{\text{op}}} - \left\lceil \frac{z}{2^{b-\gamma_{\text{op}}}} \right\rceil \\ \theta_1 &= \left\lceil \frac{z}{2^{b-\gamma_{\text{op}}}} \right\rceil - \left\lceil \frac{z-1}{2^{b-\gamma_{\text{op}}}} \right\rceil \left\lceil 1 - \frac{\gamma_{\text{op}}}{b} \right\rceil \end{aligned}$$

Proof. Let V_{TW} denote the number of queries that TW may need to identify $z \geq 2$ tags with b-bit IDs. The upper bound of V_{TW} is given as follows (proven in [66]):

$$V_{TW} \leq z(b + 1 - \log \frac{z}{2}) - 1 \tag{3.17}$$

Because $z \geq 2$, we have $V_{TW} \leq z(b + 1) - 1$.

When z tags are uniformly distributed in the ID space, TH essentially performs TW on all subtrees rooted at nodes on level γ_{op}. Let θ_0 and θ_1 denote the number of subtrees covering 0 and 1 tags, respectively. For these $\theta_0 + \theta_1$ subtrees, TH only visits the roots, which are at level γ_{op}. Let α denote the number of remaining subtrees (*i.e.*, $\alpha = 2^{\gamma_{\text{op}}} - \theta_0 - \theta_1$) and T_i denote a subtree covering $z_i \geq 2$ tags. For each subtree T_i, the maximum number of nodes that TH visits is $z_i(b - \gamma_{\text{op}} + 1) - 1$. Summing all $2^{\gamma_{\text{op}}}$ subtrees, we have

$$\begin{aligned} V &\leq \sum_{i=0}^{\alpha-1} \Big(z_i(b - \gamma_{\text{op}} + 1) - 1 \Big) + \theta_0 + \theta_1 \\ &= z(b - \gamma_{\text{op}} + 1) - 2^{\gamma_{\text{op}}} + 2\theta_0 - \theta_1(b - \gamma_{\text{op}} - 1) \end{aligned} \tag{3.18}$$

The right hand side (RHS) of Equation (3.18) is maximized when θ_0 is maximized and θ_1 is minimized, which happens when all z tag IDs are contiguous and they start from the left most leaf of a subtree at level γ_{op}. In this case, the number of subtrees with tags are $\left\lceil \frac{z}{2^{b-\gamma_{op}}} \right\rceil$ and therefore $\theta_0 = 2^{\gamma_{\text{op}}} - \left\lceil \frac{z}{2^{b-\gamma_{op}}} \right\rceil$. Furthermore in this case, when $\gamma_{\text{op}} \leq b - 1$, there is at most one subtree at level γ_{op} that has exactly one tag *i.e.*, $\theta_1 = \left\lceil \frac{z}{2^{b-\gamma_{\text{op}}}} \right\rceil - \left\lceil \frac{z-1}{2^{b-\gamma_{\text{op}}}} \right\rceil$; when $\gamma_{\text{op}} = b$, θ_1 equals z. Combining the two cases of $\gamma_{\text{op}} \leq b - 1$ and $\gamma_{\text{op}} = b$, we have $\theta_1 = \left\lceil \frac{z}{2^{b-\gamma_{op}}} \right\rceil - \left\lceil \frac{z-1}{2^{b-\gamma_{op}}} \right\rceil \left\lceil 1 - \frac{\gamma_{op}}{b} \right\rceil$. □

The proof above gives us the insight that *TH requires fewer queries when the tag IDs are distributed more uniformly in the ID space.* Intuitively, this makes sense because the more the tag IDs are distributed uniformly, the fewer the number of collisions encountered by TH. Experimentally, our results shown in Figures 3.11(a) and 3.11(b) in Section 6.9 also confirm this insight: for the same number of tags, the number of queries needed by TH when tags are uniformly distributed is less than that when tags are non-uniformly distributed.

We now conduct an analytical comparison between the maximum number of queries for TH and that for TW. Figure 3.5 shows the maximum number of queries for TH, which is calculated using the RHS of Equation (3.16), and that for TW, which is calculated using the RHS of Equation (3.17), for 64 bit tag IDs. We observe that TH again outperforms TW for the maximum number of queries, although slightly. For example, for a population of 10K tags, the maximum number of queries for TH is 93% of that for TW.

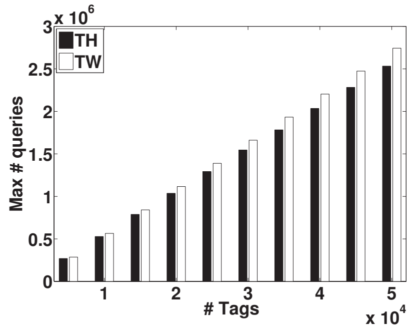


Figure 3.5 Max. # queries: TH vs. TW

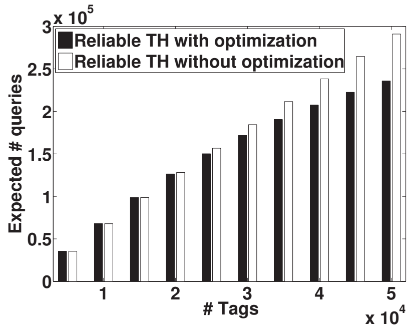


Figure 3.6 $E[Q]$ of Reliable TH

3.4 Minimizing Identification Time

The optimal value of γ calculated using the expression for $E[Q]$ in Equation (3.15) and applying the numerical method proposed in Section 3.3.2 minimizes the average number of queries, but does not minimize the average identification time because the durations of successful read, empty read, and collision are different. Next, we derive an expression for expected identification time as a function of γ. We can then use the numerical method of

Section 3.3.2 to calculate the optimal value of γ that will minimize the average identification time.

Let random variable T denote the total identification time that TH takes to identify all tags. Next, we calculate $E[T]$. Let t_s, t_c, and t_e denote the time durations of successful read, collision, and empty read, respectively. Let random variables Q_s, Q_c, and Q_e denote the number of queries resulting in successful reads, collisions, and empty reads, respectively. Thus, T can be expressed as follows:

$$T = Q_s \times t_s + Q_c \times t_c + Q_e \times t_e \tag{3.19}$$

Applying expectation operator on both sides of the equation above, the expected value of total identification time, $E[T]$, can be expressed as follows:

$$E[T] = E[Q_s] \times t_s + E[Q_c] \times t_c + E[Q_e] \times t_e \tag{3.20}$$

Next, we derive expressions for $E[Q_s]$, $E[Q_c]$, and $E[Q_e]$. Let $I_x(l, p)$ be an indicator random variable whose value is 1 if and only if node (l, p) is visited and the response type is x, where $x \in \{$s:`successful read`, c:`collision`, e:`empty read`$\}$. Thus, Q_x is the sum of $I_x(l, p)$ for all l and all p, where $x \in \{$s, c, e$\}$.

$$Q_x = \sum_{l=1}^{b} \sum_{p=0}^{2^l-1} I_x(l, p) \tag{3.21}$$

The probability that TH visits node (l, p) is $P\{(l, p)\}$. Let $\overline{P}\{x|(l, p)\}$ be the probability that given that TH visits node (l, p), the response type for the node is x, where $x \in \{$s, c, e$\}$. Thus, $E[Q_x]$ can be expressed as follows:

$$E[Q_x] = \sum_{l=1}^{b} \sum_{p=0}^{2^l-1} P\{(l, p)\} \times \overline{P}\{x|(l, p)\} \tag{3.22}$$

Recall that $P\{(l, p)\}$ has already been completely defined in Equations (3.3) through (3.14). Next, we derive expressions for $\overline{P}\{x|(l, p)\}$. Let $\overline{k}$ denote the number of tags covered by the node (l, p). Let $\overline{m} = 2^{b-l}$ denote the maximum tags node (l, p) can cover. Recall that

$n = 2^b$ denotes the max number of tags that can be accommodated in the whole ID space. The probability that node (l, p) covers $\overline{k}$ of z tags follows a hypergeometric distribution:

$$P\left\{\#\text{tags} = \overline{k}\right\} = \frac{\binom{\overline{m}}{\overline{k}}\binom{n-\overline{m}}{z-\overline{k}}}{\binom{n}{z}} \tag{3.23}$$

The probabilities $\overline{P}\{s|(l,p)\}$ and $\overline{P}\{e|(l,p)\}$ can be calculated using $\overline{k} = 1$ and $\overline{k} = 0$, respectively, in Equation (3.23).

$$\overline{P}\{s|(l,p)\} = \left\{ \begin{array}{ll} \frac{\overline{m}\binom{n-\overline{m}}{z-1}}{\binom{n}{z}} & \textbf{if } n - \overline{m} \geq z - 1 \\ 0 & \textbf{if } n - \overline{m} < z - 1 \end{array} \right\} \tag{3.24}$$

$$\overline{P}\{e|(l,p)\} = \left\{ \begin{array}{ll} \frac{\binom{n-\overline{m}}{z}}{\binom{n}{z}} & \textbf{if } n - \overline{m} > z - 1 \\ 0 & \textbf{if } n - \overline{m} \leq z - 1 \end{array} \right\} \tag{3.25}$$

Probability $P\{c|(l,p)\}$ can be calculated as follows.

$$\overline{P}\{c|(l,p)\} = 1 - (\overline{P}\{e|(l,p)\} + \overline{P}\{s|(l,p)\}) = \left\{ \begin{array}{ll} 1 - \frac{\overline{m}\binom{n-\overline{m}}{z-1}}{\binom{n}{z}} - \frac{\binom{n-\overline{m}}{z}}{\binom{n}{z}} & \textbf{if } n - \overline{m} > z - 1 \\ 1 - \frac{\overline{m}}{\binom{n}{z}} & \textbf{if } n - \overline{m} = z - 1 \\ 0 & \textbf{if } n - \overline{m} < z - 1 \end{array} \right\} \tag{3.26}$$

The expected identification time of TH can now be calculated using Theorem 8.

Theorem 8. *For a population of z tags uniformly distributed in the ID space, where each tag has an ID of b bits, if TH hops to level γ to perform DFT from each node on this level, the expected identification time for identifying all z tags is:*

$$\begin{aligned} E[T] = {} & 2^{\gamma}\left[t_c + (t_s - t_c)\overline{P}\{s|(\gamma,p)\} + (t_e - t_c)\overline{P}\{e|(\gamma,p)\}\right] \\ & + \sum_{l=\gamma+1}^{b} \left\{ \begin{array}{l} \left[t_c + (t_s - t_c)\overline{P}\{s|(l,p)\} + (t_e - t_c)\overline{P}\{e|(l,p)\}\right] \\ \times 2^{l-1}[P_l\{(l,p)\} + P_r\{(l,p)\}] \end{array} \right\} \end{aligned} \tag{3.27}$$

Proof. Equation (3.27) is obtained in three steps. First, substitute the values of $\overline{P}\{s|(l,p)\}$, $\overline{P}\{e|(l,p)\}$, and $\overline{P}\{c|(l,p)\}$ from Equations (3.24), (3.25), and (3.26) into Equation (3.22) to obtain values of $E[Q_s]$, $E[Q_e]$, and $E[Q_c]$, respectively, and further substitute these values of $E[Q_s]$, $E[Q_e]$, and $E[Q_c]$ into Equation (3.20). Second, use $P\{(l,p)\} = 0$ for $1 \leq l < \gamma$ as per Equation (3.3) and use $P\{(l,p)\} = 1$ for $l = \gamma$ as per Equation (3.4). Third, for any level $l > \gamma$, use $P\{(l,p)\} = P_l\{(l,p)\}$ for each node on this level that is left child of its parent and use $P\{(l,p)\} = P_r\{(l,p)\}$ for each node on this level that is right child of its parent. Note that there are 2^{l-1} pairs of left and right nodes on level l. □

When $\gamma = 1$, Equation (3.27) is also the analytical model for calculating expected identification time of TW protocol. Note that Equation (3.27) is a generalized form of Equation (3.15). It reduces to Equation (3.15) if the time durations of successful read, collision, and empty read are equal to unit time.

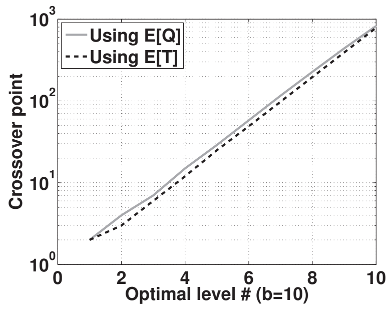


Figure 3.7 Crossover points obtained using $E[Q]$ and $E[T]$

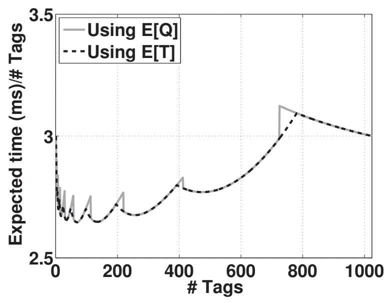


Figure 3.8 Normalized expected identification time vs. population size

According to [53] and [55], the values of t_s, t_e, and t_c are $3ms$, $0.3ms$, $1.5ms$, respectively. Figure 3.7 plots the values of crossover points obtained using expression of $E[Q]$ from Theorem 6 and expression of $E[T]$ from Theorem 8 (we used $b = 10$ for illustration). We observe from the figure that the values of crossover points obtained using the expression for $E[Q]$ are comparatively larger than those obtained using the expression for $E[T]$. The reason is that to minimize identification time instead of number of queries, TH starts the DFTs at levels with comparatively larger values of l, which results in reduction in number of collisions at an expense of slightly increased number of empty reads. The over all identification time is reduced because empty reads are five times faster than collisions and the amount of identification time increased by the increased number of empty reads is smaller than the amount of identification time reduced by the reduced number of collisions. Figure 3.8 shows the normalized expected identification times for the two cases *i.e.*, when the crossover points are calculated using $E[Q]$ and $E[T]$ (again we used $b = 10$ for illustration). We observe that for several population sizes, the normalized expected time calculated using $E[Q]$ is greater than that calculated using $E[T]$.

3.5 Discussion

3.5.1 Virtual Conversion of Population Distributions

To virtually convert a non-uniformly distributed population into a uniformly distributed population, we leverage the fact that in large populations, the expected number of tags whose IDs have the least significant bit (LSB) of 0 is approximately the same as the expected number of tags whose IDs have the LSB of 1. Similarly, the expected number of tags whose IDs have the two LSBs of 00 is approximately the same as the expected number of tags whose IDs have the two LSBs of 01, 10, or 11, and so on. Therefore, if we construct a binary tree in which level l corresponds to l^{th} LSB instead of l^{th} most significant bit (MSB), then each node of level l is expected to cover $z/2^l$ tags: a property of uniformly distributed populations. To illustrate, consider an example where there are 8 tags in a population, each with a unique 4-bit ID in the range $[0, 7]$. Figure 3.9(a) shows the binary tree constructed in the conventional way in which level l corresponds to l^{th} MSB. This population is clearly non-uniformly distributed in the ID space and TH will have to frequently perform dynamic adjustments to the optimal value of γ and the number of queries will be large compared to the number of queries for a uniformly distributed population of the same size. Figure 3.9(b) shows the binary tree constructed in the proposed way where level l corresponds to l^{th} LSB. Note from the figure that the 8 tags are now uniformly placed in the entire ID space. On the binary trees that resembles the one in Figure 3.9(b), TH will require very few dynamic adjustments and the number of queries will be approximately same as for a uniformly distributed population of the same size.

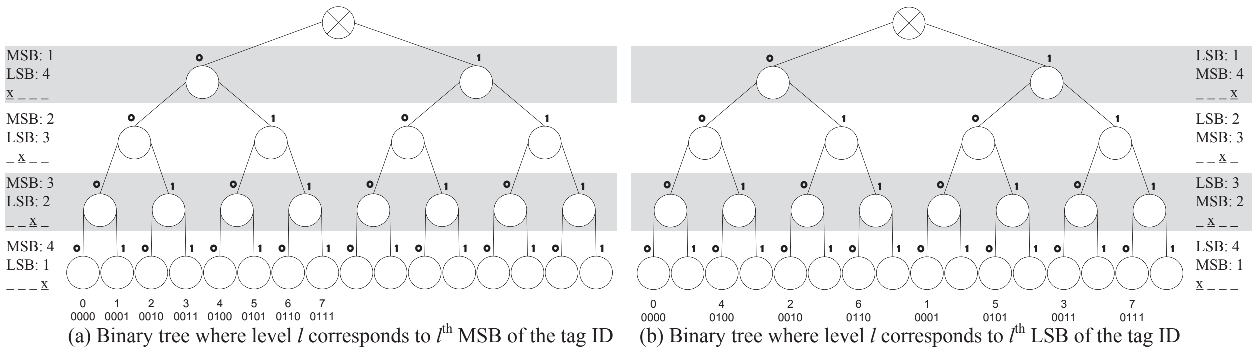


Figure 3.9 Distributions of populations with binary trees with MSBs and LSBs.

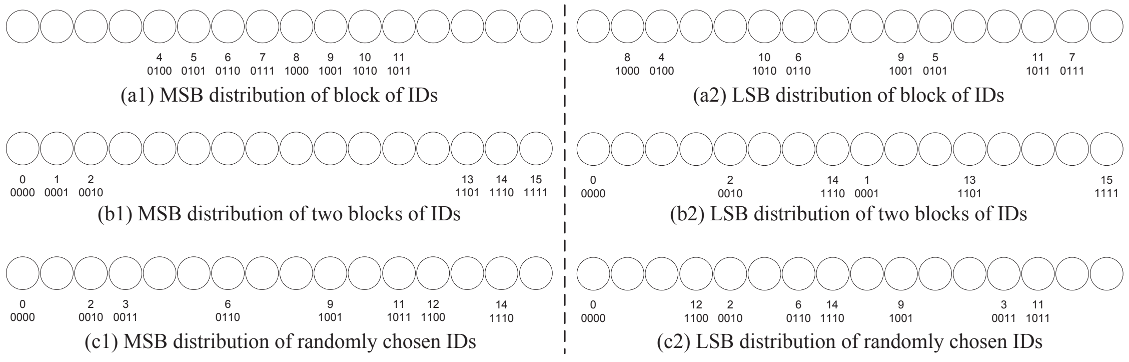


Figure 3.10 Last level $l = b = 4$ of the binary trees made with MSBs and LSBs.

Figures 3.10(a1) through 3.10(c2) show three other populations where the circles on the left side of the dashed vertical line represent level $l = b = 4$ of the binary tree in which level l corresponds the l^{th} MSB of the tag ID, and the circles on the right side of the dashed vertical line represent level $l = b = 4$ of the binary tree in which level l corresponds the l^{th} LSB of the tag ID. The population in Figures 3.10(a1) and 3.10(a2) consists of 8 tags with consecutive IDs in the range $[4, 11]$. We can see that if the binary tree is built using conventional method where l^{th} level corresponds to the l^{th} MSB of the tag ID, then the resulting population is not uniformly distributed in the binary tree. However, if the binary tree is built using our proposed modification where l^{th} level corresponds to the l^{th} LSB of the tag ID, then the resulting population is more close to a uniform distribution. Similarly, the population in Figures 3.10(b1) and 3.10(b2) consists of two blocks, each containing 3 IDs. We make the same observation that the IDs are comparatively more uniformly distributed in the binary tree made with LSBs compared to the one made with MSBs. In a scenario where a population is already uniformly distributed in the ID space, our proposed modification does not affect it and the uniformity is maintained in the tree made with LSBs. This is shown in Figures 3.10(c1) and 3.10(c2).

Next we leverage these observations to propose a simple modification in TH that reduces the number of queries and identification times of TH for non-uniformly distributed populations to approximately the same values as for uniformly distributed populations. When the reader transmits a query string, the tag compares it with its LSBs instead of MSBs to decide whether or not it will respond to the query. If the result of the query is a collision, the reader generates two new query strings by appending a 0 and a 1 at the *start* of the previous query string and queries the tags with these new query strings. All the tags whose IDs end with the new query string respond.

This modification does not require any changes to the tags and works with the C1G2 compliant tags. To make a tag compare the query string with the LSBs of its ID, we use the SELECT command standardized in the C1G2 standard. The ID of a tag is stored in its memory at a specific memory address. A tag can retrieve any bits stored in its memory by specifying an appropriate address range. Using the SELECT command, a reader broadcasts an address range and a bit mask. Each tag compares the bit mask with the bits in the specified address range in its memory and responds back only if the bit mask matches the specified bits in its memory. In TH, the bit mask contains the query string of length l, where $1 \leq l \leq b$, and the address range that the reader broadcasts is of the l LSBs of tag IDs.

3.5.2 Reliable Tag Identification

So far we have assumed that the communication channel between the reader and tags is reliable, which means that each tag can receive the query from the reader and the reader can receive either the response if only one tag responds or the collision if more than one tag respond. However, this assumption often does not hold in reality because wireless communication medium is inherently unreliable. There are two existing schemes for making tag identification reliable. Backes *et al.* proposed the scheme of letting each tag store the IDs of several other tags [31]. When the reader queries a tag, the tag transmits back its own ID as well as the IDs of other tags stored in it. When identification completes, the reader compares the set of IDs of tags that responded with the union of sets of IDs of other tags reported by each responding tag. If the sets are not equal, the whole process is repeated again to ensure that the missed tags are identified. This scheme has two weaknesses. First, this scheme does not comply with the C1G2 standard. Second, it assumes that the tag population remains static for the lifetime of tags as each tag is hard coded with some other tags' IDs. The second scheme is to run an identification protocol on the same population several times until probability of missing a tag falls below a threshold [51, 56]. They estimate the probability of missing a tag based upon the number of tags that were identified in some runs of the protocol but not in others.

While we can use the C1G2 compliant scheme proposed in [51, 56] to make TH reliable, *i.e.*, repeatedly run TH until the required reliability is achieved. We observe that in this scheme, the leaf nodes in the binary tree are queried multiple times. This is wasteful of time for the nodes that the reader successfully reads. To eliminate such waste, we propose *to query each node multiple times, instead of querying the whole binary tree multiple times.* We define the *reliability of successfully reading a tag* to be *the probability that both the tag receives the query from the reader and the reader receives the response from the tag.* For this, we calculate the maximum number of times the reader should transmit a query, which is denoted by β. Let g and u be the *given* and *required* reliability of successfully reading a tag,

Table 3.1 Comparison with Prior C1G2 Compliant Protocols (TH/Prior Art)

		Prior Nondeter. Protocol (=Aloha)			**Prior Deterministic Protocols**				**Prior Hybrid (=MAS)**		
		Max	**Min**	**Mean**	**Best prior**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**
Uniform (TH_Q)	**#queries/tag**	0.24	0.10	0.18	ATW-f	0.51	0.50	0.50	0.39	0.38	0.39
	query time/tag	0.84	0.71	0.76	ATW-c	0.92	0.89	0.90	0.81	0.78	0.79
	#responses/tag	0.85	0.59	0.69	ATW-c	0.85	0.67	0.70	0.64	0.24	0.38
	response fairness	1.15	1.10	1.13	TW	1.12	1.07	1.11	1.12	1.07	1.10
Non-Uni (TH_T)	**#queries/tag**	0.26	0.10	0.18	ATW-f	0.75	0.33	0.60	0.40	0.18	0.29
	query time/tag	0.40	0.12	0.24	ATW-f	0.60	0.19	0.41	0.21	0.09	0.15
	#responses/tag	0.63	0.11	0.32	ATW-c	0.87	0.11	0.33	0.46	0.08	0.22
	response fairness	1.38	1.25	1.35	ATW-c	1.03	1.00	1.02	1.05	0.95	1.02

respectively. Thus, the probability of successfully identifying a tag is $1-(1-g)^{\beta}$. Equating it to u gives:

$$\beta = \log_{(1-g)}(1-u) \tag{3.28}$$

Our scheme of reliable tag identification works as follows: for each non-terminal node in the binary tree that TH needs to visit, TH transmits a query corresponding to that node β times; corresponding to each terminal node, TH keeps transmitting the query until either that query has been transmitted β times or the reader successfully receives the tag ID.

The optimization technique of stop transmitting the query corresponding to a terminal node on a successful read significantly reduces the total number of queries. Figure 3.6 plots the expected number of queries per tag for the reliable TH protocol with and without this optimization. For example, for a population of 50000 tags, the number of queries per tag are reduced by 24%.

3.5.3 Continuous Scanning

In some applications, the tag population may change over time (*i.e.*, tags leave and join the population dynamically). We adapt the continuous scanning strategy proposed by Myung *et al.* in [82]. In the first scanning of the whole tag population, TH records the queries that resulted in successful or empty reads. If the tag population does not change, by perfoming DFTs on the subtrees rooted at successful and empty read nodes of the previous scan, TH experiences no collision. If some new tags join the population, some of the successful read nodes of the previous scan can now turn into collision nodes and some empty read nodes can turn into successful or collision nodes. If some old tags leave the population, some successful read nodes will become empty read nodes. If any of the new empty read nodes happens to be a sibling of another empty read node, then TH discards these two nodes from the record and stores the location of their parent because the parent is also an empty read node. This strategy works well when tag population size remains static or increases. However, when the tag population decreases, the best choice is to re-execute TH for the subsequent scan.

3.5.4 Multiple Readers

An application with a large number of RFID tags requires multiple readers with overlapping regions because a single reader can not cover all tags due to the short communication range of tags (usually less than 20 feet). The use of multiple readers introduces several new types of collisions such as reader-reader collisions and reader-tag collisions. Such collisions can be handled by reader scheduling protocols such as those proposed in [122, 36, 131, 116]. TH is compatible with all of these reader scheduling protocols.

3.6 Performance Comparison

We implemented two versions of TH. (1) TH_Q, in which γ_{op} is obtained using $E[Q]$ and the query string is matched with MSBs of tag IDs, and (2) TH_T, in which γ_{op} is obtained using $E[T]$ and the query string is matched with LSBs of tag IDs to virtually convert the population distribution into a near-uniform distribution. We also implemented all the 8 prior tag identification protocols in Matlab, namely the 3 nondeterministic protocols (Aloha [129], BS [35], and ABS [82]), the 3 deterministic protocols (TW [66], ATW [115], and STT [87]), and the 2 hybrid protocols (MAS [83] and ASAP [89]). As ATW starts DFTs from the level of $\log z$ which may not be a whole number, we present results for ATW by both ceiling and flooring the values of log z and representing them with ATW-c and ATW-f respectively. In terms of implementation complexity, TH and all the 8 prior protocols are implemented in the similar number of lines of code. We performed extensive testing, both manually and automatically, to ensure the correctness of each protocol implementation.

We performed the side-by-side comparison with TH, although this comparison is not completely fair for TH for two reasons. First, 3 of these 8 protocols (*i.e.*, BS, ABS, and ASAP) require modifications to tags and thus do not work with standard C1G2 tags, whereas TH is fully compliant with C1G2. Second, for the framed slotted Aloha, to its best advantage, we choose the frame size to be the ideal size, which is equal to the tag population size, disregarding the practical limitations on the frame sizes. We choose tag ID length to be the C1G2 standard 64 bits. We performed the comparison for both the uniform case (where the tag population is uniformly distributed in the ID space) and the non-uniform case (where the tag population is not uniformly distributed in the ID space). For the uniform case, we range tag population sizes from 100 to $100,000$ to evaluate the scalability of these protocols. For the non-uniform case, we distribute tag populations in blocks where each block is a continuous sequence of tag IDs. We range block sizes from 5 to 1000. Our motivation for simulating non-uniform distribution in blocks is that in some applications, such as supply chains, tag IDs often come in such blocks when they are manufactured. For each tag population size, we run each protocol 100 times and report the mean. We compare TH with prior protocols from both reader and tag perspectives.

3.6.1 Reader Side Comparison

For the reader side, we compared TH with the 8 prior protocols based on the following two metrics: (1) normalized reader queries and (2) identification speed. Normalized reader queries is the ratio of the number of queries that the reader transmits to identify a tag population divided by the number of tags in the population. Similarly, identification speed is the total time that the reader takes to identify a tag population divided by the number of tags in that population.

In general, more queries implies more identification time. However, identification time is not strictly in proportion to the number of queries because different queries may take different amounts of time.

For each metric, in Table 3.1, we show the value of TH divided by that for the best prior C1G2 compliant protocol for this metric in the corresponding category of nondeterministic, deterministic, or hybrid. Note that the only prior C1G2 compliant nondeterministic tag identification protocol is the framed slotted Aloha and the only prior C1G2 compliant hybrid tag identification protocol is MAS. There are 3 prior C1G2 compliant deterministic tag identification protocols: TW, ATW, and STT. We report min, max, and mean for these ratios for tag populations ranging from 100 to $100,000$.

For the two metrics defined above, the absolute performance of TH and all prior 8 tag identification protocols is shown in Figures 3.11(a) to 3.12(b), for both uniform and non-uniform distributions. Note that for non-uniform distributions, we fix the tag population size to be 5000 and range the block size from 2 to 1000.

3.6.1.1 Normalized Reader Queries

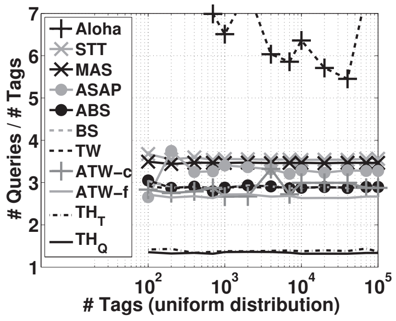
TH_Q reduces the normalized reader queries of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of 82%, 50%, *and* 61%, *respectively, for uniformly distributed tag populations. TH_T reduces the normalized reader queries of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of* 82%, 40%, *and* 71%, *respectively, for non-uniformly distributed tag populations.* Figures 3.11(a) and 3.11(b) show the normalized reader queries of all protocols for uniformly and non-uniformly distributed populations, respectively. Based on these two figures, we make the following four observations from the perspective of normalized reader queries for both uniform and non-uniform distributions. First, normalized queries of $\mathrm{TH_T}$ are slightly greater than those of $\mathrm{TH_Q}$ for uniformly dis-

tributed tag populations. This is because, to minimize identification time, TH_T starts DFTs at levels closer to the leaf nodes compared to TH_Q, which results in more empty reads and less collisions. The increase in number of empty reads is slightly greater than the decrease in number of collisions. Matching the query string with LSBs in TH_T does not bring much advantage because the population is already uniformly distributed. Second, for non-uniformly distributed tag populations, normalized queries of TH_T are, on average, 18% fewer than those of TH_Q. This significant improvement is a result of the virtual conversion of non-uniformly distributed populations into uniformly distributed populations as proposed in Section 3.5.1. Third, among all the 8 prior protocols, the traditional ATW protocol turns out to be the best. Fourth, the framed slotted Aloha in the C1G2 standard performs the worst even when we disregard the practical limitations on the frame sizes. Although BS is the best among the 3 prior nondeterministic tag identification protocols, it is not compliant with C1G2. Similarly, although ASAP is the best among the 2 prior hybrid tag identification protocols, it is not compliant with C1G2.

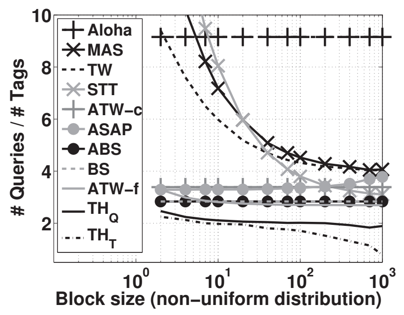
3.6.1.2 Identification Speed

TH_Q improves the identification speed of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of 24%, 10%, *and* 21%, *respectively, for uniformly distributed tag populations. TH_T improves the identification speed of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of* 76%, 59%, *and* 85%, *respectively, for non-uniformly distributed tag populations.* Figures 3.12(a) and 3.12(b) show the identification speed of all protocols for uniformly and non-uniformly distributed tag populations, respectively. Based on these two figures, we make the following four observations from the perspective of identification speed. First, the normalized identification times of TH_T are slightly smaller than those of TH_Q for uniformly distributed tag populations. This improvement is the result of using $E[T]$ to calculate γ_{op} instead of using $E[Q]$. Second, the normalized identification times of

TH_T are, on average, 36% smaller than those of TH_Q for non-uniformly distributed tag populations. This significant improvement is a result of the virtual conversion of non-uniformly distributed populations into uniformly distributed populations as proposed in Section 3.5.1. Third, among all 8 prior protocols, the traditional ATW protocol turns out to be the best for both uniform and non-uniform distributions. Fourth, although framed slotted Aloha is the worst in terms of normalized reader queries, its identification speed is not the worst. This is because in our experiments we allow it to use unrealistically large frame sizes, which leads to many empty slots and empty read is much faster than successful read and collision.

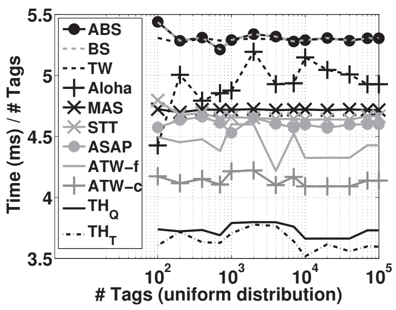


(a) Uniform

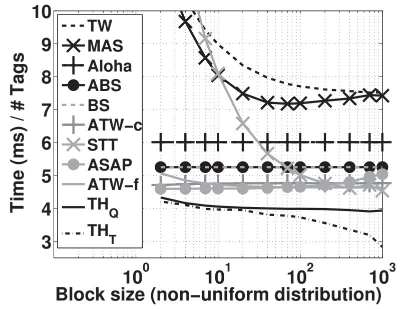


(b) Non-uniform

Figure 3.11 Normalized queries of TH and existing protocols

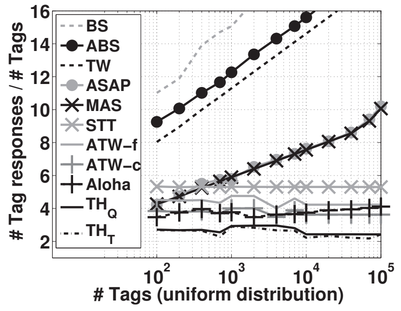


(a) Uniform

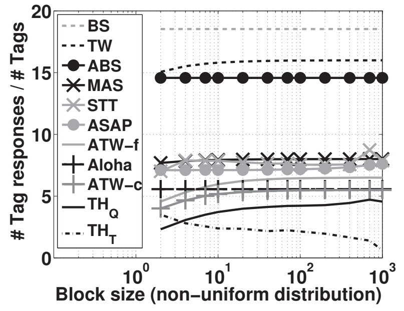


(b) Non-uniform

Figure 3.12 Identification speed of TH and existing protocols

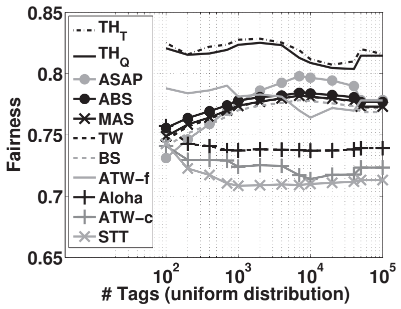


(a) Uniform

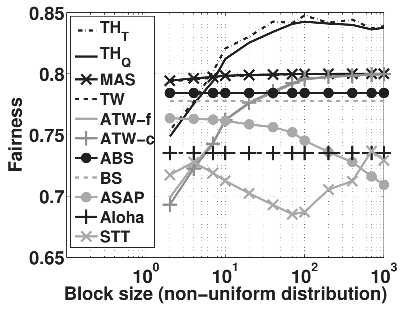


(b) Non-uniform

Figure 3.13 Normalized responses of TH and existing protocols

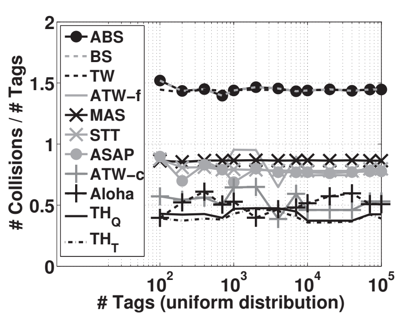


(a) Uniform

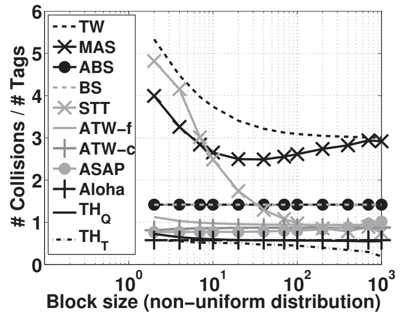


(b) Non-uniform

Figure 3.14 Response fairness of TH and existing protocols

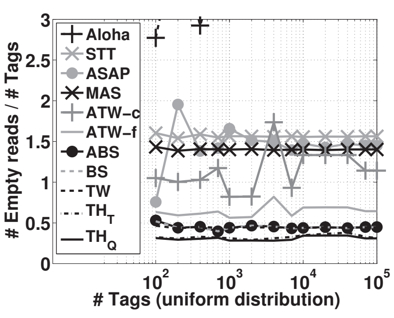


(a) Uniform

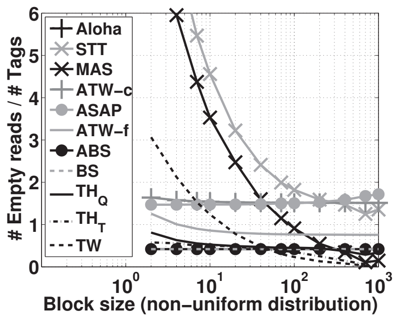


(b) Non-uniform

Figure 3.15 Normalized collisions of TH and existing protocols



(a) Uniform



(b) Non-uniform

Figure 3.16 Normalized empty reads of TH and existing protocols

3.6.2 Tag Side Comparison

On the tag side, we compare TH with the 8 prior protocols based on the following four metrics: (1) normalized tag responses, (2) response fairness, (3) normalized collisions, and (4) normalized empty reads. Normalized tag responses is the ratio of sum of responses of all tags during the identification process to the number of tags in the population. Response fairness is the Jain's fairness index given by $\frac{(\sum_{i=1}^{z} x_i)^2}{z \cdot \sum_{i=1}^{z} x_i^2}$ where x_i is the total number of responses by tag i [57]. Normalized collisions is the ratio of total number of collisions during the identification process to the number of tags in the population. Normalized empty reads is the ratio of total number of empty reads during the identification process to the number of tags in the population.

The first two metrics are important for active tags because active tags are powered by batteries. Lesser number of normalized tag responses mean lesser power consumption for active tags. Response fairness measures the variance in the number of responses per tag. Less fairness results in the depletion of the batteries of some tags more quickly compared to others. In large scale tag deployments, it is often nontrivial to identify tags with depleted batteries and replace them. Using an absolutely fair tag identification protocol, the batteries of all tags deplete at the same time and therefore all can be replaced at the same time. We use the Jain's fairness metric defined in [57]. For z tags, the fairness value is in the range

$[\frac{1}{z}, 1]$. The higher this fairness value is, the more fair the protocol is. The second two metrics are important for understanding these identification protocols.

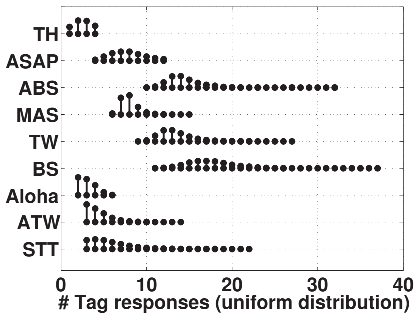
For normalized tag responses and response fairness, in Table 3.1, we show the value of TH divided by that for the best prior C1G2 compliant protocol in the corresponding category of nondeterministic, deterministic, or hybrid. The absolute performance of TH and all prior 8 tag identification protocols is shown in Figures 3.13(a) to 3.14(b), for both uniform and non-uniform distributions.

3.6.2.1 Normalized Tag Responses

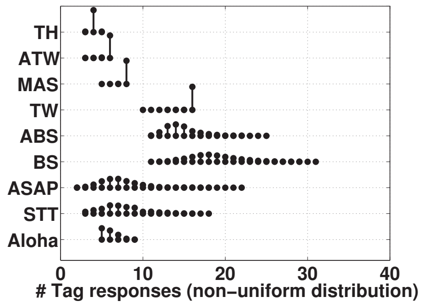
TH_Q reduces the normalized tag responses of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of 31%, 30%, *and* 62%, *respectively, for uniformly distributed tag populations. TH_T reduces the normalized tag responses of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of* 68%, 67%, *and* 78%, *respectively, for non-uniformly distributed tag populations.* Figures 3.13(a) and 3.13(b) show the normalized tag responses of all protocols for uniformly and non-uniformly distributed tag populations, respectively. We make following four observations from these two figures. First, the normalized tag responses of TH_T are, on average, 57% lesser than those of TH_Q for non-uniformly distributed tag populations. Second, the normalized tag responses of BS, ABS, TW, MAS, and ASAP increase with increasing tag population size. Third, for non-uniformly distributed tag populations, the normalized tag responses of nondeterministic protocols is not affected by the block size because their performance is independent of tag ID distribution. In contrast, the normalized tag responses of deterministic protocols slightly increase with increasing block size. Fourth, among all 8 prior protocols, Aloha has the smallest number of normalized tag responses. This is because of the unlimitedly large frame sizes that we used for Aloha. With large frame sizes, tags experience lesser collisions and thus reply fewer times.

3.6.2.2 Tag Response Fairness

TH_Q improves the tag response fairness of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of 13%, 11%, *and* 10%, *respectively, for uniformly distributed tag populations. TH_T improves the tag response fairness of the best prior C1G2 compliant nondeterministic, deterministic, and hybrid tag identification protocols by an average of* 35%, 2%, *and* 2%, *respectively, for non-uniformly distributed tag populations.* Figures 3.14(a) and 3.14(b) show the tag response fairness of all protocols for uniformly and non-uniformly distributed tag populations, respectively. We



(a) Uniform distribution



(b) Non-uniform dist.

Figure 3.17 Distribution of tag responses of TH and existing protocols

observe that among all 8 prior protocols, ASAP and ATW are the best for uniformly and non-uniformly distributed populations, respectively. We observe that TH_T achieves slightly better fairness than TH_Q.

Figures 3.17(a) and 3.17(b) show the distribution of the number of tag responses for each protocol for uniformly and non-uniformly distributed tag populations, respectively. For any protocol, the wider the horizontal span of its distribution is, the larger the range of the number of responses per tag it has. We observe that TH has the smallest range among all protocols for the number of responses per tag.

3.6.2.3 Normalized Collisions

TH_Q and TH_T, both incur smaller number of collisions than all 8 prior protocols for uniformly and non-uniformly distributed tag populations. Figures 3.15(a) and 3.15(b) show the normalized collisions for all protocols for uniformly and non-uniformly distributed tag populations, respectively. From these figures we make following three observations. First, $\mathrm{TH_T}$ incurs fewer collisions compared to $\mathrm{TH_Q}$, which is one of the reasons behind the faster identification speed of $\mathrm{TH_T}$. Second, Aloha incurs the smallest number of normalized collisions among all 8 prior protocols because of the unlimitedly large frame sizes that we used for it. Third, TW mostly incurs the largest number of normalized collisions for both types of populations.

3.6.2.4 Normalized Empty Reads

For uniformly distributed tag populations, TH_Q incurs a smaller number of empty reads than all 8 prior protocols. For non-uniformly distributed tag populations, TH_T, incurs a smaller number of empty reads than all 8 prior protocols. Figure 3.16(a) and 3.16(b) show the normalized empty reads of all protocols for uniformly and non-uniformly distributed tag populations, respectively. From these figures, we observe that although the two prior C1G2 compliant protocols, TW and MAS, have fewer empty reads compared to $\mathrm{TH_Q}$ for large block sizes, they have much larger number of collisions compared to $\mathrm{TH_Q}$, which makes their overall identification time much larger than $\mathrm{TH_Q}$. Note that the slightly larger number of empty reads for $\mathrm{TH_Q}$ for large block sizes is immaterial because the time for an empty read is 5 times lesser than that for a collision and 10 times lesser than that for a successful read. Therefore, reducing the number of collisions is more important than reducing the number of empty reads. We also observe that $\mathrm{TH_T}$ has greater number of empty reads compared to $\mathrm{TH_Q}$, which is the cost of decreasing the collisions. As collisions are 5 times slower compared to empty reads, this slight increase in number of empty reads is not of much significance.

Note that the collisions and empty reads shown in Figures 3.15(a) and 3.16(a), respectively, are consistent with the reader queries shown in Figure 3.11(a) as well as the identification speed shown in Figure 3.12(a). Similarly, the collisions and empty reads shown in Figures 3.15(b) and 3.16(b), respectively, are consistent with the reader queries shown in Figure 3.11(b) as well as the identification speed shown in Figure 3.12(b). For example, Figure 3.15(a) shows that TW has more collisions than Aloha, but 3.11(a) shows that Aloha has more queries than TW. This is because Aloha has much more empty reads than TW as shown in Figure 3.16(a). Although Aloha has more queries than TW, Figure 3.12(a) also shows that Aloha requires less identification time than TW. This is because an empty read is 5 times faster than a collision for a reader.

A common observation that we make from the plots of all the metrics of TH for uniformly distributed populations is that these plots have ups and downs and are not monotonic. This is because when the number of tags increases, the starting level from where TH performs the first DFT increases, which has an effect on all these metrics. These ups and downs are also observed in the analytical plot in Figure 3.8.

3.7 Conclusion

The technical novelty of this chapter lies in that it represents the first effort to formulate the Tree Walking process mathematically and propose a method to minimize the expected number of queries and expected identification time. The significance of this chapter in terms of impact lies in that the Tree Walking protocol is a fundamental multiple access protocol and has been standardized as an RFID tag identification protocol. Besides static optimality, our Tree Hopping protocol dynamically chooses a new optimal level after each subtree is traversed. We presented a method to make our protocol work with non-uniformly distributed populations and achieve similar performance that it achieves with uniformly distributed populations. We also presented methods to make our protocol reliable, to continuously scan tag populations that are dynamically changing, and to work with multiple readers with overlapping regions. Another key contribution of this chapter is that we conducted a comprehensive side-by-side comparison of two variants of our protocol with eight major prior tag identification protocols that we implemented. Our experimental results show that our protocol significantly outperforms all prior tag identification protocols, even those that are not C1G2 compliant, for metrics such as the number of reader queries per tag, the identification speed, and the number of responses per tag.

4 RFID Missing Tags

4.1 Introduction

4.1.1 Background & Motivation

Shoplifting, employee theft, and vendor fraud have become major causes of lost capital for retailers [113]. In 2011 alone, the retailers lost an estimated 34.5 billion dollars due to these causes [12]. With the benefits of not requiring a line-of-sight and low cost of tags (*e.g.*, 5 cents per tag [93]), radio frequency identification (RFID) systems have been deployed for monitoring products by affixing them with cheap passive RFID tags and using RFID readers, which are given the IDs of the tags that are being monitored, to detect any missing tags. A tag is a microchip with an antenna in a compact package that has limited computing power and communication range. There are two types of tags: (1) passive tags, which power up by harvesting the radio frequency energy from readers (as they do not have their own power sources) and have communication range often less than 20 feet; (2) active tags, which have their own power sources and have relatively longer communication range. A reader has a dedicated power source with significant computing power. It transmits queries to a set of tags and the tags respond over a shared wireless medium. In this chapter, we deal with both passive and active RFID tags.

4.1.2 Summary & Limitations of Prior Art

There are two types of missing tag detection protocols: *probabilistic* [114, 76] and *deterministic* [71, 128, 73]. The probabilistic protocols are faster but only report the event that some tags are missing, without pinpointing exactly which ones. The deterministic protocols return IDs of all the missing tags but are comparatively slower. Both approaches have their merits. In fact, they are complementary to each other, and should be used together. For example, a probabilistic protocol should be used to detect a missing tag event and once detected, a deterministic protocol should be invoked to identify which tags are missing. Several probabilistic protocols such as TRP [114] and EMTD [76] and deterministic protocols such as IIP [71], MTI [128], and SFMTI [73] have been proposed.

There are two key limitations of existing protocols. The first limitation is that all existing protocols assume a perfect environment with no unexpected tags, which is not a realistic assumption. In reality, tag populations often contain unexpected tags whose IDs are unknown. Here we give three examples. For the first example, in airports where an airline company uses RFID readers to monitor baggage of its passengers, the tags of other airline's baggage, which are in the vicinity of this airline's readers, also respond to the queries of this airline's readers. For the second example, in a large warehouse rented to multiple tenants, one tenant's RFID readers receive responses from tags of other tenants. For the third example, in a retail store that uses RFID readers to monitor only expensive merchandize, the readers receive responses from tags of inexpensive merchandize as well. Similar scenarios exist in other settings such as hospitals and malls. Existing protocols can not handle the presence of unexpected tags because they fill up unexpected slots in Aloha frames resulting in unexpected false positives.

The second major limitation of existing protocols is that except TRP, none of them is compliant with the EPCGlobal Class 1 Generation 2 (C1G2) RFID standard [55]. These protocols require the manufacturers to put random bit sequences in tags for calculating specialized hash functions. They also require the tags to be able to receive and interpret "pre-vector" and/or "post-vector" frames to select slots in frames. Such functionalities are not provisioned in the C1G2 standard because tags, especially the passive ones, do not have enough computational power. It is important for an RFID protocol to be compliant with the C1G2 standard because the cheap commercially available off-the-shelf (COTS) tags follow the C1G2 standard. A protocol that is not compliant with the C1G2 standard will require home brewed tags, which will not only cost more but will also work only in limited settings. For example, if an airline uses a protocol and tags that are non-compliant with the C1G2 standard, it may be able to track its baggage at its home airport but not at the airports in rest of the world, which support only the C1G2 compliant tags.

4.1.3 Problem Statement & Proposed Approach

Now we formally define the missing tag detection problem. Let $\mathbb{E}$ represent the set of IDs of the expected tags, *i.e.*, the tags that are expected to be present in a population and need to be monitored. Let an unknown number of tags, m, out of these $|\mathbb{E}|$ tags be missing, where $0 \leq m \leq |\mathbb{E}|$. Let $\mathbb{E}_p$ be the set of IDs of the remaining $|\mathbb{E}|-m$ tags that are actually present in the population. Let $\mathbb{U}$ be the set of IDs of all the unexpected tags in the population that do not need to be monitored. We neither know exactly which IDs belong to sets $\mathbb{E}_p$ and $\mathbb{U}$ nor do we know their sizes, but we do know that $\mathbb{E}_p \subseteq \mathbb{E}$. Let T be a threshold on the number of missing tags. Our objective is to design a missing tag detection protocol using which *a set of readers should quickly detect a missing tag event with a probability $\geq \alpha$ whenever the number of missing tags m is greater than or equal to the threshold T,* where α is called the required reliability and lies in the range $0 \leq \alpha < 1$. Additionally, a missing tag detection protocol should work in single as well as multiple-reader environments, and should be compliant with the C1G2 standard.

For the problem of detecting missing tags in the presence of unexpected tags, there are three seemingly obvious solutions based on previous work. The first solution is to repeatedly execute a tag collection protocol to collect IDs of all tags and compare them with the IDs in set $\mathbb{E}$ to detect if any tags are missing. This solution works; however, it is too slow. For example, our experimental results show that even the fastest existing tag collection protocol TH [105] is 14.3 times slower than our scheme. The second solution is to first execute a tag collection protocol to get the IDs of unexpected tags and then repeatedly execute an existing missing tag detection protocol. This solution has two limitations. First, it is slow because the missing tag detection protocol will have to monitor the unexpected tags in addition to the expected tags. Second, the missing tag detection protocol will report a missing tag event even when some unexpected tags go missing, which is not the requirement. Furthermore, both these solutions can not be used in settings where readers are not allowed to read the IDs of tags in set $\mathbb{U}$ due to privacy reasons. An example of such a setting is the aforementioned multi-tenant warehouse, where one tenant may not permit readers of other tenants to read the IDs of its tags. The third solution is to repeatedly execute a tag estimation protocol and look for a net change in the population size. The limitation of this solution is that if some expected tags go missing but an equal or greater number of unexpected tags join the population, the estimation protocol can not detect the missing tag event. Furthermore, missing tag detection protocols are much faster compared to estimation protocols due to the knowledge of set $\mathbb{E}$ [114, 71, 73].

In this chapter, we propose a new protocol called $\underline{\text{R}}$FID monitoring protocol with $\underline{\text{un}}$expected tags (RUN), the first protocol that can achieve required reliability in detecting a missing tag event when unexpected tags might be present in the population. RUN uses the frame slotted Aloha protocol specified in the C1G2 standard as its MAC layer communication protocol. In Aloha protocol, the reader first tells the tags a frame size f and a random seed number R. Each tag within the transmission range of the reader then uses f, R, and its ID to select a slot in the frame by evaluating a hash function $h(f, R, ID)$ whose result is uniformly distributed in $[1, f]$. Each tag has a counter initialized with the slot number it chose to reply. After each slot, the reader first transmits an end of slot signal and then each tag decrements its counter by one. In any given slot, all the tags whose counters equal

1 respond with a random sequence called RN16. If no tag replies in a slot, it is called an *empty slot*. If one or more tags reply in a slot, it is called a *nonempty slot*. As per the C1G2 standard, tags do not transmit their IDs unless the reader specifically asks them to do so. In RUN, reader checks if a slot is empty or nonempty using the RN16 sequence and never asks tags to transmit their IDs. This preserves the privacy in settings where a reader is not allowed to read IDs of tags in set $\mathbb{U}$.

To detect if any tags are missing, RUN executes multiple Aloha frames with different seeds. In each frame, each tag uses the seed for that frame to select its slot. As RUN already knows the IDs of all tags in set $\mathbb{E}$, it pre-computes which tags in $\mathbb{E}$ will select which slots in each frame. Thus, it knows which slots in the frames should be nonempty if all the tags in $\mathbb{E}$ are present in the population. When a reader executes a frame, RUN compares the response in each slot of that frame with the corresponding slot in the pre-computed frame. If it finds that a particular pre-computed slot was nonempty but the corresponding slot in the executed frame is empty, it stops and declares that some tags are missing. To minimize the effect of unexpected false positives and consequently the detection time, RUN estimates the size of $\mathbb{U}$ implicitly without running an extra estimation phase and uses this estimate to calculate optimal values of system parameters. RUN works in single as well as multiple readers environment.

4.1.4 Technical Challenges & Solutions

There are three key technical challenges in detecting a missing tag event. The first technical challenge is to handle the presence of unexpected tags. Due to the presence of such tags, it is possible that a particular slot that RUN expected to be nonempty due to a specific tag in $\mathbb{E}$ actually turns out to be nonempty even though that specific tag in $\mathbb{E}$ was missing. To address this challenge, RUN executes multiple frames with different seeds, which reduces the effects of such unexpected *false positives*. We calculate the false positive probability due to tags in $\mathbb{E}_p \cup \mathbb{U}$ and use it to calculate optimal values of frame sizes and the number of times the frames should be executed to mitigate the effects of false positives.

The second technical challenge is to estimate the number of unexpected tags $|\mathbb{U}|$ in the population, which is required to calculate the optimal values of system parameters. To address this challenge, RUN first pre-computes which slots in each frame will the tags in $\mathbb{E}$ not select. Second, it executes the frames and sees how many of such slots turn out to be nonempty. The number of such slots that are nonempty in the executed frames is a function of $|\mathbb{U}|$ but is independent of $|\mathbb{E}|$ because we know from the pre-computed frames that the tags in $\mathbb{E}$ never select these slots. Thus, by observing number of slots that are empty in the pre-computed frames and nonempty in the executed frames, RUN estimates $|\mathbb{U}|$. Note that RUN does not carry out a separate estimation phase to estimate the size of $\mathbb{U}$. It obtains the estimate while executing the Aloha frames for detecting a missing tag event and thus, does not incur any extra time cost.

The third technical challenge is to achieve the required reliability in smallest possible time. To address this challenge, we use the false positive probability to derive a "reliability condition", which, if satisfied by the system parameters, guarantees that RUN will achieve the required reliability. These values of system parameters ensure with probability α that there will be at least one slot in all the frames that is nonempty in the pre-computed frames and empty in the executed frames when $m \geq T$. To minimize RUN's execution time, we express the time in terms of the system parameters and minimize it under the constraint that the system parameters satisfy the reliability condition.

4.1.5 Key Novelty & Advantages over Prior Art

The key novelty of this chapter is twofold. First, we identify the problem of detecting missing tags in the practical scenario where unexpected tags are present. Second, we propose RUN for detecting missing tags in the presence of unexpected tags. RUN has two key advantages over prior art. First, it achieves the required reliability in the presence of unexpected tags, whereas none of the existing protocols achieves the required reliability. We have extensively evaluated and compared RUN with four state-of-the-art missing tag detection protocols (TRP [114], IIP[71], MTI[128], and SFMTI[73]) in a variety of scenarios for a large range of tag population sizes. Among existing protocols, SFMTI achieves the highest reliability of 67% whereas RUN achieves arbitrarily high reliability as per the requirement. Second, it is compliant with the C1G2 standard whereas existing protocols, except TRP, are not.

4.2 Related Work

Several probabilistic [114, 76] and deterministic [71, 128, 73] missing tag detection protocols have been proposed. The common and major drawback of all of these protocols is that none of them handle unexpected tags and assume that the readers already know the IDs of all tags that can be present in the population. Next, we review the existing probabilistic and deterministic protocols.

4.2.1 Probabilistic Protocols

The objective of the probabilistic protocols is to detect if any tags in the population are missing. Tan *et al.* proposed the first probabilistic protocol called TRP [114]. TRP pre-computes slots in a frame and compares them with the executed slots to detect missing tags. The difference with RUN, however, lies in that TRP does not consider false positives from unexpected tags. Furthermore, for large populations, TRP requires frame size that exceeds the C1G2 specified upper limit of 2^{15}, which is not possible in practical RFID systems. Among existing protocols, TRP is the only one that is compliant with the C1G2 standard as long as the frame size is below 2^{15}. Luo *et al.* proposed another probabilistic protocol called EMTD [76]. This protocol is non-compliant with the C1G2 standard because it assumes the RFID tags to be intelligent with enough computing power to implement a hash ring and calculate hashes using that ring. None of the existing probabilistic protocols have been designed to work in multiple-reader environment.

4.2.2 Deterministic Protocols

The objective of the deterministic protocols is to identify exactly which tags are missing from a population. Li *et al.* proposed a suite of protocols in [71] out of which IIP performs the best. IIP is non-compliant with the C1G2 standard due to following three reasons. First, it requires tags to interpret pre-vector frames and reply to the reader queries as described in those frames. Second, it requires frame sizes greater than 2^{15} for large populations. Last, it requires manufacturers to insert a ring of random bits in tag memory at the time of manufacturing. IIP does not handle multiple readers either. Zhang *et al.* proposed a deterministic protocol called MTI [128], which is essentially a tag collection protocol that first collects IDs of all tags and then checks which tags are missing. MTI cannot be used to achieve an arbitrary desired accuracy because the authors do not provide a frame work to calculate system parameters. Liu *et al.* proposed a deterministic protocol called SFMTI [73]. SFMTI is non-compliant with the C1G2 tags because it requires tags to interpret non-standardized vectors before and after selecting a slot in a frame.

4.3 System Model

4.3.1 Architecture

For detecting missing tags, RUN uses a central controller connected with a set of readers that cover the area where the tags in set $\mathbb{E}$ are located. The use of a central controller ensures that all readers use consistent values of frame sizes and seeds when executing frames, which helps in efficiently aggregating and processing information returned by the readers. The readers use the standardized frame slotted Aloha protocol to communicate with tags and never ask the tags to transmit their IDs. The use of multiple readers with overlapping coverage regions introduces following two problems: (1) scheduling the readers such that no two readers with overlapping regions transmit at the same time, and (2) mitigating the effect of some tags responding to multiple readers due to overlap in the coverage region of those readers. For the first problem, the controller uses one of the several existing reader scheduling protocols [116] to avoid reader-reader collisions. For the second problem, we propose solution in Section 4.4.

4.3.2 C1G2 Compliance

RUN does not require any modifications to tags or readers. It only requires the readers to receive the frame size, persistence probability, and seed number from the controller and communicate the responses in the frames back to the controller. Persistence probability p is the probability with which a tag decides whether it will participate in a frame or not before selecting a slot in that frame. Later in the chapter, we will show how we use p to handle frame sizes that exceed the C1G2 specified upper limit of 2^{15}. Such large frame sizes are required when the size of tag population is large, required reliability α is high, or the threshold T is small. As the C1G2 standard does not specify the use of p, COTS tags do not support it. To avoid making any modifications to tags, in RUN, the reader implements p by announcing a frame size of f/p but terminating the frame after the first f slots, which can be done as per the C1G2 standard.

4.3.3 Communication Channel

We assume that the communication channel between readers and tags is reliable *i.e.*, tags correctly receives queries from the readers and the readers correctly detect transmission of RN16 sequence in a slot if one or more tags in the population transmit in that slot. If the channel is unreliable, the solution proposed in [105] can be easily adapted for use with RUN.

4.3.4 Formal Development Assumption

To make the formal development tractable, we assume that instead of picking a single slot to transmit at the start of i^{th} frame of size f, a tag independently decides to transmit in each slot of the frame with probability $1/f$ regardless of its decision about previous or forthcoming slots. Vogt first used this assumption for the analysis of Aloha protocol for RFID and justified its use by recognizing that this problem belongs to a class of problems called *occupancy problem*, which deals with the allocation of balls to urns [121]. Ever since, the use of this assumption has become a norm in the formal analysis of all Aloha based RFID protocols [102, 121, 129].

The implication of this assumption is that a tag can end up choosing more than one slots in the same frame or even not choosing any at all, which is not in accordance with the C1G2 standard that requires a tag to pick exactly one slot in a frame. However, this assumption does not create any problems because the expected number of slots that a tag chooses in a frame is still one. The analysis with this assumption is, therefore, asymptotically the same as that without this assumption. Bordenave *et al.* further explained in detail why this independence assumption in analyzing Aloha based protocols provides results just as accurate as if all the analysis was done without this assumption [33]. This independence assumption is made only to make the formal development tractable. In all our simulations, a tag chooses exactly one slot at the start of a frame.

4.4 Protocol Description

To detect if any of the tags in set $\mathbb{E}$ is missing from the population, in RUN, the central controller executes up to n Aloha frames using the RFID readers. There are 6 steps involved in executing each frame. First, before executing any frame i, the controller calculates the optimal values of frame size f_i, persistence probability p_i, and generates a random seed number R_i. Second, as the controller knows the IDs in set $\mathbb{E}$, it *pre-computes* which tag in $\mathbb{E}$ will choose which slot in the i^{th} frame. Thus, it knows which slots of the executed i^{th} frame should be nonempty if all the tags in $\mathbb{E}$ were present and a single reader covered the entire population. It represents the nonempty slots in the pre-computed frame with 1s and all other slots with 0s. Third, it provides each reader with the parameters f_i, p_i, and R_i and asks each of them to execute the i^{th} frame using these parameters. The motivation behind using the same values of f_i, p_i, and R_i across all readers for the i^{th} frame is to enable RUN to work with multiple readers with overlapping regions. As all readers use the same values of f_i, p_i, and R_i in the i^{th} frame, the slot number that a particular tag chooses in the i^{th} frame of each reader covering this tag is the same *i.e.*, $h(f_i/p_i, R_i, ID)$ evaluated by the tag results in same value for each reader. Fourth, each reader executes the frame on its turn as per the reader scheduling protocol and sends the responses in the frame back to the controller. Fifth, when the controller receives the i^{th} frame of each reader, it applies logical OR operator on all the received i^{th} frames and obtains a resultant ORed frame. This resultant ORed frame is same as if received by a single reader covering all the tags. Sixth, the controller compares all the slots in the pre-computed i^{th} frame with the corresponding slots in the resultant ORed i^{th} frame. If there is any slot that is 1 in the pre-computed frame but 0 in the resultant ORed frame, the controller detects this as a missing tag event because such a slot implies that all tags in $\mathbb{E}$ that mapped to this slot in the pre-computed frame are absent from the population. At this point, the controller stops the protocol and does not execute the remaining $n - i$ frames. If the controller does not detect a missing tag event even after each reader has executed n frames, it declares that the number of missing tags m is less than the threshold T.

4.5 Parameter Optimization

Recall from the previous section that before executing any frame i, the controller calculates the optimal values of frame size f_i and persistence probability p_i. For this, the controller first estimates the value of $|\mathbb{U}|$ at the start of the i^{th} frame, represented by $|\tilde{\mathbb{U}}_i|$, based on the responses from the tag population in the previous $i-1$ frames. Details about estimating the value of $|\mathbb{U}|$ will be given in Section 4.5.1. Then, using this estimate along with the values of $|\mathbb{E}|$, α, and T, the controller calculates the optimal values of the frame size f_i and persistence probability p_i such that RUN achieves the required reliability in shortest time. Before asking the readers to execute the i^{th} frame, the controller also recalculates the maximum number of frames that it should execute, represented by n_i. As the controller executes more and more frames, *i.e.*, as i increases, the estimate $|\tilde{\mathbb{U}}_i|$ asymptotically becomes equal to $|\mathbb{U}|$. Consequently, f_i, p_i, and n_i asymptotically become equal to constants f, p, and n, respectively. When the estimate of $|\mathbb{U}|$ does not change by more than 1% in 10 consecutive frames, the controller considers the estimate to be close enough to $|\mathbb{U}|$. At this point, the controller calculates the values of f_i, p_i, and n_i and puts $f = f_i$, $p = p_i$, and $n = n_i$, and uses these fixed values of f and p to execute subsequent frames until the total number of frames executed since the first frame become equal to n. Note that the controller executes n frames only if it does not detect any missing tag event in any frame. Otherwise, it terminates the protocol as soon as it detects a missing tag event. For the first frame, *i.e.*, when $i = 1$, the controller uses $f_1 = 2 \times |\mathbb{E}|$, $p_1 = 1$, and $n_1 = \infty$. The choices of the values of f_1, p_1, and n_1 are arbitrary and do not really matter because as the controller executes more frames, the frame size, the persistence probability, and the number of frames converge to constants f, p, and n, respectively.

In rest of this section, we will derive equations that the controller uses at the start of each frame to calculate the optimal values of frame size f, number of times the frames should be repeated n, and persistence probability p to minimize the execution time of RUN while ensuring that its actual reliability is no less than the required reliability. We have dropped the subscript i from these parameters to make the presentation simple. To calculate these optimal values, the controller requires the estimate of $|\mathbb{U}|$. Next, we will first present a method to obtain this estimate at the start of any frame i based on the responses from the tag population in the previous $i - 1$ frames. Second, using the estimate of $|\mathbb{U}|$, we will derive an expression for the false positive probability, *i.e.*, the probability that a missing tag is detected as present. Third, we will use the expression for false positive probability in conjunction with the required reliability α and threshold T to obtain an equation with three unknowns f, p, and n. To ensure that the actual reliability is greater than or equal to the required reliability, the controller must use the values of f, p, and n that satisfy this equation. We call this equation the *reliability condition*. Fourth, we will derive an expression for the total execution time of RUN and minimize it with respect to n to get an expression involving p and n. The controller simultaneously solves this expression with the reliability condition using $p = 1$ to obtain the optimal values of f and n. Last, we will show how to bring the value of f within limit when the optimal value of the frame size exceeds the C1G2 specified upper limit of 2^{15}, We will also calculate the expected number of slots RUN takes to detect the first missing tag event. Next, we describe these five steps in detail.

4.5.1 Estimating Number of Unexpected Tags

In this section, we present a method to estimate the number of unexpected tags in the population at the start of any frame i. Although a lot of work has been done by the research community to estimate the number of tags present in an RFID tag population [61, 62, 102, 39], there is no work on estimating the size of some subset of RFID tag population. In our case, that subset is the set of unexpected tags in the population.

Recall from Section 4.4 that in any frame i, the slots that are 0 in the i^{th} pre-computed frame are the slots that only the tags in set $\mathbb{U}$ can select when the reader executes the i^{th} frame. This is because we have prior knowledge that the tags in set $\mathbb{E}$ will select only those slots in the i^{th} executed frame that are 1 in the i^{th} pre-computed frame. The intuition behind our estimation method is that as the number of unexpected tags in a population increases, the number of slots that are 0 in a pre-computed frame but are 1 in the corresponding executed resultant ORed frame also increase. The number of such slots in any given frame is a function of $|\mathbb{U}|$ and can, therefore, be used to estimate the value of $|\mathbb{U}|$.

Next, we derive an expression that relates the number of slots that are 0 in a pre-computed frame but are 1 in the corresponding executed resultant frame with the value of $|\mathbb{U}|$. We will use this expression to obtain the estimate of $|\mathbb{U}|$. Let the size of the i^{th} frame be f_i and let k_i out of these f_i slots be 1s in the pre-computed frames. Let j be the j^{th} 0 slot in the pre-computed frame. Thus, $1 \leq j \leq f_i - k_i$. Let X_{ij} be an indicator random variable for the event that the j^{th} 0 slot in the i^{th} pre-computed frame turns out to be 1 in the i^{th} executed resultant frame. The expected value of X_{ij} is given by

$$E[X_{ij}] = P\left\{X_{ij} = 1\right\} = 1 - \left(1 - \frac{p_i}{f_i}\right)^{|\mathbb{U}|} \approx 1 - e^{-\frac{p_i}{f_i}|\mathbb{U}|}$$

Let $\mathcal{N}_i^{01}$ be a random variable representing the number of slots that are 0 in the i^{th} pre-computed frame but 1 in the i^{th} executed resultant frame. Thus, $\mathcal{N}_i^{01} = \sum_{j=1}^{f_i - k_i} X_{ij}$. As $\left\{X_{i1}, X_{i2}, \ldots, X_{i(f_i - k_i)}\right\}$ forms a set of identically distributed random variables, $E[\mathcal{N}_i^{01}]$ is given by

$$E[\mathcal{N}_i^{01}] = E[\sum_{j=1}^{f_i - k_i} X_{ij}] = (f_i - k_i) \times E[X_{ij}] = (f_i - k_i) \times (1 - e^{-\frac{p_i}{f_i}|\mathbb{U}|}) \tag{4.1}$$

Let $\tilde{\mathcal{N}}_i^{01}$ represent the observed value of the number of slots that were 0 in the i^{th} pre-computed frame but 1 in the corresponding executed resultant frame. Replacing $E[\mathcal{N}_i^{01}]$ in the equation above with $\tilde{\mathcal{N}}_i^{01}$ and solving for $|\mathbb{U}|$ gives an estimate of $|\mathbb{U}|$. This estimate is obtained by utilizing the information from the i^{th} frame only. While this estimate may not be accurate, if we use the information from a large number of frames, the estimate will become more accurate. Specifically, we leverage the well known statistical result that the variance in the observed value of a random variable reduces by x times if we take the average of x observations of that random variable. Therefore, to obtain the estimate $|\tilde{\mathbb{U}}_i|$ of $|\mathbb{U}|$ at the start of the i^{th} frame, we obtain an estimate from each of the previous $i - 1$ frames and take their average. Solving Equation (4.1) for $|\mathbb{U}|$ and averaging over past $i - 1$ frames, the formal expression for $|\tilde{\mathbb{U}}_i|$ becomes

$$|\tilde{\mathbb{U}}_i| = -\frac{1}{i-1} \sum_{l=1}^{i-1} \frac{f_l}{p_l} \ln\left\{1 - \frac{\tilde{\mathcal{N}}_l^{01}}{f_l - k_l}\right\} \tag{4.2}$$

Finally, note that the controller obtains this estimate without executing any additional frames. It gets this estimate from the frames it was already executing to detect missing tag events.

4.5.2 False Positive Probability

A false positive occurs when all the slots that a particular missing tag maps to in the n pre-computed frames turn out to be nonempty when the frames are executed because some other tags in the population also selected those slots. Lemma 9 gives the expression to calculate the false positive probability.

Lemma 9. *Let m out of $|\mathbb{E}|$ tags be missing, and let there be $|\mathbb{U}|$ unexpected tags in the population. With persistence probability p, frame size f, and number of frames n, the false positive probability, P_{fp}, is given by:*

$$P_{fp} = \left\{1 - p\Big(1 - \frac{p}{f}\Big)^{|\mathbb{U}|+|\mathbb{E}|-m}\right\}^n \tag{4.3}$$

Proof. The total number of tags in the population are $|\mathbb{U}| + |\mathbb{E}| - m$. Consider an arbitrary tag in $\mathbb{E}$ that is missing from the population. As this tag participates in each pre-computed frame with probability p, it is possible that it does not participate in one or more of the n pre-computed frames. Let Z be the random variable for the number of pre-computed frames in which this missing tag participates. Let q be the probability that a slot that this missing tag maps to in a pre-computed frame is selected by one or more of the tags present in the population in the executed frame. Therefore,

$$P_{fp} = \sum_{z=0}^{n} P\{Z = z\} \times q^z \tag{4.4}$$

As a missing tag participates in each pre-computed frame with probability p and there are n pre-computed frames, the number of pre-computed frames in which the missing tag participates follows a binomial distribution *i.e.*, $Z \sim \text{Binom}(n, p)$. When a frame is executed, probability that at least one tag in the population chooses the same slot to which the missing tag maps in the pre-computed frame is $1-(1-\frac{p}{f})^{|\mathbb{U}|+|\mathbb{E}|-m}$, which is the value of q. Therefore, Equation (4.4) becomes

$$P_{fp} = \sum_{z=0}^{n} \binom{n}{z} p^z (1-p)^{n-z} \left\{1-(1-\frac{p}{f})^{|\mathbb{U}|+|\mathbb{E}|-m}\right\}^z$$

The binomial theorem states that $\sum_{z=0}^{n}\binom{n}{z}x^z y^{n-z} = (x+y)^n$. Substituting $x = p \times \{1 - (1-\frac{p}{f})^{|\mathbb{U}|+|\mathbb{E}|-m}\}$ and $y = 1-p$, we get Equation (4.3). □

Figure 4.1 shows the theoretically calculated false positive probability from Equation (4.3) represented by the solid line and experimentally observed values of false positive probability represented by the dots. To obtain this figure, we use $|\mathbb{E}| = 100$, $|\mathbb{U}| = 500$, $f = 300$, $p = 1$, and $n = 2$. Each dot represents the false positive probability calculated from 100 runs of simulation. We observe that the theoretically calculated values match perfectly with experimentally observed values, showing that our independence assumption that we stated in Section 4.3.4 does not cause the theoretical analysis to deviate from practically observed values. We also observe that as the number of missing tags increases, the false positive probability decreases. This means that it is hardest for RUN to detect a missing tag event when $m = T$ and becomes easier as m increases beyond T. Thus, we will use $m = T$ in all further analytical development, because if RUN is able to detect a missing tag event with probability α when $m = T$, it will be able to detect a missing tag event with probability greater than α when $m > T$.

4.5.3 Achieving Required Reliability

Following theorem gives the *reliability condition* that the values of f, p, and n need to satisfy in order for RUN to be able to achieve the required reliability.

Theorem 10. *Given a set $\mathbb{E}$ with expected IDs, set $\mathbb{U}$ with unexpected IDs, threshold T, and required reliability α, RUN will achieve the required reliability if the values of f, p, and n satisfy the reliability condition given below.*

$$f = \frac{p(T - |\mathbb{E}| - |\mathbb{U}|)}{\ln\left\{\frac{1-(1-\alpha)^{\frac{1}{nT}}}{p}\right\}} \tag{4.5}$$

Proof. Probability that RUN detects at least one of the missing tags is $1 - P_{fp}^T$. In the worst case, this probability should at least be equal to α *i.e.*, $1 - P_{fp}^T = \alpha$. Substituting the R.H.S of Equation (4.3) for P_{fp} gives

$$1-\alpha = \left\{1 - p\left(1 - \frac{p}{f}\right)^{|\mathbb{U}|+|\mathbb{E}|-T}\right\}^{nT} \approx \left\{1 - pe^{-\frac{p}{f}(|\mathbb{U}|+|\mathbb{E}|-T)}\right\}^{nT}$$

Rearranging the equation above gives Equation (4.5). □

4.5.4 Minimizing Execution Time

Following theorem gives the condition that the values of p and n need to satisfy to make the execution time of RUN minimum under the constraint that it achieves the required reliability.

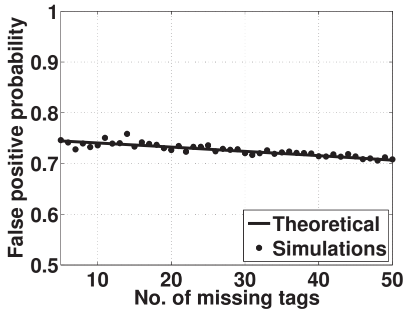


Figure 4.1P_{fp}

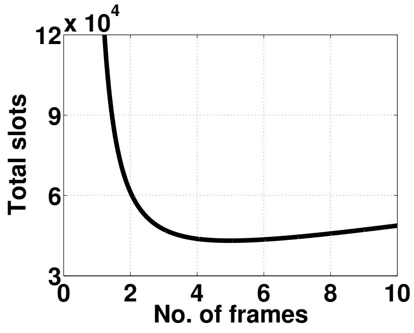


Figure 4.2 S_d vs. n

Theorem 11. *Given a threshold T and required reliability α, the execution time of RUN is minimum under the constraint that it achieves the required reliability if the values of p and n satisfy the following equation:*

$$p = \left\{1 - (1-\alpha)^{\frac{1}{nT}}\right\}\left\{(1-\alpha)^{\frac{(1-\alpha)^{\frac{1}{nT}}}{nT(-1+(1-\alpha)^{\frac{1}{nT}})}}\right\} \tag{4.6}$$

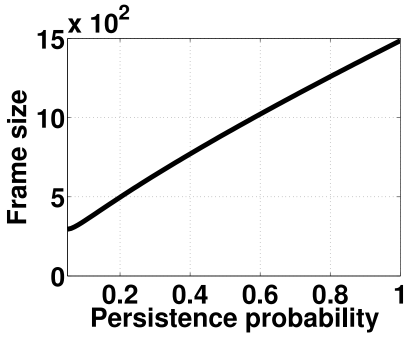


Figure 4.3 f vs. p

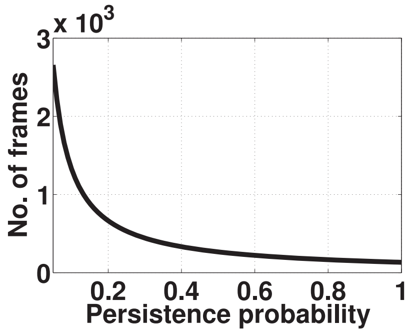


Figure 4.4 n vs. p

Proof. Execution time is directly proportional to the total number of slots required to detect the missing tag event because the duration of each slot is the same, typically $300\mu s$ for Philips I-Code RFID reader [100]. Let S_d represent the total number of slots. Thus, $S_d = f \times n$. To ensure that RUN achieves the required reliability, we use the value of f from Equation (4.5). Thus,

$$S_d = \frac{pn(T - |\mathbb{E}| - |\mathbb{U}|)}{\ln\left\{\frac{1-(1-\alpha)^{\frac{1}{nT}}}{p}\right\}} \tag{4.7}$$

Figure 4.2 plots S_d as a function of n. We observe that S_d is a convex function of n. Therefore, optimum value of n exists, represented by n_{op}, that minimizes the total number of slots S_d. To find optimal value of n, we differentiate Equation (4.7) with respect to n and equate the resulting expression to 0, which gives Equation (4.6). □

At the start of each frame, the controller replaces $|\mathbb{U}|$ with its estimate, puts $p = 1$ in Equation (4.6), and solves it numerically using Brent's method to obtain the optimal value of number of frames n_{op}. Then it puts $n = n_{\text{op}}$ and $p = 1$ in Equation (4.5) to get the optimal value of frame size f_{op}. When the controller calculates f_{op} and n_{op} like this at the start of each frame, the execution time of RUN is minimized. At the same time, as the reliability condition is satisfied, the protocol achieves the required reliability.

4.5.5 Handling Large Frame Sizes

For large populations, high required reliability, and/or small threshold, it is possible for the value of f_{op} to exceed the C1G2 specified upper limit of 2^{15}. Next, we describe how we use p to bring the frame size within limits. Bringing the frame size within limits comes at a cost of increased number of slots; greater than the minimum value of S_d that would have been achieved if the controller could use $f_{\text{op}} > 2^{15}$.

When we decrease the value of p, the number of tags that participate in a frame decrease. Therefore, intuitively, the required value of f should also decrease. Figure 4.3 confirms this intuition. This figure shows the plot of frame size vs. persistence probability, obtained using Equations (4.5) and (4.6). We can see that when p decreases, f decreases. Participation by lesser tags means that participation by the tags belonging to both the sets $\mathbb{E}$ and $\mathbb{U}$ decreases. This increases the chances that a given missing tag will not map to any slot in a given pre-computed frame, which means that chances of detecting its absence decrease. Therefore, the overall uncertainty in detection of missing tags increases. To reduce this uncertainty, intuitively, the value of n should increase when p decreases to achieve the required reliability. Figure 4.4 confirms this intuition. This figure shows the plot of number of frames vs. persistence probability, obtained using Equations (4.5) and (4.6). We observe that when p decreases, n increases.

We use these two observations to reduce the value of f whenever $f_{\text{op}} > 2^{15}$. When $f_{\text{op}} > 2^{15}$, the controller uses $f = f_{\max} = 2^{15}$ in Equation (4.5), which leaves two unknowns, p and n, in the resulting equation. The controller solves the resulting equation simultaneously with Equation (4.6) to get new values of p and n. The new value of p is less than 1 and the new value of n is greater than n_{op} because $f_{\max} < f_{\text{op}}$. Putting $f = f_{\max}$ in Equation (4.5) and solving for n, we get

$$n = \frac{\ln\{1-\alpha\}}{T \ln\left\{1 - pe^{\frac{p}{f_{\max}}(T-|\mathbb{E}|-|\mathbb{U}|)}\right\}} \tag{4.8}$$

Replacing n in Equation (4.6) with the R.H.S of the equation above, and simplifying, we get

$$\frac{p^2(T-|\mathbb{E}|-|\mathbb{U}|)}{f(e^{\frac{p}{f_{\max}}(|\mathbb{E}|+|\mathbb{U}|-T)}-p)} = \ln\left\{1-pe^{\frac{p}{f_{\max}}(T-|\mathbb{E}|-|\mathbb{U}|)}\right\}$$

The numerical solution of the equation above gives the new value of p, which the controller puts in Equation (4.8) to get the new value of n. The controller uses these new values of n and p along with $f = f_{\max}$ to pre-compute the i^{th} frame. Although the total number of slots $S_d = f_{\max} \times n > f_{\text{op}} \times n_{\text{op}}$, this is still the smallest under the constraints that the required reliability is achieved and the frame size does not exceed $f_{\max}$.

4.5.6 Expected Detection Time

The values of f and n that we calculate as described in the sections above ensure that in executing n frames, RUN will detect a missing tag event with probability greater than or equal to α if number of missing tags is greater than or equal to T. However, in many cases, the first missing tag event is detected before all n frames are executed. We calculate the expected value of the number of slots that RUN takes to detect the first missing tag event. For this, we calculate the probability that a missing tag event is detected in a given slot and use it to calculate the expected value.

Lemma 12. *Given a set $\mathbb{E}$ with expected IDs, set $\mathbb{U}$ with unexpected IDs, and threshold T, when controller executes RUN with persistence probability p and frame size f, the probability g that a missing tag event is detected in any slot is given by the following equation.*

$$g = \left\{1-\left(1-\frac{p}{f}\right)^T\right\} \times \left\{\left(1-\frac{p}{f}\right)^{|\mathbb{U}|+|\mathbb{E}|-T}\right\} \tag{4.9}$$

Proof. Probability that a missing tag event is detected in a given slot is the product of the probability that at least one missing tag maps to this slot in the pre-computed frame and the probability that no tag in the population selects that slot in the executed frame. Considering the scenario where it is hardest for RUN to detect a missing tag event *i.e.*, when $m = T$, probability that at least one of the missing tags maps to the given slot in the pre-computed frame is $1-\left(1-\frac{p}{f}\right)^{T}$. The probability that none of the tags present in the population selects that slot is $\left(1-\frac{p}{f}\right)^{|\mathbb{U}|+|\mathbb{E}|-T}$. The product of these two probabilities gives the expression for g in Equation (4.9). □

Following theorem gives the expected value of the number of slots that RUN takes to detect the first missing tag.

Theorem 13. *Let D be the random variable for the slot number when the first missing tag event is detected. Given that the probability of detecting a missing tag event in a slot is g, as calculated in Lemma 12, frame size is f, and number of frames is n, we get*

$$E[D] = \frac{1-(1-g)^{fn} - fng(1-g)^{fn}}{g} \tag{4.10}$$

Proof. The random variable D follows geometric distribution with parameter g *i.e.*, $P\{D=i\} = (1-g)^{i-1}g$. The expected value, thus, becomes

$$E[D] = \sum_{i=1}^{S_d} iP\{D=i\} = \sum_{i=1}^{f\times n} ig(1-g)^{i-1} = \frac{1-(1-g)^{fn} - fng(1-g)^{fn}}{g}$$

□

4.6 Performance Evaluation

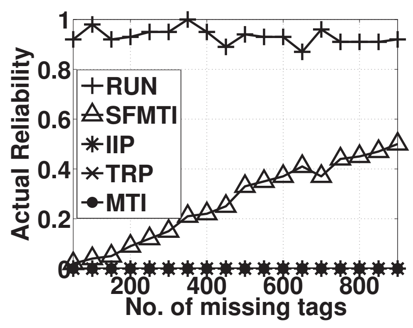
We implemented RUN in Matlab. Although, none of the existing protocols handles the presence of unexpected tags and except for TRP, none of them is C1G2 compliant, we still implemented four prior state of the art missing tag detection protocols in Matlab namely TRP [114], IIP[71], MTI[128], and SFMTI[73], and compared their performance with RUN. We calculated parameter values for these protocols by following the instructions in their respective papers. We also implemented the fastest existing tag collection protocol TH [105]. We choose tag ID length of 64 bits as specified in the C1G2 standard. Note that the distributions of the IDs of expected, unexpected, and missing tags do not matter because RUN is independent of ID distributions.

We first evaluate the actual reliability of RUN and the existing protocols for multiple values of required reliability, keeping the unexpected tag population size fixed and changing the number of missing tags. We also show the time taken by each protocol to detect the first missing tag event. Second, we evaluate the actual reliability of RUN and the existing protocols for multiple values of required reliability by keeping the number of missing tags fixed and changing the unexpected tag population size. We again show the time taken by each protocol to detect the first missing tag event. Third, we study the actual reliability achieved by each protocol when the number of tags missing from the population is different from the value of threshold T. Last, we compare the detection times of our protocols with the fastest tag collection protocol TH.

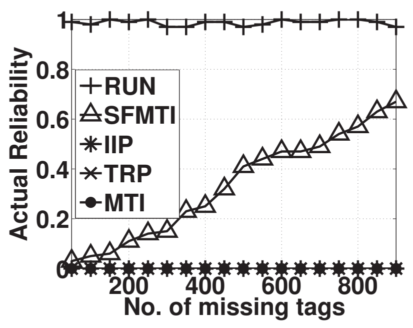
4.6.1 Impact of Number of Missing Tags

RUN is the only protocol that achieves the required reliability in the presence of unexpected tags for any number of missing tags. Figures 4.5(a) and 4.5(b) show the actual reliability achieved by RUN and all existing protocols for $\alpha = 0.9$ and 0.99, respectively. These figures are plotted using $|\mathbb{E}| = 1000$, $|\mathbb{U}| = 10000$ and m is varied from 50 to 900. The actual reliabilities are obtained using 100 runs of each protocol for each value of m. None of the existing protocols achieves the required reliability because none of them is designed to handle unexpected tags. Among the existing protocols, SFMTI has the highest actual reliability of up to 0.67

RUN is the fastest protocol that achieves the required reliability compared to the existing protocols. Figures 4.6(a) and 4.6(b) show the average times each protocol took to either detect the first missing tag event if it finds a missing tag or to complete execution if it does not find a missing tag. From these figures, MTI seems to have smaller detection time compared to RUN, but when we observe these figures in conjunction with Figures 4.5(a) and



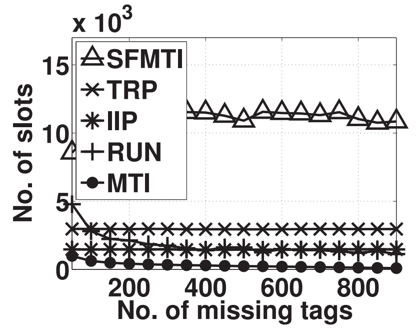
(a) $\alpha = 0.90$



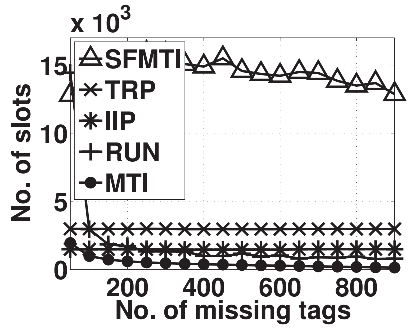
(b) $\alpha = 0.99$

Figure 4.5 Actual reliability vs. missing tags

4.5(b), we see that the actual reliability of MTI is close to 0, far lower than the required reliability. This shows that for majority of times, MTI completed execution without detecting any missing tags due to the unexpected tags.



(a) $\alpha = 0.90$

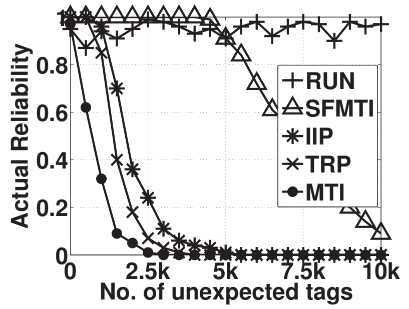


(b) $\alpha = 0.99$

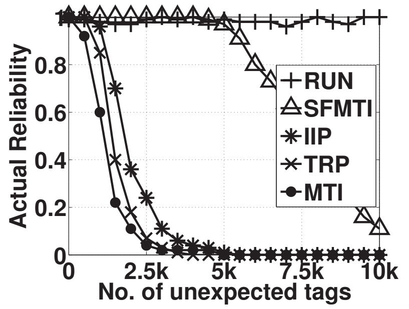
Figure 4.6 Detection time vs. missing tags

4.6.2 Impact of Number of Unexpected Tags

RUN is the only protocol that achieves the required reliability in the presence of unexpected tags while existing protocols achieve the required reliability only when there are no unexpected tags in the population. Figures 4.7(a) and 4.7(b) show the actual reliability obtained by RUN and the existing protocols for $\alpha = 0.9$, and 0.99, respectively. These figures are plotted using $|\mathbb{E}| = 1000$, $m = 200$, and $|\mathbb{U}|$ is varied from 0 to 10000. RUN always achieves the required reliability whereas the existing protocols achieve the required reliability only when $|\mathbb{U}|$ is close to zero.



(a) $\alpha = 0.90$



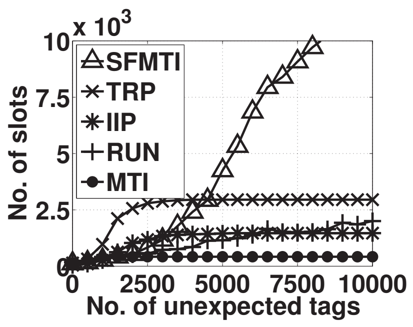
(b) $\alpha = 0.99$

Figure 4.7 Actual reliability vs. number of unexpected tags

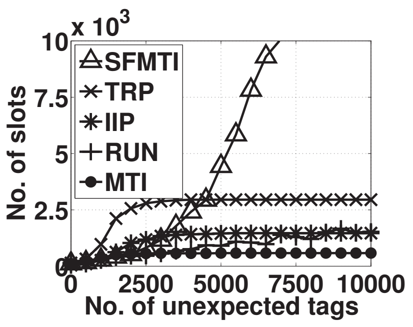
RUN is the fastest protocol that achieves the required reliability compared to the existing protocols even when there are no unexpected tags in the population. Figures 4.8(a) and 4.8(b) show the average times each protocol took to either detect the first missing tag event or complete execution without detecting any missing tags. From these figures, MTI again seems to have smaller detection time compared to RUN when number of unexpected tags in the population is large, but when we analyze these figures in conjunction with Figures 4.7(a) and 4.7(b), we see that actual reliability of MTI is close to 0 when number of unexpected tags in the population is large. Figures 4.7(a) and 4.7(b) show that SFMTI achieves the required reliability for up to 5000 unexpected tags, but then Figures 4.8(a) and 4.8(b) show that its execution time is 5 times greater than RUN.

4.6.3 Impact of Deviation from Threshold

The actual reliability of RUN exceeds the required reliability when the number of missing tags in the population exceed the threshold T. This is seen in Figure 4.9, which plots the actual reliabilities of all protocols when number of missing tags are larger or smaller compared to T. This figure is made using $|\mathbb{E}| = 1000$, $|\mathbb{U}| = 10000$, $T = 200$, $\alpha = 0.99$, and m is varied



(a) $\alpha = 0.90$



(b) $\alpha = 0.99$

Figure 4.8 Detection time vs. number of unexpected tags

from 50 to 900. The actual reliability of RUN is less than the required reliability only when the number of missing tags are less than T, but this is insignificant because we are interested in detecting the missing tags only if the number of missing tags in a population exceed the threshold T.

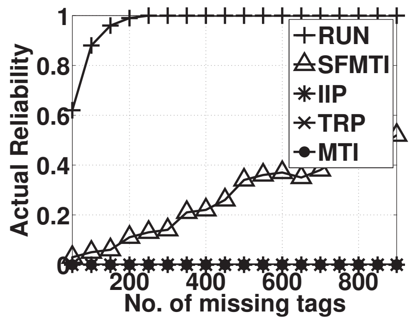


Figure 4.9 Effect of difference between m and T

4.6.4 Comparison with Tag ID Collection Protocol

RUN is faster than the fastest tag ID collection protocol, TH, in all practical scenarios. For example, for $|\mathbb{E}| = 1000$, $|\mathbb{U}| = 10000$, and $T = m = 200$, RUN is 14.3 times faster than TH for $\alpha = 0.99$. As the threshold T decreases and/or the required reliability increases, detection time of RUN increases. Therefore, there exists a value of T and/or α for a given $|\mathbb{E}|$ and $|\mathbb{U}|$ for which the tag ID collection protocol is faster than RUN. For example, for $|\mathbb{E}| = 1000$,

$|\mathbb{U}| = 10000$, and $T = 200$, TH is faster than RUN when required α is greater than 0.99999. Such high values of α are seldom required. Similarly, for $|\mathbb{E}| = 1000$, $|\mathbb{U}| = 10000$, and $\alpha = 0.99$, TH is faster than RUN for $T < 0.001$. In all practical scenarios, the threshold can not be less than 1. Therefore, practically, RUN is always faster than the tag ID collection protocols.

4.7 Conclusions

The key technical contribution of this chapter is in proposing a protocol to detect missing tag events in the presence of unexpected tags. This chapter represents the first effort on addressing this important and practical problem. The key technical depth of this chapter is in the mathematical development of the theory that RUN is based upon. The solid theoretical underpinning ensures that the actual reliability of RUN is greater than or equal to the required reliability. We have proposed a technique that our protocol uses to handle large frame sizes to ensure compliance with the C1G2 standard. We have also proposed a method to implicitly estimate the size of the unexpected tag population without requiring an explicit estimation phase. We implemented RUN and conducted side-by-side comparisons with four major missing tag detection protocols even though the existing protocols do not handle the presence of unexpected tags. Our protocols significantly outperform all prior protocols in terms of actual reliability as well as detection time.

5 Per-flow Latency Measurement

5.1 Introduction

5.1.1 Motivation

Although traditionally throughput has been the primary focus of network engineers, nowadays latency has seen growing importance because a wide variety of emerging applications and architectures require extremely low (in microseconds) and stable (low jitter) latencies. First, many emerging applications, such as financial trading applications [79], storage applications utilizing Fiber Channel over Ethernet [8], and high performance computing applications in data center networks [21], demand low latency. A small increase in latency may cause violations of service level agreements and result in significant revenue losses. For example, a one-millisecond advantage in financial trading applications can be worth $100 million a year for major brokerage firms [79]. Second, many emerging architectures, such as content delivery networks (CDNs) and mission-critical data center networks, demand low latency. CDN providers are mostly evaluated and ranked by content publishers based on latency. Companies such as Cedexis [5] and Turbobytes [18] constantly evaluate and rank CDN providers mostly based on latency. A one-millisecond disadvantage could put one CDN provider behind others and result in loss of business with content publishers. Similarly, the transit providers are primarily evaluated and ranked by CDN providers based on latency. For data centers running mission-critical applications, latency guarantee is a key requirement for the underlying networks. Low latency data center networks have become the primary focus of many data center network solution providers such as Sidera [13].

In managing networks with stringent latency demands, operators often need to measure the latency between two observation points for a particular flow. An *observation point* is either a port in a middlebox (such as a router or a switch) or a network card in an end host. Per-flow latency measurement can be used reactively by network operators to perform tasks such as detecting and localizing delay spikes in a network, isolating offending flows that are responsible for causing delay bursts, and rerouting them through other paths. It can also be used proactively by network operators to continuously monitor latencies between observation points for locating bottleneck links and replace them with higher capacity links.

Existing routers and switches provide little help for latency measurement and monitoring. SNMP counters measure the number of packets passing through a port. NetFlow measures basic statistics, such as the numbers of packets and bytes, of a flow. Both provide no measurement on latency. Network operators often rely on injecting probe packets to measure end-to-end delays and then use tomographic techniques to infer link and hop properties [40, 45]. However, to achieve latency measurement with extremely high accuracy, the required number of probe packets will be extremely large; consequently, the probe packets will consume too much bandwidth and the measured latency does not reflect the real latency without probe packets. Although some specialized latency monitoring devices are commercially available, they are too costly to be widely deployed. For example, London, Singapore, and Tokyo stock exchanges use latency monitoring devices manufactured by Corvil [19, 14, 17] costing around USD 180,000 for a 2×10Gbps box [7].

5.1.2 Problem Statement

This chapter addresses the fundamental problem of *per-flow latency measurement*: for any flow that passed through any two observation points, measure (or say estimate) the average and standard deviation of the latencies experienced by the packets of that flow in passing through the two observation points. Formally, *given a confidence interval* $\beta \in (0, 1]$ *and* *a required reliability $\alpha \in [0, 1)$, for any flow f that passed through any two observation points S and R, obtain estimates $\tilde{\mu}_f$ of the average μ_f and $\tilde{\sigma}_f$ of the standard deviation σ_f of the latencies experienced by the packets of f in passing through S and R so that $P\left\{|\tilde{\mu}_f - \mu_f| \leq \beta\mu_f\right\} \geq \alpha$ and $P\left\{|\tilde{\sigma}_f - \sigma_f| \leq \beta\sigma_f\right\} \geq \alpha$.*

An accurate per-flow latency measurement scheme should further satisfy the following two requirements. (1) *No packet probing*: As probe packets may use up a significant portion of network bandwidth, the latency measured with the insertion of probe packets may significantly deviate from the real latency. Thus, the estimates obtained with probe packets may not suffice for microsecond level accuracy. (2) *No time stamping*: First, IP headers do not have a time stamp field and the TCP time stamp option is meant for measuring end-to-end latencies. Embedding time stamps at observation points requires modifications to packet header formats, which further requires modifications to the data forwarding paths of existing routers and middleboxes. Furthermore, the added packet header fields may consume a significant portion of network bandwidth. Second, the process of attaching time stamps to each packet takes a non-negligible amount of time at observation points. Thus, the latency measured with time stamping may significantly deviate from the real latency.

5.1.3 Limitations of Prior Art

To the best of our knowledge, there are only two per-flow latency measurement schemes, namely RLI [68] and MAPLE[69]. However, neither of them satisfies both requirements because RLI uses packet probing and MAPLE attaches time stamps to every packet. Other than RLI and MAPLE, the closest work is LDA [63], which performs *aggregate latency measurement*, i.e., *given any two observation points, measure (or say estimate) the average and standard deviation of the latencies experienced by all the packets that passed through the two observation points, regardless of the flow that each packet belongs to.* Aggregate latency measurement is useful; however, it does not provide fine grained per-flow latency information. Here is an important fact: for all flows that passed through two arbitrary observation points S and R, the latencies experienced by the packets of different flows in passing through S and R can be quite different. First, there may be multiple paths from S to R and different flows may be routed via different paths. Second, at each intermediate middlebox along a path from S to R, packets of different flows may take different amount of processing time due to mechanisms such as QoS. As aggregate latency measurement does not reflect the latency of every flow, it falls short in engineering latency sensitive networks. On one hand, when the aggregate latency between two observation points appears normal, the latency experienced by an individual flow may be wildly abnormal. On the other hand, when the aggregate latency between two observation points appears abnormal, aggregate latency measurement does not provide operators the per-flow latency information needed to identify the flows being hurt.

5.1.4 Proposed Approach

In this chapter, we propose COLATE, a Counter based Per-flow Latency Estimation scheme. The key idea of COLATE is that it records timing information of packets at each observation point and purposely allows noise to be introduced in the recorded timing information for minimizing storage space. When querying the latency of a target flow, COLATE statistically denoises the recorded information to obtain an accurate latency estimate. COLATE has two phases: recording phase and querying phase. Next, we give an overview of these two phases.

5.1.4.1 Recording Phase

In this phase, at each observation point, COLATE records the timing information of each packet arriving at or departing from that point using a vector of counters in RAM, which we call *counter vector*. For each flow with a unique ID, COLATE maps it to a unique subset of these counters, which we call *counter subvector*. The ID of a flow can be any flow identifier such as the standard five tuple (*i.e.*, source IP, destination IP, source port, destination port, and protocol type). To make the mapping unique and memoryless (*i.e.*, using no memory to keep track of the mapping), COLATE maps each flow to a random subvector such that the probability of two different flows being mapped to the same subvector is practically zero. A counter may belong to multiple counter subvectors. Figure 5.1 shows an example counter vector and its three counter subvectors, from which we see that counters 5 and 8 belong to multiple counter subvectors. For each arriving or departing packet at the observation point, COLATE executes two simple steps: (1) randomly maps the packet to a counter in the counter subvector of the flow that the packet belongs to; (2) adds the current time to that counter. Before any counter overflows, COLATE dumps the counter vector to a permanent storage (such as a solid state drive (SSD)) and resets counters to zero. We call a dumped counter vector a *counter epoch*, which has two attributes: the time stamp of the first recorded packet and the time stamp of the last recorded packet. The recording module can be implemented in hardware to keep up with wire speed. For the hashing function, we can use hardware hash implementations such as those proposed in [54, 91]. For each counter, we can store its less significant bits as a counter in SRAM and the more significant bits as a counter in DRAM – when the counter in SRAM overflows, we increment the corresponding counter in DRAM.

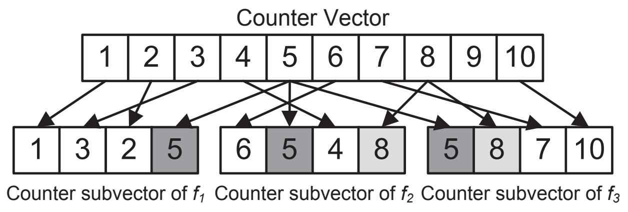

Counter Vector
1 2 3 4 5 6 7 8 9 10
1 3 2 5
6 5 4 8
5 8 7 10
Counter subvector of f_1
Counter subvector of f_2
Counter subvector of f_3

Figure 5.1 Counter vector and subvectors

5.1.4.2 Querying Phase

In this phase, given a latency measurement query of a flow f, which contains the flow ID, the starting and ending time of the flow, and two observation points that the flow passed through within the time frame, COLATE first finds all the counter epochs whose time frame overlaps with the starting and ending time of the flow f at each of the two given observation points.

Second, for each counter epoch, COLATE applies statistical techniques to estimate the sum of time stamps contributed only by flow f from each counter in the counter subvector of f. COLATE uses these extracted values to estimate the average and standard deviation of the latencies experienced by the packets of flow f in passing through the two observation points. COLATE requires the clocks of different observation points to be accurately synchronized; otherwise, the measured latencies will contain a constant offset. This synchronization can be simply achieved by the standard time synchronization protocol IEEE 1588, which provides microsecond level time synchronization [10].

Key Intuition: The intuition behind mapping a flow to multiple counters (in a counter subvector), instead of a single counter, is to mitigate counter overflow for elephant flows. The motivation behind sharing counters among multiple flows, instead of allocating unique counters for each individual flow, is to save memory. Due to the sharing of counters among multiple flows, the counter subvector of a flow f contains not only the timing information of the packets in f but also that of the packets in other flows. The intuition behind allowing this "mixing" is that later in the querying phase, we can extract the timing information of only the packets in the flow f using statistical techniques by treating the mixed-in timing information of the packets from other flows as noise.

Deployability: COLATE is designed to be efficiently implementable on network middleboxes (such as routers and switches) from both processing overhead and storage space perspectives. In terms of processing overhead, COLATE performs only one hash and one memory update per packet. On traditional memory architecture, one memory update requires two memory accesses (*i.e.*, one read and one write); however, in modern memory architecture used in high speed routers (such as the smart memory architecture developed by Huawei [77] and the bandwidth engine developed by MoSys [80]), where each memory location has built-in circuitry for handling updates on site, one memory update (such as incrementing by up to a 64-bit number) requires only one memory access. In terms of storage space, COLATE uses less than 0.1 bit per packet. To get an idea of the length of time for which COLATE can accumulate time stamps using commodity permanent storage devices, consider the 10GigE backbone link in San Jose monitored by CAIDA. At the time of writing this chapter, an interactive tool at CAIDA's website reported that on average approximately 0.484 million packets traversed this link per second between Nov. 01 and Nov. 29, 2013 [3]. With 0.1 bits per packet, on a commodity 256GB SSD, COLATE can accumulate time stamps of packets traversing this link for about 1.5 years, which means that a network operator can measure average and standard deviation of up to 1.5 years old flows. This gives not only enough time to identify and debug any problems, but also enough information to study other aspects related to packet delays such as diurnal patterns in flow latencies. On a 12TB SSD, as recently showcased by OCZ at CES-2012 [20], COLATE can accumulate time stamps of packets traversing this link for more than 69 years.

Packet Losses: COLATE, as described above, assumes no packet losses. However, to handle packet losses in COLATE, we can easily adapt the strategy proposed in [63] for handling packet losses in the aggregate latency measurement scheme LDA. According to this strategy, instead of maintaining a single counter vector, COLATE can maintain a set of counter vectors at each observation point, where each counter vector has a sampling probability and a packet counter associated with it. The sum of the sampling probabilities of all counter vectors is 1. The sampling probability of a counter vector is the fraction of packets whose time stamps COLATE will add to this counter vector. The packet counter of a counter vector keeps track of the number of packets whose time stamps have been added to this counter vector. In the recording phase, when a packet arrives at or departs from an observation point, COLATE first uses a hash function to map the packet to a counter vector such that in the long run, the fraction of packets that it maps to any counter vector is equal to its sampling probability. Then, COLATE adds the time stamp of the packet to this vector as described earlier. Note that all observation points use the same hash function to guarantee that the same packet is mapped to the same counter vector at all observation points. In the querying phase, for a target flow between two observation points, COLATE compares the packet counter of each counter vector at one observation point with that of the corresponding counter vector at the other observation point. Then, COLATE selects those two counter vectors at the two observation points that have the highest sampling probability associated with them and equal values of packet counters. After this, COLATE follows the procedure of the querying phase described earlier. For simplicity, in the rest of this chapter in describing COLATE, we assume no packet losses.

5.1.5 Technical Challenges and Solutions

The first challenge is to denoise the recorded information to extract the sum of the time stamps of the packets in the target flow from the counter subvector of the flow. To address this challenge, we first show that for each counter in the counter subvector of the target flow, the value contributed by the time stamps from the packets in the target flow and that from the packets in other flows can both be modeled with binomial distributions. We then derive an expression to calculate the expected value of each counter in the counter subvector of the target flow. From this expression, we estimate the sum of the time stamps of all the packets of the target flow. Using this estimate in conjunction with maximum likelihood estimation, we extract the sum of the time stamps of the packets in the target flow from each counter in the counter subvector.

The second challenge is to calculate the sum of the squares of the latencies of each packet in a target flow, which is needed for calculating the standard deviation of packet latencies. To address this challenge, we first use the time stamp sum extracted from counter subvectors to construct a virtual deviation vector and then use this vector to estimate this sum of the squares of the latencies.

5.1.6 Advantages of COLATE over Prior Art

COLATE brings forward the state-of-the-art in per-flow latency measurement on the following fronts: reliability, passiveness, scalability, memory, efficiency, and flexibility. For reliability, COLATE takes the required reliability and confidence interval specified by network operators as input whereas existing schemes do not. For passiveness, COLATE neither sends probe packets nor attaches time stamps to packets. For scalability, COLATE maintains only one counter vector at each observation point regardless of how many other observation points are sending and receiving packets from it; in contrast, LDA and RLI have to maintain separate vectors of counters for each pair of sender and receiver. For memory, COLATE uses less than 0.1 bit of storage space per packet, which is over 128 *times* improvement compared to 12.8 bits of storage space per packet used by MAPLE. Due to this, on a commodity 256GB SSD, where COLATE can accumulate time stamps of packets traversing the San Jose backbone link for about one and a half year, MAPLE can accumulate the time stamps for only 4.1 days. For efficiency, COLATE performs only 1 hash and 1 memory update per packet whereas MAPLE uses 9 hashes and 12 memory accesses per packet. For flexibility, COLATE allows different observation points to allocate different amount of RAM based on their available resources whereas LDA requires both the sender and receiver to allocate the same amount of RAM.

5.2 Related Work

To the best of our knowledge, there are only two per-flow latency measurement schemes, namely MAPLE[69] and RLI [68]. In MAPLE, for any packet passing through two observation points S and R, S attaches a time stamp to the packet and R calculates the latency of the packet from S to R by subtracting the time stamp from its current time. To reduce space for storing latency values of all packets, MAPLE maps the calculated latency of each packet to the closest value in a set of predetermined latency values. Thus, instead of storing the latency of every packet, MAPLE only stores these predetermined latency values and for each predetermined latency value MAPLE uses a Bloom filter to store all packets mapped to it. To query the latency of a given packet, MAPLE first finds the predetermined latency value that the packet was mapped to by querying Bloom filters and uses that value as the estimated latency of the packet. Compared with COLATE, MAPLE falls short from a few perspectives. First, MAPLE is based on a strong assumption that packet latencies between two observation points are tightly clustered around a set of predetermined latency values. We have not found any theoretical or empirical validation of this assumption in prior literature. Second, MAPLE requires attaching time stamps to packets and thus has the limitations we pointed out in Section 5.1.2 for time stamping based latency measurement schemes. Furthermore, time stamping individual packets can consume up to 10% of the available bandwidth [63]. Third, MAPLE has large memory overhead (12.8 bits/packet at each observation point) and large processing overhead (9 hash computations and 12 memory accesses per packet: 9 for hash functions, 2 for updating counters, and 1 for determining the cluster a packet's latency belongs to), whereas COLATE uses 0.1 bits/packet and performs 1 hash and 1 memory update per packet.

In RLI, for a flow passing through two observation points S and R, S inserts probe packets with time stamps into the flow and R calculates the latency of each probe packet similarly to MAPLE. To calculate the latency of the regular packets between two probe packets whose latency has been calculated as l_1 and l_2, R simply uses the straight line equation to calculate the latency of these regular packets based on their arrival time and the latency of the two probe packets. Compared with COLATE, RLI has the following limitations. First, RLI is based on the strong assumption that packet latency between S and R increases or decreases linearly in the time interval between receiving any two probe packets at R. When the time interval between two probe packets is extremely small, this assumption may practically hold but the extremely small time interval implies that the number of probe packets is extremely large. For example, to achieve an accuracy of only 81%, RLI inserts, on average, 1 probe packet every 4.78 regular packets. Furthermore, as we mentioned in Section 5.1.2, latency measured with a large number of probe packets may significantly deviate from the real latency when there are no probe packets. When the time interval between two probe packets is large, this assumption does not make intuitive sense and may not hold in practice. Similarly, we have not found any theoretical or empirical validation of this assumption in prior literature.

The other latency measurement schemes are LDA [63] and FineComb [70], which provide aggregate, not per-flow, latency measurement between a sender and a receiver. In LDA, both the sender and the receiver maintain several counter vectors where each element is a pair of counters: time stamp counter for accumulating packet time stamps and packet counter for counting the number of arriving/departing packets. Each vector has a sampling probability and a sampling function. For each arriving or departing packet, LDA first maps the packet to a counter vector such that in the long run, the fraction of packets that LDA maps to any counter vector is equal to its sampling probability. It then randomly maps the packet to a counter pair in this counter vector and adds the time stamp of the packet to the time stamp counter and increments the packet counter by one. To obtain the aggregate latency estimate between the sender and receiver, for each counter pair in each vector, LDA checks whether they have the same packet counter value and selects all counter pairs that have the same packet counter value for both the sender and receiver. Then, LDA can easily calculate the total number of successfully delivered packets and the sum of their time stamps at both sides. Finally, to obtain the aggregate average latency between the sender and receiver, it subtracts the sum of time stamps at the sender side from that at the receiver side and divides it with the total number of successfully delivered packets.

5.3 COLATE – Recording Phase

In this section, we present the recording phase and the statistical modeling of COLATE.

5.3.1 Noisy Accumulation of Time Stamps

At each observation point X, COLATE maintains a vector $\mathbf{C}_X$ of n counters where each counter $\mathbf{C}_X[i]$ $(1 \leq i \leq n)$ has b bits with initial value 0. When a packet arrives at or departs from an observation point X, COLATE extracts its flow ID f, chooses a random number j from a uniform distribution in the range $[1, m]$ where $m << n$, calculates the hash function $H(f, j)$ whose output is uniformly distributed in range $[1, n]$, and adds the time stamp of this packet (*i.e.*, the current time at observation point X) to the counter $\mathbf{C}_X[H(f, j)]$. Thus, the time stamp of all packets in flow f will be uniformly distributed to m counters: $\mathbf{C}_X[H(f, 1)], \mathbf{C}_X[H(f, 2)], \cdots, \mathbf{C}_X[H(f, m)]$. These m counters constitute the counter subvector of flow f, which is denoted by $\mathbf{S}_X^f$ where $\mathbf{S}_X^f[j] = \mathbf{C}_X[H(f, j)]$ for $j \in [1, m]$. In this recording phase, for each packet, COLATE performs one memory update to update the value of $\mathbf{C}_X[H(f, j)]$. For different flows, the probability that COLATE maps them to the same counter subvector is $\frac{m!}{n^m}$ (=2.4×10^{-62} for $n = 10000$ and $m = 20$, for example), which is practically zero. Note that an observation point is a port of a middlebox, not a middlebox itself, because the arriving and departing time of a packet at a middlebox are different due to the non-negligible packet processing time within the middlebox.

5.3.2 Analysis of Noisy Accumulation

First, by Lemma 14, we show that in any counter epoch, on average, a flow contributes the same amount to each counter in its counter subvector. We further show that the amount contributed by a flow to each counter in its counter subvector can be modeled by a binomial distribution. Second, we derive expressions for the expected value and variance of a counter in counter vector in Theorem 15. In Section 5.4, we use the expression of expected value to estimate the average and standard deviation of packet latencies for any given flow. In Section 5.5, we use the expression of variance to determine the parameter values that can ensure that the actual reliability achieved by COLATE is no less than the required reliability.

Lemma 14. *Let C_f be the random variable representing the sum of time stamps contributed by flow f to a counter $\mathbf{S}_X^f[j]$, $1 \leq j \leq m$, in its counter subvector of length m at observation point X. Let p_f be the number of packets in flow f that contributed time stamps to the current counter epoch $\mathbf{C}_X$. Let $\mathcal{T}_f$ be an independent random variable representing the time* *stamp contributed by each packet of flow f to $\mathbf{C}_X$. Then, we have* $E[C_f] = \frac{p_f \times E[\mathcal{T}_f]}{m}$.

Proof. Let $P_{f,j}$ be a random variable representing the number of packets in flow f that contributed time stamps to counter $\mathbf{S}_X^f[j]$. Therefore, $E[C_f] = E\big[\sum_{P_{f,j}} \mathcal{T}_f\big]$. Applying Wald's Lemma, we get $E[C_f] = E[P_{f,j}] \times E[\mathcal{T}_f]$. As the output of hash function H is uniformly distributed in range $[1, n]$, its output is also uniformly distributed in $[1, m]$ for the packets in flow f. Thus, the probability that COLATE adds the time stamp of a packet of the flow f to counter $\mathbf{S}_X^f[j]$ is $1/m$. The random variable $P_{f,j}$ follows the binomial distribution i.e., $P_{f,j} \sim \text{Binom}(p_f, 1/m)$. Therefore, $E[C_f] = \frac{p_f}{m} \times E[\mathcal{T}_f]$. $\square$

Figure 5.2 plots the CDF of the ratio of the observed values of C_f from simulations of our ICSI traffic trace to the $E[C_f]$ from Lemma 14. We observe from this figure that there is a steep rise in the CDF when the value of this ratio is around 1. We also observe from the simulations that the mean and median of the ratio are both equal to 1. This empirically establishes the result in Lemma 14.

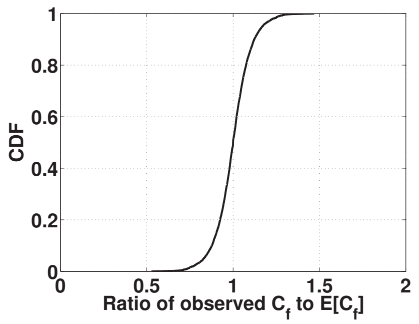


Figure 5.2 CDF of *observed* $\frac{C_f}{E[C_f]}$

Lemma 14 shows that the sum of time stamps of all packets of flow f is equally divided among all counters in its counter subvector, which conforms to binomial distribution. Thus, we approximate the distribution of C_f with a binomial distribution as $C_f \sim \text{Binom}(t_{f,X}, 1/m)$, where $t_{f,X}$ represents sum of all times-stamps contributed by flow f to the counters in its counter subvector at observation point X. Let C_r be the random variable representing the sum of time stamps contributed by packets of all flows other than

f to counter $\mathbf{S}_X^f[j]$. Using similar reasoning as for C_f, we approximate the distribution of C_r with a binomial distribution. The probability that a packet of a flow $\bar{f} \neq f$ contributes a time stamp to counter $\mathbf{S}_X^f[j]$ is the product of the probability that H maps the packet to $\mathbf{S}_X^f[j]$ given that $\mathbf{S}_X^f[j] \in \mathbf{S}_X^{\bar{f}}$, which is $1/m$, and the probability that counter $\mathbf{S}_X^f[j]$ is in the counter subvector of $\bar{f}$, which is denoted by $P\{\mathbf{S}_X^f[j] \in \mathbf{S}_X^{\bar{f}}\}$ and calculated as

$$\begin{aligned} P\{\mathbf{S}_X^f[j] \in \mathbf{S}_X^{\bar{f}}\} &= 1 - \left\{ \binom{m}{0} \left(\frac{1}{n}\right)^0 \left(1 - \frac{1}{n}\right)^m \right\} \\ &= 1 - \left\{ 1 - \frac{m}{n} + \frac{(m)(m-1)}{n^2 \times 2!} - \dots \right\} \approx \frac{m}{n} \qquad \because m << n \end{aligned} \tag{5.1}$$

Thus, the probability that a packet of a flow $\bar{f} \neq f$ contributes a time stamp to counter $\mathbf{S}_X^f[j]$ is $\frac{1}{m} \times \frac{m}{n} = \frac{1}{n}$. Thus, $C_r \sim \text{Binom}(t_X - t_{f,X}, 1/n)$.

Theorem 15. *Let C be the random variable representing the value of a counter $\boldsymbol{S}_X^f[j]$, $1 \leq j \leq m$, in the counter subvector of a flow f. Let $t_{f,X}$ be the sum of all time stamps contributed by packets of flow f to all counters in its counter subvector and t_X be the sum of all time stamps contributed by packets of all flows in the counter epoch at observation point X. Let $C_f \sim Binom(t_{f,X}, 1/m)$ represent the sum of time stamps contributed by packets of flow f to $\mathbf{S}_X^f[j]$ and let $C_r \sim Binom(t_X - t_{f,X}, 1/n)$ represent the sum of time stamps contributed by packets of all flows other than f to counter $\boldsymbol{S}_X^f[j]$. Let C_f and C_r be independent of each other. The expected value and variance of C are calculated as follows*

$$E[C] = \frac{t_{f,X}}{m} + \frac{t_X - t_{f,X}}{n} \tag{5.2}$$

$$\text{Var}(C) = \left(\frac{t_{f,X}}{m}\right)\left(1 - \frac{1}{m}\right) + \left(\frac{t_X - t_{f,X}}{n}\right)\left(1 - \frac{1}{n}\right) \tag{5.3}$$

Proof. As $C = C_f + C_r$, we get, $E[C] = E[C_f] + E[C_r] = \frac{t_{f,X}}{m} + \frac{t_X - t_{f,X}}{n}$. As C_f and C_r are assumed to be independent, $\text{Var}(C) = \text{Var}(C_f) + \text{Var}(C_r)$. As variance of $Binom(v, w)$ is $vw(1-w)$, we get $\text{Var}(C_f) = \left(\frac{t_{f,X}}{m}\right)\left(1 - \frac{1}{m}\right)$ and $\text{Var}(C_r) = \left(\frac{t_X - t_{f,X}}{n}\right)\left(1 - \frac{1}{n}\right)$. □ □

In Theorem 15, we assumed C_f and C_r to be independent of each other, which is true whenever $t_{f,X} << t_X$. In practice $t_{f,X}$ indeed is much smaller than t_X because $t_{f,X}$ is the sum of the time stamps added by a single flow in the counter epoch while t_X is the sum of the time stamps added by all flows (in the order of tens and hundreds of thousands). Furthermore, in Theorem 15, we approximate the distributions of C_f and C_r with binomial distributions because when there are a large number of packets that contribute time stamps to the counters in the counter vector, we can approximate each time stamp to be smeared over the counters. This approximation makes the formal development of variance of C and its subsequent use in calculating parameters for COLATE tractable. If the exact equation for variance of C is desired, it can be obtained as follows. Consider any packet with time stamp t_p not belonging to a flow f. This packet has a probability $\frac{1}{n}$ of being mapped to a counter in the counter subvector of the flow f. Thus, the time stamp for each such packet will contribute a variance of $t_p^2(\frac{1}{n})(1-\frac{1}{n})$ to the overall variance of C. In Theorem 15, $t_{f,X}/m$ models the average sum of the time stamps contributed by flow f, and $(t_X - t_{f,X})/m$ models the average *noise* contributed by all other flows to counter $\mathbf{S}_X^f[j]$.

5.4 COLATE – Querying Phase

In this section, we present the methods that COLATE uses to estimate the average and standard deviation of the latencies of the packets of a flow in passing any two points.

5.4.1 Estimating Latency Average

For a flow f passing through observation point S and then observation point R, we want to calculate $\tilde{\mu}_f$, the estimate of average latency μ_f of flow f. For a packet z in flow f, let $u_{f,S}[z]$ be its time stamp at observation point S and $u_{f,R}[z]$ be its time stamp at observation point R. The delay experienced by this packet in traveling from S to R is thus $u_{f,R}[z] - u_{f,S}[z]$. Let $t_{f,S}$ and $t_{f,R}$ be the sums of all time stamps of the packets in flow f at S and R, respectively. Recall that p_f denotes the number of packets in f. Thus,

$$\begin{aligned}\mu_f &= \frac{1}{p_f}\sum_{\forall z}\big(u_{f,R}[z] - u_{f,S}[z]\big)\\ &= \frac{1}{p_f}\Big(\sum_{\forall z} u_{f,R}[z] - \sum_{\forall z} u_{f,S}[z]\Big) = \frac{1}{p_f}(t_{f,R} - t_{f,S})\end{aligned}$$

Note that the value of p_f can be measured by tools such as NetFlow available on Cisco routers or by schemes proposed in [74, 64]. Thus, to estimate the value of μ_f, we need to obtain the estimated values $\tilde{t}_{f,S}$ and $\tilde{t}_{f,R}$ for $t_{f,S}$ and $t_{f,R}$, respectively. Then, we can calculate

$$\tilde{\mu_f} = \frac{1}{p_f}(\tilde{t}_{f,R} - \tilde{t}_{f,S}) \tag{5.4}$$

Theorem 16 shows how to obtain $\tilde{t}_{f,X}$ at X.

Theorem 16. *Given a counter epoch $\mathbf{C}_X$ of length n at observation point X where each counter subvector is of length m, let t_X denote the sum of all counters in $\mathbf{C}_X$, the estimate $\tilde{t}_{f,X}$ of the sum of all time stamps of the packets in flow f is calculated as follows*

$$\tilde{t}_{f,X} = \frac{1}{n-m}\Big\{n\sum_{j=1}^{m}\mathbf{S}_X^f[j] - mt_X\Big\} \tag{5.5}$$

Proof. Given $\mathbf{C}_X$ and flow f, we can easily obtain the values of every counter in the counter subvector $\mathbf{S}_X^f$ of f. Thus, we can calculate $E[C]$ as $E[C] = \frac{1}{m}\sum_{j=1}^{m}\mathbf{S}_X^f[j]$. Substituting $E[C]$ in Equation (5.2) by $\frac{1}{m}\sum_{j=1}^{m}\mathbf{S}_X^f[j]$, replacing $t_{f,X}$ by $\tilde{t}_{f,X}$, and solving for $\tilde{t}_{f,X}$, gives Equation (5.5). □

5.4.2 Estimating Latency Standard Deviation

Let D_f be the random variable representing the latency experienced by a packet in flow f. The standard deviation of D_f can be calculated by $\tilde{\sigma}_f = \sqrt{\text{Var}(D_f)}$, where $\text{Var}(D_f)$ can be calculated as follows:

$$\begin{aligned}\mathrm{Var}(D_f) &= E[D_f^2] - E^2[D_f]\\ &= \left\{\frac{1}{p_f}\sum_{\forall z}\big(u_{f,R}[z]-u_{f,S}[z]\big)^2\right\}-\left\{\frac{1}{p_f}\sum_{\forall z}\big(u_{f,R}[z]-u_{f,S}[z]\big)\right\}^2\\ &= \left\{\frac{1}{p_f}\sum_{\forall z}\big(u_{f,R}[z]-u_{f,S}[z]\big)^2\right\}-\left\{\frac{1}{p_f}\big(t_{f,R}-t_{f,S}\big)\right\}^2 \qquad (5.6)\end{aligned}$$

We can calculate the second term $\{\frac{1}{p_f}(t_{f,R}-t_{f,S})\}^2$ in Equation (5.6) based on Theorem 16. Now the key challenge is to calculate the first term $\{\frac{1}{p_f}\sum_{\forall z}(u_{f,R}[z]-u_{f,S}[z])^2\}$ in Equation (5.6). Our solution to this challenge is based on the statistical technique proposed by Alon *et al.* in [25]. The main idea is to introduce a random variable G_z where the value g_z that this random variable takes on is either $+1$ or -1 with equal probability. Before adding the time stamp of a packet z to a counter, if we randomly multiply the time stamp with g_z and then add it to the counter, then we will get Equation (5.7).

$$E\left[\left\{\sum_{\forall z}\big(g_z u_{f,R}[z]-g_z u_{f,S}[z]\big)\right\}^2\right]=\sum_{\forall z}\big(u_{f,R}[z]-u_{f,S}[z]\big)^2 \qquad (5.7)$$

This can be proven as follows:

$$\begin{aligned}E\left[\left\{\sum_{\forall z} g_z\big(u_{f,R}[z]-u_{f,S}[z]\big)\right\}^2\right]=E\Bigg[&\sum_{\forall z} g_z^2\big(u_{f,R}[z]-u_{f,S}[z]\big)^2\\ &+\sum_{\forall z\neq\overline{z}} g_z g_{\overline{z}}\big(u_{f,R}[z]-u_{f,S}[z]\big)\big(u_{f,R}[\overline{z}]-u_{f,S}[\overline{z}]\big)\Bigg]\end{aligned}$$

Using the well known result that expectation of sum of random variables is the sum of their individual expectations and that $g_z^2=1$, we get:

$$=\sum_{\forall z}\big(u_{f,R}[z]-u_{f,S}[z]\big)^2+\sum_{\forall z\neq\overline{z}}\big(u_{f,R}[z]-u_{f,S}[z]\big)\big(u_{f,R}[z]-u_{f,S}[z]\big)\times E[G_zG_{\overline{z}}]$$

Note that $\{G_1, G_2, G_3, \dots\}$ is a set of independent and identically distributed random variables. So, $E[G_zG_{\overline{z}}] = E[G_z]\times E[G_{\overline{z}}]$. As $E[G_z]=0$ for all values of z, this implies

$E[G_z G_{\overline{z}}] = 0$. Thus, the second term in the equation above is 0, which proves Equation (5.7).

We now present our method for calculating the first term in Equation (5.6). First, for each counter in the counter subvector of f, whose value is contributed by the time stamps of the packets in flow f and those of the packets in other flows, we use Theorem 17 to extract an estimate of the value that is contributed by only the time stamps of the packets in f. In other words, we eliminate the noise introduced by the packets of flows other than f from the counter subvector of f. Second, we statistically simulate the process of multiplying the time stamp of a packet in f with the random variable G_z and then adding the multiplication result to the corresponding counter in the counter subvector of f. By repeating this process a statistically sufficient number of times, we obtain an accurate estimate of $\sum_{\forall z}(u_{f,R}[z] - u_{f,S}[z])^2$ based on Theorem 18. Note that this simulation does not require any changes to the recording phase. Next, we present our denoising solution and statistical simulation process.

5.4.2.1 Denoising Counter Subvectors

Our denoising solution is based on Theorem 17. The numerical solution of Equation (5.8) gives us the estimate of the value that is contributed by only the time stamps of the packets in f for each counter in f's subvector.

Theorem 17. *Let $w_{f,X}[j]$ $(1 \leq j \leq m)$ be the sum of the time stamps of flow f's packets that are mapped to counter $\mathbf{S}_X^f[j]$ at observation point X. Let $\tilde{t}_{f,X}$ be the estimate of sum of all time stamps contributed by packets of flow f to all counters in f's counter subvector and t_X be the sum of all time stamps contributed by packets of all flows in the counter epoch at observation point X. The maximum likelihood estimate $\tilde{w}_{f,X}[j]$ of $w_{f,X}[j]$ satisfies the following equation:*

$$rCl \ln\{n-1\} = \psi^{(0)}\{t_X - \tilde{t}_{f,X} - \mathbf{S}_X^f[j] + \tilde{w}_{f,X}[j] + 1\} - \psi^{(0)}\{\mathbf{S}_X^f[j] - \tilde{w}_{f,X}[j] + 1\} \quad (5.8)$$

where $\psi^{(0)}\{.\}$ is the 0th order polygamma function.

Proof. The maximum likelihood estimate of $w_{f,X}[j]$ is the value of $w_{f,X}[j]$ that maximizes the probability that the counter $\mathbf{S}_X^f[j]$ takes the observed value.

$$\arg\max_{w_{f,X}[j]} P\{C = \mathbf{S}_X^f[j] | w_{f,X}[j]\}$$

This value of $w_{f,X}[j]$ can be obtained by differentiating $P\{C = \mathbf{S}_X^f[j]|w_{f,X}[j]\}$ w.r.t $w_{f,X}[j]$ and equating to 0.

$$\frac{d}{d(w_{f,X}[j])} P\{C = \mathbf{S}_X^f[j] | w_{f,X}[j]\} = 0$$

As $C = C_f + C_r$ and $C_f = w_{f,X}[j]$, the L.H.S. becomes

$$\frac{d}{d(w_{f,X}[j])} P\{C_r = \mathbf{S}_X^f[j] - w_{f,X}[j]\}$$

For simplicity, let $\xi = \mathbf{S}_X^f[j] - w_{f,X}[j]$ and $\tau = t_X - t_{f,X}$. As C_r is a binomial random variable, this derivative further becomes

$$\begin{aligned}\frac{d}{d(w_{f,X}[j])}\binom{\tau}{\xi}\left(\frac{1}{n}\right)^{\xi}\left(1-\frac{1}{n}\right)^{\tau-\xi} &= \left\{\binom{\tau}{\xi}\left(\frac{1}{n}\right)^{\xi}\left(1-\frac{1}{n}\right)^{\tau-\xi}\right\} \\ &\times\left\{\psi^{(0)}\{\xi+1\} - \psi^{(0)}\{\tau-\xi+1\} + \ln\left\{1-\frac{1}{n}\right\} - \ln\left\{\frac{1}{n}\right\}\right\} \qquad (5.9)\end{aligned}$$

Due to space limitations, we have skipped the intermediate derivation steps, which use the following identity:

$$\frac{d}{dw}\binom{v}{w} = \binom{v}{w}\left(\psi^{(0)}\{v-w+1\} - \psi^{(0)}\{w+1\}\right)$$

By replacing $t_{f,X}$ with $\tilde{t}_{f,X}$, which is calculated using Theorem 16, in $\tau = t_X - t_{f,X}$ and further in the R.H.S of Equation (5.9) and equating it to zero, we obtain the maximum likelihood estimate $\tilde{w}_{f,X}[j]$ of $w_{f,X}[j]$. As $\binom{\tau}{\xi}\left(\frac{1}{n}\right)^{\xi}\left(1-\frac{1}{n}\right)^{\tau-\xi}$ is $P\{C_f = \mathbf{S}_X^f[j]|w_{f,X}[j]\}$, which is not equal to zero, we have

$$\left\{\psi^{(0)}\{\xi+1\} - \psi^{(0)}\{\tau-\xi+1\} + \ln\left\{1-\frac{1}{n}\right\} - \ln\left\{\frac{1}{n}\right\}\right\} = 0$$

Simplifying the ln{.} terms results in Equation (5.8). □

5.4.2.2 Statistical Simulations

We have obtained the m values extracted using Theorem 17 from f's counter subvector at observation point X. For flow f that passes observation point X, each unique permutation of the m distinct integers from 1 to m, denoted by vector Q, defines a unique *deviation vector* $v_{f,X}$ of size $m/2$ as follows. To ensure that $m/2$ is an integer, we choose m to be an even number.

$$v_{f,X}[l] = \tilde{w}_{f,X}\big[Q[2l]\big] - \tilde{w}_{f,X}\big[Q[2l-1]\big] \quad 1 \le l \le m/2 \tag{5.10}$$

Each unique permutation of the m distinct integers from 1 to m is essentially a unique simulation of the aforementioned statistical process of multiplying half the time stamps with $G_z = +1$ and the other half with $G_z = -1$. Theorem 18 gives us the way to estimate $\sum_{\forall z}(u_{f,R}[z] - u_{f,S}[z])^2$, which is needed for calculating $\text{Var}(D_f)$ based on Equation (5.6).

Theorem 18. *Given any two observation points S and R, for any permutation Q of the m distinct integers from 1 to m, let $v_{f,S}$ and $v_{f,R}$ be the corresponding deviation vectors of flow f at observation points S and R, respectively. The following equation holds:*

$$\sum_{\forall z}\big(u_{f,R}[z] - u_{f,S}[z]\big)^2 = E\Big[\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2\Big] \tag{5.11}$$

Proof. Let $\mathcal{Y}_{lS}$ be the set of all the time stamps contributed by the packets of flow f to counters $\boldsymbol{S}_S^f[Q[2l]]$ and $\boldsymbol{S}_S^f[Q[2l-1]]$. Similarly, let $\mathcal{Y}_{lR}$ be the set of all the time stamps contributed by the packets of flow f to counters $\boldsymbol{S}_R^f[Q[2l]]$ and $\boldsymbol{S}_R^f[Q[2l-1]]$. Let $y_{lS}[i]$ be the i-th element of $\mathcal{Y}_{lS}$, where $1 \le i \le |\mathcal{Y}_{lS}|$. Similarly, let $y_{lR}[i]$ be the i-th element of $\mathcal{Y}_{lR}$, where $1 \le i \le |\mathcal{Y}_{lR}|$. Starting from the R.H.S of Equation (5.11), we have:

$$
\begin{aligned}
&= \sum_{l=1}^{\frac{m}{2}} E\Big[(v_{f,R}[l] - v_{f,S}[l])^2\Big] = \sum_{l=1}^{\frac{m}{2}} E\Bigg[\Big\{\sum_{i=1}^{|\mathcal{Y}_{lR}|} g_i{\cdot}y_{lR}[i] - \sum_{i=1}^{|\mathcal{Y}_{lS}|} g_i{\cdot}y_{lS}[i]\Big\}^2\Bigg] \\
&= \sum_{l=1}^{\frac{m}{2}} E\Bigg[\Big\{\sum_{i=1}^{|\mathcal{Y}_{lR}|} (g_i{\cdot}y_{lR}[i] - g_i{\cdot}y_{lS}[i])\Big\}^2\Bigg] \because |\mathcal{Y}_{lR}| = |\mathcal{Y}_{lS}| \\
&= \sum_{l=1}^{\frac{m}{2}} \sum_{i=1}^{|\mathcal{Y}_{lR}|} \big(y_{lR}[i] - y_{lS}[i]\big)^2, \text{ using Equation (5.7)} \\
&= \sum_{\forall z} \big(u_{f,R}[z] - u_{f,S}[z]\big)^2
\end{aligned}
$$

The last equality follows from the fact that each $y_{lX}[i]$ is actually the value of some time stamp $u_{f,X}[z]$ at observation point X. □

For any permutation Q of the m distinct integers from 1 to m, we calculate $v_{f,R}[l]$ and $v_{f,S}[l]$ for each $1 \le l \le m/2$ based on Equation (5.10), and then calculate $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$, which is one estimate of $\sum_{\forall z}\big(u_{f,R}[z] - u_{f,S}[z]\big)^2$ according to Theorem 18. As $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$ is a random variable, its variance can be reduced by γ times if we repeat the above process γ times using a different random permutation Q each time and use the average of the γ values of $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$ as the estimate of $\sum_{\forall z}\big(u_{f,R}[z] - u_{f,S}[z]\big)^2$. As the R.H.S. of Equation (5.11) is an expected value, to get an accurate estimate of $\sum_{\forall z}\big(u_{f,R}[z] - u_{f,S}[z]\big)^2$, we need to calculate $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$ for a statistically sufficient number of unique permutations of the m distinct integers from 1 to m. The process of calculating $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$ for different permutations of Q is essentially simulating the aforementioned *random* statistical process of multiplying the time stamp of each packet with random variable G_z that takes the value of $+1$ and -1 with equal probability *without having to perform this process in the recording phase.* We name this process of calculating $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$ using different permutations of Q as *virtual repetitions.* The number of distinct ways in which we can repeat this process is $\prod_{i=1}^{\frac{m}{2}} \binom{m-2(i-1)}{2}$, which is large enough for us to obtain any required reliability α for estimating the standard deviation. For example, when $m = 20$, $\prod_{i=1}^{\frac{m}{2}} \binom{m-2(i-1)}{2} = 2.38 \times 10^{15}$.

5.4.2.3 Steps of Estimating Standard Deviation

To summarize, COLATE performs the following six steps to estimate the standard deviation of the latencies that the packets in flow f experienced in traversing from observation points S to R. (1) Obtain the number of packets in flow f, denoted by p_f, using NetFlow or the schemes proposed in [74, 64]. (2) Obtain the estimates of $t_{f,S}$ and $t_{f,R}$, which are the sum of the time stamps of all packets in flow f at observation points S and R respectively, using Theorem 16. (3) Extract the values of $w_{f,S}[j]$ and $w_{f,R}[j]$, which are the sum of the time stamps of flow f's packets that are mapped to counter $\boldsymbol{S}_S^f[j]$ at observation point S and to counter $\boldsymbol{S}_R^f[j]$ at observation point R, respectively, for all $1 \leq j \leq m$, using Theorem 17. (4) Randomly choose γ permutations of the m distinct integers from 1 to m. For each permutation Q, first calculate $v_{f,R}[l]$ and $v_{f,S}[l]$ for all $1 \leq l \leq m/2$ using Equation (5.10) and then calculate $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$. (5) Calculate the average of the γ values of $\sum_{l=1}^{\frac{m}{2}}(v_{f,R}[l] - v_{f,S}[l])^2$, which is the estimated value of $\sum_{\forall z}\left(u_{f,R}[z] - u_{f,S}[z]\right)^2$ (6) Estimate of the standard deviation using Equation (5.6).

5.5 COLATE – Reliability

COLATE has four parameters: (1) the total number of counters denoted by n, (2) the number of counters in each counter subvector denoted by m, (3) the number of bits in each counter denoted by b, and (4) the vector threshold denoted by T. Note that when the sum of all n counters in a counter vector reaches T, COLATE dumps the counter vector into permanent storage as a counter epoch and then resets all counter values to be zero. In this section, we present solutions to find the values for these parameters so that our estimated average latency achieves the required reliability $\alpha \in [0, 1)$ for the given confidence interval $\beta \in (0, 1]$. Note that for standard deviation, we have already presented a method in Section 5.4 that can achieve arbitrarily high required reliability. Recall $\tilde{\mu}_f = \frac{1}{p_f}(\tilde{t}_{f,R} - \tilde{t}_{f,S})$ (in Equation (5.4)), which shows that the estimate $\tilde{\mu}_f$ depends on two other estimates $\tilde{t}_{f,S}$ and

$\tilde{t}_{f,R}$. Next, we find the confidence interval B and required reliability A that the estimate $\tilde{t}_{f,X}$ for $t_{f,X}$ at each observation point X must satisfy so that the estimate $\tilde{\mu}_f$ for μ_f satisfies the confidence interval β and required reliability α. That is, we want to find the values of B and A so that if for every observation point X we have $P\left\{|\tilde{t}_{f,X} - t_{f,X}| \le Bt_{f,X}\right\} \ge A$, then we will have $P\left\{|\tilde{\mu}_f - \mu_f| \le \beta\mu_f\right\} \ge \alpha$. After we find the values for B and A, we present a solution to calculate the optimal values of the four parameters n, m, b, and T.

5.5.1 Individual Reliability Requirements

Individual Required Reliability: The maximum fraction of estimated values $\tilde{\mu}_f$ that can violate the requirement of $|\tilde{\mu}_f - \mu_f| \le \beta\mu_f$, while the overall estimate still satisfies the required reliability α, is $1 - \alpha$. Thus, the maximum fraction of estimates $\tilde{t}_{f,X}$ at either observation points of S and R that can violate the requirement $|\tilde{t}_{f,X} - t_{f,X}| \le Bt_{f,X}$ must be no greater than $(1-\alpha)/2$. Thus,

$$A = 1 - (1-\alpha)/2 = (1+\alpha)/2 \tag{5.12}$$

Individual Confidence Interval: The estimate $\tilde{\mu}_f$ obtained by COLATE needs to satisfy the requirement of $|\tilde{\mu}_f - \mu_f| \le \beta\mu_f$ with probability of at least α. As $\tilde{\mu}_f = \frac{1}{p_f}(\tilde{t}_{f,R} - \tilde{t}_{f,S})$ and $\mu_f = \frac{1}{p_f}(t_{f,R} - t_{f,S})$, the confidence interval requirement $|\tilde{\mu}_f - \mu_f| \le \beta\mu_f$ becomes:

$$\left|(\tilde{t}_{f,R} - t_{f,R}) - (\tilde{t}_{f,S} - t_{f,S})\right| = \left|(\tilde{t}_{f,R} - \tilde{t}_{f,S}) - (t_{f,R} - t_{f,S})\right| \le \beta(t_{f,R} - t_{f,S})$$

The largest value of $\left|(\tilde{t}_{f,R} - t_{f,R}) - (\tilde{t}_{f,S} - t_{f,S})\right|$ is $Bt_{f,R} + Bt_{f,S}$, which must be no greater than $\beta(t_{f,R} - t_{f,S})$.

$$Bt_{f,R} + Bt_{f,S} \le \beta(t_{f,R} - t_{f,S})$$

Thus, we get

$$B \le \beta\left(\frac{t_{f,R} - t_{f,S}}{t_{f,R} + t_{f,S}}\right) \tag{5.13}$$

To determine B for a given network, we conduct measurement of $(t_{f,R} - t_{f,S})/(t_{f,R} + t_{f,S})$ on the network to find the appropriate value so that Equation (5.13) statistically holds.

5.5.2 Reliability Centered Parameter Selection

As we have four unknown parameters (*i.e.*, n, m, b, and T), we need at least four equations so that we can calculate the values of these parameters by solving the four equations. Next we develop these four equations.

Equation 1: Let M be the total number of bits of the RAM that an observation point can allocate for storing the counter vector, which requires $n \times b$ bits. Thus,

$$n \times b = M \tag{5.14}$$

Equation 2: Based on Lemma 14, the sum of all the time stamps of all packets of all flows is equally divided among all n counters on average. Thus, the value of each counter on average can go up to T/n. Thus, the number of bits in each counter, which is b, needs to satisfy the following equation.

$$b = \log_2 \left\{ \frac{T}{n} \right\} + 1 \tag{5.15}$$

Note that we add 1 in the R.H.S of this equation to double the capacity of each counter to avoid overflows.

Equation 3: As the expected value of a counter in a counter subvector, which is specified in Equation (5.2), should never exceed the maximum capacity of the counter, we have

$$\frac{t_{f,X}}{m} + \frac{T - t_{f,X}}{n} \leq 2^b - 1$$

Let $t_{f,X}^{max}$ be the maximum value of $t_{f,X}$ for all flows on a network. Thus, the value of b should satisfy the following equation:

$$\frac{t_{f,X}^{max}}{m} + \frac{T - t_{f,X}^{max}}{n} = 2^b - 1 \tag{5.16}$$

Here $t_{f,X}^{max}$ can be obtained by some measurement on the sum of the time stamps of all packets on a per-flow basis on the given network.

Equation 4: To achieve the required reliability, $P\left\{|\tilde{t}_{f,X} - t_{f,X}| \leq Bt_{f,X}\right\}$ should at least be equal to its lower bound A, *i.e.*,

$$P\left\{(1-B)t_{f,X} \leq \tilde{t}_{f,X} \leq (1+B)t_{f,X}\right\} = A \tag{5.17}$$

Based on Equation (5.2), we can represent $E[C]$ as a function of $t_{f,X}$; denoting this function by g, we have $E[C] = g\{t_{f,X}\}$. Thus, $t_{f,X} = g^{-1}\{E[C]\}$. Let $\tilde{C}$ be the observed value of $E[C]$. Then, we have $\tilde{t}_{f,X} = g^{-1}\{\tilde{C}\}$. Equation (5.17) becomes

$$\begin{aligned} A &= P\left\{(1-B)t_{f,X} \le g^{-1}\{\tilde{C}\} \le (1+B)t_{f,X}\right\} \\ &= P\left\{g\left\{(1-B)t_{f,X}\right\} \le \tilde{C} \le g\left\{(1+B)t_{f,X}\right\}\right\} \end{aligned} \tag{5.18}$$

Similarly, based on Equation (5.3), we can represent standard deviation of C as a function of $t_{f,X}$; denoting this function by h, we have $\mathrm{Var}(C) = h^2\{t_{f,X}\}$. Based on the fact that the variance of a random variable reduces by m times if the random event is repeated m times, by observing the values of C from m counters, the variance of C becomes $\frac{h^2\left\{t_{f,X}\right\}}{m}$ and the standard deviation of C becomes $\frac{h\left\{t_{f,X}\right\}}{\sqrt{m}}$. Let Z denote $\frac{\tilde{C}-g\left\{t_{f,X}\right\}}{h\left\{t_{f,X}\right\}/\sqrt{m}}$. Thus, Equation (5.18) becomes

$$P\left\{\frac{g\left\{(1-B)t_{f,X}\right\}-g\left\{t_{f,X}\right\}}{h\left\{t_{f,X}\right\}/\sqrt{m}} \le Z \le \frac{g\left\{(1+B)t_{f,X}\right\}-g\left\{t_{f,X}\right\}}{h\left\{t_{f,X}\right\}/\sqrt{m}}\right\} = A \tag{5.19}$$

By the central limit theorem, Z approximates a standard normal random variable. The area under the standard normal curve gives the success probability, which is the required reliability in our context. As our confidence interval requirement is symmetric on both the upper and lower sides of $t_{f,X}$, we can represent the required reliability A in terms of a constant k as follows:

$$P\{-k \le Z \le k\} = A \tag{5.20}$$

Let Φ be the cumulative distribution function (CDF) of a standard normal distribution and erf$\{.\}$ be the standard error function, we get

$$P\{-k \le Z \le k\} = \Phi(k) - \Phi(-k) = \mathrm{erf}\left\{\frac{k}{\sqrt{2}}\right\} \tag{5.21}$$

From Equations (5.20) and (5.21), we get

$$k = \sqrt{2}\,\mathrm{erf}^{-1}\{A\}$$

We observe that the absolute values of the upper and lower bounds of Z in Equation (5.19) are the same. Thus, equating the lower bound with $-k$ or the upper bound with k results in the following equation:

$$B^2 t_{f,X}^2 = \frac{k^2 n^2 m}{n-m} \Big\{ \Big(\frac{t_{f,X}}{m}\Big)\Big(1-\frac{1}{m}\Big) + \Big(\frac{T-t_{f,X}}{n}\Big)\Big(1-\frac{1}{n}\Big) \Big\} \tag{5.22}$$

By rearranging Equation (5.22), we get.

$$\begin{aligned} B^2 = & \frac{1}{t_{f,X}} \Big[\frac{k^2 n^2 m}{n-m} \Big\{ \Big(\frac{1}{m}\Big)\Big(1-\frac{1}{m}\Big) - \Big(\frac{1}{n}\Big)\Big(1-\frac{1}{n}\Big) \Big\} \Big] \\ & + \frac{1}{t_{f,X}^2} \Big[\frac{k^2 n^2 m}{n-m} \Big(\frac{1}{n}\Big)\Big(1-\frac{1}{n}\Big) T \Big] \end{aligned}$$

This equation shows that B is inversely proportional to $t_{f,X}$ when other parameters are fixed. This makes intuitive sense because the more packets in flow f in passing observation point X (*i.e.*, the larger $t_{f,X}$ is), the more timing information we can obtain from the packets, and the smaller confidence interval can be achieved. Thus, we should use the statistically minimum observable value of $t_{f,X}$, denoted $t_{f,X}^{min}$, for the given network, in Equation (5.22). Here $t_{f,X}^{min}$ can be obtained by some measurement on the sum of the time stamps of all packets on a per-flow basis on the given network. The parameter values obtained using $t_{f,X}^{min}$ in Equation (5.22) will ensure that the estimates for all flows whose sum of time stamps are $\geq t_{f,X}^{min}$ satisfy $P\left\{|\tilde{t}_{f,X} - t_{f,X}| \leq B t_{f,X}\right\} \geq A$. By replacing $t_{f,X}$ by $t_{f,X}^{min}$ in Equation (5.22), we get

$$B^2 {t_{f,X}^{min}}^2 = \frac{k^2 n^2 m}{n-m} \Big\{ \Big(\frac{t_{f,X}^{min}}{m}\Big)\Big(1-\frac{1}{m}\Big) + \Big(\frac{T-t_{f,X}^{min}}{n}\Big)\Big(1-\frac{1}{n}\Big) \Big\} \tag{5.23}$$

Solving Equations: COLATE takes M, α, β, $t_{f,X}^{min}$, and $t_{f,X}^{max}$ as input. The values of required reliability α and confidence interval β are provided by network operators. The value of RAM space M depends on the amount of RAM available at an observation point. For $t_{f,X}^{min}$ and $t_{f,X}^{max}$, network operators can obtain them by measurements on targeted flows in the given network. With the values of M, α, β, $t_{f,X}^{min}$, and $t_{f,X}^{max}$, COLATE simultaneously solves the four equations (*i.e.*, (5.14), (5.15), (5.16), and (5.23)) to obtain the appropriate values of the four parameters n, m, b, and T. To simultaneously solve these equations, we express m, b, and T in terms of n using Equations (5.14), (5.15), and (5.16) and replace them in Equation (5.23). This results in an expression with only one unknown parameter n. We numerically solve this expression to obtain n and then the other three unknown parameters.

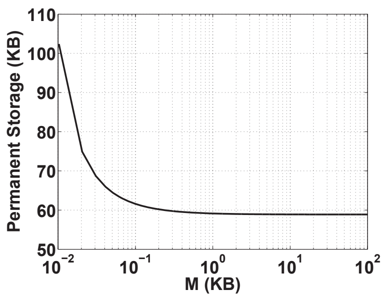


Figure 5.3 Permanent storage vs. RAM

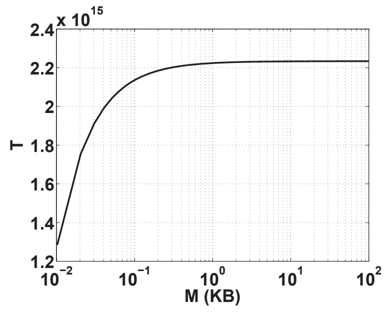


Figure 5.4 Threshold T vs. RAM size

RAM Space vs. Storage Space: From the simultaneous solution of these four equations, we have an interesting observation that COLATE requires smaller amount of permanent storage space for storing counter epochs when it is allocated with more RAM for storing counter vectors. Figure 5.3 plots an example graph of the permanent storage size vs. RAM size for COLATE.

While this observation seems surprising, it makes intuitive sense because the sum of the two maximum values of two b-bit numbers is $2 \times 2^b = 2^{b+1}$ whereas the maximum value of a $2b$-bit number is $2^{2b} >> 2^{b+1}$. As we increase the total number of bits in a counter vector, *i.e.*, M, the counter value threshold T increases as shown in Figure 5.4. This implies that the frequency of writing the counter vector into permanent storage is reduced, and although each counter epoch takes more space, the overall required storage space is reduced as shown in Figure 5.3. The M value at the knee of the curve in Figure 5.3 represents the best tradeoff point between RAM space and permanent storage space.

5.5.3 Flexibility in Parameter Selection

If we only measure average per-flow latency, each observation point can choose its own values for the parameters of n, m, b, and T based on its available resources and traffic condition without global coordination among observation points. If we also need to measure standard deviation, then all observation points need to use the same value of m because Theorem 18 requires that the two sets $\mathcal{Y}_{lS}$ and $\mathcal{Y}_{lR}$ contain the time stamps from the same set of packets at the two observation points, which is possible only when the value of m is the same at both observation points. The remaining three parameters can still be chosen independently at each observation point.

5.6 Performance Evaluation

We implemented COLATE in Matlab. We also implemented RLI [68] in Matlab for comparison purposes. We did not implement LDA [63] and MAPLE [69] because LDA can not provide per-flow latency measurement and MAPLE requires attaching time stamps to every packet *i.e.*, MAPLE is not a latency estimation scheme but a storage scheme. In this section, we present our evaluation results of COLATE in comparison with RLI. We first give details of the three network traces that we used. Second, we evaluate the accuracy of COLATE as well as the impact of RAM space on permanent storage space used by COLATE. Last, we compare COLATE with RLI and with Count-Min (CM) sketch [43], a summarizing data structure for queries on data streams.

5.6.1 Network Traces

To evaluate COLATE, we need real packet traces with high-resolution time stamps collected simultaneously from at least two observation points. Unfortunately, no such traces are publicly available. Thus, we resort to three real network traces where each is collected at a single observation point at a time. These traces include CHIC [4], ICSI [88], and DC

[32]. CHIC is a backbone header trace, published by CAIDA, which includes the arrival times of packets at a 10GigE link interface. We used traces generated from 5 minutes of packet capture. Note that the authors of RLI and MAPLE also evaluated their schemes on the header trace of this backbone link collected by CAIDA. ICSI is an enterprise network traffic trace, collected at a medium-sized site, which includes the arrival times of packets on an ethernet link for a duration of over 41.1 hours. ICSI is available in the form of 41 trace files collected at 17 different ports in an enterprise network. DC is a data center traffic trace, collected at a university data center, which includes the arrival times of packets on an ethernet link for a duration of a little more than an hour. DC is available in the form of 20 trace files collected at the same port. Figure 5.5 shows the CDFs of sizes of flows in each trace. We observe that the traces contain both mice flows as well as elephant flows. Table 5.1 reports the total duration, number of packets, number of flows, and average data rate of each trace.

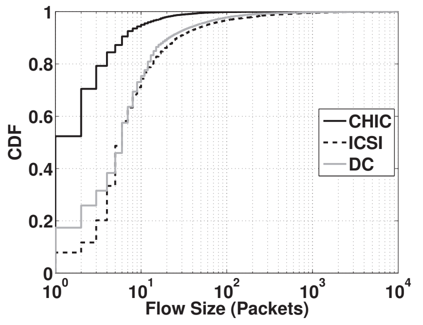


Figure 5.5 Flow sizes CDFs

As these network traces contain only the arrival time stamps of packets, we adopt the simulation strategy used by RLI and MAPLE, which simulates the traversal of packets in each trace through a queue to get a departure time stamp for each packet and uses random early detection (RED) [48] as the queue management strategy because RED is most popular. As modern routers typically use a queue size that can hold 0.5 seconds of traffic at their maximum line rates, we also use the same queue size. For the remaining parameters of RED queue management strategy, we use $\min_{th} = 0.475\times$queue size, $\max_{th} = 0.95\times$queue size,

$w_q = 0.002$, and $\max_p = \frac{1}{50}$ as per the guidelines in [48].

Table 5.1 Summary of network traces

Trace	**Duration**	**# pkts**	**# flows**	**Mbps**
CHIC	5 mins	37.3M	3.01M	411
ICSI	41.1 hours	46.9M	0.387M	1.31
DC	1.08 hours	19.9M	0.439M	49.6

5.6.2 COLATE Accuracy

Now we evaluate the accuracy of the average and standard deviation of the flows in the three network traces estimated by COLATE.

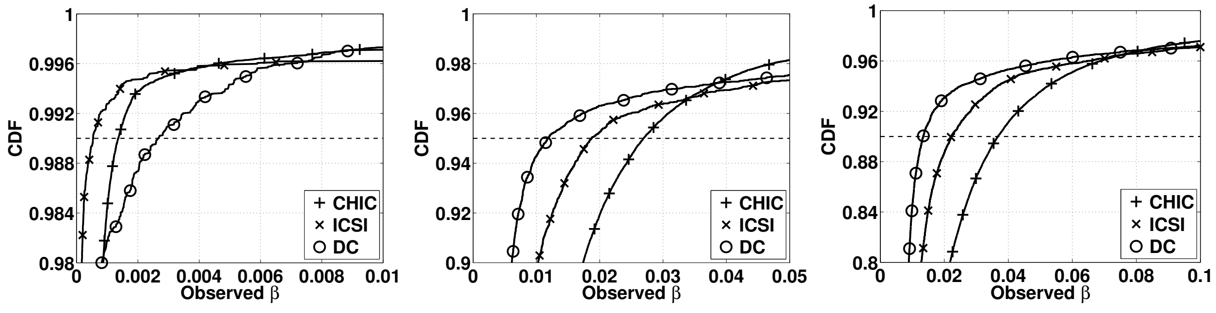
5.6.2.1 Average Latency

We evaluated COLATE for both the scenario of only two observation points (where one sender sends and one receiver receives) and that of more than two observation points (where multiple senders send and multiple receivers receive). For the scenario with more than two observation points, we choose three observation points forming a triangle topology where everyone sends to and receives from everyone else. We choose three observation points and the triangle topology for the sake of simplicity as the number of senders and receivers does not affect the accuracy of COLATE. For a triangle topology, there are 6 unidirectional links. We used the largest 6 of the 41 trace files from ICSI data set, each trace file representing the traffic on one of the 6 links. This choice is arbitrary.

We performed our evaluation for three different accuracy requirements: low ($\alpha = 0.90$, $\beta = 0.10$), medium ($\alpha = 0.95$, $\beta = 0.05$), and high ($\alpha = 0.99$, $\beta = 0.01$). For each of these three accuracy requirements, we evaluated COLATE using three values of available RAM: small ($M = 1$MB), medium ($M = 10$MB), and large ($M = 100$MB). We obtained the values of $t_{f,X}^{min}$ and $t_{f,X}^{max}$ from simple measurement of the network traces. We calculated the values of the parameters b, m, n, and T using the method described in Section 5.5. For example, for

$M = 1\text{MB}$, $\alpha = 0.95$, and $\beta = 0.05$, typical values of these parameters are $b = 19$, $m = 20$, $n = 455334$, and $T = 8 \times 10^{10}$.

Our results show that COLATE always achieves the required reliability. Figures 5.6(a), 5.6(b), and 5.6(c) show the CDFs of the observed values of β *i.e.*, $|\tilde{\mu}_f - \mu_f|/\mu_f$, for the average latency estimated by COLATE for the three traces under the scenario of one sender and one receiver using the low, medium, and high accuracy requirements, respectively. Figures 5.7(a), 5.7(b), and 5.7(c) show the CDFs of the observed values of β *i.e.*, $|\tilde{\mu}_f - \mu_f|/\mu_f$, for the average latency estimated by COLATE for the six links under the scenario of multiple senders and receivers using the low, medium, and high accuracy requirements, respectively. The horizontal line in each of these figures shows the required reliability. We see that every plot of CDF always crosses the horizontal line for an observed value of β that is smaller than the required confidence interval. This shows that COLATE always achieves the required reliability. Due to lack of space, we only show plots for $M = 10\text{MB}$. Observations from $M = 1\text{MB}$ and $M = 100\text{MB}$ are the same.

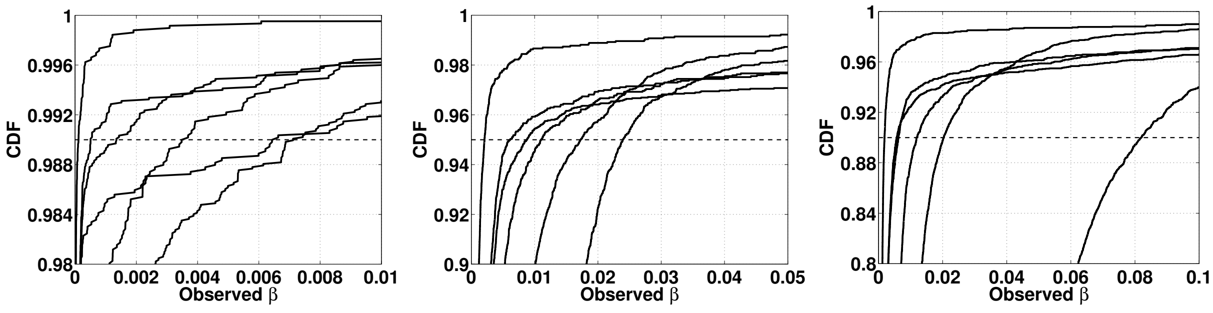


(a) $\alpha = 0.99, \beta = 0.01$ (b) $\alpha = 0.95, \beta = 0.05$ (c) $\alpha = 0.9, \beta = 0.1$

Figure 5.6 CDF of observed β in average estimate (1-S, 1-R)

5.6.2.2 Standard Deviation

Our results show that the relative error in the standard deviation estimates of over 91% *flows is less than* 0.05 *with only 1000 virtual repetitions.* Relative error is defined as

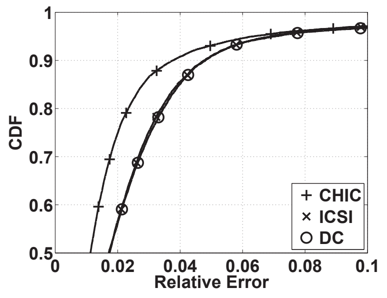


(a) $\alpha = 0.99, \beta = 0.01$ (b) $\alpha = 0.95, \beta = 0.05$ (c) $\alpha = 0.9, \beta = 0.1$

Figure 5.7 CDF of observed β in average estimate (multiple S,R)

$(\texttt{actual value} - \texttt{estimated value})/\texttt{actual value}$. Figure 5.8 plots the CDFs of the relative errors in standard deviation estimated by COLATE for each of the three traces.

Our results also show that the percentage of flows, for which the relative error is less than 0.05*, increases with the increase in the number of virtual repetitions.* Figure 5.9 plots this percentage versus the number of virtual repetitions for the three traces. With 10^5 iterations, this percentage reaches 98%. Although 10^5 iterations may take some time depending on the available computing power, it is not an issue as the estimation of standard deviation is an offline process and does not have to keep up with high line rates.



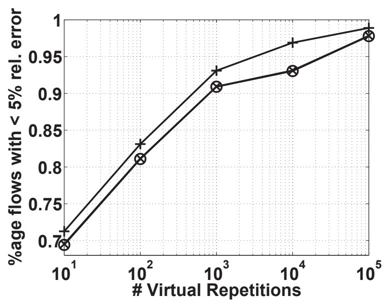


Figure 5.8 CDFs of relative errors in STD Figure 5.9 Rel. error in STD vs. # reps

5.6.3 RAM and Storage Size

Our results show that COLATE uses less than 0.1 bit of permanent storage per packet. Figure 5.10 shows the bar graph of the number of bits per packet used by each of the three traces

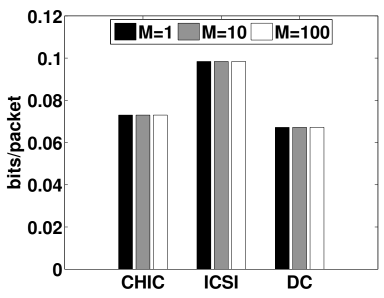


Figure 5.10 Storage bits per packet

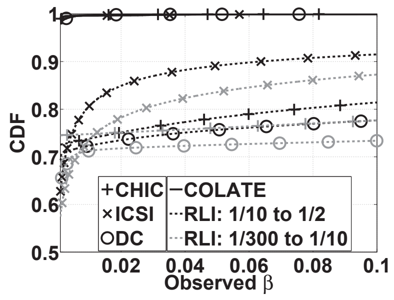


Figure 5.11 Comparison of delay estimates

for all three values of M, when $\alpha = 0.99$ and $\beta = 0.01$. This small size of storage required per packet results in a very low frequency of transferring the counter vectors from RAM to permanent storage. For example, for ICSI network trace, COLATE transfers a counter vector to SSD every 24.6 hours when $M = 1$MB, $\alpha = 0.95$, and $\beta = 0.05$. Transferring 1MB of content from RAM to SSD once a day is trivial for modern devices. The number of bits per packet in permanent storage decreases with the increase in RAM size M, which confirms our analysis based on Figures 5.3 and 5.4. However, this decrease is hard to observe from Figure 5.10 because the difference is small. Nevertheless, this difference becomes significant for longer time durations (on the order of say days and weeks).

5.6.4 Comparison with RLI

Our results show that COLATE always achieves higher accuracy than RLI. RLI requires two inputs, namely the lower and upper limits of the Probe packet Injection Rate (PIR). The authors of RLI used the lower limit as 1 probe packet per 300 regular packets and the upper limit as 1 probe packet per 10 regular packets in [68]. We first evaluated RLI using this pair of PIR values. Because the accuracy of RLI increases as PIR increases, to improve the estimation accuracy of RLI, although at the cost of larger bandwidth usage, we also evaluated RLI using much higher PIRs – 1 probe packet per 10 regular packets as the lower limit and 1 probe packet per 2 regular packets as the upper limit. Figure 5.11 plots the CDFs of the observed value of β in the average latency estimated by RLI in the three traces for these two configurations of PIRs. For comparison, we also plot the CDF of the observed value of β in the estimates obtained using COLATE for high accuracy requirement ($\alpha = 0.99, \beta = 0.01$). Note that the observed value of β is essentially the relative error in the estimated values of average latency. Figure 5.11 shows that the relative error of COLATE is much smaller than that of RLI. This relative error can be made arbitrarily small by specifying smaller values of β and larger values of α. This figure further shows that the relative error of RLI is smaller when PIR is larger. With the PIR proposed by the authors, on average, only 77% flows have relative error less than 5%. At this rate, on average, RLI inserts one probe packet after 21.66 regular packets. With our high PIR configuration, on average, only 81% flows have a relative error less than 5%, but at this rate, on average, RLI inserts one probe packet every 4.78 regular packets in the three traces. Table 5.2 shows the average number of regular packets after which RLI inserts a probe packet for each of the three traces. In contrast, COLATE does not insert any probe packet at the cost of a small amount of memory at observation points.

Table 5.2 Average number of regular packets after which RLI inserts a probe packet

Trace	**# reg. pkts**	**# reg. pkts**
	1:300 to 1:10	1:10 to 1:2
CHIC	18.66	10.0
ICSI	17.19	2.97
DC	245.7	9.06

5.6.5 Comparison with Count-Min Sketch

Count-Min (CM) sketches can theoretically be used for latency measurement but practically, there is a fundamental limitation. Let t_{f_i} represent the sum of time stamps of all packets in flow f_i and let there be j flows in total whose time stamps need to be stored for latency measurements. We can use a CM-sketch to store time stamps of multiple flows and obtain the estimate $\tilde{t}_{f_i}$ of the sum of time stamps of any flow f_i as per the method described in [43]. The estimate $\tilde{t}_{f_i}$ obtained through CM-sketch can satisfy the condition $\tilde{t}_{f_i} \leq t_{f_i} + B \times \sum_{\forall j} t_{f_j}$ with probability α. This is problematic because we require the estimate to satisfy the condition $\tilde{t}_{f_i} \leq t_{f_i} + B \times t_{f_i}$ with probability α. Therefore, to achieve the required reliability using CM-sketch, we need to ensure that $\sum_{\forall j} t_{f_j} \leq t_{f_i}$, which is possible if and only if we do not add time stamps of any flow other than f_i to the CM-sketch. Consequently we need to maintain a CM-sketch for each flow, which results in large memory requirements. For example, for $\alpha = 0.95$ and $\beta = 0.05$, CM-sketch requires 165 counters per flow and 3 hash functions and 3 memory accesses per packet. Assuming the same counter size of 19 bits as for COLATE in this scenario, CM-sketch requires $165 \times 19 = 3135$ bits per flow. In comparison, COLATE requires 0.1 bit per packet. The memory requirement of CM-sketch matches that of COLATE only if each flow has at least 31350 packets, which is impractical as seen in Figure 5.5. The number of memory accesses and the number of hash functions per packet for CM-sktech are always greater than those for COLATE.

5.7 Conclusion

The key contribution of this chapter is in proposing an accurate and efficient per-flow latency measurement scheme without packet probing and time stamping. The key novelty of this work is that we purposely allow noise to be introduced in recording packet timing information for minimizing storage space and use statistical techniques to denoise the recorded information to obtain accurate latency estimates when latency of a target flow is queried. The key technical depth of this chapter is in the mathematical development of the estimation theory that our scheme is based upon. Our theoretical analysis and experimental results show that our scheme always achieves the required reliability. Our scheme has a much smaller processing overhead in terms of number of hash computations and memory updates compared to existing schemes, which further require sending probe packets or attaching time stamps to every packet. Our scheme is scalable in that the amount of memory required at each observation point is only dependent on the number of packets and not on the number of sending and receiving observation points. The memory requirement is so low that a commodity storage device can store time stamps of several years worth of flows.

6 User Security

6.1 Introduction

6.1.1 Motivation

Touch screens have revolutionized and dominated the user input technologies for mobile computing devices (such as smart phones and tablets) because of high flexibility and good usability. Mobile devices equipped with touch screens have become prevalent in our lives with increasingly rich functionalities, enhanced computing power, and more storage capacity. Many applications (such as email and banking) that we used to run on desktop computers are now also being widely run on such devices. These devices therefore often contain privacy sensitive information such as personal photos, email, credit card numbers, passwords, corporate data, and even business secrets. Losing a smart phone with such private information could be a nightmare for the owner. Numerous cases of celebrities losing their phones with private photos and secret information have been reported on news [2]. Recently, security firm Symantec conducted a real-life experiment in five major cities in North America by leaving 50 smart phones in streets without any password/PIN protection [15]. The results showed that 96% of finders accessed the phone with 86% of them going through personal information, 83% reading corporate information, 60% accessing social networking and personal emails, 50% running remote admin, and 43% accessing online bank accounts.

Safeguarding the private information on such mobile devices with touch screens therefore becomes crucial. The widely adopted solution is that a device locks itself after a few minutes of inactivity and prompts a password/PIN/pattern screen when reactivated. For example, iPhones use a 4-digit PIN and Android phones use a geometric pattern on a grid of points, where both the PIN and the pattern are secrets that users should configure on their phones. These password/PIN/pattern based unlocking schemes have three major weaknesses. First, they are susceptible to shoulder surfing attacks. Mobile devices are often used in public settings (such as subway stations, schools, and cafeterias) where shoulder surfing often happens either purposely or inadvertently, and passwords/PIN/patterns are easy to spy [117, 96]. Second, they are susceptible to smudge attacks, where imposters extract sensitive information from recent user input by using the smudges left by fingers on touch screens. Recent studies have shown that finger smudges (*i.e.*, oily residues) of a legitimate user left on touch screens can be used to infer password/PIN/pattern [30]. Third, passwords/PINs/patterns are inconvenient for users to input frequently, so many people disable them leaving their devices vulnerable.

6.1.2 Proposed Approach

In this chapter, we propose GEAT, a gesture based authentication scheme for the secure unlocking of touch screen devices. A gesture is a brief interaction of a user's fingers with the touch screen such as swiping or pinching with fingers. Figure 6.1 shows two simple gestures on smart phones. Rather than authenticating users based on *what* they input (such as a password/PIN/pattern), which are inherently subjective to shoulder surfing and smudge attacks, GEAT authenticates users mainly based on *how* they input. Specifically, GEAT first asks a user to perform a gesture on touch screens for about 15 to 25 times to obtain training samples, then extracts and selects behavior features from those sample gestures, and finally builds models that can classify each gesture input as legitimate or illegitimate using machine learning techniques. The key insight behind GEAT is that people have consistent and distinguishing behavior of performing gestures on touch screens. We implemented GEAT on Samsung Focus, a Windows based phone, as seen in Figure 6.1 and evaluated it using

15009 gesture samples that we collected from 50 volunteers. Experimental results show that GEAT achieves an average Equal Error Rate (EER) of 0.5% with 3 gestures using only 25 training samples.

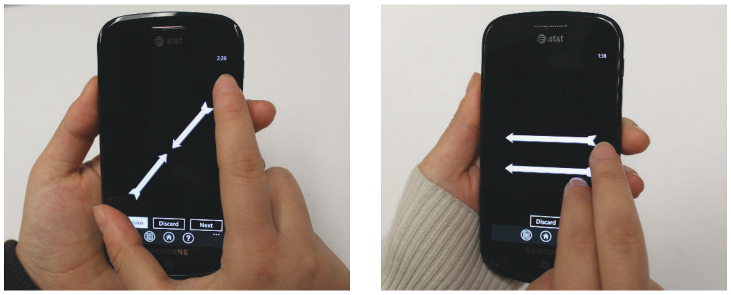


Figure 6.1 GEAT implemented on Windows Phone 7

Compared to current secure unlocking schemes for touch screen devices, GEAT is significantly more difficult to compromise because it is nearly impossible for an imposter to reproduce the behavior of others doing gestures through shoulder surfing or smudge attacks. Unlike password/PIN/pattern based authentication schemes, GEAT allows users to securely unlock their touch screen devices even when imposters are spying on them. GEAT actually displays the gesture that the user needs to perform on the screen for unlocking. Compared with biometrics (such as fingerprint, face, iris, hand, and ear) based authentication schemes, GEAT has two key advantages on touch screen devices. First, GEAT is secure against smudge attacks whereas some biometrics, such as fingerprint, are subject to such attacks as they can be copied. Second, GEAT does not require additional hardware for touch screen devices whereas biometrics based authentication schemes often require special hardware such as a fingerprint reader.

For practical deployment, we propose to use password/PIN/pattern based authentication schemes to help GEAT to obtain the training samples from a user. In the first few days of using a device with GEAT enabled, in each unlocking, the device first prompts the user to do a gesture and then prompts with the password/PIN/pattern login screen. If the user successfully logged in based on his password/PIN/pattern input, then the information that GEAT recorded during the user performing the gesture is stored as a training sample; otherwise, that gesture is discarded. Of course, if the user prefers not to set up a password/PIN/pattern, then the password/PIN/pattern login screen will not be prompted and the gesture input will be automatically stored as a training sample. During these few days of training data gathering, users should specially guard their password/PIN/pattern input from shoulder surfing and smudge attacks. In reality, even if an imposter compromises the device by shoulder surfing or smudge attacks on the password/PIN/pattern input, the private information stored on the device during the initial few days of using a new device is typically minimal. Plus, the user can easily shorten this training period to be less than a day by unlocking his device more frequently. We only need to obtain about 15 to 25 training samples for each gesture. After the training phase, the password/PIN/pattern based unlocking scheme is automatically disabled and GEAT is automatically enabled. It is possible that a user's behavior of doing the gesture evolve over time. Such evolution can be handled by adapting the scheme proposed by Monrose *et al.* [81].

6.1.3 Technical Challenges and Solutions

The first challenge is to choose features that can model *how* a gesture is performed. In this work, we extract the following seven types of features: velocity magnitude, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction. The first five feature types capture the dynamics of performing gestures while the remaining two capture the static shapes of gestures. (1) *Velocity Magnitude:* the speed of finger motion at different time instants. (2) *Device Acceleration:* the acceleration of touch screen device movement along the three perpendicular axes of the device. (3) *Stroke Time:* the time duration that the user takes to complete each stroke. (4) *Inter-stroke Time:* the time duration between the starting time of two consecutive strokes for multi-finger gestures. (5) *Stroke Displacement Magnitude:* the Euclidean distance between the centers of the bounding boxes of two strokes for multi-finger gestures, where the bounding box of a stroke is the smallest rectangle that completely contains that stroke. (6) *Stroke Displacement Direction:* the direction of the line connecting the centers of the bounding boxes of two strokes for multi-finger gestures. (7) *Velocity Direction:* the direction of finger motion at different time instants.

The second challenge is to segment each stroke into sub-strokes for a user so that the user has consistent and distinguishing behavior for the sub-strokes. It is challenging to determine the number of sub-strokes that a stroke should be segmented into, the starting point of each sub-stroke, and the time duration of each sub-stroke. On one hand, if the time duration of a sub-stroke is too short, then the user may not have consistent behavior for that sub-stroke when performing each gesture. On the other hand, if the time duration of a sub-stroke is too large, then the distinctive information from the features is too much averaged out to be useful for authentication. The time duration of different sub-strokes should not be all equal because at different locations of a gesture a user may have consistent behaviors that last different amounts of time. In this work, we propose an algorithm that automatically segments each stroke into sub-strokes of appropriate time duration where for each sub-stroke the user has consistent and distinguishing behavior. We use coefficient of variation to quantify consistency.

The third challenge is to learn multiple behaviors from the training samples of a gesture because people exhibit different behaviors when they perform the same gesture in different postures such as sitting and lying down. In this work, we distinguish the training samples that a user made under different postures by making least number of *minimum variance partitions*, where the coefficient of variation for each partition is below a threshold, so that each partition represents a distinct behavior.

The fourth challenge is to remove the high frequency noise in the time series of coordinate values of touch points. This noise is introduced due to the limited touch resolution of capacitive touch screens. In this work, we pass each time series of coordinate values through a low pass filter to remove high frequency noise.

The fifth challenge is to design effective gestures. Not all gestures are equally effective for authentication purposes. In our study, we designed 39 simple gestures that are easy to perform and collected data from our volunteers for these gestures. After comprehensive evaluation and comparison, we finally chose 10 most effective gestures shown in Figure 6.2. The number of unconnected arrows in each gesture represents the number of fingers a user should use to perform the gesture. Accordingly we can categorize gestures into single-finger gestures and multi-finger gestures.

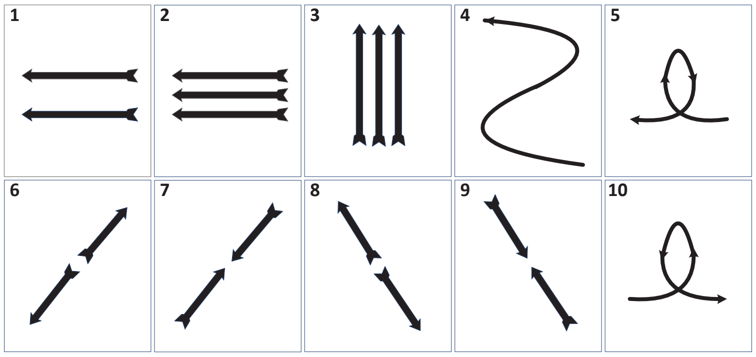


Figure 6.2 The 10 gestures that GEAT uses

The sixth challenge is to identify gestures for a given user that result in low false positive and false negative rates. In our scheme, we first ask a user to provide training samples for as many gestures from our 10 gestures as possible. For each gesture, we develop models of user behaviors. We then perform elastic deformations on the training gestures so that they stop representing legitimate user's behavior. We classify these deformed samples and calculate EER for a given user for each gesture and rank the gestures based on their EERs. Then we use the top n gestures for authentication using majority voting where n is selected by the user. Although larger n is, higher accuracy GEAT has, for practical purposes such as unlocking smart phone screens, $n = 1$ (or 3 at most) gives high enough accuracy.

6.1.4 Threat Model

During the training phase of a GEAT enabled touch screen device, we assume imposters cannot have physical access to it. After the training phase, we assume imposters have the following three capabilities. First, imposters have physical access to the device. Such physical access can be gained in ways such as thieves stealing a device, finders finding a lost device, and roommates temporarily holding a device when the owner is taking a shower. Second, imposters can launch shoulder surfing attacks by spying the owner when he performs gestures. Third, imposters have necessary equipment and technologies to launch smudge attacks.

6.1.5 Key Contributions

In this chapter, we make following five key contributions.

1. We proposed, implemented, and evaluated a gesture based authentication scheme for the secure unlocking of touch screen devices.
2. We identified a set of effective features that capture the behavioral information of performing gestures on touch screens.
3. We proposed an algorithm that automatically segments each stroke into sub-strokes of different time duration where for each sub-stroke the user has consistent and distinguishing behavior.
4. We proposed an algorithm to extract multiple behaviors from the training samples of a given gesture.
5. We collected a comprehensive data set containing 15009 training samples from 50 users and evaluated the performance of GEAT on this data set.

6.2 Related Work

6.2.1 Gesture Based Authentication on Phones

A work parallel to ours is that Luca *et al.* proposed to use the timing of drawing the password pattern on Android based touch screen phones for authentication [75]. Their work has following two major technical limitations compared to our work. First, unlike ours, their scheme has low accuracy. They feed the time series of raw coordinates of the touch points of a gesture to the dynamic time wrapping signal processing algorithm. They do not extract any behavioral features from user's gestures. Their scheme achieves an accuracy of 55%; in comparison, ours achieves an accuracy of 99.5%. Second, unlike ours, they can not handle the multiple behaviors of doing the same gesture for the same user.

Another work parallel to ours is that Sae-Bae *et al.* proposed to use the timing of performing five-finger gestures on multi-touch capable devices for authentication [95]. Their work has following four major technical limitations compared to our work. First, their scheme requires users to use all five fingers of a hand to perform the gestures, which is very inconvenient on small touch screens of smart phones. Second, they also feed the time series of raw coordinates of the touch points to the dynamic time wrapping signal processing algorithm and do not extract any behavioral features from user's gestures. Third, they can not handle the multiple behaviors of doing the same gesture for the same user. Fourth, they have not evaluated their scheme in real world attack scenarios such as resilience to shoulder surfing.

6.2.2 Phone Usage Based Authentication

Another type of authentication schemes leverages the behavior in using several features on the smart phones such as making calls, sending text messages, and using camera [112, 42]. Such schemes were primarily developed for continuously monitoring smart phone users for their authenticity. These schemes take a significant amount of time (often more than a day) to determine the legitimacy of the user and are not suitable for instantaneous authentication, which is the focus of this chapter.

6.2.3 Keystrokes Based Authentication

Some work has been done to authenticate users based on their typing behavior [126, 81]. Such schemes have mostly been proposed for devices with physical keyboards and have low accuracy [60]. It is inherently difficult to model typing behavior on touch screens because most people use the same finger(s) for typing all keys on the keyboard displayed on a screen. Zheng *et al.* [130] reported the only work in this direction in a technical report, where they did a preliminary study to check the feasibility of using tapping behavior for authentication.

6.2.4 Gait Based Authentication

Some schemes have been proposed that utilize accelerometer in smart phones to authenticate users based upon their gaits [78, 52, 65]. Such schemes have low true positive rates because gaits of people are different on different types of surfaces such as grass, road, snow, wet surface, and slippery surface.

6.3 Data Collection and Analysis

In this section, we first describe our data collection process for gesture samples from our volunteers. Second, we extract the seven types of features from our data and validate our hypothesis that people have consistent and distinguishing behaviors of performing gestures on touch screens.

6.3.1 Data Collection

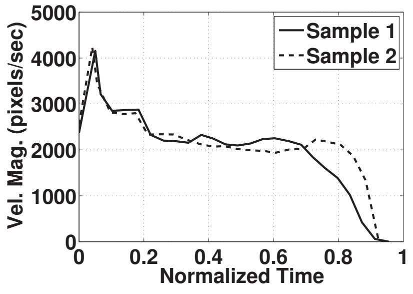
We developed a gesture collection program on Samsung Focus, a Windows based phone. During the process of a user performing a gesture, our program records the coordinates of each touch point and the accelerometer values and time stamps associated with each touch point. The duration between consecutive touch points provided by the Windows API on our device is 18ms. To track movement of multiple fingers, our program ascribes each touch point to its corresponding finger.

We found 50 volunteers with age ranging from 19 to 55 and jobs ranging from students, faculty, to corporate employees. We gave a phone to each volunteer for a period of 7 to 10 days and asked them to perform gestures over this period. Our data collection process consists of two phases. In the first phase, we chose 20 of the volunteers to collect data for the 39 gestures that we designed and each volunteer performed each gesture for at least 30 times. We conducted experiments to evaluate the classification accuracy of each gesture. An interesting finding is that different gestures have different average classification accuracies. We finally choose 10 gestures that have the highest average classification accuracies and discarded the remaining 29 gestures. These 10 gestures are shown in Figure 6.2. In the second phase, we collected data on these 10 gestures from the remaining 30 volunteers, where each volunteer performed each gesture for at least 30 times. Finally, we obtained a total of 15009 samples for these 10 gestures. The whole data collection took about 5 months.

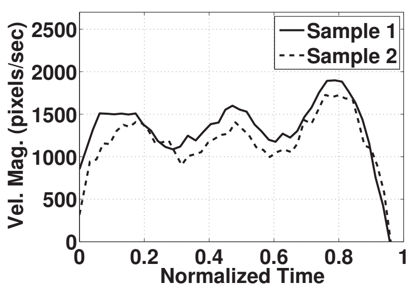
6.3.2 Data Analysis

We extract the following seven types of features from each gesture sample: velocity magnitude, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction.

• **Velocity and Acceleration Magnitude:** From our data set, we observe that people have consistent and distinguishing patterns of velocity magnitudes and device accelerations along its three perpendicular axes while doing gestures. For example, Figure 6.3(a) shows the time series of velocity magnitudes of two samples of gesture 4 in Figure 6.2 performed by a volunteer. Figure 6.3(b) shows the same for another volunteer. Similarly Figures 6.4(a) and 6.4(b) show the time series of acceleration along the x-axis in two samples of gesture 4 by two volunteers. We observe that the samples from same user are similar and at the same time different from samples from another user.

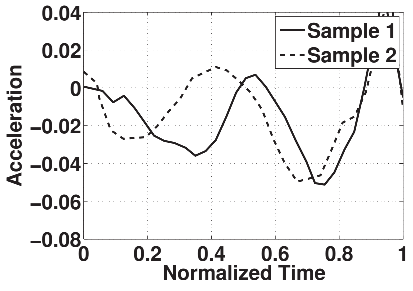


(a) Volunteer 1

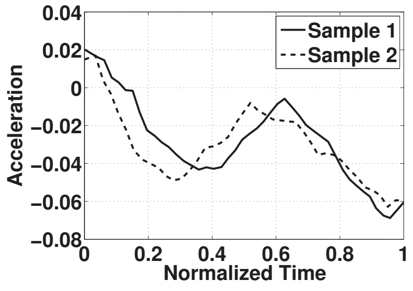


(b) Volunteer 2

Figure 6.3 Velocity magnitudes of gesture 4



(a) Volunteer 1



(b) Volunteer 2

Figure 6.4 Device acceleration of gesture 4

To quantify the similarity between any two time series, f_1 with m_1 values and f_2 with m_2 values, where $m_1 \leq m_2$, we calculate the root mean squared (RMS) value of the time series obtained by subtracting the normalized values of f_1 from the normalized values of f_2. Normalized time series $\hat{f}_i$ of a time series f_i is calculated as below, where $f_i[q]$ is the q^{th} value in f_i.

$$\hat{f}_i[q] = \frac{f_i[q] - \min(f_i)}{\max\big(f_i - \min(f_i)\big)} \quad \forall q \in [1, m_i] \tag{6.1}$$

Normalizing the time series brings all its values in the range of [0, 1]. We do not use metrics such as correlation to measure similarity between two time series because their values are not bounded.

To subtract one time series from the other, the number of elements in the two need to be equal; however, this often does not hold. Thus, before subtracting, we re-sample f_2 at a sampling rate of m_1/m_2 to make f_2 and f_1 equal in number of elements. The RMS value of a time series f containing $\mathcal{N}$ elements, represented by P_f, is calculated as:

$$P_f = \sqrt{\frac{1}{\mathcal{N}} \sum_{m=1}^{\mathcal{N}} f^2[m]} \tag{6.2}$$

Normalizing the two time series before subtracting them to obtain f ensures that each value in f lies in the range of $[-1, 1]$ and consequently the RMS value lies in the range of $[0, 1]$. An RMS value closer to 0 implies that the two time series are highly alike while an RMS value closer to 1 implies that the two time series are very different. For example, the RMS value between the two time series from the volunteer in Figure 6.3(a) is 0.119 and that between the two time series of the volunteer in Figure 6.3(b) is 0.087, whereas the RMS value between a time series in Figure 6.3(a) and another in Figure 6.3(b) is 0.347. Similarly, the RMS values between the two time series of each volunteer in Figures 6.4(a) and 6.4(b) are 0.159 and 0.144, respectively, whereas the RMS value between one time series in Figure 6.4(a) and another in Figure 6.4(b) is 0.362.

• **Stroke Time, Inter-stroke Time, and Stroke Displacement Magnitude:** From our data set, we observe that people take consistent and distinguishing amount of time to complete each stroke in a gesture. For multi-finger gestures, people have consistent and distinguishing time duration between the starting times of two consecutive strokes in a gesture and have consistent and distinguishing magnitudes of displacement between the centers of any two strokes. The distributions of stroke times of different users are centered at different means and the overlap is usually small, which becomes insignificant when the feature is used with other features. Same is the case for inter-stroke times and stroke displacement magnitudes. Figures 6.5, 6.6, and 6.7 plot the distribution of stroke time of gesture 4, inter-stroke time of gesture 6, and stroke displacement magnitude of gestures 7, respectively, for different volunteers. The figures show that the overlap in distributions for different users is small and are centered at different means.

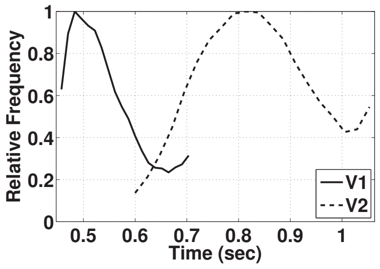


Figure 6.5 Distributions of stroke time

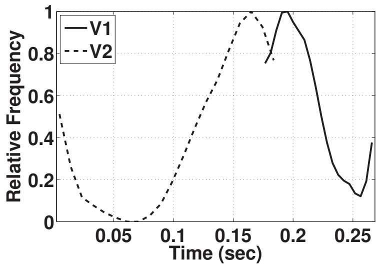


Figure 6.6 Dists. of inter-stroke time

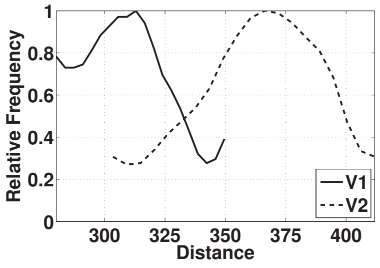


Figure 6.7 Distributions of disp. mag.

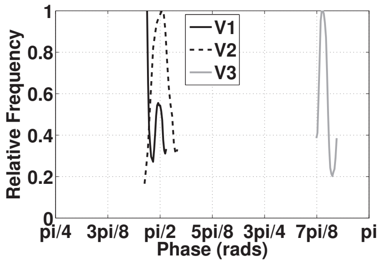


Figure 6.8 Distributions of disp. dir.

• **Stroke Displacement and Velocity Directions** From our data set, we observe that people have consistent, but not always distinguishing, patterns of velocity and stroke displacement directions because different people may produce gestures of similar shapes. For example, Figure 6.8 plots the distributions of the displacement direction of gesture 1 for three volunteers. Figure 6.9 shows the time series of velocity directions of gesture 10 for three volunteers. Volunteers V1 and V2 produced similar shapes of gesture 1 as well as gesture 10, so they have overlapping distributions and time series. Volunteer V3 produced shapes of the two gestures different from the corresponding shapes produced by volunteers V1 and V2, and thus has a non-overlapping distribution and time series.

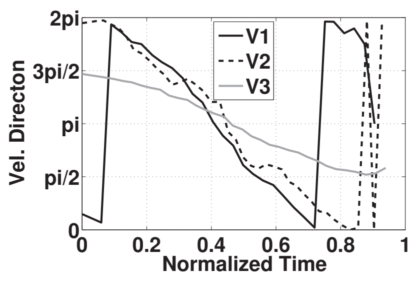


Figure 6.9 Velocity direction of gesture 10

6.4 GEAT Overview

To authenticate a user based on his behavior of preforming a gesture, GEAT needs to have a model of the legitimate user's behaviors of preforming that gesture. Given the training samples of the gesture performed by the legitimate user, GEAT builds this model using Support Vector Distribution Estimation (SVDE) in the following five steps.

The first step is *noise removal*, where GEAT passes the time series of touch point coordinates in each gesture sample through a filter to remove high frequency noise.

The second step is *feature extraction*, where GEAT extracts the values of the seven types of features from the gesture samples and concatenates these values to form a feature vector. To extract feature values of velocity magnitude, velocity direction, and device accelerations, GEAT segments each stroke in a gesture sample into sub-strokes at multiple time resolutions and extracts values from these sub-strokes. We call these three types of features *sub-stroke based features*. For the remaining four types of features, GEAT extracts values from the entire strokes in each gesture. We call these four types of features *stroke based features*.

The third step is *feature selection*. For each feature element, GEAT first partition all its N values, where N is the total number of training samples, into the least number of minimum variance partitions, where the coefficient of variation for each partition is below a threshold. If the number of minimum variance partitions is less than or equal to the number of postures in which the legitimate user provided the training samples, then we select this feature element; otherwise, we discard it. For this purpose, ideally the user should inform

GEAT the number of postures in which he performed the training gestures. However, if the user does not provide this information, the classification accuracy of GEAT decreases, but only very slightly, as shown in our experimental results in Section 6.9.

The fourth step is *classifier training*. GEAT first partitions all N feature vectors into the minimum number of groups so that within each group, all feature vectors belong to the same minimum variance partition for any feature element. We call each group a *consistent training group*. Then, for each group of feature vectors, GEAT builds a model in the form of an ensemble of SVDE classifiers trained using these vectors. Note that we do not use any gestures from imposters in training GEAT because in the real-world deployment of authentication systems, training samples are typically available only from the legitimate user.

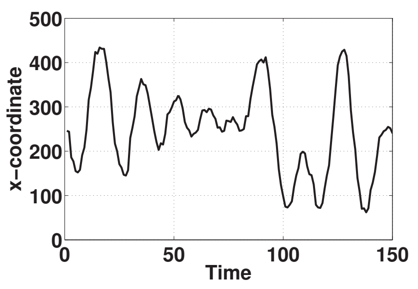
The fifth step is *gesture ranking*. For each gesture, GEAT repeats the above four steps and then ranks the gestures based on their EERs. A user can pick $1 \leq n \leq 10$ gestures to be used in each user authentication. Although the larger n is, the higher accuracy GEAT has, for practical purposes such as unlocking smart phone screens, $n = 1$ (or 3 at most) gives us high enough accuracy. To calculate the EER of a gesture, GEAT needs the true positive rates (TPR) and false positive rates (FPR) for that gesture. TPRs for each gesture are calculated using 10 fold cross validation on legitimate user's samples of the gesture. To calculate FPRs, GEAT needs imposter samples, which are not available in real world deployment at the time of training. Therefore, GEAT generates synthetic imposter samples by elastically deforming the samples of legitimate user using cubic B-splines and calculates the FPRs using these synthetic imposter samples. Note that the synthetic imposter samples are used only in ranking gestures, the performance evaluation of GEAT that we present in Section 6.9 is done entirely on real world imposter samples. These synthetic imposter samples are not used in classifier training either.

When a user tries to login on a touch screen device with GEAT enabled, the device displays the n top ranked gestures for the user to perform. Then authentication process behind the scene works as follows. First, for each gesture, GEAT extracts the values of all the feature elements selected earlier by the corresponding classification model for this gesture. Second, GEAT feeds the feature vector consisting these values to the ensemble of SVDE classifiers of each consistent training group and gets a classification decision. If the classification decision of any ensemble is positive, which means that the gesture has almost the same behavior as one of the consistent training groups that we identified from the training samples of the legitimate user, then GEAT accepts that gesture input to be legitimate. Third, after GEAT makes the decision for each of the n gestures, GEAT makes the final decision on whether to accept the user as legitimate based on the majority voting on the n decisions.

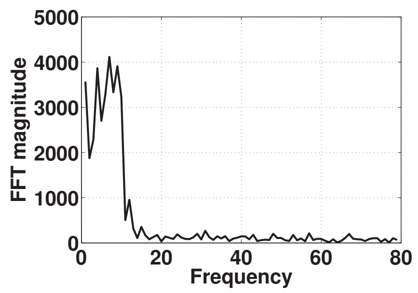
6.5 Noise Removal

The time series of x and y coordinates of the touch points of each stroke contain high frequency noise as we can see from the time series of x coordinates for a sample gesture in Figure 6.10(a). There are two major contributors to this noise. First, the touch resolution of capacitive touch screens is limited. Second, because capacitive touch screens determine the coordinates of each touch point by calculating the coordinates of the centroid of the area on the screen touched by a finger, when a finger moves on the screen, its contact area varies and the centroid changes at each time instant, resulting in high frequency noise. Such noise should be removed because it affects velocity magnitude and direction values.

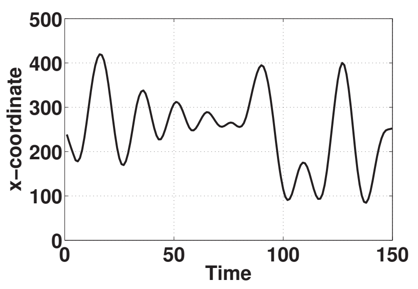
We remove such high frequency noise by passing the time series of x and y coordinates of touch points through a low pass filter. We consider frequencies above 20Hz as high frequencies because the time series of touch points contain most of their energy in frequencies lower than 20Hz, as we can see from the magnitude of the fourier transform of this time series in Figure 6.10(b). In this work, we use a simple moving average (SMA) filter, which is the unweighted mean of previous α data points. We choose the value of α to be 10. Figure 6.10(c) shows the time series of Figure 6.10(a) after passing through the SMA filter. We can see that the



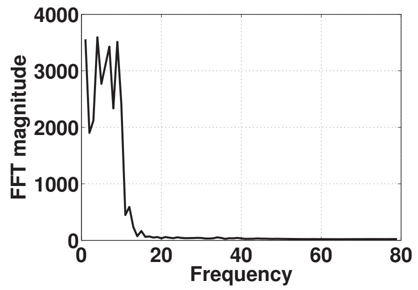
(a) Unfiltered



(b) FFT of unfiltered



(c) Filtered



(d) FFT of filtered

Figure 6.10 Unfiltered and filtered time series

filtered time series is much smoother compared to the unfiltered time series. Figure 6.10(d) shows the magnitude of fourier transform of the filtered time series. We observe from this figure that the magnitudes of frequency components above 20Hz are negligible.

6.6 Feature Extraction & Selection

In this section, we describe the feature extraction and selection process in GEAT. We categorize the seven types of features into *stroke based features*, which include stroke time, inter-stroke time, stroke displacement magnitude, and stroke displacement direction, and *sub-stroke based features*, which include velocity magnitude, velocity direction, and device acceleration.

6.6.1 Stroke Based Features

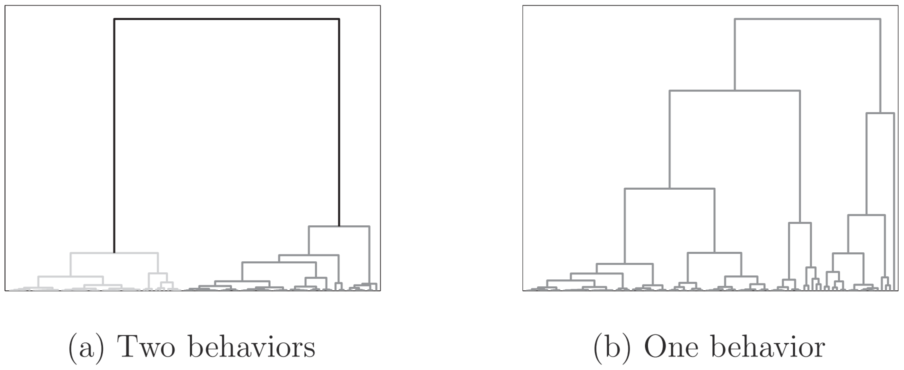
6.6.1.1 Extraction

To extract the stroke time of each stroke, we calculate the time duration between the time of the first touch point and that of the last touch point of the stroke. To extract the inter-stroke time between two consecutive strokes in a gesture, we calculate the time duration between the time of the first touch point of the first stroke and that of the second stroke. To extract the stroke displacement magnitude between any two strokes in a gesture, we calculate the Euclidean distance between the centers of the two bounding boxes of the two strokes. To extract stroke displacement direction between any two strokes in a gesture, we calculate the arc-tangent of the ratio of the magnitudes of the vertical component and the horizontal component of the stroke displacement vector directed from the center of one bounding box to the center of the other bounding box. We calculate inter-stroke time and stroke displacement magnitude and direction from all pairs of strokes in a gesture.

6.6.1.2 Selection

Given N training samples, for each feature element, we first partition all its N values into the least number of minimum variance partitions (MVPs) where the coefficient of variation (cv) for each partition is below a threshold. Let $\mathcal{P}_k$ and $\mathcal{Q}_k$ represent two different partitionings of N values, each containing k partitions. Let $\sigma_i^2(\mathcal{P}_k)$ and $\sigma_i^2(\mathcal{Q}_k)$ represent the variance of values in partition i $(1 \leq i \leq k)$ of partitioning $\mathcal{P}_k$ and $\mathcal{Q}_k$, respectively. Partitioning $\mathcal{P}_k$ is the MVP if for any $\mathcal{Q}_k$, $\max_i\left(\sigma_i^2(\mathcal{P}_k)\right) \leq \max_i(\sigma_i^2(\mathcal{Q}_k)\big)$. We empirically determined the threshold of the cv to be 0.1. The detailed empirical evaluation of this threshold is given in Section 6.9.

To find the least number of MVPs, we start by increasing the number of MVPs from one until cv of all partitions is below the threshold. To obtain MVPs, we use agglomerative hierarchical clustering with Ward's method [58]. Ward's method allows us to make any number of partitions by cutting the dendrogram built by agglomerative hierarchical clustering at an appropriate level. Figure 6.11 shows dendrograms made through hierarchical clustering with Ward's method form the values of stroke time of two volunteers for gesture 5. A dendrogram visually illustrates the presence and arrangement of clusters in data. The dendrogram in Figure 6.11(a) is for a volunteer who performed gestures in two postures, sitting and laying down. The dendrogram in Figure 6.11(b) is for a volunteer who performed gestures in one posture. We make two MVPs for Figure 6.11(a) and one for Figure 6.11(b).



(a) Two behaviors (b) One behavior

Figure 6.11 Dendrograms for feature values with one and two behaviors

After we find the least number of MVPs, where the *cv* for each partition is below the threshold, we decide whether to select this feature element. If the number of partitions in these MVPs is less than or equal to the number of postures in which the training samples are performed, then we select this feature element; otherwise, we discard it. We ask the user to enter the number of postures in which he performed training samples. If the user does not provide this input, we assume the number of postures to be 1.

6.6.2 Sub-stroke Based Features

Sub-stroke based features include velocity magnitude, velocity direction, and device acceleration. To extract values for these features, GEAT needs to segment each stroke into sub-strokes because of two major reasons. First, at different segments of a stroke, the finger often has different moving speed and direction. Second, at different segments of a stroke, the device often has different acceleration. If we measure the feature values from the entire stroke, we will only utilize the information measured at the starting and ending points of the stroke, by which we will miss the distinguishing information of velocity magnitude, velocity direction, and device acceleration at different segments of the stroke.

Our goal is to segment a stroke into sub-strokes so that the velocity magnitude, velocity direction, and device acceleration information measured at each sub-stroke characterizes the distinguishing behaviors of the user who made the stroke. There are three key technical challenges to this goal. The first technical challenge is how we should segment N stroke samples of different time durations assuming that we are given an appropriate time duration as the segmentation guideline. The second technical challenge is how to find the appropriate time duration as the segmentation guideline. The third technical challenge is how to select sub-strokes whose velocity magnitude, velocity direction, and device acceleration information will be included in the feature vector used by GEAT for training. Next, we present our solutions to these three technical challenges.

6.6.2.1 Stroke Segmentation and Feature Extraction

Given N strokes performed by one user and the appropriate time duration p as the segmentation guideline, we need to segment each stroke into the same number of segments so that for each stroke we obtain the same number of feature elements. However, because different strokes have different time durations, segmenting each stroke into sub-strokes of time duration p will not give us the same number of segments for different strokes. To address this issue, we first calculate $\lceil \frac{\mathfrak{t}}{p} \rceil$ for each stroke where $\mathfrak{t}$ is the time duration of the stroke. From the resulting N values, we use the most frequent value, denoted s, to be the number of sub-strokes that each stroke should be segmented into. Finally, we segment each stroke into s sub-strokes where each sub-stroke within a stroke has the same time duration.

After segmenting all strokes into sub-strokes, we extract velocity magnitude, velocity direction, and device acceleration from each sub-stroke. To calculate velocity magnitude and direction, we first obtain the coordinates of the starting and ending points of the sub-stroke. The starting and ending points of a sub-stroke, which is segmented from a stroke based on time duration, often do not lie exactly on touch points reported by the touch screen device. For any end point that lies between two consecutive touch points reported by the touch screen device, we calculate its coordinates by interpolating between these two touch points. Let (x_i, y_i) be the coordinates of a touch point with time stamp t_i and (x_{i+1}, y_{i+i}) be the coordinates of the adjacent touch point with time stamp t_{i+1}. Suppose the time stamp of an end point is t where $t_i < t < t_{i+1}$. Then, we calculate the coordinates (x, y) of this end point based on the straight line between (x_i, y_i) and (x_{i+1}, y_{i+i}) as follows:

$$x = \frac{(t - t_i)}{(t_{i+1} - t_i)} \times (x_{i+1} - x_i) + x_i \tag{6.3}$$

$$y = \frac{(t - t_i)}{(t_{i+1} - t_i)} \times (y_{i+1} - y_i) + y_i \tag{6.4}$$

We extract the velocity magnitude of each sub-stroke by calculating the Euclidean distance between the starting and ending points of the sub-stroke divided by the time duration of the sub-stroke. We extract the velocity direction of each sub-stroke by calculating the arc-tangent of the ratio of the magnitudes of the vertical and horizontal components of the velocity vector directed from the starting point to the ending point of the sub-stroke. We extract the device acceleration during each sub-stroke by averaging the device acceleration values reported by the touch screen device at each touch point in that sub-stroke in all three directions.

6.6.2.2 Sub-stroke Time Duration

Next, we investigate how to find the appropriate sub-stroke time duration. On one hand, when the sub-stroke time duration is too small, the behavior information extracted from each sub-stroke of the same user may become inconsistent because when feature values become instantaneous, they are unlikely to be consistent for the same user. For example, from Figure 6.12, which shows the *cv* for the velocity magnitude values extracted from the first sub-stroke from all samples of a gesture performed by a random volunteer in our data set, when we vary the sub-stroke time duration from 5ms to 100ms, we observe that the *cv* is too large to be usable when the sub-stroke time duration is too small and the *cv* decreases as we increase sub-stroke time duration. On the other hand, when the sub-stroke time duration is too large, the behavior information extracted from each sub-stroke of different users may become similar because all unique dynamics of individual users are too averaged out to be distinguishable. For example, treating all the samples of a gesture performed by all our volunteers as if they are all performed by the same person, Figure 6.13 shows that when the sub-stroke time duration is 80ms, over 60% of feature elements of velocity magnitude are consistent, which means that they do not have any distinguishing power among different users. It is therefore challenging to trade off between consistency and distinguishability in choosing the appropriate time duration for sub-strokes.

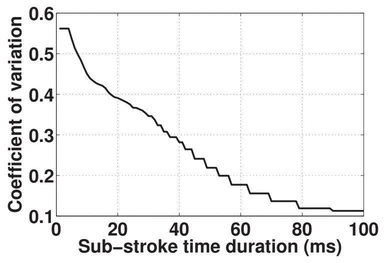


Figure 6.12 *cv* vs. time periods

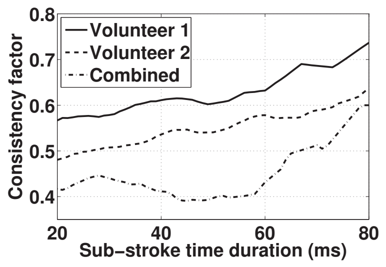


Figure 6.13 Consistency factor

Next, we present the way that we achieve this tradeoff and find the appropriate time duration for sub-strokes. We first define a metric called *consistency factor*. Given a set of samples of the same stroke, which are segmented using time duration p as the guideline, let s be the number of sub-strokes, c be the number of sub-strokes that have the consistent behavior for a particular feature, we define the consistency factor of this set of samples under time duration p to be $\frac{c}{s}$. For simplicity, we use *combined consistency factor* to mean the consistency factor of the set of all samples of the same stroke from all volunteers, and *individual consistency factor* to mean the consistency factor of the set of all samples of the same stroke from the same volunteer. Figure 6.13 shows the combined consistency factor plot and two individual consistency factor plots of an example gesture. We have two important observations from this figure. First, the individual consistency factors mostly keep increasing as we increase sub-stroke time duration p. Second, the combined consistency factor has a significant dip when p is in the range from 30ms to 60ms. We conducted the similar measurement for other strokes from other gestures for velocity magnitude, velocity direction, and device acceleration and made the same two observations. This means that when sub-stroke time duration is between 30ms to 60ms, people have distinguishing behavior for the features of velocity magnitude, velocity direction, and device acceleration. Therefore, we choose time duration p to be between 30ms to 60ms.

6.6.2.3 Sub-stroke Selection at Appropriate Resolutions

So far we have assumed that all sub-strokes segmented from a stroke have the same time duration. However, in reality, people have consistent and distinguishing behavior for sub-strokes of different time durations. Next, we discuss how we find such sub-strokes of different durations. For each type of sub-stroke based features, we represent the entire time duration of a stroke as a line with the initial color of white. Given a set of samples of a stroke performed by one user under b postures, we first segment the stroke with the time duration $p = 60$ms and the number of MVPs $k = 1$. For each resulting sub-stroke, we measure cv of the feature values extracted from the sub-stroke. If it is lower than the threshold, then we choose this sub-stroke with k MVPs as a feature element and color this sub-stroke in the line as black. After this round of segmentation, if any white sub-stroke is left, we move to the next round of segmentation on the entire stroke with $p = 55$ms and the number of MVPs k still being 1. In this round, for any sub-stroke whose color is completely white, we measure its cv; if it is lower than the threshold, then we choose this sub-stroke with k MVPs as a feature element and color this sub-stroke in the line as black. We continue this process, decrementing the time duration p by 5ms in each round until either there is no white region of length greater than or equal to 30ms left in the line or p is decremented to 30. If p is decremented to 30 but there are still white regions of length greater than or equal to 30ms, we increase k by 1, reset p to be 60ms, and repeat the above process again. The last possible round is the one with $k = b$ and $p = 30$ms. The process also terminates whenever there is no white region of length greater than or equal to 30ms.

6.7 Classifier Training

In this section, we explain the internal details of GEAT on training its classifiers. After feature extraction and selection, we obtain one feature vector for each training sample of a gesture. For a single-finger gesture, the feature vector contains the values of the selected feature elements such as stroke time and the velocity magnitude, velocity direction, and device acceleration from selected sub-strokes. For a multi-finger gesture, the feature vector additionally contains the selected feature elements such as inter-stroke time, displacement magnitude, and direction between all pairs of strokes.

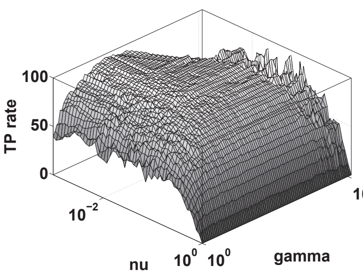
6.7.1 Partitioning the Training Sample

Before we use these N feature vectors to train our classifiers, we partition them into consistent training groups so that the user has the consistent behavior for each group for any feature element. Recall that for each feature element, we have already partitioned the N feature vectors into the least number of MVPs. For different feature elements, we may have partitioned the N feature vectors differently. Thus, we partition the N feature vectors into the least number of consistent training groups so that for each feature element, all feature vectors within a training group belong to one minimum variance partition. If the number of feature vectors in a resulting consistent training group is below a threshold, then it is not used to train classifiers.

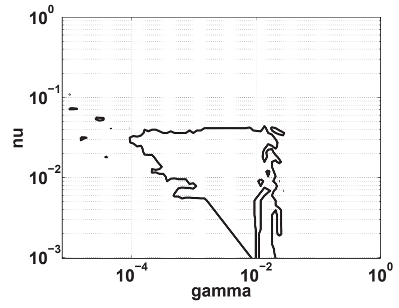
6.7.2 Training the SVDE Classifiers

In real world deployment of authentication schemes, training samples are often all from the legitimate user. When training data is only from one class (*i.e.*, the legitimate user in our scenario) while test samples can come from two classes (*i.e.*, both the legitimate user and imposters), Support Vector Distribution Estimation (SVDE) with the Radial Basis Function (RBF) kernel is effective and efficient [97, 59]. We use the open source implementation of SVDE in libSVM [38].

We build an ensemble of classifiers for each consistent training group. First, for each feature element, we normalize its N values to be in the range of $[0, 1]$; otherwise feature elements with larger values will dominate the classifier training. Second, we empirically find the appropriate values for γ, a parameter for RBF kernel, and ν, a parameter for SVDE, by performing a grid search on the ranges $2^{-17} \leq \gamma \leq 2^0$ and $2^{-10} \leq \nu \leq 2^0$ with 10-fold cross validation on each training group. As the training samples are only from one class (*i.e.*, the legitimate user), cross validation during grid search only measures the true positive rate (TPR). Figure 6.14(a) plots a surface of TPR resulting from cross validation during the grid search for a training group of a gesture for one volunteer. We see that TPR values are different



(a) TPR surface plot



(b) 95% TPR contour

Figure 6.14 Parameter selection

for different parameter values and there is a region where the TPR values are particularly high. The downside of selecting parameter values with higher TPR is that it increases the false positive rate (FPR). While selecting parameter values with lower TPR decreases the FPR, it is inconvenient for the legitimate user if he cannot successfully authenticate in several attempts. Therefore, we need to tradeoff between usability and security in selecting parameter values. We choose the highest value of TPR such that 1−TPR equals FPR, which results in the lowest EER. To calculate FPRs, GEAT needs imposter samples, which are not available in real world deployment at the time of training. Therefore, GEAT generates synthetic imposter samples by elastically deforming the samples of legitimate user using cubic B-splines and calculates the FPRs using these synthetic imposter samples. Note that these synthetic imposter samples are not used in classifier training.

Once we decide on TPR, we obtain the coordinates of the points on the contour of that TPR from the surface formed by the grid search. Figure 6.14(b) shows the 95% TPR contour on the surface in Figure 6.14(a). From the points on this contour, we randomly select z (say $z = 10$) points, where each point provides us with the parameter values of γ and ν. For each of the z pairs of parameter values of γ and ν, GEAT trains an SVDE classifier on a consistent training group. Thus, for each consistent training group, we get an ensemble of z classifiers for modeling the behavior of the legitimate user. This ensemble can now be used to classify any test sample. The decision of this ensemble of classifiers for a test sample is based on the majority voting on the decision of the z classifiers in the ensemble. Larger value of z increases the probability of achieving the TPR at which the contour was made, however, the computation required to perform authentication also increases. Therefore, we need to tradeoff between classification reliability and efficiency in choosing the value of z. We choose $z = 10$ in our experiments.

6.7.3 Classifying the Test Samples

Given a test sample of a gesture on a touch screen device, we first extract values from this test sample for the selected feature elements of the legitimate user of this device and form a feature vector. Then, we feed this feature vector to all ensembles of classifiers. If any ensemble of classifiers accepts this feature vector as legitimate, which means that this test sample gesture is similar to one of the identified behavior of the legitimate user, we accept this test sample as legitimate and skip the remaining ensembles of classifiers. If no ensemble accepts this test sample as legitimate, then this test sample is deemed as illegitimate.

6.8 Ranking and Classification

For each gesture, GEAT repeats the above three steps given in Sections 6.5, 6.6, and 6.7 and then ranks the gestures based on their EERs. The user chooses the value of n, the number of gestures with lowest EERs that the user needs to do in each authentication attempt. Although larger n is, higher accuracy GEAT has, for practical purposes, $n = 1$ (or 3 at most) gives high enough accuracy.

When a user tries to unlock, the device displays the n top ranked gestures for the user to perform. GEAT classifies each gesture input as discussed in Section 6.7.3, and uses majority voting on the n decisions to make the final decision about the legitimacy of the user.

6.9 Experimental Results

In this section, we present the results from our evaluation of GEAT. First, we report EERs from Matlab simulations on gestures in our data set. Second, we study the impact of the number of training samples on the EER of GEAT. Third, we study the impact of the threshold of cv on the EER of GEAT and justify our choice of using 0.1 as the threshold. Fourth, we report the results from real world evaluation of GEAT implemented on Windows smart phones. Last, we compare the performance of GEAT with the scheme proposed in [75]. We report our results in terms of equal error rates (EER), true positive rates (TPR), false negative rates (FNR), and false positive rates (FPR). EER is the error rate when the classifier parameters are selected such that FNR equals FPR.

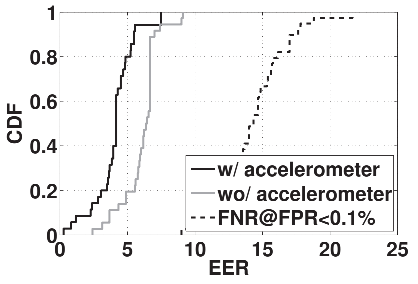
6.9.1 Accuracy Evaluation

First, we present our error rates when the number of postures b is equal to 1, which means that GEAT only looks for a single consistent behavior among all training samples. Second, we present the error rates of GEAT when $b > 1$, which means that GEAT looks for multiple consistent behaviors in training samples. We present these error rates for $n = 1$ and $n = 3$ where n is the number of gestures that the user needs to do for authentication. Recall that GEAT allows a user to choose the n top ranked gestures. Third, we present the average error rates for each of the 10 gestures. We calculated the average error rates by treating each volunteer as a legitimate user once and treating the remaining as imposters for the current legitimate user. To train SVDE classifiers on legitimate user for a given gesture, we used a set of 15 samples of that gesture from that legitimate user. For testing, we used remaining samples from the legitimate user and 5 randomly chosen samples of that gesture from each imposter. We repeated this process of training and testing on the samples of the given gesture for 10 times, each time choosing a different set of training samples. We did not use imposter samples in training.

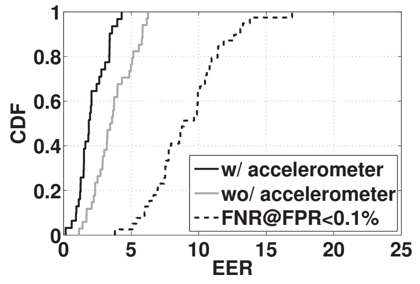
For the training samples of a gesture performed by a user, ideally, we would like to know the number of postures b in which the user performed the gesture. Knowing the value of b helps us to achieve higher classification accuracy. However, in real deployment, the value of b may not be available. In such scenarios, actually our classification accuracy is still very high. Next, we first present the evaluation results if we do not know the value of b. In such cases, we treat all training samples to be from the same posture by setting $b = 1$. Then, we present the evaluation results if we know the value of b.

6.9.1.1 Single Behavior Results

In this case, we assume $b = 1$. Figure 6.15(a) plots the cumulative distribution functions (CDFs) of the EERs of GEAT with and without accelerometers, and the FNR of GEAT when FPR is less than 0.1%, for $n = 1$. Similarly, Figure 6.15(b) shows the corresponding plots for $n = 3$. We make following two observations when device acceleration features are used in training and testing. First, the average EER of users in our data set for $n = 1$ and $n = 3$ is 4.8% and 1.7%, respectively. Second, over 80% of users have their EERs less than 4.9% and 3.4% for $n = 1$ and $n = 3$, respectively. We make following two observations when device acceleration features are not available. First, the average EER of users in our data set for $n = 1$ and $n = 3$ is 6.8% and 3.7%, respectively. That is, EER increases by 2% for both $n = 1$ and $n = 3$ when accelerometers are not available. This shows that even when accelerometers are not available, GEAT still has high classification accuracy. Second, over 80% of users have their EERs less than 6.7% and 5.2% for $n = 1$ and $n = 3$, respectively. We also observe that the average FNR is less than 14.4% and 9.2% for $n = 1$ and $n = 3$, respectively when FPR is taken to be negligibly small (*i.e.*FPR $< 0.1\%$). These CDFs show that if the parameters of the classifiers in GEAT are selected such that the legitimate user is rejected only once in 10 attempts *i.e.*, for TPR$\approx$ 90%, an imposter will almost never be accepted *i.e.*FPR$\approx$ 0%.



(a) $n = 1$

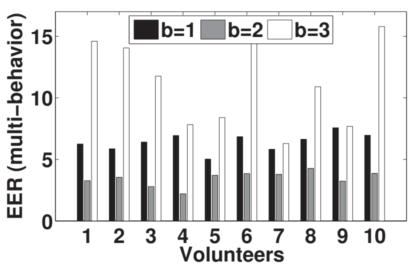


(b) $n = 3$

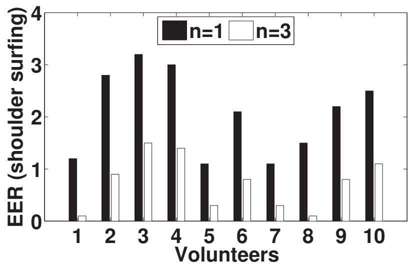
Figure 6.15 EERs with and without accelerometer and FNR at FPR $< 0.1\%$

6.9.1.2 Multiple Behaviors

Among our volunteers, we requested ten volunteers to do each of the 10 gestures in 2 postures (*i.e.*, sitting and laying down). In this case, $b = 2$. Figure 6.16(a) shows the EER for these ten volunteers for $b = 1$, 2, and 3. We see that the EER is minimum when $b = 2$ for these ten

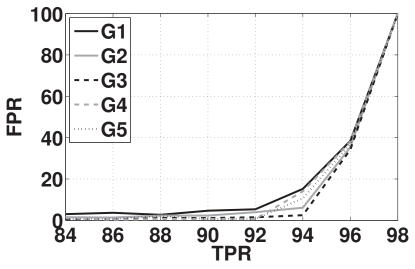


(a) EER w.r.t b

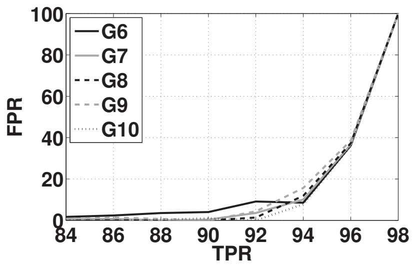


(b) Shoulder surfing

Figure 6.16 EER under different scenarios



(a) Gestures 1 to 5



(b) Gestures 6 to 1

Figure 6.17 Avg. FPR vs. TPR for all gestures

volunteers because these volunteers provided training samples of gestures in two postures. Figure 6.16(a) shows that the use of $b < 2$ results in a larger EER because it renders most of the sub-strokes inconsistent, which leaves lesser consistent information to train the classifiers. Figure 6.16(a) shows that the use of $b > 2$ results in a larger EER as well because dividing the training samples made under b postures into more than b consistent training groups reduces the training samples in each group, resulting in increased EER.

6.9.1.3 Individual Gestures

The FPR of each gesture averaged over all users is always below 5% for a TPR of 90% and decreases with the decrease in TPR. Figures 6.17(a) and 6.17(b) show the plots of FPRs vs. TPRs for each of the 10 gestures, averaged over all users. Table 6.1 shows AUC, the area under the receiver operating characteristic (ROC) curve, of all gestures for both filtered and unfiltered samples. Unfiltered samples are the samples before the noise is removed. We see that the AUC values are greater than 0.95 for most gestures. Note that an ideal classification scheme that never misclassifies any samples has AUC=1. We also see from Table 6.1 that AUC values for unfiltered gestures are slightly lower compared to AUC values for filtered gestures showing that filtering before feature extraction improves classification accuracy. We have presented both FPR and TPR for all gestures individually only to show how individual gestures perform. In real world implementation, a user will only perform n top ranked gestures, resulting in much lower FPR at much higher TPR as shown by the small values of EER in 6.15(b).

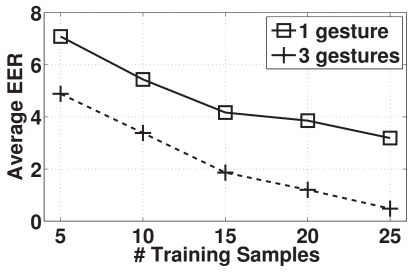
Table 6.1 AUC for filtered and unfiltered gestures

Filtered									
G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
0.94	0.96	0.95	0.95	0.95	0.96	0.96	0.96	0.96	0.96

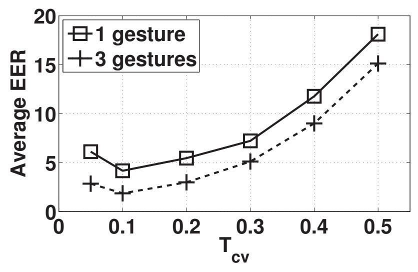
Unfiltered									
G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
0.92	0.95	0.94	0.93	0.94	0.95	0.94	0.95	0.95	0.94

6.9.2 Impact of Training Samples Size

The EER decreases with the increase in the number of training samples. Figure 6.18(a) plots the EERs averaged over all users for $n = 1$ and $n = 3$ for the increasing number of training samples. For $n = 1$ and $n = 3$, average EER falls to 3.2% and 0.5%, respectively, with just 25 training samples. An EER of 0.5% means TPR=99.5% and FPR=0.5%, which are very good results for classification schemes. A user can achieve these rates by providing only 25 training samples for each gesture. Providing more training samples over time further lowers the EER.



(a) # of training samples



(b) Effect of T_{cv}

Figure 6.18 Effect of system parameters on EER

6.9.3 Determining Threshold for cv

The average EER is a convex function in terms of the threshold of cv, denoted by T_{cv}. On one hand, if T_{cv} is too small, then it is difficult to find sub-strokes in which the user has consistent behavior, which gives us less information for classifier training. On the other hand, if T_{cv} is too large, then the feature elements with less consistent behavior will be selected, which adds noise in the user behavior models. Figure 6.18(b) shows the average EER for $n = 1$ and $n = 3$. We see that the average EER is the smallest for $T_{cv} = 0.1$.

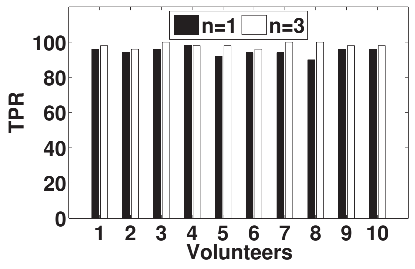
6.9.4 Real-world Evaluation

We evaluated GEAT on two sets of 10 volunteers each in real-world settings by implementing it on Samsung Focus running Windows. We used the first set to evaluate GEAT's resilience to attacks by imposters that have not observed the legitimate users while doing the gestures. We used the second set to evaluate GEAT's resilience to shoulder surfing attack, where imposters have observed the legitimate users while doing the gestures.

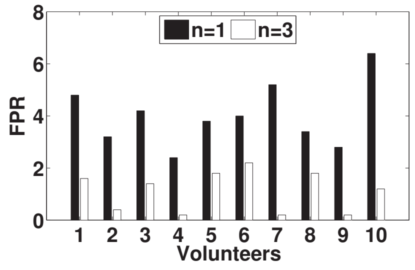
6.9.4.1 Non-shoulder Surfing Attack

In this case, our implementation requests the user to provide training samples for all gestures and trains GEAT on those samples. We asked each volunteer in the first set to provide at least 15 training samples for each gesture. GEAT also asks the user to select a value of n.

We used $n = 1$ and 3 in our experiments. Once trained, we asked the legitimate user to do his n top ranked gestures ten times and recorded the authentication decisions to calculate TPR. After this, we randomly picked 5 out of 9 remaining volunteers to act as imposters and did not show them how the legitimate user does the gestures. We asked each imposer to do the same top n ranked gestures, and recorded the authentication decisions to calculate FPR. We repeated this process for each volunteer by asking him to act as the legitimate user once. Furthermore, we repeated this entire process for all ten volunteers five times on five different days. The average (TPR, FPR) over all volunteers for $n = 1$ and $n = 3$ turned out



(a) TPR for $n = 1, 3$



(b) FPR for $n = 1, 3$

Figure 6.19 Real world results of GEAT

to be (94.6%, 4.02%) and (98.2%, 1.1%), respectively. Figures 6.19(a) and 6.19(b) show the bar plots of TPR and FPR of each of the 10 volunteers for $n = 1$ and 3, respectively.

6.9.4.2 Shoulder Surfing Attack

For this scenario, we made a video of a legitimate user doing all gestures on the touch screen of our Samsung Focus phone and showed this video to each of the 10 volunteers in the second set. The volunteers were allowed to watch the video as many times as they wanted and then requested them to perform each gesture ten times. The average FPR over all 10 volunteers turned out to be 0% for $n = 1$ as well as $n = 3$ when we set the TPR at 80%. The average EER over all volunteers for $n = 1$ and $n = 3$ turned out to be only 2.1% and 0.7%, respectively. These results show that GEAT is very resilient to shoulder surfing attack. Figure 6.16(b) shows the bar plots of EER for the 10 volunteers in second set for $n = 1, 3$.

6.9.5 Comparison with Existing Schemes

We compared the performance of GEAT with the only work in this direction reported in [75] where Luca *et al.* used the following four gestures: swipe left with one finger, swipe down with one finger, swipe down with two fingers, and swipe diagonally up from bottom left of the screen to top right. The highest FPR, when TPR= 93%, that they achieved is 43%, which is way higher than our average FPR of 4.77% at TPR of 95.23%. For a fair comparison, we also collected data for these 4 gestures from 45 volunteers and calculated the value of FPR at the TPRs reported in [75]. Table 6.2 reports the FPR achieved by GEAT and the scheme in [75]. We see that the FPRs of GEAT on these gestures are at least 4.66 times lesser than the corresponding FPRs in [75] for the TPRs used in [75]. We do not use these 4 gestures because their average EERs are larger compared to the average EERs of the 10 gestures we have proposed.

Table 6.2 Comparison of GEAT with [75]

	TPR	FPR	
		Luca *et al.* [75]	GEAT
Swipe left	85.11	48	5.12
Swipe down–1 finger	95.71	50	10.71
Swipe down–2 fingers	89.58	63	8.12
Swipe diagonal	90.71	43	8.01

6.10 Conclusions

In this chapter, we propose a gesture based user authentication scheme for the secure unlocking of touch screen devices. Compared with existing passwords/PINs/ patterns based schemes, GEAT improves both the security and usability of such devices because it is not vulnerable to shoulder surfing attacks and smudge attacks and at the same time gestures are easier to input than passwords and PINs. Our scheme GEAT builds single-class classifiers using only training samples from legitimate users. We identified seven types of features (namely velocity magnitude, device acceleration, stroke time, inter-stroke time, stroke displacement magnitude, stroke displacement direction, and velocity direction). We proposed algorithms to model multiple behaviors of a user in performing each gesture. We implemented GEAT on real smart phones and conducted real-world experiments. Experimental results show that GEAT achieves an average equal error rate of 0.5% with 3 gestures using only 25 training samples.

7 Software Security

7.1 Introduction

In computer software, a vulnerability is a loophole in the software code that enables an attacker to circumvent the deployed security measures [99]. Each software vulnerability has a life cycle that consists of distinct phases characterized by the events of its discovery, disclosure, exploitation, and patching. Each phase has a certain level of risk associated with it. The first phase of the life cycle of a vulnerability starts when it is discovered by the vendor, a hacker, or any third-party software analyst. The security risk associated with a vulnerability is particularly high if it is first discovered by hackers. The next phase starts with the public disclosure of the vulnerability, which can again be done by the vendor, a hacker, or any third-party software analyst. After disclosure, the information about a vulnerability is freely available to everyone; therefore, the level of security risk increases further because the hacker community is active in developing and releasing *zero-day exploits* [27]. The aim of the vendor is to release a *patch* for the vulnerability as soon as possible. It is noteworthy that many users of the affected software do not instantly install the patch released to fix the vulnerability. The life cycle of a vulnerability ends when all users of a software install the patch to fix the vulnerability. A vulnerability can be exploited by hackers at any time during its entire life cycle.

The exploratory analysis of vulnerability life cycles can uncover interesting patterns for vendors and software products that are helpful in following ways: First, a thorough analysis is helpful in the deployment of best practices in the software development processes. Second, such analysis is useful to develop the security policies that can handle future attacks and threats more effectively. Third, an exploratory analysis provides insights about the previous security incidents that are helpful in their audit. Finally, it also helps customers to assess the security risks associated with the software products of a particular vendor.

To the best of our knowledge, no previous work has been done to analyze the evolution of life cycle of different types of vulnerabilities for different software products and vendors. The only work in this direction was reported by Frei *et al.* [49, 50]. In [49], Frei *et al.* studied the performance of the software industry as a whole but did not characterize the behavior of individual vendors. In [50], the authors only compared the vulnerability handling process of two vendors and based their analysis on a small data set. Some researchers have focused on the modeling of vulnerability discovery process [27, 24, 92]. The goal of such work is to estimate the number of vulnerabilities in new software products. Another direction of work aims to study the changes in the patching behavior of vendors in response to vulnerability disclosures and the existence of competitors [29, 28]. These studies analyze only small vulnerability data sets and do not cover the behavior of individual vendors.

In this chapter we make following three contributions. (1) We have aggregated a large software vulnerability data set from three vulnerability repositories: (a) National Vulnerability Database (NVD) [11], (b) Open Source Vulnerability Database (OSVDB) [16], and (c) the vulnerability data collected by Frei *et al.* (FVDB) [49]. Our aggregated software vulnerability data set contains 46310 vulnerabilities since 1988 to 2011. (2) We have comprehensively analyzed software vulnerabilities along the seven dimensions mentioned in the abstract. Our observations are supported by statistical tests for significance. (3) To systematically analyze patterns in our vulnerability data set, we have utilized association rule mining to extract rules that represent exploitation behavior of hackers and the patching behavior of vendors.

The rest of the chapter is organized as: Section 7.2 explains the terminology and notations used in the chapter and provides details about our vulnerability collection process and the aggregated data set. In Section 7.3, we analyze the evolution of vulnerability disclosure rates, access methodology for vulnerability exploitation, impact of the exploitation, risk associated with vulnerabilities and evolution of different types of vulnerabilities. In Sections 7.4 and 7.5, we study the exploitation and patching behavior of hackers and vendors respectively. In Section 7.6, we cross examine the exploitation behavior of hackers and the patching behavior of vendors. In Section 7.7, we present the implications of our work followed by the related work and conclusion.

7.2 Preliminaries

In this section, we first explain the terms and notations used in rest of the chapter and then present the data set used for analysis.

7.2.1 Terminology and Notations

Vendor is an entity (an individual, a group of individuals, or an organization) that develops a software product and is responsible to keep it secure. An ideal vendor would discover and patch all the vulnerabilities in its products before they are exploited.

Hacker is an entity that releases exploits for the vulnerabilities in the software products.

Independent organization is an entity that independently discovers and discloses vulnerabilities as well as their corresponding exploits and patches but is not involved in the development of patches or exploits.

Disclosure Date (t_d) refers to the date when information about a vulnerability is made publicly available after establishing that the vulnerability poses a potential risk.

Patch Date (t_p) is the date when a vendor provides a solution (*i.e.*patch) for a vulnerability to neutralize the threat posed by it. We consider only those patches that are released by the corresponding vendor.

Exploit Date (t_e) is the earliest date when a vulnerability is exploited. An exploit can be in the form of an automatic script, a virus, a tool, or any such thing that can breach the security of a software.

Exploit – Disclosure (t_{ed}) is the duration (in days) between the date an exploit for a given vulnerability was provided by hackers and the date the vulnerability was disclosed.

Patch – Disclosure (t_{pd}) is the duration (in days) between the date a patch for a vulnerability was released by the vendor and the date the vulnerability was disclosed.

Patch – Exploit (t_{pe}) represents the duration (in days) between the dates of availability of a patch and an exploit for a given vulnerability.

Risk Score is assigned to a vulnerability by Common Vulnerability Scoring System (CVSS) [9] and establishes the magnitude of risk associated with that vulnerability. We divide vulnerabilities into three categories of *low*, *medium*, and *high* risk severity based on their CVSS scores.

Access Vector (AV $\in$ {Local, Adjacent Network, Network}) indicates if local or network access to the hardware is required to exploit the vulnerability.

Access Complexity (AC $\in$ {Low, Medium, High}) is a measure of the complexity of the attack required to exploit the vulnerability.

Integrity Impact (Ii $\in$ {None, Partial, Complete}) measures the potential impact of a successfully exploited vulnerability on the integrity of the system. Integrity refers to the trustworthiness of information.

7.2.2 Data Set

In this section, we provide details of our data aggregation process and the basic statistics of the data. We provide details about the selection criteria of vendors and products for our study. We have collected vulnerability information from three sources: (1) NVD [11], (2) OSVDB [16], and (3) FVDB [49].

7.2.2.1 Data Aggregation

NVD and FVDB identify each vulnerability with Common Vulnerability and Exposures Identifier (CVE-ID) [6]. OSVDB also provides CVE-IDs of about 70% of vulnerabilities. We leverage the CVE-IDs to aggregate the vulnerability data from the three sources. We take CVSS scores, CVSS vectors, vendor and product names, text description, and disclosure dates from NVD. From OSVDB and FVDB, we take disclosure dates, exploit dates, and patch dates.

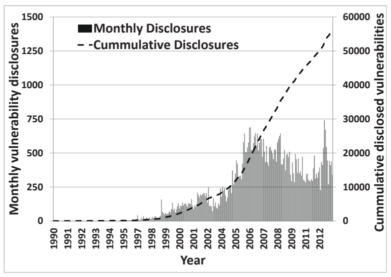
The total number of vulnerabilities in our aggregate data set are 46310 and the number of vulnerabilities for which disclosure dates, patch dates, and exploit dates are available are 46310, 9667, and 15456 respectively. We do not have exploit dates and patch dates for all the vulnerabilities in our aggregate data set. Due to the shear size of the data set, it is not feasible to find them manually. To systematically conduct our study, we divide our aggregate data set into following three subsets:

ED-subset consists of 15456 vulnerabilities and contains those vulnerabilities for which both exploit and disclosure dates are known. *PD-subset* consists of 9667 vulnerabilities and contains those vulnerabilities for which we have both patch and disclosure dates. *PE-subset* consists of 1424 vulnerabilities and contains those vulnerabilities for which both patch and exploit dates are known.

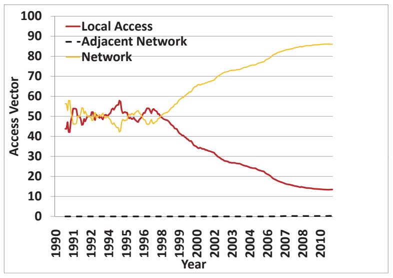
7.2.2.2 Selection of Vendors and Products

The aggregate data set contains vulnerabilities from more than 11 thousand vendors and over 17 thousand software products. Figure 7.2 plots the number of vulnerabilities of each vendor in the descending order. It can be seen that over 95% of the vendors have less than 10 vulnerabilities. Therefore, to make statistically sound observations, we focus our attention only on the top 8 vendors each of which has at least 500 vulnerabilities. For our study, we select Microsoft, Apple, Sun, Oracle, Linux[1], Mozilla, Red Hat, and Google. We also study

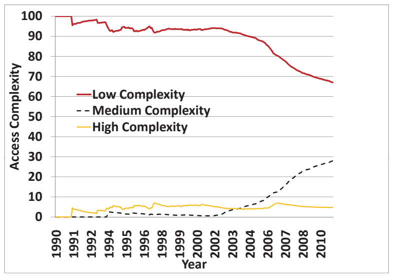
[1]Linux is not a vendor. It only represents the vulnerabilities in Linux kernel.



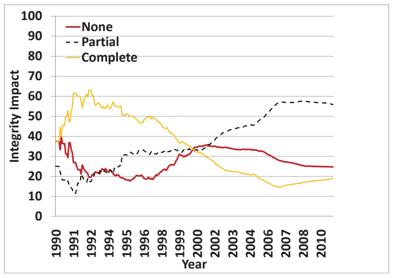
(a) Vulnerability disclosure trend



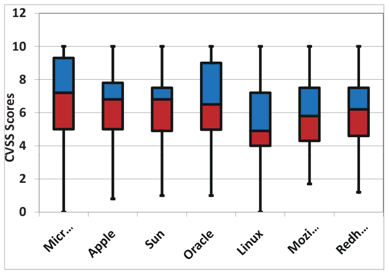
(b) Access Vector Evolution



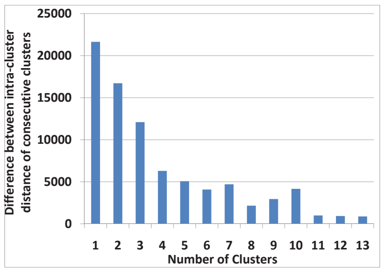
(c) Access Complexity Evolution



(d) Integrity Impact Evolution



(e) Boxplots of the CVSS scores of selected vendors



(f) Difference between intra-cluster dissimilarity of consecutive clusters

Figure 7.1 Vulnerability trends in the data set

popular software products of these vendors that include Internet Explorer, Safari, Firefox, Chrome, Windows, MAC OS X, Solaris, and several Linux based operating systems.

7.3 General Vulnerability Analysis

In this section, we study the trends in vulnerability disclosure and CVSS-vector metrics (*i.e.*, access vector, access complexity, and integrity impact) over the past 2 decades. We also categorize the vulnerabilities into groups and study their evolution.

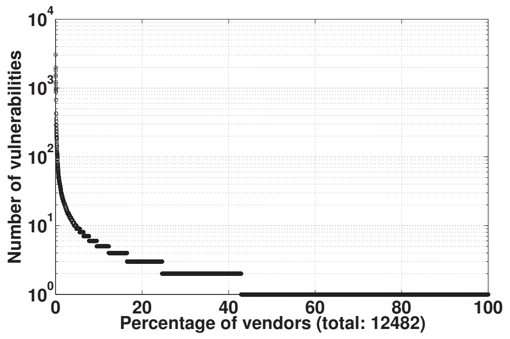


Figure 7.2 # of vulnerabilities for each vendor (in descending order

7.3.1 Vulnerability Disclosure Trend

The rate of vulnerability disclosures experienced an exponential growth since 1997 and lasted till 2006 as can be seen in Figure 7.1(a). The vertical lines in the figure show the number of vulnerabilities disclosed every month since January 1990 and the dashed line shows the cumulative number of vulnerabilities. The number of vulnerability disclosures has not been increasing since 2006. In fact, on average, the number of vulnerabilities being disclosed every month have been decreasing since 2008 despite the ever increasing use of software products.

7.3.2 Evolution of CVSS-Vector Metrics

Figures 7.1(b) to 7.1(d) show the evolution of three metrics of CVSS-vector. For each metric, we have calculated the percentage of vulnerabilities corresponding to each of its three values for every month since January 1990. We observe from Figure 7.1(b) that the percentage of remotely exploitable vulnerabilities has been increasing since 1998. The fact that most computer systems are connected to Internet has made it possible for hackers to exploit these systems remotely. Figure 7.1(c) shows the change in access complexity of vulnerabilities over the years. We observe that the percentage of low complexity vulnerabilities has decreased over time indicating that the hackers have to use more sophisticated techniques to exploit new vulnerabilities. From Figure 7.1(d), we also observe a reduction in the percentage of vulnerabilities having complete integrity impact.

7.3.3 General Trend of CVSS Score for Short-listed Vendors

Recall from Section 7.2 that every vulnerability has an associated risk quantified by CVSS score. Figure 7.1(e) shows the box plots of CVSS scores for vulnerabilities in the products of the selected vendors. We note that CVSS scores of most vulnerabilities in our study lie in *medium* to *high* range. The median CVSS scores for closed-source vendors are greater than the median scores for open-source vendors.

7.3.4 Evolution of Types of Vulnerabilities

To determine the prevalent types of vulnerabilities and to study their evolution, we utilize unsupervised k-means clustering to group different types of vulnerabilities. We leverage the text information provided by NVD and OSVDB for each vulnerability to cluster them into groups of distinct types. We extracted the keywords from the text description of each vulnerability that characterize its functionality and used them as features to cluster all the vulnerabilities into groups. Some example keywords include *denial, service, buffer, injection etc.*We had a total of 608 relevant keywords.

It is well known that k-means clustering algorithm is well suited for large data sets with large number of attributes. To set an appropriate value of k in k-means algorithm, we used *Euclidean distance* as the intra-cluster dissimilarity metric due to the binary nature of the attributes [119]. Figure 7.1(f) shows the difference in the intra-cluster dissimilarity between consecutive clusters. It can be seen that the distance decreases as the number of clusters increases for lower values of k. The bar above any value x in Figure 7.1(f) represents the difference between intra-cluster distances of x and $x + 1$ clusters. Note that increasing the number of clusters to 8 increases the intra-cluster distance (the bar above 6 is smaller than that above 7). Therefore, the optimum value of k is 7. For statistical rigor, we repeated k-means clustering algorithm 20 times with different seeds for each value of k. The coefficient of variation in each case was less than 0.05 which shows the statistical significance of results. We analyzed the centroids of clusters to determine their dominant keywords. Table 7.1

tabulates dominant keywords for each centroid. From the observed keywords, we label the vulnerability clusters as PHP vulnerabilities (PHP), executable code (EXE), denial of service (DoS), buffer overflow (BO), SQL injection (SQL), cross-site scripting (XSS), and miscellaneous vulnerabilities (Misc).

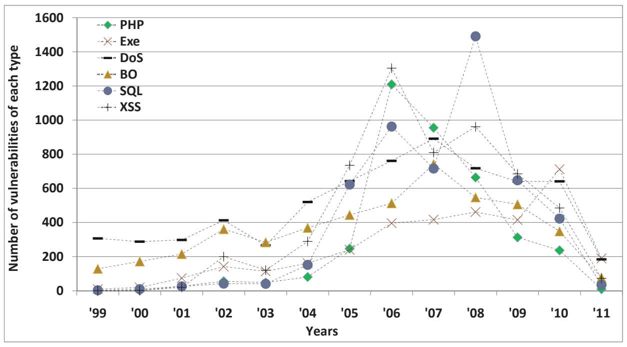


Figure 7.3 Evolution of vulnerability clusters over the years

Figure 7.3 shows the number of vulnerabilities belonging to each cluster disclosed since 1999.

Only BO, DoS, and EXE vulnerabilities were prevalent till 2001. These types of vulnerabilities constitute a major portion of software vulnerabilities even today which indicates that the vendors have not been able to devise effective strategies to limit these types of vulnerabilities. Since 2002, we observe an increase in the XSS vulnerabilities, which peak in 2006. PHP vulnerabilities were prevalent in 2006 and 2007 and SQL vulnerabilities became dominant since 2005. These trends highlight the shift in focus of hackers to exploit new

Table 7.1 Results of vulnerability clustering

C#	Keywords	Label	Size
1	php, parameter, execute, file, code, url	PHP	8.32%
2	–	MISC	36.6%
3	execute, code	EXE	7.25%
4	service, denial	DoS	14.2%
5	buffer, execute, code, overflow	BO	10.2%
6	injection, sql, execute, commands	SQL	11.2%
7	cross, scripting, site, script, html, inject	XSS	12.3%

services as they become popular.

In the sections that follow, we present the behavior of hackers and vendors towards vulnerabilities.

7.4 Exploitation Behavior

In this section, we study the behavior of hackers in releasing exploits for vulnerabilities. For this, we analyze trends in t_{ed} values of vulnerabilities. The analysis presented in this section is done on *ED-subset*. We study three ranges of t_{ed} values.

$\mathbf{t_{ed} < 0}$ shows that an exploit for a given vulnerability was released before its public disclosure. The vulnerabilities falling in this range represent a big threat to the security of end-users as the vendor could be oblivious about them. A total of 2.8% software vulnerabilities fall into this range.

$\mathbf{t_{ed} = 0}$ refers to the case when an exploit for a given vulnerability was released on the day it was disclosed. The exploits corresponding to such vulnerabilities are called *zero-day* exploits. In our ED-subset, a total of 88.2% vulnerabilities have zero-day exploits.

$\mathbf{t_{ed} > 0}$ means that the exploit for a vulnerability was released after its public disclosure. The vulnerabilities for which $t_{ed} > 0$ represent the case where a vulnerability is disclosed by the vendor or an independent organization and the hackers used this information to release an exploit in more than a day. 9.7% vulnerabilities fall in this range. To do more detailed analysis, we subdivide this range into three parts: (1) $0 < t_{ed} \leq 7$ gives us the percentage of exploits released within a week of disclosure, (2) $7 < t_{ed} \leq 30$ gives us the percentage of exploits released after a week and within a month of disclosure, and (3) $t_{ed} > 30$ gives us the percentage of exploits released a month after the disclosure.

7.4.1 Evolution of Exploitation

To extract and construe the dominant trends, we first divided the vulnerabilities in ED-subset into *groups* where each group contains vulnerabilities disclosed in one distinct year. Then we subdivided the vulnerabilities in each group into five *subgroups* corresponding to the five ranges of t_{ed}. We then calculated the percentage of vulnerabilities in each subgroup (called the *percentage size* of the subgroup) in its respective group and plotted the results in Figure 7.4 in the form of stacked bars where each bar corresponds to the group of vulnerabilities disclosed each year and each block in every bar represents the percentage size of the corresponding subgroup in its respective group. The number inside each block is the value of the percentage size of the corresponding subgroup. The number at the top of each bar represents the total number of vulnerabilities in the corresponding group. All figures in rest of the chapter have been made using similar methodology.

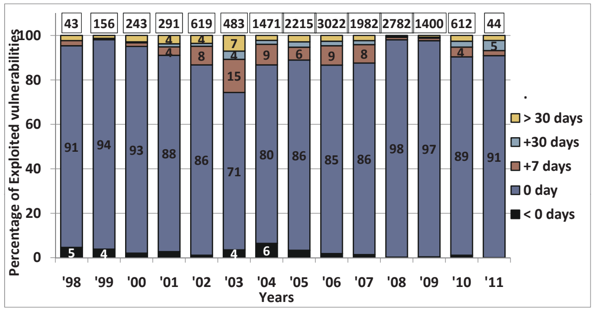


Figure 7.4 Yearly change in exploitation behavior for different t_{ed} ranges

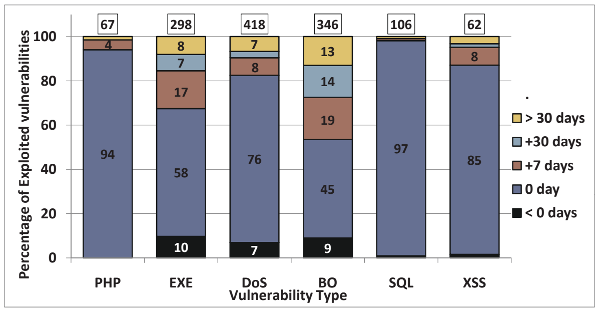


Figure 7.5 Exploitation trend in clusters

It can be seen from Figure 7.4 that majority of vulnerabilities have always been exploited on their disclosure dates (having $t_{ed} = 0$). Till 2004, the percentage size of the subgroup of $t_{ed} < 0$ was non-negligible which shows that the hackers were finding a significant number of vulnerabilities themselves and exploiting them. At the same time we observe a decrease in the percentage size of the subgroup of $t_{ed} = 0$. This does not mean that hackers were getting sluggish because we also observe a significant increase in the total number of exploited vulnerabilities. Since 2004, although we observe a decrease in the percentage size of the subgroup of $t_{ed} < 0$, an increasing trend in the percentage size of subgroup of $t_{ed} = 0$ still shows that the hackers are becoming more and more active.

7.4.2 Exploitation of Types of Vulnerability

We now see the exploitation of different types of vulnerabilities. Figure 7.5 has been made in the same way as Figure 7.4 except that now the *groups* are the *types* of vulnerabilities. It can be seen that over 80% of vulnerabilities of each type (except BO and EXE) are exploited on or before the day of disclosure. In case of BO and EXE, a significant percentage of vulnerabilities is exploited several weeks after the disclosure. According to our data set, 79% of BO and EXE pose *high risk* and only 7% have *high access complexity*, so intuitively, they should attract more attention from hackers. The total number of exploited vulnerabilities of these two types are large which justifies the intuition.

7.4.3 Exploitation Trend for Vendors and Products

We study the behavior of hackers in exploiting the vulnerabilities for different vendors and their respective products. Figures 7.6 and 7.7 show the exploit data for the selected vendors and products respectively. These figures have been made for vendors and products in the same way as Figure 7.5 was made for vulnerability types.

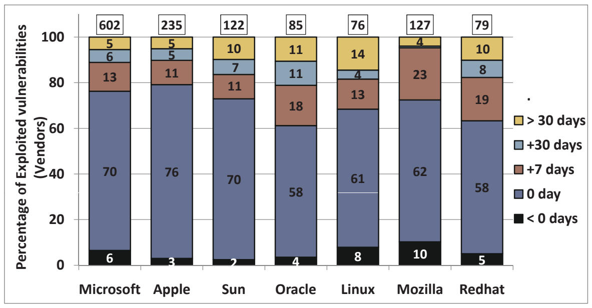


Figure 7.6 Exploited vulnerabilities for vendors relative to disclosure dates

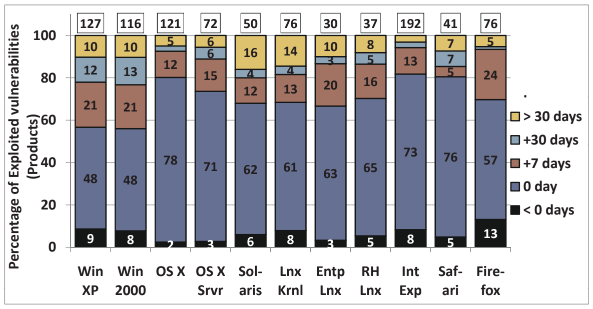


Figure 7.7 Exploited vulnerabilities for products relative to disclosure dates

Lets first compare the vulnerability exploitation in open vs. closed-source vendors. In comparison to closed-source vendors, for open-source vendors *e.g.*, Linux, Red Hat *etc.*, comparatively smaller percentage of vulnerabilities is exploited till the day of disclosure while a larger percentage of vulnerabilities is exploited before the disclosure. To generate statistically significant conclusion from these two conflicting observations, we do statistical hypothesis testing.

As our samples for open-source and closed-source vendors contain large number of data points, therefore, the most appropriate statistical test for this scenario (and all the subsequent scenarios) is the standard one-tailed t-test. t-test is considered to be the most appropriate when the number of data points in the samples are large (typically > 50) regardless of the distributions they come from.

To remove any bias in testing, we state the null hypothesis as: the mean value of t_{ed} for open-source vendors, $\mu_{t_{ed}}(O)$, is equal to the mean value for closed source vendors, $\mu_{t_{ed}}(C)$. The alternative hypothesis is: $\mu_{t_{ed}}(C)$ is greater than $\mu_{t_{ed}}(O)$. We apply the right tailed t-test to the null hypothesis. If the null hypothesis is rejected, it would be statistically sound to claim that the average time to exploit a vulnerability in closed-source software is larger compared to open-source software. We give a general equation for hypothesis testing that will be used for all the subsequent tests:

$$\begin{aligned} &H_0 : \mu_A(X) = \mu_B(Y) \\ &H_1 : \mu_A(X) > \mu_B(Y) \end{aligned} \tag{7.1}$$

where $X = C$ represents closed-source vendors, $Y = O$ represents open-source vendors, and $A = B = t_{ed}$ represents that the data points of t_{ed} are being considered. We do the hypothesis testing for a 95% confidence interval *i.e.*, $\alpha = 0.05$. Our test resulted in a p-value of 0.003 which is much smaller than α, thus we reject H_0 to accept H_1. Therefore, it is statistically sound to state that the exploitation of vulnerabilities in closed-source software is slower compared to open-source software.

Figure 7.6 shows that hackers release most exploits till the disclosure dates for Microsoft and Apple. This is primarily because hackers find it more rewarding to exploit these products due to their wider market capitalization. For the selected products, we see the similar trend in Figure 7.7 as for vendors in Figure 7.6 except for Windows. The percentage of exploited vulnerabilities for Windows till disclosure date is lesser as compared to OS X but at the same time the percentage of exploited vulnerabilities for Windows before disclosure is greater than that for OS X. In fact, the mean value of t_{ed} for Windows is negative while that for OS X is positive. The t-test with $X =$ "OS X", $Y =$ "Windows", and $A = B = t_{ed}$ yields $p = 0.031$ proving that the exploitation in Windows is quicker compared to OS X.

Among web browsers, Firefox has the smallest percentage of vulnerabilities exploited till disclosure date compared to Internet Explorer and Safari but at the same time has the highest percentage of vulnerabilities exploited before the disclosure. The t-test with $X =$ "Safari" and $Y =$ "Internet Explorer" yields $p = 0.05$ showing that exploitation in Internet Explorer is quicker compared to Safari. The t-test with $X =$ "Safari" and $Y =$ "Firefox" yields $p = 0.09$, and therefore, fails to reject the null hypothesis.

7.4.4 Exploitation Behavior: CVSS Scores

Recall from Section 7.2 that each vulnerability is assigned a CVSS score depending upon the level of risk associated with it. Based on CVSS scores, we divide vulnerabilities into three categories. *Low:* $0 \leq$ CVSS Score < 4; *Medium:* $4 \leq$ CVSS Score < 7; *High:* $7 \leq$ CVSS Score ≤ 10. Figure 7.8 has been generated in the same way as Figure 7.6 except that we plotted the vulnerabilities belonging to low, medium, and high categories separately. The white lines with round markers represent the percentage of total vulnerabilities belonging to low, medium, or high categories.

It is intuitive to think that hackers would be less interested in exploiting low risk vulnerabilities because such vulnerabilities usually cause lesser damage. This is exactly what the markers for low risk vulnerabilities show in Figure 7.8. The bars in Figure 7.8 show that the

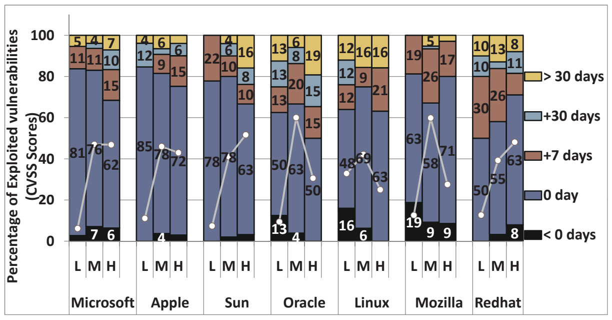


Figure 7.8 Exploited vulnerabilities for different CVSS scores

percentage of medium risk vulnerabilities for which exploits are released till the disclosure date is greater than that for high risk vulnerabilities for all closed-source vendors and some open-source vendors.

7.4.5 Interesting Exploitation Rules

Now we present some interesting association rules about the exploitation behavior in the products of the short-listed vendors. We used implementation of Apriori association rule mining algorithm in WEKA to extract the rules with confidence greater than 95% [23, 124]. For association rule mining, we used following 7 attributes of each vulnerability: Vendor Name <`vnd`>, Product Name <`prd`>, Vulnerability Type <`typ`>, Severity <`sev`>, t_{ed}, t_{pd}, and t_{pe}. For the rules presented in this section, we used t_{ed} as class attribute.

We found that in case of Microsoft, majority of vulnerabilities including DoS, XSS, and BO are exploited on the day they are disclosed. One such rule obtained from association rule mining is: `vnd=Microsoft typ=XSS sev=H` $\rightarrow$ t_{ed}`=0-day`.

In case of Apple, the vulnerabilities are exploited on or before their disclosure date. For example, as shown in the following rule, vulnerabilities in Safari browser are mostly exploited on the day of disclosure: `vnd=Apple prod=Safari typ=BO sev=H` $\rightarrow$ t_{ed}`=0-day`.

For Solaris, association rules show that high risk vulnerabilities are exploited on the day of disclosure while medium risk vulnerabilities are mostly exploited within a week after their disclosure. The latter trend is shown by the following rule: `vnd=Sun prod=Solaris sev=M`

$\rightarrow$ `0<`t_{ed}`≤ +1 week`.

For Mozilla, we get interesting rules showing that hackers do not exploit a vulnerability that has already been patched while they quickly exploit those that have not been patched. Two rules stating this observation are: (1) `vnd=Mozilla prod=Firefox typ=BO` t_{pd}`=0-day` $\rightarrow$ t_{ed}`> +1 month`, (2) `vnd = Mozilla prod=Firefox typ=BO +1 week <`t_{pd}`≤+1 month` $\rightarrow$ t_{ed}`=0-day`.

7.5 Patching Behavior

Now we study the behavior of vendors in providing patches for vulnerabilities in their products. For this, we study the trends in t_{pd} values of vulnerabilities. The analysis presented in this section is based upon *PD-subset*. The three ranges for t_{pd} that we study are described below.

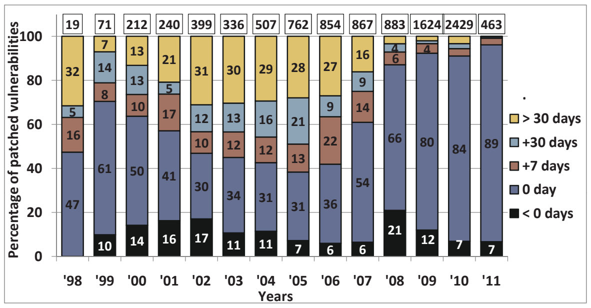


Figure 7.9 Yearly change in the patching behavior for different t_{pd} ranges

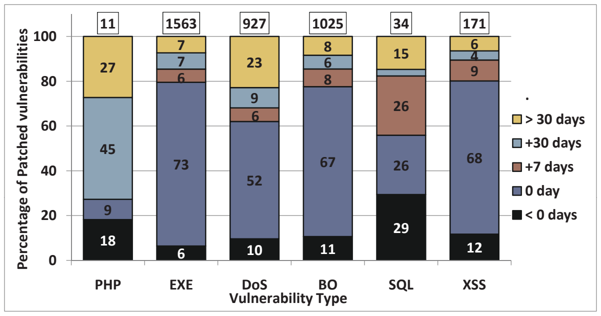


Figure 7.10 Patching trend in clusters

$\mathbf{t_{pd} < 0}$ shows that the patch for a given vulnerability was released before its public disclosure. A total of 10.1% vulnerabilities have $t_{pd} < 0$ which is greater than the corresponding value for $t_{ed} < 0$. One possible reason is that the independent organizations inform the vendors about the vulnerabilities they discover and give them a reasonable time to release a patch before disclosing the vulnerabilities.

$\mathbf{t_{pd} = 0}$ means that the patch for a vulnerability was released on the disclosure day. Such patches provide *zero-day* protection against exploitation. In our data set, zero-day patches are provided for 62.2% of the vulnerabilities.

$\mathbf{t_{pd} > 0}$ refers to the case where the patch for a given vulnerability was released after its public disclosure. In our PD-subset, 27.7% of all the vulnerabilities are patched more than a day after their disclosures. We further subdivide the range $t_{pd} > 0$ into the same three parts as in Section 7.4.

The t-test with $A = t_{pd}$, $B = t_{ed}$, and $X = Y =$ "aggregate data set" yields $p \approx 0$ which leads us to accepting the alternative hypothesis that, compared to hackers, vendors take more time on average to patch a vulnerability (considering disclosure date as reference).

7.5.1 Evolution of Patching Behavior

In Figure 7.9 we observe that till 2005, the percentage of vulnerabilities patched on or before disclosure dates consistently decreased. Keeping in view the fact that independent organizations inform the vendors about vulnerabilities well before disclosing them, such a poor patching behavior of vendors indicates that security was not a major concern for vendors at that time. However, we see a significant improvement after 2005. Since 2008, vendors have been providing patches for more than 80% of total vulnerabilities till their disclosure dates. A possible reason for this can be that it has become more common to not report vulnerabilities publicly, rather, the vendors "pay" for vulnerabilities.

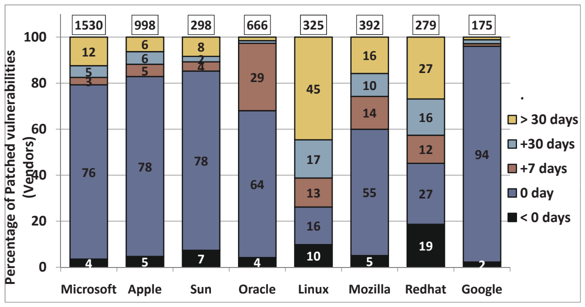


Figure 7.11 Patched vulnerabilities for vendors relative to disclosure dates

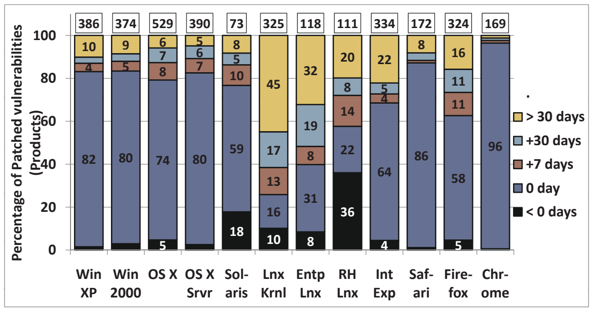


Figure 7.12 Patched vulnerabilities for products relative to disclosure dates

7.5.2 Patching of Types of Vulnerabilities

From Figure 7.10, we can note that the vendors are generally slower in patching the PHP and SQL vulnerabilities. Recall from Section 7.4.2 that hackers tend to quickly exploit these groups of vulnerabilities. On the other hand, the vendors are quicker in patching the EXE and BO vulnerabilities because these vulnerabilities are quickly exploited and thus pose high security risk.

7.5.3 Patching Trend for Vendors and Products

Here we study the behavior of the selected vendors in patching the vulnerabilities in their products. Figures 7.11 and 7.12 show the patch data for selected vendors and products.

Closed-source vendors are typically profit based organizations and have more resources to secure their products as compared to open-source vendors. Therefore, we expect better patching behavior from closed-source vendors. Figure 7.11 confirms this intuition as Microsoft, Apple, and Oracle release patches for about 70% or more of all the vulnerabilities on or before disclosure dates. In comparison, we observe significantly smaller percentages and quantity of patched vulnerabilities for open-source vendors. Applying the t-test with $X = O$, $Y = C$, and $A = B = t_{pd}$, we obtained $p \approx 0$ which statistically justifies the observation that open source-vendors are slower in patching as compared to closed-source vendors.

We see the similar trend for the selected products in Figure 7.12 as for vendors in Figure 7.11. We also see that over 85% of the vulnerabilities in Windows are patched on or before the disclosure dates. If we compare Figure 7.12 with Figure 7.7, we observe that the percentage of zero-day patches for Windows is greater than the percentage of zero-day exploits.

Among web browsers, Figure 7.12 shows that Google Chrome is the fastest patched web browser followed by Apple's Safari. t-test with $Y =$ Chrome and $X =$ (Internet Explorer, Safari, Firefox) respectively yields $p = (0, 0.024, 0)$ confirming that our observation about Chrome from Figure 7.12 is statistically significant. t-test with $Y =$ Safari and $X =$ (Internet Explorer, Firefox) yields $p = (0.009, 0.078)$ confirming Safari is patched quicker compared to

Internet Explorer but the test fails to reject the null hypothesis of Safari against Firefox.

7.5.4 Patching Behavior: CVSS Scores

One would expect the vendors to be quicker in patching the medium and high risk vulnerabilities compared to low risk vulnerabilities. This is exactly what we observe in Figure 7.13. Open-source vendors are slower as compared to closed-source vendors for vulnerabilities belonging to all risk categories.

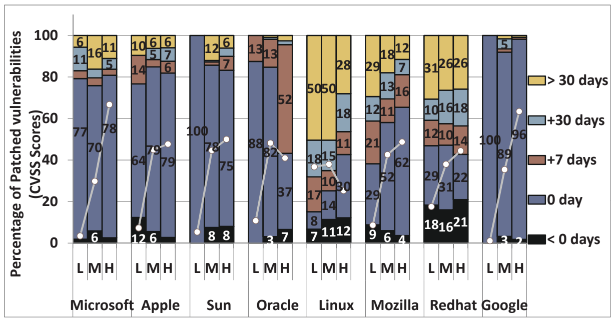


Figure 7.13 Patched vulnerabilities for different CVSS scores

7.5.5 Interesting Patch Rules

We present some association rules about the patching behavior of the vendors extracted using t_{pd} as class attribute.

Microsoft is quicker in patching vulnerabilities in Windows as compared to its remaining products. The following two rules show this: (1) `vnd=Microsoft prod=Windows XP typ=BO` $\rightarrow$ t_{pd}`=0-day`, (2) `vnd=Microsoft prod=Internet Explorer typ=BO` $\rightarrow$ t_{pd}`>+1 month`.

Apple also patches vulnerabilities in its operating systems as soon as they are disclosed. The following rule highlights this trend: `vnd=Apple prod=MAC OS typ=BO` $\rightarrow$ t_{pd}`=0-day`. Following rule shows that Apple generally takes about a week to fix DoS vulnerabilities even if they are exploited on the day they are disclosed: `vnd=Apple prod=MAC OS typ=DoS` $\rightarrow$ `0<`t_{pd}`≤+1 week`. Other rules show that Apple takes about a month after disclosure to patch the EXE and PHP vulnerabilities although they are always exploited before the patch is released and are prevalent types of vulnerabilities.

Sun is quicker in patching all kinds of vulnerabilities except XSS. Sun fixes DoS vulnerabilities before their disclosure which is a better performance as compared to Microsoft and Apple.

For Mozilla, BO and EXE vulnerabilities are mostly patched till the day of disclosure; however, SQL vulnerabilities are not patched for months. Following rules state this: (1) `vnd=Mozilla prod=Seamonkey typ=BO sev=M` $\rightarrow$ t_{pd}`=0-day`, (2) `vnd= Mozilla prod= Firefox typ=SQL sev=H` $\rightarrow$ t_{pd}`>+1 month`.

7.6 Patching vs. Exploitation

In this section, we compare the quickness of vendors with hackers. We study the trends in t_{pe} values of vulnerabilities present in the *PE-subset*.

$\mathbf{t_{pe} < 0}$ shows that a vulnerability was patched before its exploitation irrespective of whether or not it was disclosed. The inherent time-lag between the release of patches by vendors and their installation by end-users motivates the hackers to write exploits for vulnerabilities even after corresponding patches have been released. In our PE-subset, 31.7% of all the vulnerabilities fall in this range.

$\mathbf{t_{pe} = 0}$ means that a given vulnerability was exploited on the day its patch was released. 21.8% of the vulnerabilities fall in this range.

$\mathbf{t_{pe} > 0}$ shows that an exploit for a given vulnerability was released before the vendor patched it. A total of 46.4% of vulnerabilities have $t_{pe} > 0$. The larger percentage of $t_{pe} > 0$ compared to $t_{pe} < 0$ indicates that hackers have generally been quicker in exploiting the vulnerabilities as compared to vendors in patching. This observation affirms the result of the first t-test presented in Section 7.5.

7.6.1 Patching vs. Exploitation: Over the Years

From Figure 7.14 we can see the same behavior as observed in Section 7.5.1: patching response of vendors was poor till around 2005 and a large percentage of vulnerabilities was being exploited before being patched. In 2006, the situation was so bad that the patches for about 38% of the vulnerabilities were released more than a month after their exploitation. However, after 2007 a significant improvement can be observed in the vendor response. It is encouraging to see that since 2008, over 70% of all the vulnerabilities have been patched on or before the release date of their exploits. From the discussion in this section and Sections 7.4.1 and 7.5.1, we can conclude that the security state of the software industry has been improving for the last 3 years.

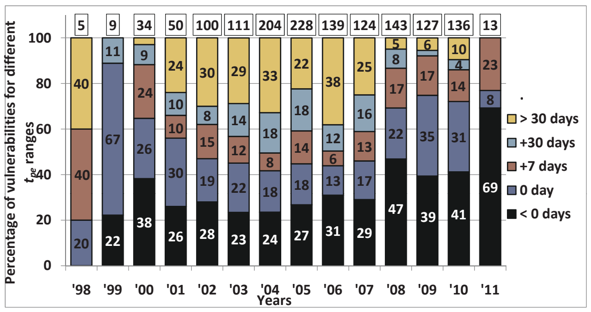


Figure 7.14 Yearly change in patching vs. exploitation trend for t_{pe}

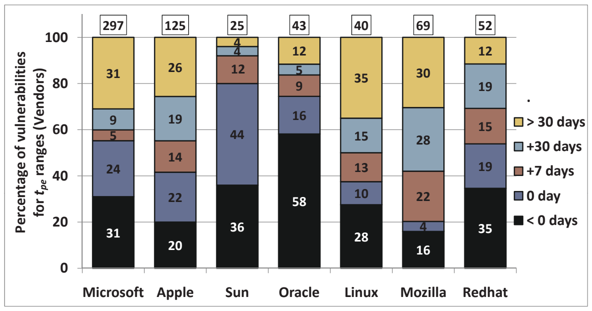


Figure 7.15 Patched vulnerabilities for vendors relative to exploit dates

7.6.2 Patching vs. Exploitation: Vendors and Products

It can be seen from Figure 7.15 that for all vendors except Oracle and Sun, the percentage size of the subgroups corresponding to $t_{pe} > 0$ is greater than that for $t_{pe} < 0$. The magnitude of the difference between the percentage sizes of $t_{pe} < 0$ and $t_{pe} > 0$ can serve as a measure to gauge the agility of the vendors in reference to hackers. We can see that among the vendors, only Oracle and Sun are faster than hackers, whereas hackers are, on average, faster than all other vendors. From Figure 7.16 we can see that, compared to hackers, Microsoft and Sun are quicker for Windows and Solaris respectively.

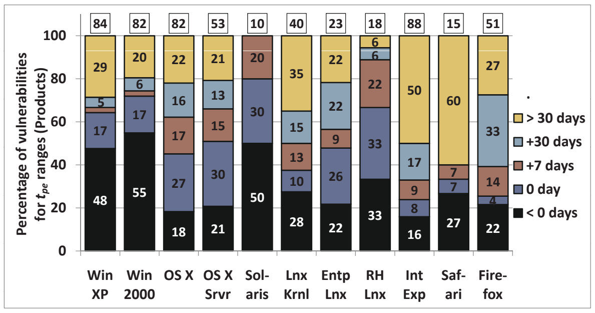


Figure 7.16 Patched vulnerabilities for products relative to exploit dates

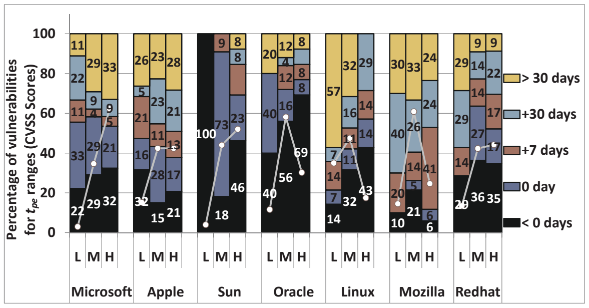


Figure 7.17 Patched vulns. relative to exploited vulns.: CVSS

7.6.3 Patching vs. Exploitation: CVSS Scores

From Figure 7.17, it can be seen that for Microsoft and Apple, approximately the same percentage of vulnerabilities belonging to medium and high risk categories are patched before the release of their exploits. However, the percentage of vulnerabilities for which $t_{pe} = 0$ is generally greater for medium risk vulnerabilities as compared to high risk vulnerabilities. It can be seen that closed-source vendors are quicker in patching the medium and high risk vulnerabilities compared to open-source vendors.

7.7 Implications

Observations from our study have important implications in software design, development, deployment, and management. We separately discuss them in the following text.

7.7.1 Software Design

The analysis of access requirements, functionality, and risk level of vulnerabilities presented in Sections 7.3.2, 7.3.4, and 7.3.3 respectively, can reveal inherent flaws in software design process for specific products and vendors. For instance, if a particular software series has more than typical instances of buffer overflow vulnerabilities, then this may reflect lack of sanity checks in socket read processes. From our data set, we observed that DoS is the most exploited vulnerability type in Solaris accounting for 38.85% of all its vulnerabilities. At the same time, only 11.7% of vulnerabilities in OS X involve DoS, which shows that Solaris is more susceptible to DoS attacks compared to OS X. The observation mentioned above implies that Solaris developers need to take additional steps to make the design more robust to DoS attacks.

7.7.2 Code Development Practices

The analysis of vulnerability life cycles during the evolution of a given software can reveal insights about potential flaws in its code development and testing practices. In particular, a correlation analysis of count of vulnerabilities across different software and vendors can highlight important differences in code development practices. For instance, we observe in Figure 7.11 that the percentage sizes of the subgroups corresponding to $t_{pd} > 0$ for open-source vendors (Linux, Redhat) are significantly greater than those of closed-source vendors (Microsoft, Apple). This observation highlights an important insight into the code development practices of open-source vendors which typically rely on contributions from a group of volunteer developers. On the other hand, closed-source vendors have dedicated resources to fix newly disclosed vulnerabilities as soon as possible. Therefore, open-source vendors tend to have a slower patch response compared to closed-source vendors.

7.7.3 Customer Assessment of Vendors and Products

The analysis presented in this chapter also has direct implications in product assessment, certification, and security recommendations to consumers. Several commercial products *e.g.*eEye Digital Security (`http://www.eeye.com`), Arellia (`http://www.arellia.com/`), can leverage the presented analysis for product recommendation and design of future security policies. For example, given that the exploits of vulnerabilities have already been released, our measurement analysis showed that Sun releases patches for 96% of the vulnerabilities within a month; whereas, Microsoft, Apple, and Linux provide patches for only 69%, 74%, and 65% of vulnerabilities in the same time period. Therefore, if the patch response of vendor is of prime importance to a customer, then the products from Sun should be preferred. As another example, if a customer's infrastructure has less tolerance for DoS attacks, then it is more suitable to deploy Mac OS X, which has the lowest percentage of DoS vulnerabilities compared to other operating systems. Likewise, if a customer requires more robustness to buffer overflow attacks, then it is more suitable to deploy Solaris because BO vulnerabilities account for about 20% of all the vulnerabilities in Windows and Mac but only 13% in Solaris.

7.8 Related Work

The major focus of the work on large scale analysis of vulnerabilities has been on the development of vulnerability discovery models (VDMs). Some work has also been done to understand the economic impacts of vulnerability disclosures in software. We briefly describe the work that has been done in these areas in relation to our work.

7.8.1 Large Scale Vulnerability Analysis

The work most relevant to ours was reported in [49] in which the authors presented a large scale analysis of vulnerabilities keeping in view the discovery, disclosure, exploit, and patch dates. They analyzed about 14000 vulnerabilities and showed that till 2006, the hackers had been quicker than vendors. This observation is in accordance with what we presented in this chapter but we also show that in the last three years, the response of vendors has been improving. Their work does not differentiate between vendors and types of vulnerabilities.

In [41], authors study the life-cycle of vulnerabilities from the time a software is released till the time the first vulnerability is discovered. They show that the time till the discovery of the first vulnerability is a function of the familiarity with the system and the amount of legacy code. In [125], the authors propose to use semantic templates to help the developers understand the vulnerabilities and their artifacts. This work only focuses on understanding the technical details of a disclosed vulnerability and does not study any large scale trend in vulnerabilities.

7.8.2 Studies on Disclosure and Patching

In [26], authors have studied the economic aspects of the quickness of vendors in releasing patches for Internet based vulnerabilities. In [118], authors show that on average a vendor loses 0.6% of the stock price with the disclosure of a vulnerability. In [28], authors show that a vendor with more competitors patches the vulnerabilities more quickly. In [29], they show that the vulnerability disclosure accelerates the patch release. Although their work is based upon a small data set of just 354 vulnerabilities disclosed till 2003, they make similar observation as ours that the closed-source vendors are quicker in patching the disclosed vulnerabilities. These studies, however, do not develop any insight into understanding individual behaviors of vendors and hackers.

In [98], using a small data set, authors make a claim that there is no difference between the patching behavior of open and closed-source vendors. They make this observation because they only consider the percentage of patched vulnerabilities as a measure of goodness of a vendor which is unreasonable because without analyzing the duration between disclosure dates and patch dates, one can not determine how active a vendor is in fixing vulnerabilities in its products.

7.8.3 Modeling and Classification

The motivation behind the work on VDMs is to enable the prediction of quantity and timing of vulnerability discoveries in new software. Four notable VDMs have been proposed: (1) Anderson Thermodynamic Model [27], (2) Rescorla Linear Model [92], (3) Rescorla Exponential Model [92], and (4) Alhazmi-Malaiya Logistic Model [24]. Another work focused on modeling the time interval between disclosure date of vulnerabilities and their corresponding exploit, patch, and discovery dates [120]. A recent work extracted various features from NVD and OSVDB and used SVM to predict whether a recently disclosed vulnerability will be exploited within a given time or not [34]. Our focus, however, is not the prediction rather the study of phases of vulnerability life cycle in reference to different variables along with several aspects associated with the nature of vulnerabilities.

7.9 Conclusion

In this chapter, we presented a large scale study of various aspects associated with software vulnerabilities during their life cycle. We aggregated a large software vulnerability data set containing 46310 vulnerabilities disclosed till 2011. Our study showed that the number of vulnerabilities being disclosed every year has stopped increasing since 2008. We showed that the most primitive and most exploited form of vulnerabilities are DoS, BO, and EXE; however, SQL, XSS, and PHP have also become significantly large. We also observed that the percentage of remotely exploitable vulnerabilities has gradually increased to over 80% of all the vulnerabilities. Since 2008, the vendors have been becoming more agile in patching the vulnerabilities and the access complexity of vulnerabilities has been increasing. However, even then, the average time taken by hackers to exploit a vulnerability is smaller than that taken by the vendor. Our findings highlight that patching in closed-source software is faster compared to open-source software and at the same time the exploitation is slower.

8 Conclusion

In this thesis, I presented statistical algorithms for the design, analysis, measurement, and modeling of RFID systems, network metrics, user authentication, and software security. For RFID systems, I first presented a new estimator, the average run size of 1s, for estimating RFID tag population size of arbitrarily large sizes. Using analytical plots, I showed that our estimator has much smaller variance compared to other estimators, which makes our scheme faster than the previous ones. Our experimental results show that our estimation scheme is significantly faster than all prior schemes. Second, I presented our new RFID identification scheme. It represents the first effort to formulate the Tree Walking process mathematically and proposed a method to minimize the expected number of queries and expected identification time. The significance of this work in terms of impact lies in that the Tree Walking protocol is a fundamental multiple access protocol and has been standardized as an RFID tag identification protocol. Our experimental results show that TH significantly outperforms all prior tag identification protocols, even those that are not C1G2 compliant, for metrics such as the number of reader queries per tag, the identification speed, and the number of responses per tag. Third, I proposed a protocol to detect missing tag events in the presence of unexpected tags. It represents the first effort on addressing the important and practical problem of detecting missing tags in the presence of unexpected tags. We have proposed a technique that our protocol uses to handle large frame sizes to ensure compliance with the C1G2 standard. Our experimental results show that our protocols significantly outperform all prior protocols in terms of actual reliability as well as detection time even though the existing protocols do not handle the presence of unexpected tags. Fourth, I proposed an accurate and efficient per-flow latency measurement scheme that does not require packet probing and time stamping. The key novelty of this work is that we purposely allow noise to be introduced in recording packet timing information for minimizing storage space and use statistical techniques to denoise the recorded information to obtain accurate latency estimates when latency of a target flow is queried. Our theoretical analysis and experimental results show that our scheme always achieves the required reliability. Our scheme has a much smaller processing overhead in terms of number of hash computations and memory updates compared to existing schemes, which further require sending probe packets or attaching time stamps to every packet. Fifth, I proposed GEAT, a gesture based user authentication scheme for the secure unlocking of touch screen devices. Compared with existing passwords/PINs/ patterns based schemes, GEAT improves both the security and usability of such devices because it is not vulnerable to shoulder surfing attacks and smudge attacks and at the same time gestures are easier to input than passwords and PINs. I also proposed algorithms to model multiple behaviors of a user in performing each gesture. We implemented GEAT on real smart phones and conducted real-world experiments. Last, I presented a large scale study of various aspects associated with software vulnerabilities during their life cycle. Our study showed that the number of vulnerabilities being disclosed every year has stopped increasing since 2008. We showed that the most primitive and most exploited form of vulnerabilities are DoS, BO, and EXE; however, SQL, XSS, and PHP have also become significantly large. Our findings also highlighted that patching of vulnerabilities in closed-source software is faster compared to open-source software and at the same time the exploitation is slower.

The vision of this thesis can be extended to many other similar research directions. Within RFID systems, the theoretical framework of the proposed schemes can be leveraged to enable other applications such as RFID tag search for product recall, dynamic RFID population tracking, multi-category RFID estimation, and fair RFID identification for active RFID tags.

For network measurements, the theoretical framework of the proposed scheme for latency measurement can be extended to measure other network performance metrics such as loss, throughput, jitter, flow size distributions, quality of service, and quality of experience. For user authentication, the feature extraction and modeling aspect of the proposed scheme can be extended to authenticate users with the help of wearable devices and even authenticate devices themselves in the emerging internet of things infrastructure.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] http://en.wikipedia.org/wiki/Distribution_center.

[2] 25 leaked celebrity cell phone pics. http://www.holytaco.com/25-leaked-celebrity-cell-phone-pics/.

[3] CAIDA passive network monitors. http://www.caida.org/data/realtime/passive/.

[4] The CAIDA UCSD anonymized 2011 internet traces. http://www.caida.org/data/passive/passive_2011_dataset.xml.

[5] Cedexis. http://www.cedexis.com/.

[6] Common Vulnerabilities and Exposures, http://cve.mitre.org/.

[7] Corvil claims to minimize network latency. http://www.pcworld.idg.com.au/article/196828/corvil_claims_minimize_network_latency/.

[8] *Fibre Channel Backbone - 5 (FC-BB-5) REV 2*.

[9] Forum for Incident Response and Security Teams, http://www.first.org/cvss.

[10] IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems.

[11] National Vulnerability Database, http://nvd.nist.gov/.

[12] Preliminary national retail security survey findings. https://nrf.com/news/national-retail-security-survey-retail-shrinkage-totaled-345-billion-2011.

[13] Sidera. http://www.sidera.net/.

[14] Singapore exchange (sgx) selects corvil for latency management. http://www.corvil.com/News/Press-Releases/Singapore-Exchange-(SGX)-Selects-Corvil-For-Latenc.aspx.

[15] The symantec smartphone honey stick project. *http://www.symantec.com/content/en/us/about/presskits/b-symantec-smartphone-honey-stick-project.en-us.pdf?om_ext_cid=biz_socmed_twitter_facebook_marketwire_linkedin_2012Mar_worldwide_honeystick*.

[16] The Open Source Vulnerability Database, http://osvdb.org/.

[17] Tokyo stock exchange select corvil. `http://www.corvil.com/News/Press-Releases/Tokyo-Stock-Exchange-Select-Corvil.aspx`.

[18] Turbobytes. `http://www.turbobytes.com/`.

[19] While london stock exchange selects corvil for low latency network monitoring and analysis solution. `http://low-latency.com/article/\%E2\%80\%A6-while-london-stock-exchange-selects-corvil-low-latency-network-monitoring-and-analysis-sol`.

[20] Z-Drive R4 and R5 PCIe SSD. `http://lensfire.in/2012/01/ocz-launches-new-z-drive-r4-and-r5-pcie-ssd-ces-2012-2012/`.

[21] HP expands high-performance computing offering with infiniband solutions from cisco. `http://www.hp.com/hpinfo/newsroom/press/2007/070524xa.html`, May 2007.

[22] The amazon warehouses. `http://imgur.com/gallery/uHZbW`, 2013.

[23] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of of 20th International Conference of on Very Large Data Bases*, pages 487–499, 1994.

[24] Omar H. Alhazmi and Yashwant K. Malaiya. Quantitative vulnerability assessment of systems software. In *Proceedings of Annual Reliability and Maintainability Symposium*, pages 615–620, 2005.

[25] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of ACM SoTC*, pages 20–29, 1996.

[26] Ross Anderson. Why information security is hard – an economic perspective. In *Proceedings of 17th Annual Computer Security Applications Conference of* , pages 358–365, 2001.

[27] Ross Anderson. Security in open versus closed systems – the dance of boltzmann, coase and moore. In *Proceedings of Open Source Software: Economics, Law, and Policy Confocuce*, June 2002.

[28] Ashish Arora, Chris Forman, Anand Nandkumar, and Rahul Telang. Competition and patching of security vulnerabilities: An empirical analysis. *Information Economics and Policy*, 22(2):164–177, 2010.

[29] Ashish Arora, Ramayya Krishnan, Rahul Telang, and Yubao Yang. An empirical analysis of software vendors patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, 21(1):115–132, 2010.

[30] Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith. Smudge attacks on smartphone touch screens. In *Proceedings of 4th USENIX conference on Offensive technologies*, pages 1–10, 2010.

[31] Michael Backes, Thomas R. Gross, and Guenter Karjoth. Tag identification system, 2008.

[32] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of IMC*, pages 267–280, 2010.

[33] Charles Bordenave, David McDonald, and Alexandre Proutiere. Performance of random medium access control, an asymptotic approach. In *Proceedings of ACM SIGMETRICS*, 2008.

[34] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond heuristics: Learning to classify vulnerabilities and predict exploits. In *Proceedings of of 16th International Conference of on Knowledge discovery and data mining*, pages 105–114, 2010.

[35] John I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory*, 25:505–515, 1979.

[36] Bogdan Carbunar, Murali Krishna Ramanathan, Mehmet Koyuturk, Christoph Hoffmann, and Ananth Grama. Redundant reader elimination in RFID systems. In *Proceedings of IEEE Communications Society Conference of on SECON*, pages 576–580, 2005.

[37] Jae-Ryong Cha and I. Jae-Hyun Kim. Novel anti-collision algorithms for fast object identification in rfid system. In *Proceedings of of International Conference of on Parallel and Distributed Systems*, 2005.

[38] Chih-Chung Chang and Chin-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27, 2011.

[39] Binbin Chen, Ziling Zhou, and Haifeng Yu. Understanding rfid counting protocols. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 291–302. ACM, 2013.

[40] Yan Chen, David Bindel, Hanhee Song, and Randy H. Katz. An algebraic approach to practical and scalable overlay network monitoring. In *Proceedings of ACM SIGCOMM*, pages 55–66, 2004.

[41] Sandy Clark, Stefan Frei, Matt Blaze, and Jonathan Smith. Familiarity breeds contempt: The honeymoon effect and the role of legacy code in zero-day vulnerabilities. In *Proceedings of 26th International Annual Computer Security Applications Conference of* , pages 251–260, 2010.

[42] Mauro Conti, Irina Zachia-Zlatea, and Bruno Crispo. Mind how you answer me!: transparently authenticating the user of a smartphone when answering or placing a call. In *Proceedings of ACM Symposium on Information, Computer and Communications Security*, pages 249–259, 2011.

[43] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[44] Robert Dorfman. The detection of defective members of large populations. *Annals of Mathematical Statistics*, 14:436–440, 1943.

[45] Nick Duffield. Simple network performance tomography. In *Proceedings of ACM IMC*, pages 210–215, 2003.

[46] Klaus Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication*. Wiley, 2010.

[47] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[48] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[49] Stefan Frei, Martin May, Ulrich Fiedler, and Bernhard Plattner. Large-scale vulnerability analysis. In *Proceedings of 2006 SIGCOMM workshop on Large-Scale Attack Defense*, pages 131–138, September 2006.

[50] Stefan Frei, Bernhard Tellenbach, and Bernhard Plattner. 0-day patch exposing vendors (in) security performance. In *Proceedings of Black Hat Technical Security Conference of* , volume 14, 2009.

[51] Karsten Fyhn, , Rasmus Melchior Jacobsen, Petar Popovski, and Torben Larsen. Fast capture – recapture approach for mitigating the problem of missing rfid tags. *IEEE Transactions on Mobile Computing*, 11(3):518–528, 2012.

[52] D. Gafurov, K. Helkala, and T. Søndrol. Biometric gait authentication using accelerometer sensor. *Journal of computers*, 1(7):51–59, 2006.

[53] Hao Han, Bo Sheng, Chiu C. Tan, Qun Li, Weizhen Mao, and Sanglu Lu. Counting RFID tags efficiently and anonymously. In *Proceedings of IEEE International Conference of on Computer Communications*, 2010.

[54] Nan Hua, Eric Norige, Sailesh Kumar, and Bill Lynch. Non-crypto hardware hash functions for high performance networking ASICs. In *Proceedings of ACM/IEEE ANCS*, pages 156–166, 2011.

[55] EPCGlobal Inc. *Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz–960 MHz.* EPCGlobal Inc, 1.2.0 edition, 2008.

[56] Rasmus Jacobsen, Karsten Fyhn Nielsen, Petar Popovski, and Torben Larsen. Reliable identification of rfid tags using multiple independent reader sessions. In *Proceedings of IEEE International Conference of on RFID*, pages 64–71, 2009.

[57] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, Digital Equipment Corporation, 1984.

[58] Jr Joe H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[59] S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[60] Kevin Killourhy and Roy Maxion. Why did my detector do that?! In *Proceedings of Recent Advances in Intrusion Detection*, pages 256–276, 2010.

[61] Murali Kodialam and Thyaga Nandagopal. Fast and reliable estimation schemes in RFID systems. In *Proceedings of 12th International Conference of on Mobile Computing and Networking*, pages 322–333, 2006.

[62] Murali Kodialam, Thyaga Nandagopal, and Wing Cheong Lau. Anonymous tracking using RFID tags. In *Proceedings of IEEE International Conference of on Computer Communications*, 2007.

[63] Ramana Rao Kompella, Kirill Levchenko, Alex C. Snoeren, and George Varghese. Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator. In *Proceedings of ACM SIGCOMM*, pages 255–266, 2009.

[64] Abhishek Kumar, Jun (Jim) Xu, Jia Wang, Oliver Spatschekt, and Li (Erran) Lit. Space-code bloom filter for efficient per-flow traffic measurement. In *Proceedings of IEEE INFOCOM*, pages 1762–1773, 2004.

[65] J.R. Kwapisz, G.M. Weiss, and S.A. Moore. Cell phone-based biometric identification. In *Proceedings of IEEE International Conference of on Biometrics: Theory Applications and Systems*, pages 1–7, 2010.

[66] Ching Law, Kayi Lee, and Kai-Yeung Siu. Efficient memoryless protocol for tag identification. In *Proceedings of 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.

[67] Chun Hee Lee and Chin-Wan Chung. Efficient storage scheme and query processing for supply chain management using RFID. In *Proceedings of ACM International Conference of on Management of data*, pages 291–302, 2008.

[68] Myungjin Lee, Nick Duffield, and Ramana Rao Kompella. Not all microseconds are equal: fine-grained per-flow measurements with reference latency interpolation. In *Proceedings of ACM SIGCOMM*, pages 27–38, 2010.

[69] Myungjin Lee, Nick Duffield, and Ramana Rao Kompella. A scalable architecture for maintaining packet latency measurements. In *Proceedings of IMC*, pages 101–114, 2012.

[70] Myungjin Lee, Sharon Goldberg, Ramana Rao Kompella, and George Varghese. Fine-grained latency and loss measurements in the presence of reordering. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 329–340. ACM, 2011.

[71] Tao Li, Shigang Chen, and Yibei Ling. Identifying the missing tags in a large RFID system. In *Proceedings of MobiHoc*, pages 1–10, 2010.

[72] Tao Li, Samuel Wu, Shigang Chen, and Mark Yang. Energy efficient algorithms for the RFID estimation problem. In *Proceedings of IEEE International Conference of on Computer Communications*, 2010.

[73] Xiulong Liu, Keqiu Li, Geyong Min, Yanming Shen, A Liu, and Wenyu Qu. Completely pinpointing the missing RFID tags in a time-efficient way. *IEEE Transactions on Computers*, pages 1–11, 2013.

[74] Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *Proceedings of ACM SIGMETRICS*, pages 121–132, 2008.

[75] Alexander De Luca, Alina Hang, Frederik Brudy, Christian Lindner, and Heinrich Hussmann. Touch me once and i know it's you!: implicit authentication based on touch screen patterns. In *Proceedings of ACM Annual Conference on Human Factors in Computing Systems (SIGCHI)*, pages 987–996, 2012.

[76] Wen Luo, Shigang Chen, Tao Li, and Yan Qiao. Probabilistic missing-tag detection and energy-time tradeoff in large-scale RFID systems. In *Proceedings of MobiHoc*, pages 95–104, 2012.

[77] Bill Lynch and Sailesh Kumar. Smart memory for high performance network packet forwarding. In *Proceedings of Hot Chips Symposium*, 2010.

[78] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S.M. Makela, and HA Ailisto. Identifying users of portable devices from gait pattern with accelerometers. In *Proceedings of IEEE International Conference of on Acoustics, Speech, and Signal Processing*, volume 2, pages 973–976, 2005.

[79] Richard Martin. Wall street's quest to process data at the speed of light. *Information Week*, 4(21), 2007.

[80] Michael J. Miller. Bandwidth engine serial memory chip breaks 2 billion accesses/sec. In *Proceedings of Hot Chips Symposium*, 2011.

[81] Fabian Monrose, Michael K. Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. In *Proceedings of ACM CCS*, pages 73 – 82, 1999.

[82] Jihoon Myung and Wonjun Lee. Adaptive splitting protocols for rfid tag collision arbitration. In *Proceedings of 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 202–213, 2006.

[83] Vinod Namboodiri and Lixin Gao. Energy-aware tag anticollision protocols for RFID systems. In *Proceedings of 5th IEEE International Conference of on Pervasive Computing and Communications*, pages 23–36, 2007.

[84] Badri Nath, Franklin Reynolds, and Roy Want. RFID technology and applications. *IEEE Pervasive Computing*, 5:22–24, 2006.

[85] Aditya Nemmaluri, Mark D. Corner, and Prashant Shenoy. Sherlock: Automatically locating objects for humans. In *Proceedings of International Conference of on Mobile Systems, Applications, and Services*, pages 187–198, 2008.

[86] Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P. Patil. Landmarc: Indoor location sensing using active RFID. *LANDMARC: Indoor Location Sensing Using Active RFID*, 10:701–710, 2004.

[87] Lei Pan and Hongyi Wu. Smart trend-traversal: A low delay and energy tag arbitration protocol for large RFID systems. In *Proceedings of 30th IEEE International Conference of on Computer Communications*, 2009.

[88] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. A first look at modern enterprise traffic. In *Proceedings of ACM IMC*, pages 15–28, 2005.

[89] Chen Qian, Yunhuai Liu, Hoilun Ngan, and Lionel M. Ni. ASAP: Scalable identification and counting for contactless rfid systems. In *Proceedings of 30th IEEE International Conference of on Distributed Computing Systems*, pages 52–61, 2010.

[90] Chen Qian, Hoilun Ngan, and Yunhao Liu. Cardinality estimation for large-scale RFID systems. In *Proceedings of 6th IEEE PerCom*, pages 30–39, 2008.

[91] M.V. Ramakrishna, E. Fu, and E. Bahcekapili. Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers*, 46(12):1378–1381, 1997.

[92] Eric Rescorla. Is finding security holes a good idea? *IEEE Security and Privacy*, 3(1):14–19, Januray 2005.

[93] Mark Roberti. A 5-cent breakthrough. *RFID Journal*, 5(6), 2006.

[94] Walter A. Rosenkrantz and Donald Towsley. On the instability of slotted aloha multiaccess algorithm. *IEEE Transactions on Automatic Control*, 28(10):994–996, 1983.

[95] Napa Sae-Bae, Kowsar Ahmed, Katherine Isbister, and Nasir Memon. Biometric-rich gestures: a novel approach to authentication on multi-touch device. In *Proceedings of ACM Annual Conference on Human Factors in Computing Systems (SIGCHI)*, 2012.

[96] Florian Schaub, Ruben Deyhle, and Michael Weber. Password entry usability and shoulder surfing susceptibility on different smartphone platforms. In *Proceedings of 11th International Conference of on Mobile and Ubiquitous Multimedia*, 2012.

[97] Bernhard Schlkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[98] Guido Schryen. A comprehensive and comparative analysis of the patching behavior of open source and closed source software vendors. In *Proceedings of 5th International Conference of on IT Security Incident Management and IT Forensics*, pages 153–168, 2009.

[99] E. Eugene Schultz, David S. Brown, and Thomas A. Longstaff. *Responding to Computer Security Incidents: Guidelines for Incident Handling.* Lawrence Livermore National Laboratory, Livermore, CA, 1990.

[100] Philips Semiconductors. *SL2 ICS11 I.Code UID Smart Label IC Functional Specification Datasheet http://www.advanide.com/datasheets/sl2ics11.pdf*, 2004.

[101] Vahid Shah-Mansouri and Vincent W.S. Wong. Anonymous cardinality estimation in RFID systems with multiple readers. In *Proceedings of IEEE Global Communications Conference of* , 2009.

[102] Muhammad Shahzad and Alex X. Liu. Every bit counts – fast and scalable RFID estimation. In *Proceedings of 18th International Conference of on Mobile Computing and Networking (Mobicom)*, pages 365–376, 2012.

[103] Muhammad Shahzad and Alex X Liu. Every bit counts: Fast and scalable RFID estimation. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2012.

[104] Muhammad Shahzad and Alex X Liu. Probabilistic optimal tree hopping for RFID identification. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013.

[105] Muhammad Shahzad and Alex X. Liu. Probabilistic optimal tree hopping protocol for RFID identification. In *submission, ACM International Conference of on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2013.

[106] Muhammad Shahzad and Alex X Liu. Fast and accurate estimation of RFID tags. *IEEE/ACM Transactions on Networking (ToN)*, 2014.

[107] Muhammad Shahzad and Alex X Liu. Noise can help: Accurate and efficient per-flow latency measurement without packet probing and time stamping. In *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2014.

[108] Muhammad Shahzad and Alex X Liu. Probabilistic optimal tree hopping for RFID identification. *IEEE/ACM Transactions on Networking (ToN)*, 2014.

[109] Muhammad Shahzad and Alex X Liu. Expecting the unexpected: Fast and reliable detection of missing RFID tags in the wild. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2015.

[110] Muhammad Shahzad, Alex X Liu, and Arjmand Samuel. Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, 2013.

[111] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In *International Conference on Software Engineering (ICSE)*, 2012.

[112] Muhammad Shahzad, Saira Zahid, and Muddassar Farroq. A hybrid GA-PSO fuzzy system for user identification on smart phones. In *Proceedings of 11th Annual Conference of on Genetic and Evolutionary Computation (GECCO)*, pages 1617–1624, 2009.

[113] Alan D Smith, Amber A Smith, and David L Baker. Inventory management shrinkage and employee anti-theft approaches. *International Journal of Electronic Finance*, 5(3):209–234, 2011.

[114] Chiu Chiang Tan, Bo Sheng, and Qun Li. How to monitor for missing RFID tags. In *Proceedings of IEEE ICDCS*, pages 295–302, 2008.

[115] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall, 2002.

[116] ShaoJie Tang, Jing Yuan, Xiang-Yang Li, Guihai Chen, Yunhao Liu, and JiZhong Zhao. Raspberry: A stable reader activation scheduling protocol in multi-reader RFID systems. In *Proceedings of IEEE International Conference of on Network Protocols*, pages 304–313, 2009.

[117] F. Tari, A. Ozok, and S.H. Holden. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *Proceedings of SOUPS*, pages 56–66, 2006.

[118] Rahul Telang and Sunil Wattal. An empirical analysis of the impact of software vulnerability announcements on firm stock price. *IEEE Transactions on Software Engineering*, 33(8):544–557, 2007.

[119] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[120] Géraldine Vache. Vulnerability analysis for a quantitative security evaluation. In *Proceedings of 3rd International Symp. on Empirical Software Engineering and Measurement*, 2009.

[121] Harald Vogt. Efficient object identification with passive RFID tags. *Pervasive Computing*, 2414:98–113, 2002.

[122] James Waldrop, Daniel W. Engels, and Sanjay E. Sarma. Colorwave: A MAC for RFID reader networks. In *Proceedings of IEEE Wireless Communications and Networking*, pages 1701–1704, 2003.

[123] Chong Wang, Hongyi Wu, and Nian-Feng Tzeng. RFID-based 3-d positioning schemes. In *Proceedings of IEEE International Conference of on Computer Communications*, pages 1235–1243, 2007.

[124] Ian H. Witten, Eibe Frank, Len Trigg, Mark Hall, Geoffrey Holmes, and Sally Jo Cunningham. *Weka: Practical Machine Learning Tools and Techniques with Java Implementations*. Citeseer, 1999.

[125] Yan Wu, Harvey Siy, and Robin Gandhi. Empirical results on the study of software vulnerabilities: NIER track. In *Proceedings of 33rd International Conference of on Software Engineering*, pages 964–967, 2011.

[126] Saira Zahid, Muhammad Shahzad, Syed Ali Khayam, and Muddassar Farooq. Keystroke-based user identification on smart phones. In *Proceedings of 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 224–243, 2009.

[127] Andrea Zanella. Estimating collision set size in framed slotted aloha wireless networks and RFID systems. *IEEE Communications Letters*, 16(3):300–303, 2012.

[128] Rui Zhang, Yunzhong Liu, Yanchao Zhang, and Jinyuan Sun. Fast identification of the missing tags in a large RFID system. In *Proceedings of IEEE SECON*, 2011.

[129] Bin Zhen, Mamoru Kobayashi, and Masashi Shimizu. Framed ALOHA for multiple RFID objects identification. *IEICE Transactions on Communications*, 88:991–999, 2005.

[130] Nan Zheng, Kun Bai, Hai Huang, and Haining Wang. You are how you touch: User verification on smartphones via tapping behaviors. Technical report, College of William and Mary, 2012.

[131] Zongheng Zhou, Himanshu Gupta, Samir R. Das, and Xianjin Zhu. Slotted scheduled tag access in multi-reader RFID systems. In *Proceedings of IEEE International Conference of on Network Protocols*, pages 61–70, 2007.