ANALYTICAL CHARACTERIZATION OF INTERNET SECURITY ATTACKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Sarah H. Sellke

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2010

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

I am indebted to my advisors, Professor Saurabh Bagchi and Professor Ness Shroff, for their guidance and support during the entire course of my Ph.D. research, to Professor Chih-Chun Wang for being an active and crucial part of my research on timing channels, and to my fine committee members Professor Sonia Fahmy and Professor Ninghui Li for their invaluable feedback and assistance on my work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Sellke, Sarah H. Ph.D., Purdue University, May 2010. Analytical Characterization of Internet Security Attacks. Major Professors: Saurabh Bagchi and Ness B. Shroff.

Internet security attacks have drawn significant attention due to their enormously adverse impact. These attacks includes Malware (Viruses, Worms, Trojan Horse), Denial of Service, Packet Sniffer, and Password Attacks. There is an increasing need to provide adequate defense mechanisms against these attacks. My thesis proposal deals with analytical aspects of the Internet security attacks, as well as practical solutions based on our analysis.

First, We focus on modeling and containment of internet worms. We present a branching process model for the propagation of worms. Our model leads to the development of automatic worm containment strategies, which effectively contain both uniform scanning worms and local preference scanning worms. Incremental deployment of our scheme provides worm containment for local networks when combined with traditional firewalls.

Next, we study the capacity of Bounded Service Timing Channels. We derive an upper bound and two lower bounds on the capacity of such timing channels. We show that when the length of the support interval is small, the uniform BSTC has the smallest capacity among all BSTCs. Based on our analysis, we design and implement a covert timing channel over TCP/IP networks. We are able to quantify the achievable data rate (or leak rate) of such a covert channel. Moreover, by sacrificing data rate, we are able to mimic normal traffic patterns, which makes detecting such communication virtually impossible.

1. INTRODUCTION

The Internet has become critically important to the financial viability of the national and the global economy. Meanwhile, we are witnessing an upsurge in the incidents of attacks on the Internet. In this report, we will provide analytical characteristics of some of the attacks on the Internet including covert communications, computer worms and viruses.

Covert communication can be used by a party who has access to sensitive information to leak such information to unauthorized party while avoid being detected. A network covert timing channels can be used by a Trojan horse to leak information in a noisy network environment by the timing of the packets, not the content of the packets. Such channels are hard to detect because by examining the content of the packet alone, it will reveal no information.

Worms and DDoS attacks are more common attacks on the Internet. Code Red, SQL Slammer, and Sasser are some of the more famous examples of worms that have caused considerable damage. Network worms have the potential to infect many vulnerable hosts on the Internet before human countermeasures take place. The infected hosts can also be controlled by a master machine to launch DDoS attacks on the Internet.

In the rest of this chapter, we introduce our analytical results and practical solutions in covert network timing channels and Internet worms.

1.1 Internet Worms

Most models of Internet-scale worm propagation are based on deterministic epidemic models [6, 26, 29]. They are acceptable for modeling worm propagation when the number of infected hosts is large. However, it is generally accepted that they are inadequate to model the early phase of worm propagation accurately because the number of infected hosts early on is very small [19].

In our work, we propose a stochastic branching process model for the early phase of uniform scanning worm propagation. This model captures the worm spreading dynamics for worms of arbitrary scanning rate, including stealth worms that may turn themselves off at times. Our analysis shows that it is the total number of scans that an infected host attempts, and not the more restrictive scanning rate, that determines whether worms can spread. Moreover, we can probabilistically bound the total number of infected hosts. The insight from our model provides us with a mechanism for containing both fast scanning worms and slow scanning worms without knowing the worm signature in advance or needing to detect whether a host is infected. This scheme is non-intrusive in terms of its impact on legitimate traffic.

Our modeling and containment schemes are further generalized to preferential scanning worms. Our analysis and simulation shows that our method can contain worm infection in a local network, and allows for incremental deployment.

1.2 Theory and Practice of Covert Network Timing Channels

1.2.1 Capacity of Timing Channels with Bounded Service Time

It is well known that queues with exponentially distributed service times have the smallest Shannon capacity among all single-server queues with the same service rate. In my thesis, we study the capacity of timing channels in which the service time distributions have bounded support, i.e., Bounded Service Timing Channels (BSTC). We derive an upper bound and two lower bounds on the capacity of such timing channels. The tightness of these bounds is investigated analytically as well as via simulations. We find that the uniform BSTC serves a role for BSTCs that is similar to what the exponential service timing channel does for the case of timing channels with unbounded service time distributions. That is, when the length of the support interval is small, the uniform BSTC has the smallest capacity among all BSTCs.

1.2.2 Network Timing Channels: Theory to Practice

There has been significant recent interest in covert communication using timing channels. In network timing channels, information is leaked by controlling the time between transmissions of consecutive packets. In TCP/IP timing channels, the service time distributions have *bounded support*. Moreover, we observe that in TCP/IP networks, end-to-end delays are much larger than the jitter [1].

Based on our theoretical results, we have designed a robust, low-cost, and efficient *L-bits to n-packets* scheme for covert communications using timing channels. Our approach has been to develop easy-to-use encoding and decoding schemes that allow us to operate at close to the achievable data rate of the timing channel. We have also implemented a TCP/IP based timing channel on the real Internet and our experiments show that our scheme achieves between two to five times the data rate over the state-of-the art.

1.2.3 Concealable Timing Channels

We provide two designs of concealable timing channels. The first one mimics normal traffic whose packet inter-arrival times are *i.i.d.*, and it can be made computationally indistinguishable from that of *i.i.d.* normal traffic. The second design of our concealable timing channel mimics traffic that are long range dependent. It is statistically indistinguishable from LRD legitimate traffic and evades current covert timing channel detections.

It has been shown that the telnet traffic can be modeled by an *i.i.d.* Pareto distribution. We demonstrated our design of a timing channel that mimics telnet packet inter-transmission time distribution such that it is indistinguishable from legitimate telnet traffic. Since Web traffic accounts for more than half of Internet traffic today, Camouflaging covert timing channels in Web traffic would be more advantageous for concealment. Extensive research has shown that Internet traffic, including HTTP

[1]jitter is defined as the variation of the delay

traffic, exhibits self-similarity and long range persistence. The covert timing channels that mimic *i.i.d.* legitimate traffic cannot imitate HTTP traffic because these covert traffic patterns are not long range dependent. Therefore, we design a covert timing channel whose inter-arrival times are long range dependent and have the same marginal distribution as the inter-arrival times for new HTTP connection traffic. These inter-arrival times are constructed by combining a Fractional Auto-Regressive Integrated Moving Average (FARIMA) time series and an i.i.d. cryptographically secure random sequence.

Experiments are conducted on PlanetLab, and the results are validated against recent real traffic trace data. Our experiments demonstrate that the traffic from this timing channel traffic is statistically indistinguishable from legitimate HTTP traffic and undetectable by all current detection schemes for timing channels. The concealable timing channels result in a drop of the achievable data rate using the timing channel, however the resultant non-detectable scheme is still able to achieve a higher data rate than previous easily detectable timing channel schemes.

Our work on network timing provides three main contributions. The first is to derive an upper bound and two lower bounds on the capacity of timing channels with bounded service time. These bounds are asymptotically tight when the service time is uniformly distributed. The second is to quantify the threat posed by covert network timing channels by demonstrate that the data rate can be much larger than previously thought. The third is to use timing channels to communicate at a low data rate without being detected. This suggests that TCP/IP timing channels can be far stealthier than previously thought possible.

The rest of report is organize as follows: in chapter 2 and 3, we will provide our theoretical results on the capacity of timing channels and our design, implementation, and experimental results on the covert TCP/IP timing channels. In chapter 4, we will present our research in modeling of scanning worms and our automatic worm containment mechanisms for both uniform scanning worms and preference scanning worms.

2. MODELING AND AUTOMATED CONTAINMENT OF INTERNET SCANNING WORMS

2.1 Introduction

The Internet has become critically important to the financial viability of the national and the global economy. Meanwhile, we are witnessing an upsurge in the incidents of malicious code in the form of computer viruses and worms. One class of such malicious code, known as random scanning worms, spreads itself without human intervention by using a scanning strategy to find vulnerable hosts to infect. Code Red, SQL Slammer, and Sasser are some of the more famous examples of worms that have caused considerable damage. Network worms have the potential to infect many vulnerable hosts on the Internet before human countermeasures take place. The aggressive scanning traffic generated by the infected hosts has caused network congestion, equipment failure, and blocking of physical facilities such as subway stations, 911 call centers, etc. As a representative example, consider the Code Red worm version 2 that exploited a buffer overflow vulnerability in the Microsoft IIS web servers. It was released on July 19th, 2001 and over a period of less than 14 hours infected more than 359,000 machines. The cost of the epidemic, including subsequent strains of Code Red, has been estimated by Computer Economics to be $2.6 billion [32]. While Code Red was particularly virulent in its economic impact (e.g., see [3, 21]) it provides an indication of the magnitude of the damage that can be inflicted by such worms. Thus, there is a need to carefully characterize the spread of worms and develop efficient strategies for worm containment.

The goal of our research is to provide a model for the propagation of random scanning worms and the corresponding development of automatic containment mechanisms that prevent the spread of worms beyond their early stages. This containment scheme is then extended to protect an enterprise network from a preference scanning worm. A host infected with random scanning worms finds and infects other vulnerable hosts by scanning a list of randomly generated IP addresses. Worms using other strategies to find vulnerable hosts to infect are not within the scope of this work. Some examples of non-random-scanning worms are email worms, peer-to-peer worms, and worms that search the local host for addresses to scan.

Most models of Internet-scale worm propagation are based on deterministic epidemic models [6, 26, 29]. They are acceptable for modeling worm propagation when the number of infected hosts is large. However, it is generally accepted that they are inadequate to model the early phase of worm propagation accurately because the number of infected hosts early on is very small [19]. The reason is that epidemic models capture only expected or mean behavior, while not being able to capture the variability around this mean, which could be especially dramatic during the early phase of worm propagation. While stochastic epidemic models can be used to model this early phase, they are generally too complex to provide useful analytical solutions.

In this paper, we propose a stochastic branching process model for the early phase of worm propagation [1]. We consider the generation-wise evolution of worms, with the hosts that are infected at the beginning of the propagation forming generation zero. The hosts that are directly infected by hosts in generation n are said to belong to generation $n + 1$. Our model captures the worm spreading dynamics for worms of arbitrary scanning rate, including stealth worms that may turn themselves off at times.

We show that it is the total number of scans that an infected host attempts, and not the more restrictive scanning rate, that determines whether worms can spread. Moreover, we can probabilistically bound the total number of infected hosts. These insights lead us to develop an automatic worm containment strategy. The main idea is to limit the total number of distinct IP addresses contacted (denote the limit as M_C)

[1]Branching process approximations to stochastic epidemic processes have been studied rigorously. A tutorial on this topic can be found in [4].

per host over a period we call the *containment cycle*, which is of the order of weeks or months. We show that the value of M_C does not need to be as carefully tuned as in the traditional rate control mechanisms. Further, we show that this scheme will have only marginal impact on the normal operation of the networks. Our scheme is fundamentally different from rate limiting schemes because we are not bounding instantaneous scanning rates.

Preference scanning worms are a common class of worms but have received significantly less attention from the research community. Unlike uniform scanning worms, this type of worm prefers to scan random IP addresses in the local network to the overall Internet. We show that a direct application of the containment strategy for uniform scanning worms to the case of preference scanning worms makes the system too restrictive in terms of the number of allowable scans from a host. We therefore propose a local worm containment system based on restricting a host's total number of scans to local unused IP addresses (denoted as N). We then use a stochastic branching process model to come up with a bound on the value of N to ensure that the worm spread is stopped.

The main contributions of the paper are summarized as follows. We provide a means to accurately model the early phase of propagation of uniform scanning worms. We also provide an equation that lets a system designer probabilistically bound the total number of infected hosts in a worm epidemic. The parameter that controls the spread is the number of allowable scans for any host. The insight from our model provides us with a mechanism for containing both fast scanning worms and slow scanning worms without knowing the worm signature in advance or needing to detect whether a host is infected. This scheme is non-intrusive in terms of its impact on legitimate traffic. Our model and containment scheme are validated through analysis, simulation, and real traffic statistics.

The rest of the paper is organized as follows. In Section 2.2, we review relevant research on network worms. In Section 2.3, we present our *branching process* model with corresponding analytical results on the spread of the infection. In Sections 2.4

and 2.5, we describe an automatic worm containment scheme for random scanning worms and adaptation to the case of local preference scanning worms. In Section 2.6, we provide numerical results that validate our model and confirm the effectiveness of our containment scheme. In Section 2.7, we summarize our contributions and provide some discussion and directions for future work.

2.2 Related Work

As mentioned in the introduction, deterministic epidemic models have been used to study worm propagation [26, 29]. For illustration, consider the two-factor worm model proposed by Zou *et al.* [29]:

$$\frac{dI(t)}{dt} = \beta(t)[V - R(t) - I(t) - Q(t)]I(t) - \frac{dR(t)}{dt}, \tag{2.1}$$

where V is the total number of susceptible hosts on the Internet, and $I(t), R(t), Q(t)$ represent the number of infectious hosts, the number of removed hosts from the infectious population, and the number of removed hosts from the susceptible population at time t, respectively. The parameter $\beta(t)$ is the infection rate at time t and reflects the impact of the Internet traffic on the worm propagation. The parameters $R(t)$ and $Q(t)$ reflect the human countermeasures in patching.

When there is no patching and when the infection rate is constant, the two factor model equation is the random constant spread model (RCS) proposed by Staniford *et al.* [26]:

$$\frac{dI(t)}{dt} = \beta I(t)(V - I(t))$$

These types of models are suitable when there are a large number of infected hosts. However, during the early stage of the worm propagation, the number of infected hosts is small and such deterministic models may not accurately characterize the spread of worms. Nonetheless, most existing models for Internet worms are based on deterministic epidemic models.

Early worm detection systems have been proposed by several researchers. Zou *et al.* use a Kalman filter [30] to detect the worms. The Kalman filter is used to detect the presence of a worm by detecting the trend, not the rate, of the observed illegitimate scan traffic. The filter is used to separate worm traffic from background non-worm scan traffic. Liljenstam *et al.* [19] and Berk [5] develop an early worm detection system called DIB:S/TRAFEN in which a select group of routers forward ICMP T-3 packets to the analysis station. It is shown in [19] that the total number of participating routers can be small, but these routers must be distributed across a significant fraction of the Internet address space to ensure timely and accurate worm detection. They develop a worm simulation model that is used for generating worm traffic for evaluating the DIB:S/TRAFEN detection system. Their simulation model uses a combination of the deterministic epidemic model and a general stochastic epidemic model to model the effect of large scale worm attacks. They found the stochastic epidemic model to be useful for modeling the early stage of the worm spread. However, the complexity of the general stochastic epidemic model makes it difficult to derive insightful results that could be used to contain the worm.

Rate-control based countermeasures, such as *Virus throttling* by Williamson [27], have been shown to be successful in detecting and slowing down fast scanning worms. Wong *et al.* [28] studied the effect of rate control on suppressing the spread of the worms when this mechanism is deployed at various points (e.g., host, LAN, and core router) of the network. The rate control is effective in slowing down fast worms but is not effective against slow scanning worms. In addition, the limit on the rate must be carefully tuned in order to let the normal traffic through.

Zou *et al.* propose and analyze a dynamic quarantine scheme for Internet worms [31]. They assume that the underlying worm detection system has a certain false alarm rate. Their system confines all hosts that have triggered the alarm, and automatically releases them after a short time. They found that this scheme can slow down the worm spread but cannot guarantee containment.

Moore *et al.* [23] examined the reaction time required to contain Internet scale worms using countermeasures such as blacklisting the infected hosts and content filtering at routers. Their study concluded that to effectively contain Internet worms, it is necessary to take actions early, within minutes of the worm outbreak.

Ganesh *et al.* investigated how topology affects the spread of an epidemic by modeling the epidemic spread as a contact process in a finite undirected graph [9]. Most recently, Ganesh *et al.* proposed an interesting framework [10] to examine the impact of the worm countermeasures, such as worm throttling and worm quarantine, on the spread of the worms. The interaction between the worm detection strategy and worm propagation is viewed as a game. Based on their analysis, they designed optimal detection rules against scanning worms and further proposed methods of coordinating information among the end hosts to speed up the detection using Bayesian decision theory.

Another approach to worm detection is based on *Sequential Hypothesis Testing* [13, 37, 41]. To determine if a host is infected, a sequence of connection attempts, both successful and failed, is examined to validate the null hypothesis that the host is not infected. If a host is deemed infected, the scans from it are suppressed. The shortcoming is that the infected hosts may avoid detection if they fake more successful connections.

2.3 Branching Process Model for Random Scanning Worms

We now present the branching process model we use to characterize the propagation of random scanning worms. Scanning worms are those that generate a list of random IP addresses to scan from an infected host. The uniform scanning worms are those in which the addresses are chosen completely randomly while preference scanning worms weight the probability of choosing an address from different parts of the network differently. In this paper, we first describe the approach for uniform scanning worms, and then we present the extension for preference scanning worms. In our model, a host under consideration is assumed to be in one of three states: *susceptible, infected*, or *removed.* An infected host generates a list of random IP address to scan. If a susceptible host is found among the scans, it will become infected. A removed host is one that has been removed from the list of hosts that can be infected.

We use V to denote the total number of initially vulnerable hosts. The initial probability of successfully finding a vulnerable host in one scan is $p = \frac{V}{2^{32}}$, where 2^{32} is the size of current IPv4 address space. We call p the density of the vulnerable hosts, or *vulnerability density*.

We use M to denote the total number of scans from an infected host. This M is the "natural" limit of the worm itself, and is always finite during a finite period of time. For example, when a worm scans 6 IPs per second, it will scan $M = 518,400$ times in a day. We will characterize the values of M that ensure extinction of a worm in the Internet, and provide the probability distribution of the total number of infected hosts as a function of M. To that end, we first describe our branching process model.

2.3.1 Galton-Watson Branching Process

The Galton-Watson Branching process[2] is a Markov process that models a population in which each individual in generation n independently produces some random number of individuals in generation $n+1$, according to a fixed probability distribution that does not vary from individual to individual [14, 25].

All infected hosts can be classified into generations in the following manner. The initially infected hosts belong to the $0 - th$ generation. All hosts that are directly infected by the initially infected hosts are 1^{st} generation hosts, regardless of when they are infected. In general, an infected host H_b is an $(n + 1)$-st generation host if it is infected *directly* by a host H_a from the n-th generation. H_b is also called an offspring of H_a. All infected hosts form a tree if we draw a link between a host and its offspring. Figure 2.3.1 illustrates the notion of generation-wise evolution. In this model, there is no direct relationship between generation and time. A host in a

[2]Branching process models have already been successfully used in modeling the spread of infectious diseases in the early phase of the outbreak [4].

higher generation may precede a host in a lower generation, as host D (generation 2) precedes host B (generation 1) in Figure 2.3.1 ($t(D) < t(B)$). Figure 2.3.1 illustrates the Code Red propagation in the early stage showing the growth of the number of infected hosts of the first 6 generations.

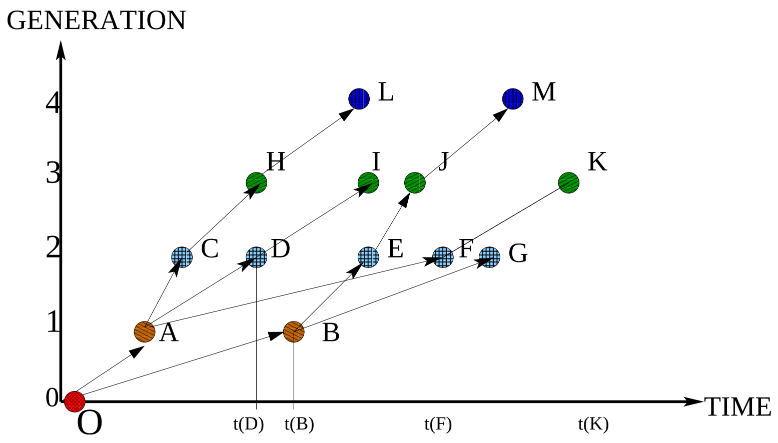

Fig. 2.1. Generation wise evolution in a tree structure, with O the initially infected host. O has two offsprings: host A and host B.

Let ξ be the random variable representing the offsprings of (i.e., the number of vulnerable hosts infected by) one infected host scanning M times. During the early phase of the propagation, the vulnerability density p remains constant since the number of infected hosts is much smaller than the number of vulnerable hosts in the population. Thus, during the initial phase of the worm propagation, ξ is a $binomial(M, p)$ random variable. Hence,

$$P\{\xi = k\} = \binom{M}{k} p^k (1-p)^{M-k}, \qquad k = 0, 1, \cdots, M. \tag{2.2}$$

Let I_n be the number of infected hosts in the n-th generation. I_0 is the number of initial hosts that are infected. During early phase of worm propagation, each infected host in the n^{th} generation infects a random number of vulnerable hosts, independent of one another, according to the same probability distribution. These newly infected hosts are the $(n+1)^{st}$ generation hosts. Let $\xi_k^{(n)}$ denote the number of hosts infected by the k^{th} infected host in the n^{th} generation. The number of infected hosts in the

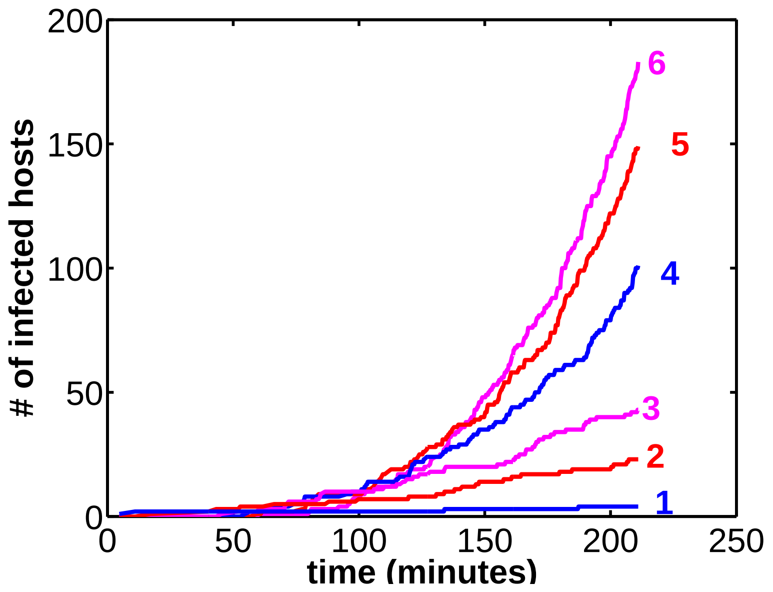

Fig. 2.2. Growth of the infected hosts in generations. The generation number is shown next to the growth curve.

$(n+1)^{st}$ generation can be expressed as $I_{n+1} = \sum_{k=1}^{I_n} \xi_k^{(n)}$, where $\xi_k^{(n)}$ are independent $binomial(M,p)$ random variables.

During the initial worm epidemic, each infected host produces offsprings independently and according to the same probability distribution as in Equation (2.2). Therefore, the spread of infected hosts in each generation $\{I_n, n \geq 0\}$ forms a branching process. The branching process accurately models the early phase of worm propagation. When the worm propagation is beyond the early phase, the branching process model gives an upper bound on the spread of worms. This is because the vulnerability density is smaller and the probability of two infected hosts scanning the same vulnerable host is higher later on. Thus, if we can ensure that the branching process does not spread, it also ensures that the worm propagation is contained.

For convenience, we provide a list of the symbols used in our model and their corresponding definitions in Table 2.1.

We next use the branching process model to answer questions on how the worm propagates as a function of the total number of allowable scans M.

Table 2.1
Notations

Symbol	Explanation
V	The size of the uninfected host population.
p	The vulnerability density, specified in terms of entire network address space, used and unused. e.g. $p = \frac{V}{2^{32}}$ for IPv4.
M	The number of scans from a host
M_C	The maximum number of scans allowed from a host within a containment cycle.
ξ	Random number of offsprings generated by each infected host
$\xi_k^{(n)}$	Number of offsprings produced by k^{th} host in n^{th} generation
I_0	Number of initially infected hosts
I_n	Number of n^{th} generation infected hosts
I	Total number of all infected hosts $[I = \sum_{n=0}^{\infty} I_n]$
π	Extinction probability
P_n	Extinction probability at n^{th} generation, i.e.,$P[I_n = 0]$

2.3.2 Extinction Probability for Scanning Worms

We model worm propagation as a branching process $\{I_n\}_{n=0}^{\infty}$, where I_n is the number of infected hosts in the n^{th} generation. The *extinction probability* of a branching process $\{I_n\}_{n=0}^{\infty}$ is the probability that the population dies out eventually. It is defined as

$$\pi = P\{I_n = 0, \quad \text{for some } n\} \tag{2.3}$$

When the branching process model is used for the spread of random scanning worms, the extinction probability measures the likelihood of the worm dying out after a certain number of generations. When $\pi = 1$, we are certain that the infections from the worm cannot be spread for an arbitrarily large number of generations. The following *Proposition* provides the necessary and sufficient condition for worm extinction.

Theorem 2.3.1 *Let $\{I_n\}_{n=0}^{n=\infty}$ be a branching process model for worm propagation, where I_n models the number of infected hosts in the n^{th} generation. Let p be the density* *of vulnerable hosts and M be the total number of scans attempted by an infected host. Then $\pi = 1$ if and only if $M \leq \frac{1}{p}$.*

Proof: According to Theorem 4.5.1 in [25], the extinction probability of a branching process is 1 if and only if the expected number of offsprings from each individual is no more than 1.

Let ξ be the random variable representing the number of offsprings produced by an infected host. ξ is a Binomial random variable with parameters (M, p). The expected value of ξ is $E(\xi) = Mp$. By Theorem 4.5.1 in [25], $\pi = 1$ if and only if $E(\xi) \leq 1$. Therefore, $\pi = 1$ if and only if $M \leq \frac{1}{p}$. ■

Since $E[\xi] = Mp$, Mp is the *basic reproduction number*, R_0, in the epidemiology literature.

Using *Code Red* and *SQL Slammer* as examples, if M is no more than 11,930 and 35,791 respectively, the worms would eventually die out[3]. The value of M corresponds to the number of unique addresses that can be contacted and, therefore, the restriction on M is not expected to significantly interfere with normal user activities. This is borne out by actual data for traffic originated by hosts at the Lawrence Berkeley National Laboratory and Bell Labs presented in Section 2.4.

If M is different for different hosts, we can use multi-type branching processes to obtain conditions under which the extinction probability is 1. If r and $1 - r$ are the fractions of hosts that scan M_1 and M_2 times, respectively, the extinction probability is 1 if $M_1 r + M_2(1 - r) < 1/p$.

2.3.3 Probability Distribution of Total Infections

While the probability of extinction gives us a bound on the maximum number of allowable scans per host, the true effectiveness of a worm containment strategy is measured by *how many hosts are infected before the worm finally dies out.* We next provide a probability density function for the total number of infections when

[3]V=360,000 for Code Red, and V=120,000 for SQL Slammer are used in this calculation.

the total number of scans per host is below $1/p$. The probability density function is applicable only when the total number of scans per host is below $1/p$. It has as a parameter M, which is typically kept below $1/p$.

The total number of infections, denoted by I, is the sum of the infections in all generations ($I = \sum_{n=0}^{\infty} I_n$). Our objective is to provide a simple closed-form equation that accurately characterizes $P\{I = k\}$, the probability that the total number of hosts infected is k, for a given value of M.

We consider any uniform scanning worm with I_0 initially infected hosts. We allow all hosts to scan $M \leq 1/p$ times, where the vulnerability density is p, and the total number of infected hosts is $I = \sum_{n=0}^{\infty} I_n$. As stated earlier, $\{I_n\}$ is a branching process. The infected hosts independently infect a random number of vulnerable hosts that obeys the same probability distribution as ξ. Since the total number of scans per infected host is M, ξ is a binomial random variable $B(M, p)$. Further, since p is typically small in practice (e.g., $p \approx 8.4 \times 10^{-4}$ for Code Red), and M is typically large, the probability distribution of ξ can be accurately approximated by a Poisson distribution with mean $\lambda = Mp$. Hence, the probability density function for ξ is:

$$P\{\xi = k\} \approx e^{-\lambda} \frac{(\lambda)^k}{k!}.$$

It then follows directly from [7] that the total progeny of the branching process has a Borel-Tanner distribution, i.e.,

$$P\{I = k\} = \frac{I_0}{k(k - I_0)!} (k\lambda)^{(k-I_0)} e^{-k\lambda}, \qquad k = I_0, I_0 + 1, \cdots \tag{2.4}$$

where $\lambda = Mp$. The mean and variance of I are given by [7]:

$$E(I) = \frac{I_0}{1 - \lambda} \qquad VAR(I) = \frac{I_0}{(1 - \lambda)^3}$$

More importantly,

$$p_k = P\{I \leq k\} = \sum_{i=I_0}^{k} P\{I = i\} = \sum_{i=I_0}^{k} \frac{I_0}{i(i - I_0)!} (i\lambda)^{(i-I_0)} e^{-i\lambda}, \qquad k = I_0, I_0 + 1, \cdots \tag{2.5}$$

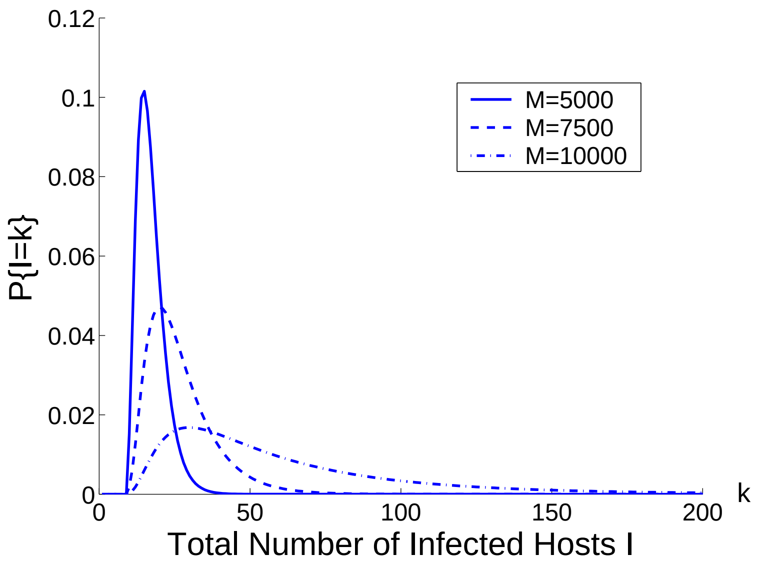

Fig. 2.3. Probability Density of I, the total number of infected hosts.

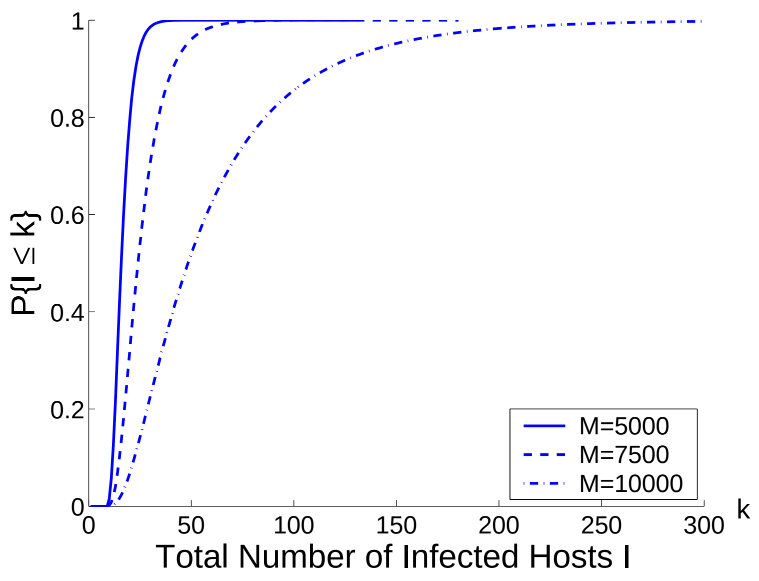

Fig. 2.4. Cumulative Distribution of I, the total number of infected hosts.

Equation (2.5) for $p_k = P\{I \leq k\}$ is important because it probabilistically determines the spread of the worm for a given limit on the number of scans M. Thus, p_k can be used to design a containment strategy that can be tuned to specifications.

Consider Code Red with 10 initially infected hosts, as used in [31]. If we would like $p_{360} = P\{I \leq 360\} = 0.99$, we could use equation (2.5) to set $M = 10,000$ ($\lambda = Mp = 0.83$). That is, if we want the total number of infected hosts to be less than 360 with probability 0.99, we set $M = 10000$ to achieve this.

Figure 2.3.3 shows the plot of the probability density function of I for three different values of M for Code Red with 10 initial infections. Figure 2.3.3 plots the cumulative probability distribution of I for three different values of M for Code Red with 10 initial infections. As we can see from Figure 2.3.3, with high probability (0.95), Code Red will not spread to more than $150, 50$, and 27 total infected hosts if the values of M are chosen to be $10000, 7500$, and 5000, respectively.

We also consider the SQL Slammer worm with 10 initial infected hosts. If we use the same value for M ($M = 10000$), with high probability (0.97), no more than 20 vulnerable hosts (or 10 additional vulnerable hosts) will be infected. This corresponds to 0.008% of the total vulnerable population. If we further reduce M to 5000, with high probability (0.97), no more than 4 additional vulnerable hosts will be infected.

Now, we compare this result to existing worm detection systems [19], which provide detection when approximately 0.03% (Code Red) and 0.005% (slammer) of the susceptible hosts are infected. This performance is achieved by careful selection of the routers at which worm detection mechanisms are put in place. With our scheme, when M is kept below a pre-defined threshold, with very high probability, the infection will not be allowed to spread that widely. Further, our results also hold for slow worms, which most other detection techniques, including [19], have trouble detecting.

Based on our analysis, we now develop an automatic worm containment system that can inhibit the spread of the worms.

2.4 Automated Worm Containment System

The results in Section 2.3 provide us with a blueprint of a *host based* worm containment strategy. The containment system is based on the idea of restricting the total number of scans to unique IP addresses by any host. We assume that we can estimate or bound the percentage of infected hosts in our system. Our proposed automated worm containment strategy has the following steps.

1. Let M_C be the total number of unique IP addresses that a host can contact in a *containment cycle*. At the beginning of each new containment cycle, set a counter that counts the number of unique IP addresses for each host to be zero.

2. Increment this counter for each host when it scans a new IP address.

3. If a host reaches its scan limit before the end of the containment cycle, it is removed and goes through a heavy-duty checking process to ensure that it is free of infection before being allowed back into the system. When allowed back into the system, its counter is reset to zero.

4. Hosts are thoroughly checked for infection at the end of a containment cycle (one by one to limit the disruption to the network) and their counters reset to zero.

Choose M_C to probabilistically bound the total number of infected hosts (I) to less than some acceptable value ϵ, (e.g. $p_\epsilon = P\{I \leq \epsilon\} \geq 0.99$), as given by Equation (2.5). Further, the containment cycle can be obtained through a learning process. For example, initially one could choose a containment cycle of a fixed but relatively long duration, e.g., a month. Since the value of M_C that we can allow is fairly large (on the order of thousands, as indicated by analysis with SQL Slammer and Code Red) we do not expect that normal hosts will be impacted by such a restriction. We can then increase (or, for the rare hosts, decrease) the duration of the containment cycle depending on the observed activity of scans generated by correctly operating hosts. Also, the containment cycle is determined to ensure that the number of scans from legitimate hosts is highly probable to be less than M_C. Since typical values of M_C are large (i.e., most legitimate hosts will not reach this value even over a month's time frame), the containment cycle is quite large. Typically, computer security patches are pushed out weekly or biweekly and machines are brought down during the patching process. For example, Purdue's engineering machines are brought down once a week. The worm containment cycle check can be done during this maintenance period.

The "heavy duty checking" in *step 3* could even include human intervention. Since the number of offending hosts is small, administrators should be able to take the machine offline and perform a thorough checking. The first step of the heavy-duty checking should be to follow a common security best-practice procedure. For example, one must make sure that the anti-virus software is up-to-date and is not disabled. One also needs to run a file integrity checker to make sure the critical files are not modified and no new executables are installed. After routine checking with all the available tools, an experienced system administrator should be able to make a final decision as to whether or not to let this machine be back online. Figure 2.5 illustrates our worm containment scheme.

We use the 30 day trace of wide-area TCP connections (LBL-CONN-7) [33] originating from 1645 hosts in the Lawrence Berkeley Laboratory to analyze the growth of the number of unique destination IP addresses per host (this is clean data over a period when there was no known worm traffic in the network). Our study indicates that 97% of hosts contacted less than 100 distinct destination IP addresses during this period. Only six hosts contacted more than 1000 distinct IP addresses, and the most active host contacted approximately 4000 unique IP addresses. Figure 2.6 shows the growth trend of the total unique destination IP addresses for these six most active hosts.

We also analyze HTTP traffic using more recent trace data from NLANR [2]. The trace data consists of contiguous Internet access IP headers collected at Bell Labs during one week in May 2002. The network serves about 400 people. We find that there are three Web proxies that each have contacted a little over 20K Web servers. The Web servers being contacted are highly correlated. The total number of distinct Web servers contacted by all the three proxies is 46K, not 60K. This correlation

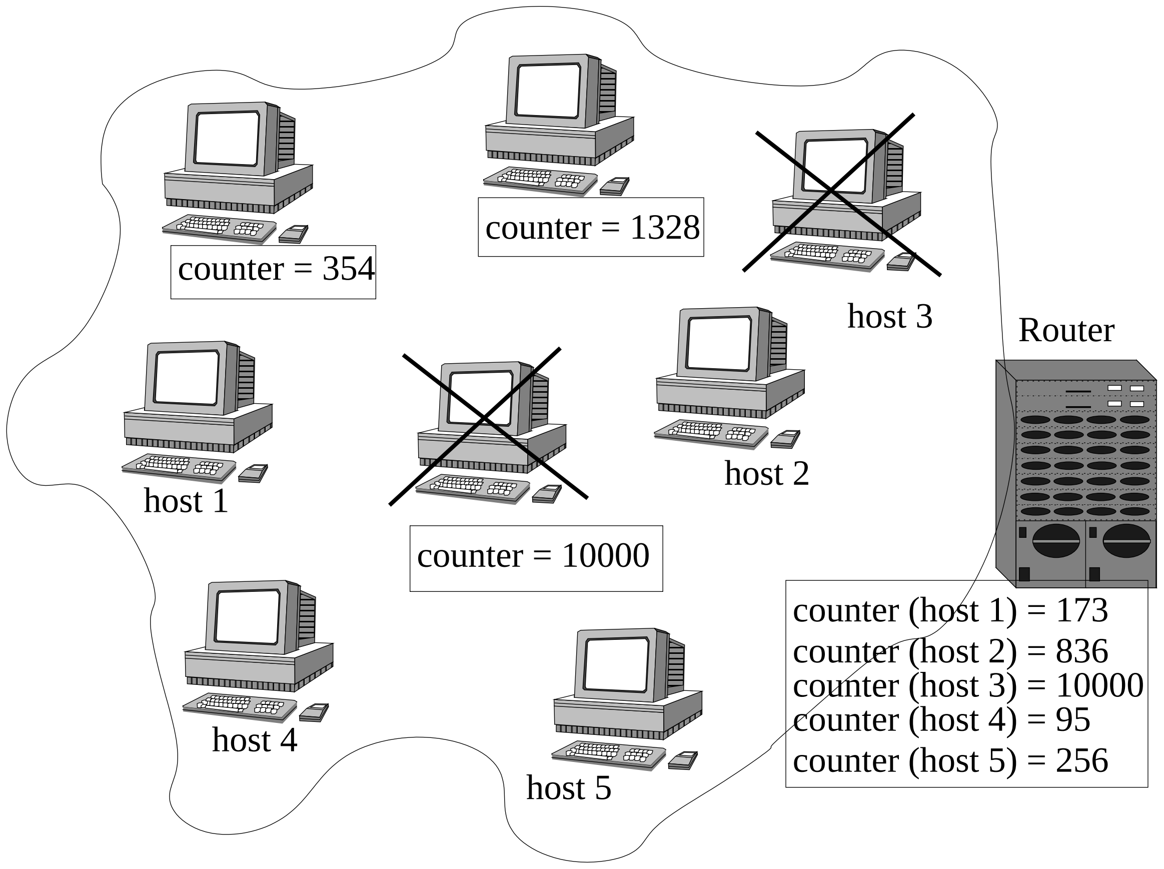

Fig. 2.5. Worm containment system for the uniform scanning worms. M_C is set to $10,000$. The total number of scans for each host is monitored. The monitoring system can be implemented on each host, or on the edge router of a local network. Two hosts marked are removed from the network automatically because their total number of scans (counter) have reached $10,000$.

may aid in reducing the storage requirement and correspondingly speeding the search performance when the unique destinations are maintained.

The data on the traffic between the internal hosts and the web proxies is not available. There are two other hosts which contacted between 5K and 6K Web servers. The rest of the hosts contacted only a few hundred Web servers. The histogram of those hosts (excluding the three proxies and two others mentioned) are shown in Figure 2.4.

If our containment system is used with the containment cycle to be one week and M_C is set to be 5000, none of the above hosts (except for the five mentioned above) will trigger an alarm. As shown in section 2.3, with high probability the total infections caused by Code Red will be under 27 hosts when $M = 5000$. *This suggests that our containment system is not likely to interfere significantly with normal traffic, yet it contains the spread of the worms.* If our scheme is implemented in a network with proxies, the best solution would be to have the proxies do the counting of scans on an individual host basis.

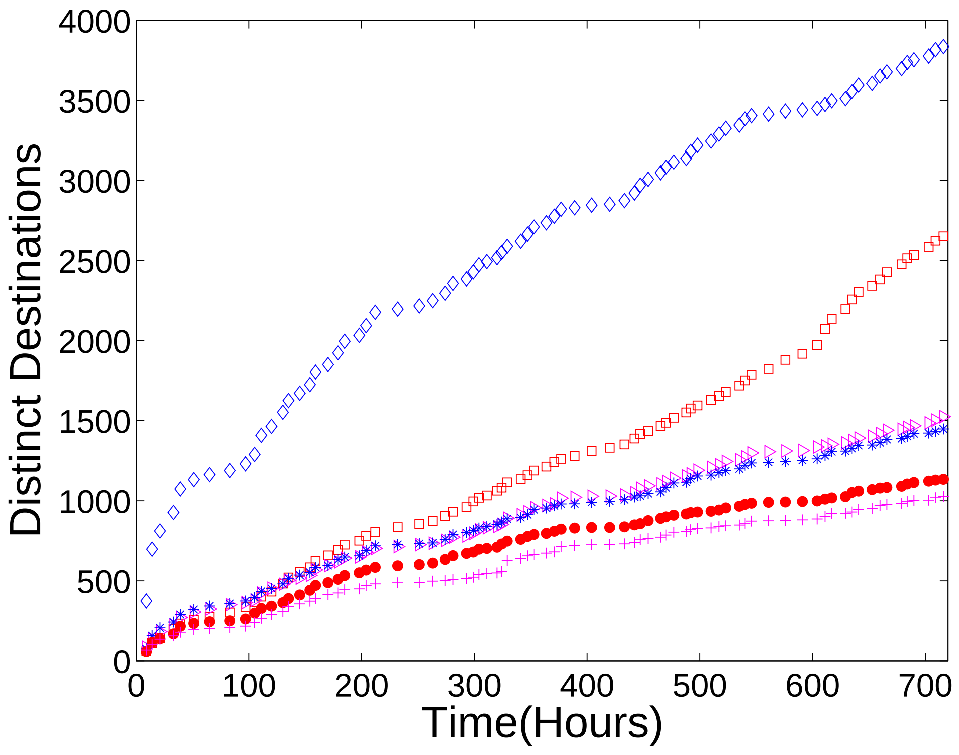

Fig. 2.6. Number of Distinct IPs contacted over 30 days for the six most active hosts (LBL trace data)

The containment cycle can also be adaptive and dependent on the scanning rate of a host. If the number of scans originating from a host gets close to the threshold,

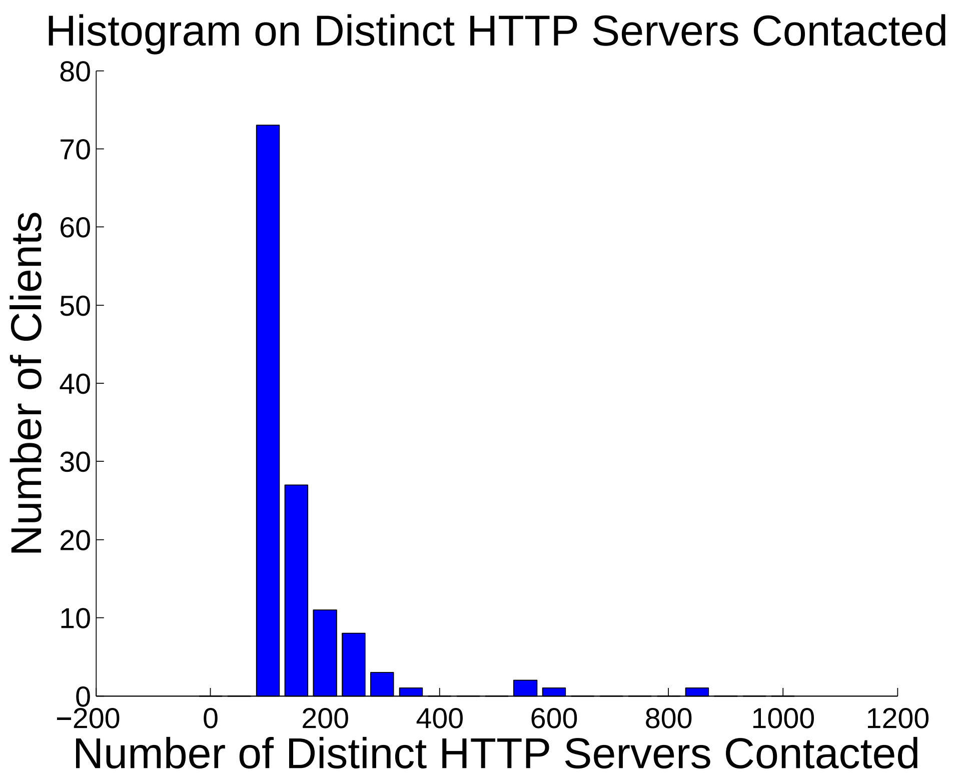

Fig. 2.7. Histogram of Distinct HTTP Servers contacted over one week, excluding the three proxies and two other active hosts with 5K to 6K destinations (Bell Lab trace data).

say it reaches a certain fraction f of the threshold, then the host goes through a complete checking process. The advantage of this worm containment system is that it does not depend on diagnosis of infected hosts over small time-granularities. It is also effective in preventing an Internet scale worm outbreak because the total number of infected hosts is extremely low, as shown in the examples in the previous section.

Traditional rate based techniques attempt to limit the spread of worms by limiting the scanning rate. The limit imposed must be carefully tuned so as to not interfere with normal traffic. For example, the *rate throttling* technique [27] limits the host scan rate to 1 scan per second. The rate limiter can inhibit the spread of fast worms without interfering with normal user activities. However, slow scanning worms with scanning rate below 1 Hz and stealth worms that may turn themselves off at times will elude detection and spread slowly.

In contrast, our worm containment system can contain fast worms, slow worms, and stealth worms. The fast scanning worms will reach the limit on M_C sooner, while the slow worms will reach this limit after a longer period of time. As long as the host is disinfected before the threshold is reached, the worm cannot spread in the Internet. This strategy can effectively contain the spread of uniform scanning worms in local networks. When the *global* total number of infected hosts is low, the number of infected hosts in any local networks must also be very low.

Our worm containment scheme can be deployed at end hosts and edge routers. When it is deployed at routers, special care must be taken when the network uses proxies or NAT. The counting of distinct destination IPs for each host is best placed at the proxies or inside the NAT. When the worm containment system is deployed outside NAT and proxies, a higher limit on M_C can be allocated based on the normal traffic patterns. Deploying our scheme at routers also requires protection against spoofing of IP addresses in order to correctly identify the offending hosts. The host's MAC address and IP address should be used together for identifying the offending host. Commercial products , such as *CounterPoint* by Mirage Networks, uses a host's MAC address for identifying an infected host.

In the future, we would like to design and implement our worm containment module at the edge routers. We plan to use real trace data to test the performance, usability and scalability of our scheme. Meanwhile, we are also interested in evaluating the deployment of our scheme at strategically positioned routers. Defending against worms that exploit high vulnerability density is another interesting topic for future research.

2.5 Local Preference Scan Worms

Local preference scanning (LPS) worms, such as Code Red II, scan more intensely in local networks. Code Red II scans a completely random address only 1/8 of the time. It scans the the same /16 network 1/2 of the time, and it scans the same /8 network 3/8 of the time. When vulnerable hosts are more dense in the local networks, the LPS worms spread much faster in the enterprise network. The faster spread is the result of the following two factors. One is that LPS worms scan the local network thousands of times more than do uniform scanning worms. The other factor is the higher vulnerability density in the local network due to similar configurations among machines in a subnet.

In this section, we extend our worm containment system developed in Section 2.4 to contain LPS worms in enterprise networks by removing the infected hosts in a timely manner. In order to prevent DOS attacks using IP spoofing, we need to use the host's MAC address and IP address to identify the offending host before it is taken off line, or use anti-spoofing software with our scheme. We show that our LPS containment scheme prevents the worm from spreading inside the local networks. Moreover, we provide analysis and simulations of global containment of LPS worms. When the LPS worm containment scheme is deployed in an enterprise network with traditional firewalls around the enterprise network boundary, worm containment inside the local network can be achieved without the requirement of participation and coordination from outside networks. Our simulations show that when this scheme is partially deployed, the LPS worm is likely to be contained on a global scale when the initially infected hosts are in the protected networks. Our analysis and simulation shows that when this scheme is deployed 100%, we can achieve not only local containment, but also global containment.

Our approach relies on the fundamental argument that there naturally exist large swaths of unused IP address space in today's IPv4 infrastructure (roughly 25% of the address space is used). The legitimate hosts are unlikely to scan multiple times to the unused addresses and therefore the number of such scans can be used to estimate the number of *worm* scans. When the number of worm scans exceeds the limit due to the epidemic theory, the machine is quarantined to prevent the spread of the worm.

This approach is also used in LaBrea [18], DIB/TRAN [19], and the worm monitoring system proposed by Zou et al. [30]. Network Intrusion Detection System Bro [24] can also be configured to perform dark-address detection. Commercial worm containment systems such as CounterPoint by Mirage Networks [20] and CounterACT by Forescout [12] also use the dark-address scan detection approach. In the dark-address detection approach, a worm infection is declared when a host has attempted to connect to the dark-addresses a number of times (denoted by N).

It is generally recognized that the choice of N is important - too small an N value will result in high number of false alarms, while too large an N value will result in a detection delay, allowing the worms to spread. Our contribution is that we provide an upper bound on N to guarantee worm containment. N is given in terms of the local vulnerability density p_1 and the dark-address density u. To contain worms, we require $N \leq \frac{u}{p_1}$.

2.5.1 Model

As in our model for uniform scanning worms, we take a host-centric view of worm propagation. We consider an infected host along with the /16 network and the /8 network to which it belongs. Let p_1, p_2, and p_3 be the vulnerability densities of the /16 network, /8 network, and the Internet respectively. Let M_1, M_2, and M_3 be the total number of scans by an infected host in the /16 network, /8 network and the Internet. Let f_1, f_2 and f_3 be the fraction of times a worm scans the /16 network, the /8 network, and the Internet. The total number of scans is $M = M_1 + M_2 + M_3$ and $M_i = f_i M$ for $i = 1, 2, 3$.

Let ξ_1, ξ_2, and ξ_3 be the number of offsprings an infected host produces in the /16 network, the /8 network, and the Internet when it scans M times. The random variables ξ_i are binomially distributed with parameters (M_i, p_i), $i = 1, 2, 3$. Moreover, $E[\xi_i] = M_i p_i = M f_i p_i$. The total number of offsprings ξ produced by an infected host is then $\xi = \xi_1 + \xi_2 + \xi_3$, and $E[\xi] = M(f_1 p_1 + f_2 p_2 + f_3 p_3)$. When the vulnerable hosts are much denser inside the internal networks (i.e. $p_1 >> p_3$ and/or $p_2 >> p_3$), LPS worms pose a more serious threat to the internal networks than to the rest of the Internet. This is because a single infected host produces far more offsprings inside local networks.

Our worm containment system for uniform scanning worms restricts the total number of outgoing connections to below $M \leq 1/p$. If we adopt the same scheme to contain LPS worms in an enterprise network consisting of several /16 networks, in each /16 network, M_1 needs to be considerably smaller because of the higher vulnerability density p_1 in the network. In a /16 network with 600 vulnerable hosts ($p_1 \approx 0.01$), the original containment scheme requires $M_1 \leq 100$, that is, no host is allowed to contact more than 100 distinct destinations in that /16 network. It is over-restrictive because some benign hosts may in fact contact 100 distinct IP addresses in the local network over a period of time.

In the following, we will show how to extend our worm containment system to contain LPS worms in enterprise networks. We use a /16 network to illustrate our new scheme.

2.5.2 LPS Worm Containment System Analysis

Our LPS worm containment system limits the total scans a host can make to the dark-address space. In a containment cycle (weeks or months), if a host scans the dark-addresses N times, it is automatically disconnected from the network for a thorough checkup. It is desirable that N be large so that normal activities will not be disrupted, while still containing LPS worms effectively.

If a host is infected, its total number of scans to dark-addresses N provides essential information on the total number of worm scans $M_{1,w}$ and the number of offsprings

ξ_1 it produces. To prevent the worms from spreading, we must limit the expected number of offsprings to below 1. This can be achieved by limiting N, the total number of dark-address scans per host. We provide the probability distribution of $M_{1,w}$ and ξ_1 for a given value of N in the following proposition.

Theorem 2.5.1 *Let u be the density of the dark IP addresses and p_1 be the vulnerable density in an enterprise network ($u + p_1 \leq 1$). Suppose an infected host is removed from the network when it hits the dark-address the N^{th} time. Let $M_{1,w}$ be the total worm scans in the local network by an infected host and ξ_1 be the number of offsprings produced by a single infected host in the network. Then*

(1) $M_{1,w}$ is a negative binomial random variable having distribution:

$$P\{M_{1,w} = m\} = \binom{m-1}{N-1} u^N (1-u)^{m-N}, \qquad m = N, N+1, N+2, \cdots$$

and $E[M_{1,w}] = N/u$.

(2) ξ_1 is a negative binomial random variable having distribution:

$$P\{\xi_1 = k\} = \binom{k+N-1}{k} (\frac{p_1}{p_1+u})^k (\frac{u}{p_1+u})^N, \qquad k = 0, 1, 2, \cdots$$

and $E[\xi_1] = Np_1/u$.

Proof: (1) Suppose an infected host scans m times and is stopped at the N^{th} scan to the un-used IP addresses, $(m \geq N)$. The m^{th} scan falls into the un-used IP addresses. Among the first $m-1$ scans, it scans the un-used IP addresses $N-1$ times and scans the vulnerable and healthy hosts $m-N$ times. Thus,

$$P\{M_{1,w} = m\} = \binom{m-1}{N-1} u^N (1-u)^{m-N}, \qquad m = N, N+1, \cdots$$

Therefore, $M_{1,w}$ is a negative binomial random variable with mean $E[M_{1,w}] = N/u$.

(2) When the infected host is removed after N scans to the un-used IP addresses,

$$P\{\xi_1 = k | M_{1,w} = m\} = \binom{m-N}{k} (\frac{p_1}{1-u})^k (1 - \frac{p_1}{1-u})^{m-N-k}$$

$$\begin{aligned}
\text{Hence,} \quad P(\xi_1 = k) = & \sum_{m=N+k}^{\infty} P(\xi_1 = k | M_{1,w} = m) P(M_{1,w} = m) \\
= & \sum_{m=N+k}^{\infty} \binom{m-N}{k} (\frac{p_1}{1-u})^k (\frac{1-u-p_1}{1-u})^{m-N-k} \times \binom{m-1}{N-1} u^N (1-u)^{m-N} \\
= & \sum_{m=N+k}^{\infty} \binom{m-N}{k} \binom{m-1}{N-1} p_1^k (1-u-p_1)^{m-N-k} u^N \\
= & \binom{k+N-1}{k} p_1^k u^N \times \sum_{m=N+k}^{\infty} \binom{m-1}{k+N-1} (1-u-p_1)^{m-N-k} \\
= & \binom{k+N-1}{k} p_1^k u^N (p_1+u)^{-N-k} \\
= & \binom{k+N-1}{N-1} (\frac{p_1}{p_1+u})^k (\frac{u}{p_1+u})^N
\end{aligned}$$

Thus, ξ_1 is a negative binomial random variable with parameters $(N, \frac{u}{p_1+u})$, and $E[\xi_1] = \frac{Np_1}{u}$. ■

Proposition 2 shows that the number of worm scans $M_{1,w}$ is not affected by p_1, but the number of internal offsprings ξ_1 is a function of u, N, and p_1. Figure 2.8 shows the pmf of $M_{1,w}$ for $u = 0.5$ and $N = 20$. Figure 2.5.2 illustrates the pmf's of ξ_1 for the same u and N values as in Figure 2.8, with p_1 values 0.01, 0.03, and 0.05. In particular, when $p_1 = 0.05$, $E[\xi_1] = 2$. That is, if $u = 50\%$, $N = 20$, and there are roughly 3000 vulnerable hosts in a /16 network, one infected host on average infects 2 vulnerable hosts during the initial worm outbreak.

To prevent the worms from spreading, we will limit the expected number of offsprings per host $E[\xi_1]$ to below 1 by the extinction theorem for branching processes. Since $E[\xi_1] = Np_1/u$, we require $N \leq u/p_1$, We denote $N_{max} = \lfloor u/p_1 \rfloor$.

The local vulnerability density p_1 will decrease when more vulnerable hosts become infected. The above bound on N is derived using the vulnerability density in the early stage when it is the largest, so the bound N_{max} is still valid when more hosts become infected. Moreover, when this bound is enforced, the total number of the infected hosts would be small and a large fraction of vulnerable hosts is automatically protected from the LPS worms.

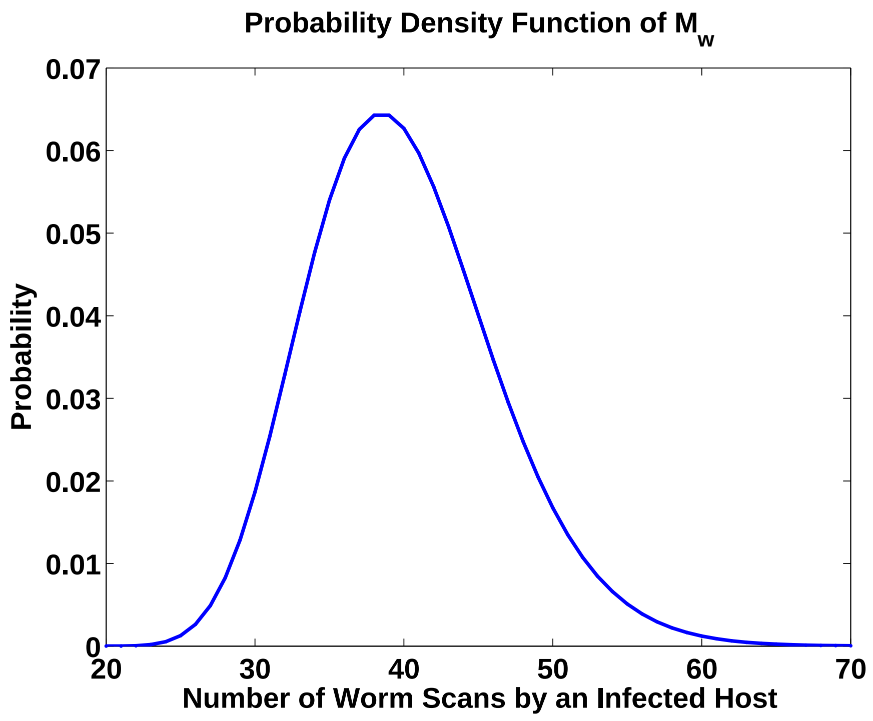

Fig. 2.8. Probability Density Function of $M_{1,w}$ (u=0.5, N=20).

Our next proposition provides analysis on the global impact of the LPS worm containment.

Theorem 2.5.2 *Let p_1, p_2, p_3 be the vulnerability densities for all /16 networks, /8 networks, and in the whole Internet respectively. The LPS worm scans /16 networks, /8 networks and the Internet f_1, f_2, and f_3 fraction of times, $f_1 + f_2 + f_3 = 1$ and $f_1 > f_2 > f_3$. Let ξ be the number of offsprings an infected host produces. Then $E[\xi] = \frac{N}{u}(p_1 + p_2\frac{f_2}{f_1} + p_3\frac{f_3}{f_1})$.*

Proof: Let ξ_2 be the number of offsprings an infected host produces in the /8 network but outside the /16 network. When the infected host scans the /16 network $M_{1,w}$ times, it scans /8 networks it belongs to $M_{1,w} \cdot (f_2/f_1)$ times.

$$\begin{aligned} E[\xi_2] &= p_2 E[\text{number of scans in /8 network}] \\ &= p_2 E[M_{1,w} \cdot \frac{f_2}{f_1}] = p_2 \frac{N}{u} \cdot \frac{f_2}{f_1} = \frac{Np_2}{u} \cdot \frac{f_2}{f_1} \end{aligned}$$

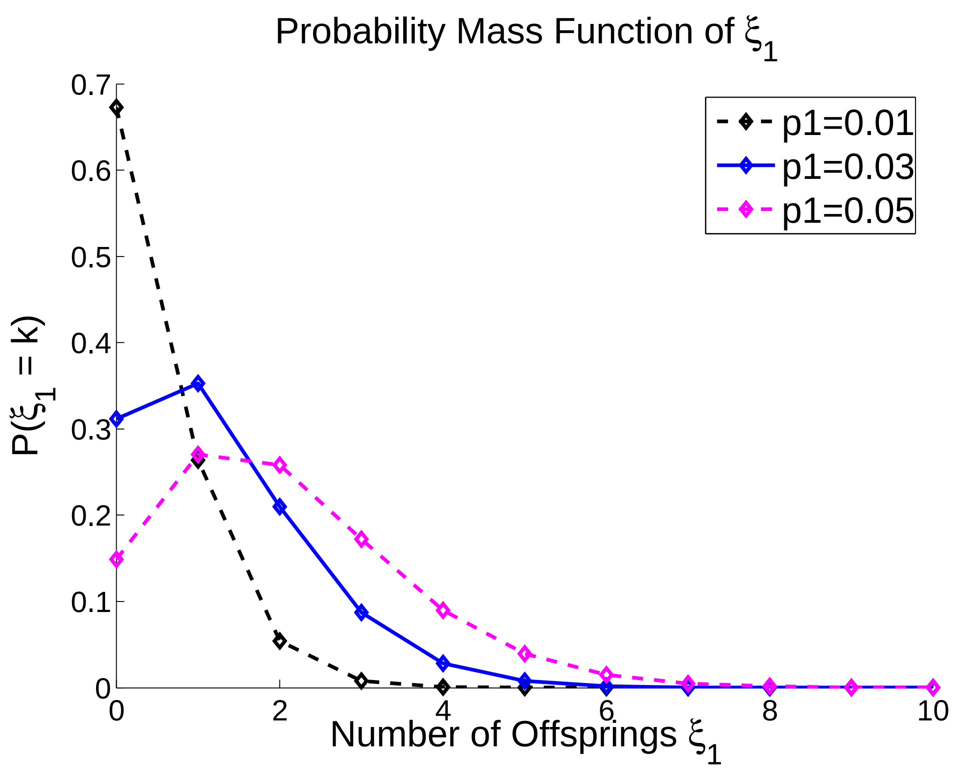

Fig. 2.9. Probability Density Functions of ξ_1 (u=0.5, N=20).

Similarly, the expected number of offsprings an infected host produces outside the /8 network is:

$$E[\xi_3] = \frac{Np_3}{u} \cdot \frac{f_3}{f_1}.$$

Therefore, the expected number of offsprings each infected host produces is:

$$E[\xi] = E[\xi_1 + \xi_2 + \xi_3] = \frac{Np_1}{u} + \frac{Np_2}{u} \cdot \frac{f_2}{f_1} + \frac{Np_3}{u} \cdot \frac{f_3}{f_1} = \frac{N}{u}(p_1 + p_2\frac{f_2}{f_1} + p_3\frac{f_3}{f_1})$$

■

When $E[\xi] < 1$, the LPS worm can be contained globally, and the average global total number of infected hosts is $E[I] = I_0/(1 - E[\xi])$. Proposition 3 also tells us that our LPS worm containment system cannot contain uniform scanning worms, because their scanning pattern has the following parameters: $f_2/f_1 = 256$ and $f_3/f_1 = 65536$.

This will result in a large number of infections outside the local networks. Therefore, we need both containment schemes to contain all scanning worms.

The scanning strategy of Code Red II has the following parameters: $f_1 = 1/2, f_2 = 3/8$ and $f_3 = 1/8$. Thus, $f_2/f_1 = 3/4$, $f_3/f_1 = 1/4$. Assume our networks have the following parameters: $p_1 = 0.02$, $p_2 = 10^{-3}$ and $p_3 = 10^{-4}$. If we use $N = 20, u = 0.5$ for LPS containment, then $E[\xi] = 40(0.02 + 10^{-3} \cdot 0.75 + 10^{-4} \cdot 0.25) = 1.101$. In this scenario, the worm will spread to the Internet. If the LPS containment system uses $N = 10, u = 0.5$, then the worm will be contained globally since $E[\xi] = 0.55 < 1$.

2.5.3 LPS Worm Containment System Deployment

The above analysis suggests that in order to protect local networks from worm infections, the local network addresses should be allocated in such a way that the ratio $\frac{u}{p_1}$ is larger than the number of dark-address connections expected from a normal host (denoted as N_t). Choosing N so that $N_t < N < \frac{u}{p_1}$ allows worm containment without intruding on normal-user activities.

To deploy the new worm containment system in an existing network, we need to know the parameters u, p_1, and N_t. Here, u is the density of dark-address space; p_1 is the estimated vulnerability density inside the network; N_t is a tolerable threshold value—a benign host will not mistakenly scan the dark addresses more than this value in a containment cycle. N_t is an optional parameter that may be set by the system administrator. The value of the local vulnerability density p_1 can be estimated based on the most common applications with open server ports. The value of u is known to the network administrator from the number of machines in the local network. Using u and p_1, we can calculate $N_{max} = \lfloor (\frac{u}{p_1}) \rfloor$. The actual threshold N for the containment system should be less than N_{max} to guarantee containment and greater than N_t to not interfere with normal user activities. When $N_{max} \geq N_t$, we can set $N = N_t$. However, when $N_{max} < N_t$, we could set $N = N_{max}$ and tolerate a higher rate of false alarms in order to contain the worms.

Our analysis shows that if a local network has very high vulnerability density, the dark-address based scheme will be too restrictive in the number of dark address space scans allowed. One solution for the high vulnerability density networks is to adopt the 24-bit block private address scheme. For example, networks utilizing private addresses 10.0.0.0/24 can accommodate 360,000 hosts with $u \approx 0.98$. If we make a conservative estimate on p_1 by assuming all these hosts are vulnerable, $p_1 \approx 0.02$. The value of $N_{max} = 49$.

We agree that an organization that has u/p_1 close to 1 and is unwilling to take measures (such as private address space) to increase the value, our scheme will not be useful. In general, local networks with dense and homogeneous software deployment are at a higher risk of worm infections. However, our simulation of incremental deployment shows that our scheme is beneficial even for those unprotected networks when the initial infection starts from the protected networks.

The goal of our worm containment system is to probabilistically contain the worms in a non-intrusive manner without explicitly detecting the infected hosts. Our scheme allows hosts to stay in the infected state for some time until their dark address scans reach the limit, and it still guarantees containment. This is fundamentally different from worm detections.

By combining traditional firewalls around the enterprise network boundary and our worm containment system within the enterprise network, worm containment can be achieved without the requirement of participation and coordination from outside networks. This allows for incremental deployment of the worm containment system for individual networks. Figure 2.10 illustrates the worm containment system for preference scanning worms.

A strong firewall at the enterprise network boundary is necessary to achieve worm containment inside the enterprise network. Consider an enterprise network consisting of a /16 network. Consider the situation when worms are spreading outside the network with more than 65K hosts already infected. One random scan from each infected host to the enterprise network will cause a significant fraction of vulnerable

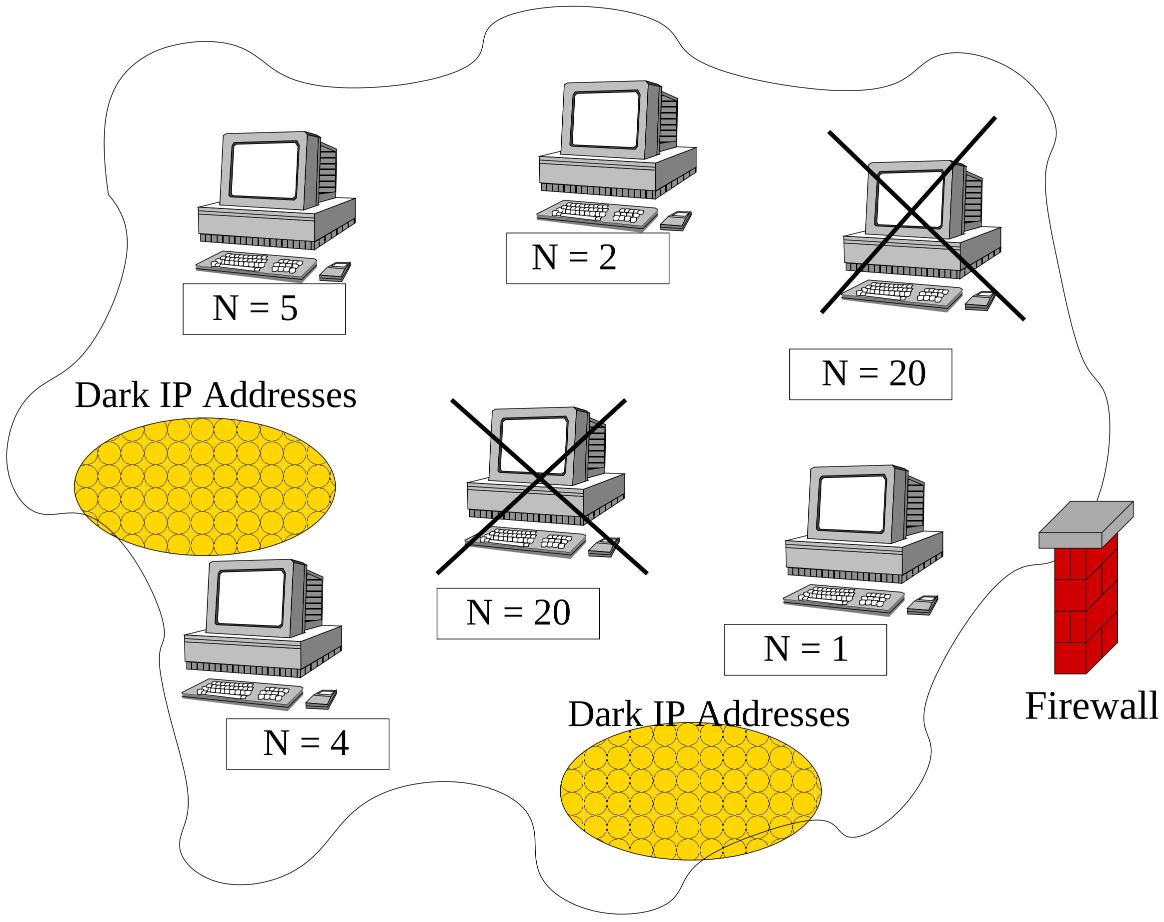

Fig. 2.10. Worm containment systems for the preference scan worms. The value N_{max} is chosen to be 20. Two hosts are disconnected from the network because their total dark address space scans has reached 20.

hosts to be infected. The first approach to counter the problem is to apply the same limit on the number of scans to unused address space to an external host as to an internal host. When the external host reaches this limit, a firewall rule is introduced to block any incoming scans from the suspected host. The challenges in creating accurate firewall rules (countering address spoofing, for example) are out of the scope of this paper and the reader is referred to [39] (pp. 340-351) for details. However, this approach fails if this suspected external host is not prevented from infecting other vulnerable external hosts. Failing that, more infected hosts are generated in the external network, which in turn scan to the internal network. Repeating the process of blacklisting the external hosts leads to the situation where all susceptible internal hosts are eventually infected. Since the spread of worms outside the networks cannot be controlled by an internal worm containment system, it suggests a complementary approach. The approach is that we restrict the number of hosts (servers) that are accessible from outside the enterprise network. All other hosts should be hidden behind the firewall and not accessible from outside networks. In the event of Internet scale worm outbreaks, the small number of externally accessible servers may be infected. However, these servers cannot spread the worms to the internal networks because the worm containment system will disconnect the servers when they become infected. Moreover, the threshold value N for the servers can be carefully tuned down to a smaller number to further restrict the worm propagation. The subsetting to create a few externally accessible servers seems feasible in many enterprise settings since only a few servers (e.g., web server) need be exposed to the external network.

It is challenging to contain hitlist worms. Hitlist worms create a target list of probable victims. However, as noted in [40], building the hit-list is non-trivial. A small hit-list can be compiled from readily accessible public sources, such as a metaserver that maintains a list of servers for different games (GameSpy is one example). Such a hit-list may accelerate a scanning worm, but is unlikely to be very damaging by itself. Building comprehensive hit-lists takes more effort. A distributed scan to find vulnerable hosts appears a likely strategy. The scans must be low rate to avoid detection. Since our worm containment system can contain the slow spreading worms, it will prevent the hitlist from being built through such low rate random scanning.

2.6 Simulation Results

We use a discrete event simulator to simulate scanning worm propagation with our defense strategies. We first simulate a *uniform scanning worm* with our containment scheme. In this simulator, each vulnerable host is assigned a IPv4 address randomly, and each will be in one of the three states: *susceptible, infected*, and *removed*. A host is in *removed* state if it has sent M scans. The *infected* hosts independently generate random IP addresses to find the victims. If the random IP address matches any of the IP addresses of the hosts in the *susceptible* state, the susceptible host will become infected.

In our simulation for Code Red, we used $V = 360,000$ for the vulnerable population size and $I_0 = 10$ for the number of initially infected hosts. We used $M = 10,000$, which is below the threshold required for worm extinction. In this case, $p = 8.38 \times 10^{-5}$ and $\lambda = Mp = 0.838$.

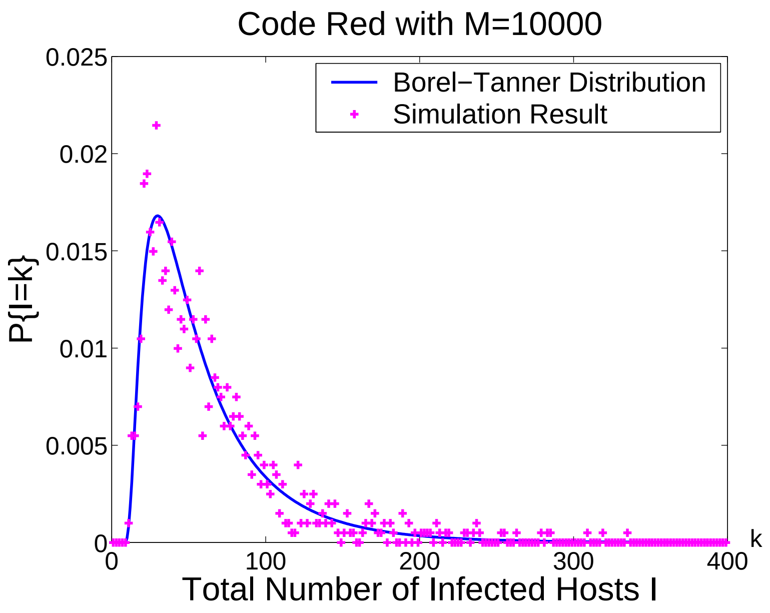

Fig. 2.11. Probability density for total number of infected hosts

As discussed earlier, the total number of infected hosts I measures how well a worm is contained. We ran this simulation 1000 times and collected the values of I. Figure 2.6 shows the relative frequency of I from our simulations and the probability density function of I obtained from our theoretical results in Section 2.3. Figure 2.12

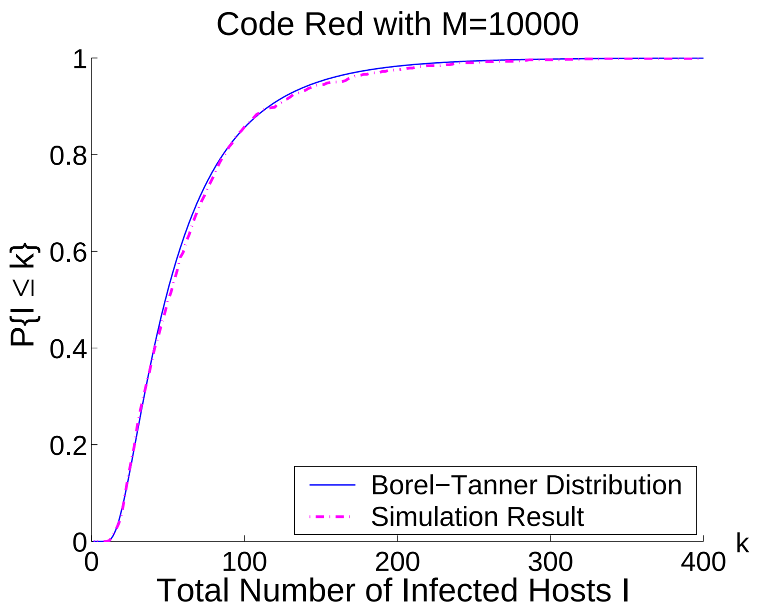

Fig. 2.12. Cumulative Distribution of total number of infected hosts

shows the relative *cumulative* frequency of I from our simulations and the *cumulative distribution function* of I from the theoretical analysis.

The simulation results validate the accuracy of our model and the effectiveness of our containment strategy. Figures 2.6 and 2.12 demonstrate that our simulation results match closely with the theoretical results from Section 2.3. We can see from Figure 2.12 that with high probability (0.95), the total number of infected hosts is held below 150 hosts. As mentioned in Section 2.3, one can reduce the spread of infection by further reducing the value of M.

To demonstrate the variability of worm propagation, we show two sample paths among our 1000 simulation runs in Figures 2.6 and 2.6. In one scenario depicted in Figure 2.6, there are a total of approximately 300 hosts infected. Figure 2.6 shows another scenario when there are 55 total infected hosts. In the first scenario, the active number of infected hosts (number infected - number removed) is held below 30 at all times. This is due to our countermeasure that when a host scans $M = 10,000$ times, it is removed. The worm ceased spreading after all infected hosts were removed.

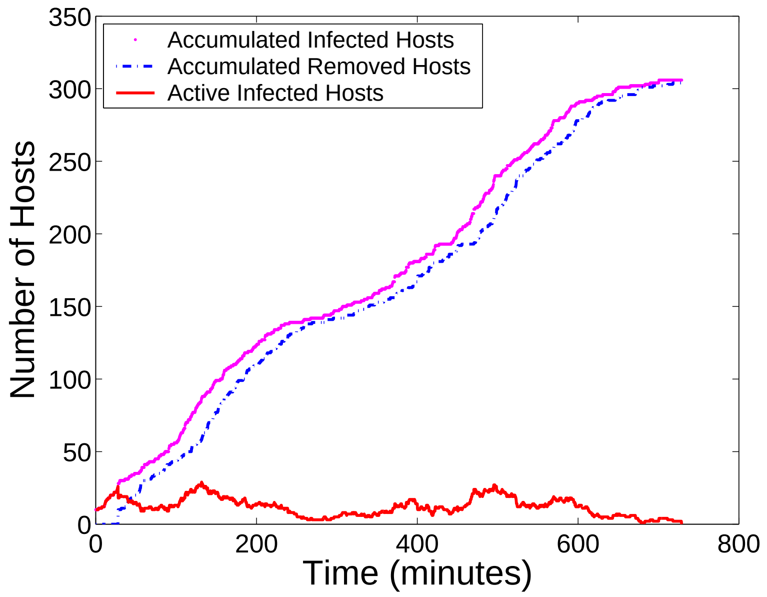

Fig. 2.13. Variability of Worm Propagation: Two Simulation Runs with Identical Parameters (Code Red).

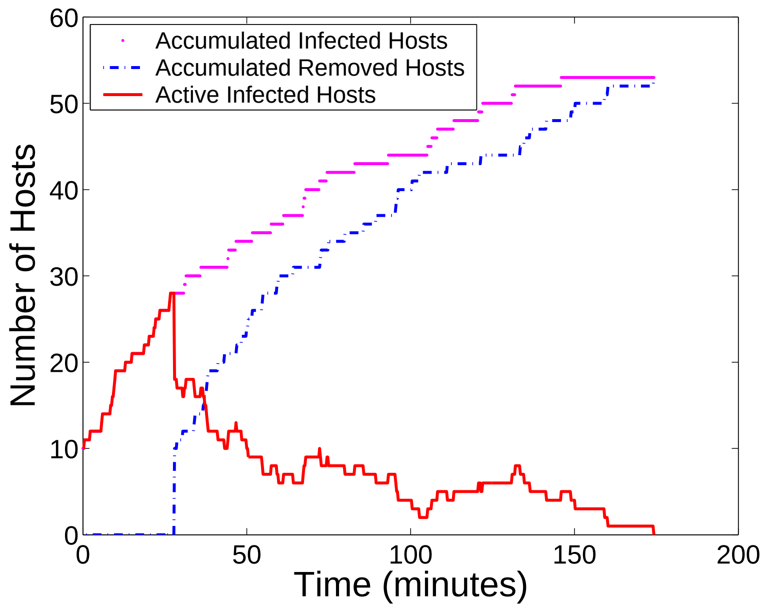

Fig. 2.14. Variability of Worm Propagation: Two Simulation Runs with Identical Parameters (Code Red).

In the second scenario, the removal process quickly catches the infection process, so that the worm dies out rapidly. We used $M = 10,000$ for both scenarios. Using formula provided in section 2.3, we obtain $E(I) = 58$ and $\sigma(I) = 45$. The variation is large and can be significant in the early stage, which will have significant effect in modeling the latter stage of growth of the worm. Stochastic models need to be used to capture this variation.

We used a scan rate of 6 scans/second for Code Red for the purpose of illustrating worm propagation and containment with respect to time. This scan rate is taken from the empirical study in [21]. Figure 2.6 shows that the worm dies out completely in 750 minutes. In the simulation run shown in Figure 2.6, the worm dies out in less than 200 minutes.

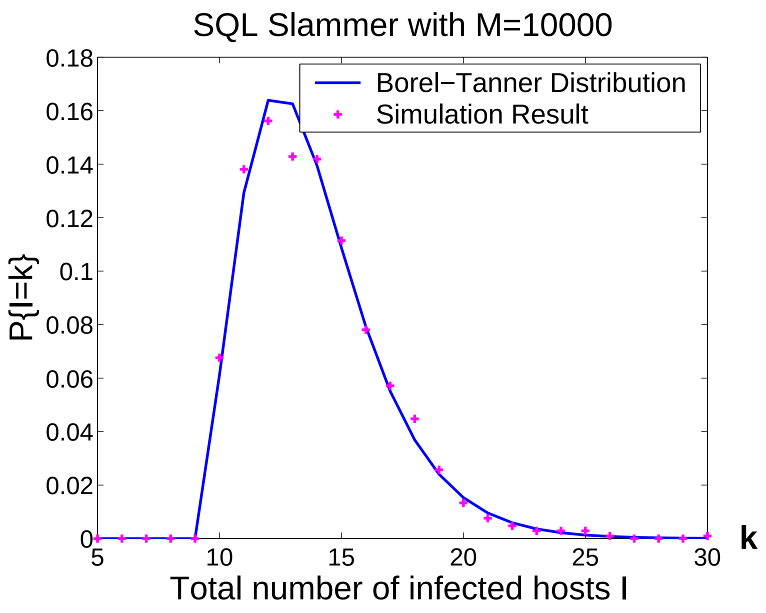

Fig. 2.15. Probability density and Cumulative distribution of I (SQL Slammer)

We also ran our simulations with SQL Slammer parameters. Here we used V=120,000 as used in [19], $I_0 = 10$, and $M = 10,000$. The experiments with Slammer show a close match between predicted and observed values, and this is borne out by Fig-

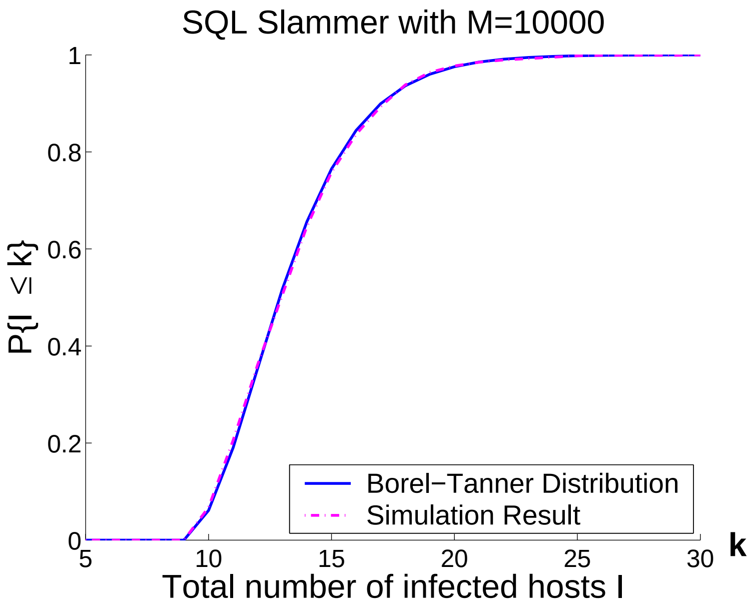

Fig. 2.16. Probability density and Cumulative distribution of I (SQL Slammer)

ures 2.15 and 2.16. The worm containment scheme contains the infection to below 20 hosts (i.e., only 10 newly infected hosts) with a very high probability.

To illustrate the effectiveness of our LPS worms, we also simulated our LPS worm containment system in a /16 network. In this simulator, the addresses for the vulnerable hosts and the dark IP addresses are randomly chosen from the /16 network. The *infected* hosts independently generate random /16 IP addresses to find victims. If the random IP address matches any of the IP addresses of the hosts in the *susceptible* state, the susceptible host will become infected. If it matches any of the dark IP addresses, the counter for its dark address scans will be increased. An infected host is *removed* if it has sent N scans to the dark addresses.

In the first three scenarios for the LPS worm containment system, we set the unused IP address density to equal 50% ($u = 0.5$), as in [19], and the number of initial infections I_0 is set to 20. We ran each of the scenarios 1000 times and collected the number of total infections in the /16 network for each run.

Figure 2.6 shows the effectiveness of the worm containment scheme when $N = 20$ for various vulnerability densities. As shown in Figure 2.6, with high probability (0.99), the total number of infected hosts in the network is less than 45 when there are 300 and 600 vulnerable hosts. When number of vulnerable hosts is increased to 1200, the containment system is less effective due to increased vulnerability density.

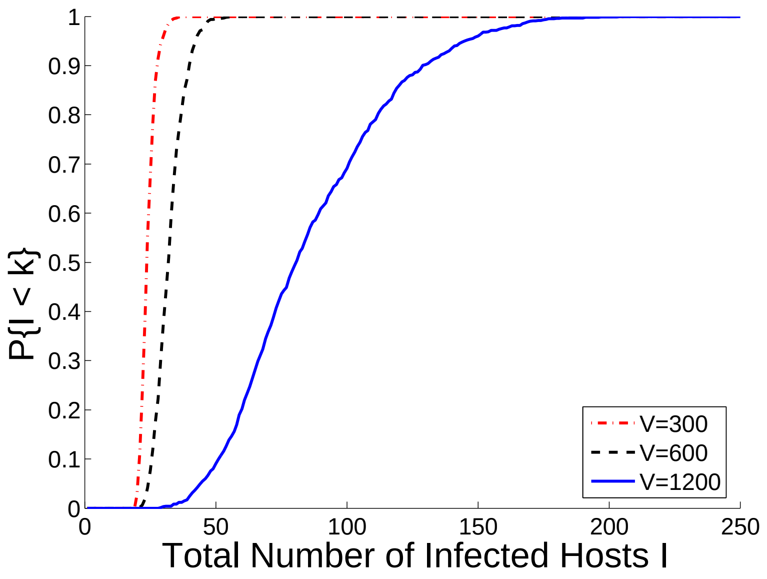

Fig. 2.17. The Cumulative Distribution of Total Number of Infected Hosts ($u = 0.5, N = 20$, and $I_0 = 20$). In (a) $V = 300, 600, 1200$ and in (b) $V = 1800$.

When the number of vulnerable hosts is 1800, having the same parameters as in the above scenarios ($u = 0.5$ and $N = 20$) will not contain the worms. Figure 2.6 shows the cumulative distribution function of the the total number of infections I ($u = 0.5$, $N = 20$, and $I_0 = 20$). As shown in Figure 2.6, with probability more than 0.9, the total number of infections will exceed 500. This is because in this case, $p_1 = 0.0275$ so that $N_{max} = \lfloor 0.5/0.0275 \rfloor = 18$. When we set $N = 20$ for the worm containment systems, the average number of offsprings from an infected host during the early phase is approximately 1.1, only slightly higher than 1. But, this results in a large fraction of

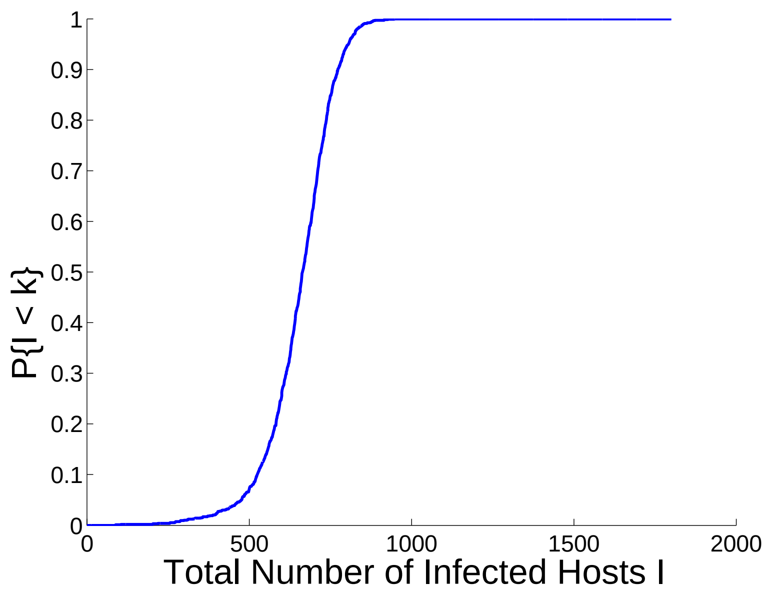

Fig. 2.18. The Cumulative Distribution of Total Number of Infected Hosts ($u = 0.5, N = 20$, and $I_0 = 20$). In (a) $V = 300, 600, 1200$ and in (b) $V = 1800$.

vulnerable hosts getting infected. Because the local vulnerability density p_1 decreases as more vulnerable hosts become infected, the number of offsprings from later infected hosts will decrease. Therefore, not all the vulnerables in the local network will be infected. The total number of the infected infected hosts is less than 1000 (out of 1800) with high probability.

Our analysis gives a network designer an easily quantifiable trade-off. First, reduce N to give tighter probabilistic bound on the total number of infected hosts at the risk of higher false alarm. Second, increase dark address space to again give the tighter bound at the risk of not fully utilizing the allocated IP address space.

Next, we simulated incremental deployment scenarios on a single /8 network that contains 256 /16 networks. We simulated a LPS worm that randomly scans the local /16 network 1/2 of the time and the /8 network 1/2 of the time. Each /16 network has V=1200 vulnerable hosts.

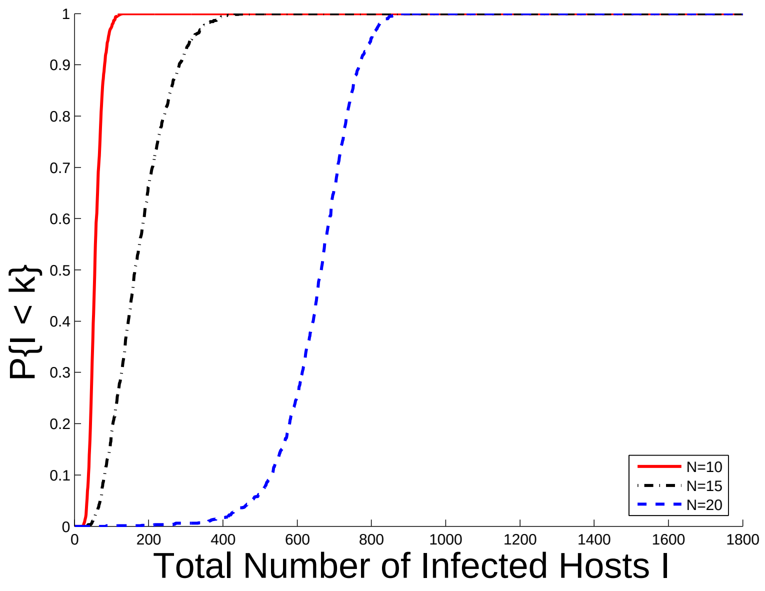

Fig. 2.19. The Cumulative Distribution of Total Number of Infected Hosts ($V = 1800$, $I_0 = 20$). In (a), $u = 0.5$, N varies; In (b) $N = 20$, u varies.

In our simulations with 100% deployment (i.e., all the /16 networks within the /8 have the defense mechanism deployed), there are 5 to 20 of these vulnerable hosts which are externally visible (denoted as EV in the plot). Only these externally visible hosts can be infected by hosts scanning from other networks. In one scenario, the network with initially infected hosts has $V_0 = 1800$ vulnerable hosts. In three other scenarios, the network with initially infected hosts has $V_0 = 1200$ vulnerable hosts. In all the scenarios, the initially infected hosts lie within a single /16 network and are 20 in number.

Our results are shown in Figure 2.21. We can see that when $V_0 = 1200$, the number of infections in the local network is between 50 to 150, and the total number of infections in all outside networks is very small. When $V_0 = 1800$, the total number of infections outside of the network is also small (about 100), despite a large number of hosts being infected inside the network. Our analysis and simulations both show

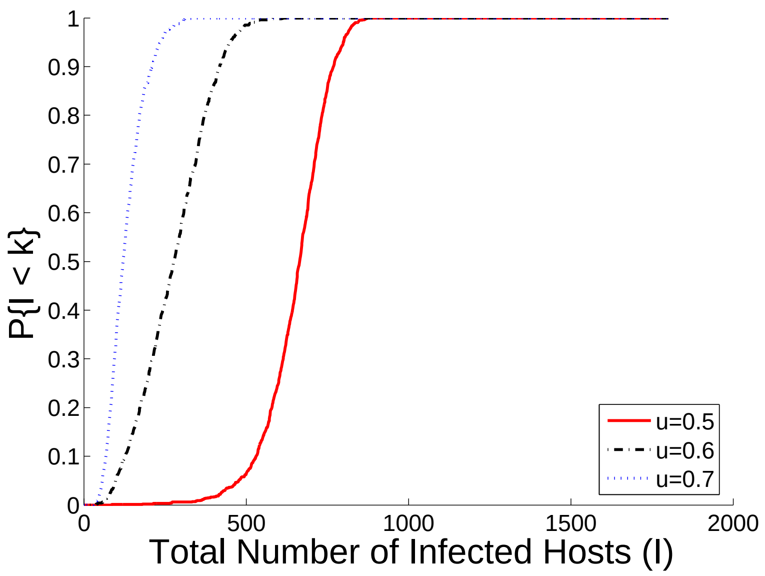

Fig. 2.20. The Cumulative Distribution of Total Number of Infected Hosts ($V = 1800$, $I_0 = 20$). In (a), $u = 0.5$, N varies; In (b) $N = 20$, u varies.

that our LPS worm containment scheme can indeed provide global containment with universal deployment.

Next, we simulated partial deployment scenarios with only one protected network, as well as 50%, 75%, and 90% of the /16 networks being protected by our LPS worm containment scheme. There are 20 externally visible hosts ($EV = 20$) in each of the /16 networks. As mentioned earlier, $V = 1200$ for each /16 network. We only simulate the scenarios when the initially infected hosts are in protected networks. When the initially infected host reside in an unprotected network, the worm will spread to all the unprotected networks eventually. The number of the initially infected hosts is 5 ($I_0 = 5$) in our simulations.

We ran our simulation 1000 times in each partial deployment scenario. Our results are summarized in Table 2.2. In all scenarios, the worm is completely contained within the originating network over 76% of the time. When only 50% of the networks are protected, the worm escapes to the unprotected networks only 12.7% of

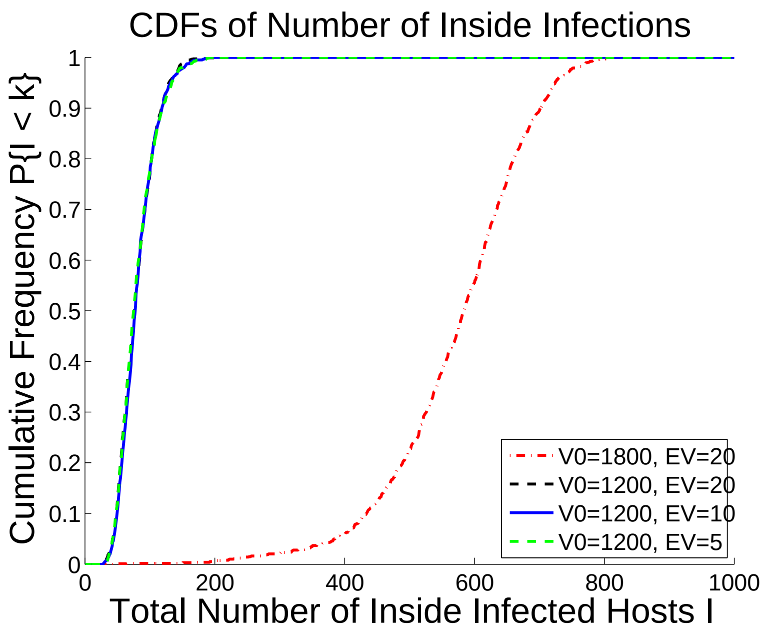

Fig. 2.21. Full Deployment of LPS containment scheme, with infections inside the original infections network

Table 2.2
Simulation Results on Partial Deployment

Networks with LPS Worm Containment	90%	75%	50%	ONE
Contained in Originating Network	76.6%	77.6%	76.4%	76.9%
Contained in Protected Networks	20%	15.2%	10.9%	0
Escaped to Unprotected Networks	3.4%	7.2%	12.7%	23.1%

the time; it is contained in other protected networks 10.9% of the time and is completely contained in the originating network 76.4% of the time. When the LPS worm containment system is deployed in only one network in which the initially infected host originates, there is also a 76.9% chance that the worm is completely contained in the local network. This is not surprising since the protected networks have the same configurations, and the probability that the infection will spread outside the

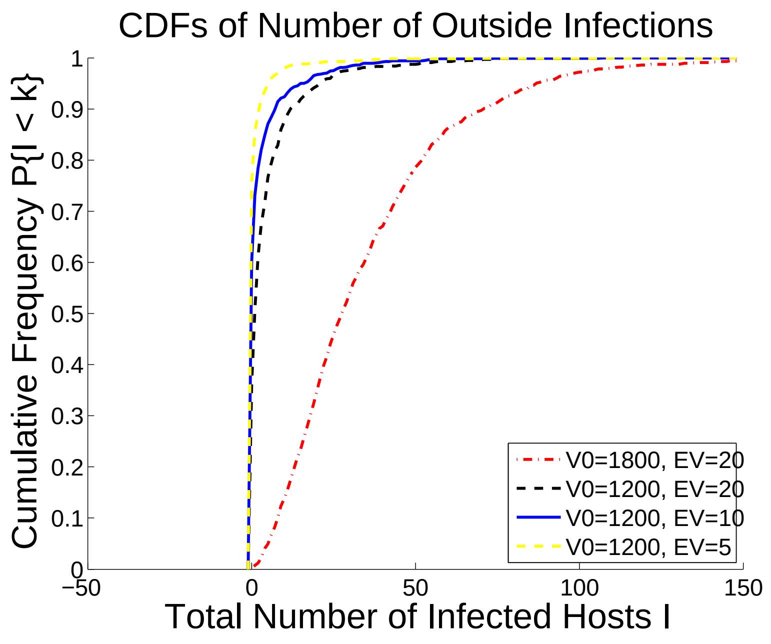

Fig. 2.22. Full Deployment of LPS containment scheme, with infections outside the original infected networks.

network is the same. However, the more networks deploy the LPS worm containment system, the less likely the worm infection will spread to the unprotected networks. Higher participation in LPS worm containment also makes it less likely for the initial infection to start in the unprotected networks.

The reason is that the average total internal infections is about 19, starting with 5 infected hosts. Average scans per host going outside the /8 network is about 40 before it gets quarantined. So the total number of scans from the originating /16 netowrk on average is approximately 800. The vulnerability density in the /8 network $p_2 = 20/65536 = 3 \times 10^{-4}$ when EV = 20, assuming pessimistically that all the external visible host are vulnerable. The probability of finding an externally visible host in the /8 network through the preferential scan is very small (less than 0.25). Also, the expected number of external hosts that would be infected by the originating /16 network is very small (about 0.23).

Our simulation result shows that partial deployment of our LPS worm containment scheme can benefit the global network. If the initially infected host is in an unprotected network, the worm spread will be slowed down, depending on the how large the deployment base is. Let V_u be the total number of vulnerable hosts from unprotected network, and $p_u = V_u/2^{32}$. If the infection originates from the unprotected networks, the worm propagation is at least the speed as if the vulnerability density is $p = p_u$.

Our analysis and simulation shows that our LPS worm containment system is very effective when there is a 100% deployment. When there is only a partial deployment, it protects the local networks and provides global benefit. Since this scheme benefits the local network, it is likely to be adopted by more organizations over time.

2.7 Conclusion

In this paper, we have studied the problem of combating Internet worms. To that end, we have developed a branching process model to characterize the propagation of Internet worms. Unlike deterministic epidemic models studied in the literature, this model allows us to characterize the early phase of worm propagation. Using the branching process model we are able to provide a precise bound M on the total number of scans that ensure that the worm will eventually die out. Further, from our model we also obtain the probability that the total number of hosts that the worm infects is below a certain level, as a function of the scan limit M. The insights gained from analyzing this model also allow us to develop an effective and automatic worm containment strategy that does not let the worm propagate beyond the early stages of infection. Our strategy can effectively contain both fast scan worms and slow scan worms without knowing the worm signature in advance or needing to explicitly detect the worm. We show via simulations and real trace data that the containment strategy is both effective and non-intrusive.

We extended the worm containment scheme to local preference scanning worms. In this scheme, we restrict the total number of scans per host to the dark address space. We derive the precise bound N on the total number of scans to the dark address space which ensures that the worm will be contained. This containment scheme, combined with firewalls at the network boundary, allows for incremental deployment of the worm containment system without participation of outside networks.

For further work, we would like to propose a statistical model for the spread of topology-aware worms and subsequently design mechanisms for automatic containment of such worms. We would also like to characterize the deviation of our proposed branching process model from the “ideal” stochastic epidemic model, assuming that the values of its rich set of parameters were available. Finally, we would like to design and implement our worm containment schemes with real data from enterprise networks.

2.8 Acknowledgement

We would like to thank Professor William Cleveland and Joerg Micheel for their tremendous help with the trace data from Bell Lab.

3. CAPACITY BOUNDS ON TIMING CHANNELS WITH BOUNDED SERVICE TIMES

3.1 Introduction

A timing channel is a non-conventional communication channel, in which a message is encoded in terms of the arrival times of bits. The receiver observes the time of the departing bits and decodes the message. It has been shown in [42] that when the service time of the queue is exponentially distributed, the channel capacity, $e^{-1}\mu$ nats/sec, is the lowest among all the servers with the same service rate μ. Most of the existing work such as in [42–49] has been focused on Exponential Service Timing Channels (ESTC). The discrete-time counterpart has been studied in [51, 52].

While ESTC has the lowest capacity among all servers with the same service rate, deterministic service timing channels have infinite capacity. In [53], we estimated the lower bounds on the capacities of single-server timing channels in which the service time distributions are uniform (uniform BSTC), Gaussian (GSTC), and truncated Gaussian (Gaussian BSTC). The capacities of these channels are on the order of $\mu \log_2(\mu\sigma)^{-1}$ bits/sec as $\sigma \to 0$, where μ is the service rate and σ is the standard deviation of the service time.

In many real world applications, the service time distributions have *bounded support*. By bounded support, we mean that there exist some constants $a, \Delta > 0$, such that the *i.i.d.* service times $S_1, S_2, \cdots$ satisfy $P(a < S_k < a + \Delta) = 1$. Such timing channels are called Bounded Service Timing Channels (BSTC), and $(a, a + \Delta)$ is called a support interval of the BSTC. We are especially interested in BSTC with small relative fluctuation of the service time, i.e. $\Delta/a \ll 1$.

In this paper, we focus on the capacity of BSTCs with support intervals symmetric about the mean service time $\frac{1}{\mu}$, i.e. support intervals of the form $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$. The service time distribution does not need to be symmetric about $1/\mu$. We derive an upper bound $C_U(\epsilon)$ on the capacity of BSTC using a feedback argument, and two *zero-error* capacity lower bounds, $C_{L,1}(\epsilon)$ and $C_{L,2}(\epsilon)$, using geometrically distributed inter-arrival times. While $C_U(\epsilon)$ is dependent on the service time distribution, both $C_{L,1}(\epsilon)$ and $C_{L,2}(\epsilon)$ are independent of the distribution of the service time, given the support interval. All these results can easily be extended to BSTCs that have support intervals asymmetric about $\frac{1}{\mu}$, i.e. of the form $(\frac{1}{\mu} - \epsilon_L, \frac{1}{\mu} + \epsilon_R)$.

We further show that these bounds are asymptotically tight for the uniform BSTC. By the tightness, we mean, when ϵ is small, the *capacity* of the uniform BSTC is $C_{L,2}(\epsilon) + o(1)$ (or $C_{L,1}(\epsilon) + O(1)$). Since our lower bound $C_{L,2}(\epsilon)$ is universal for all BSTC, the uniform BSTC serves a role similar to that of the ESTC in the paper by Anantharam and Verdu [42]. Namely, when ϵ is small, the uniform BSTC has the smallest capacity among all BSTCs, just as the exponential service time has the smallest capacity when considering unbounded service time distributions.

The rest of the paper is organized as follows: In Section 3.2 we provide an upper bound $C_U(\epsilon)$ on the capacity of BSTC using a feedback argument. In Section 3.3, we provide two lower bounds $C_{L,1}(\epsilon)$ and $C_{L,2}(\epsilon)$, both of which are asymptotically tight but in different senses. Furthermore, we show that the second lower bound, exploiting the absolute timing information, is extremely close to the capacity of the uniform BSTC when small ϵ is considered and is hence asymptotically optimal. We conclude our paper in Section 3.4.

3.2 An Upper Bound on the Capacity of BSTCs

Bounded Service Timing Channels (BSTC) are single-server queue based timing channels in which the service times $S_1, S_2, \cdots$ are *i.i.d.* with bounded support. In this paper, we consider servers with support intervals symmetric about the mean service time $E[S_k] = \frac{1}{\mu}$, i.e. $\exists \epsilon, 0 < \epsilon < \frac{1}{\mu}$ such that $P(\frac{1}{\mu} - \epsilon < S_k < \frac{1}{\mu} + \epsilon) = 1$. The service times $S_1, S_2, \cdots$ of the uniform BSTC are *i.i.d.* $U(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$ uniform random variables. The service times $S_1, S_2, \cdots$ of the Gaussian BSTC are *i.i.d.* truncated Gaussian random variables with density function:

$$f(x) = \frac{1}{K\sqrt{2\pi}\sigma} \exp(-\frac{(x-1/\mu)^2}{2\sigma^2})\ I(\frac{1}{\mu} - 3\sigma, \frac{1}{\mu} + 3\sigma)$$

where $\epsilon = 3\sigma < \frac{1}{\mu}$, and $K = \int_{-\infty}^{\infty} f(x)dx = 0.997$.

We will first provide an upper bound on the capacity of the BSTC by using a feedback argument of these channels.

Proposition 1 *Consider a BSTC with service rate μ and support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$.*

(a) An upper bound $C_U(\epsilon)$ on the capacity of the BSTC is $C_U(\epsilon) = \mu \sup_{0<\gamma<1} G(\epsilon, \gamma)$ bits/sec, where

$G(\epsilon, \gamma) = \gamma[\log_2(\epsilon\mu + \frac{1}{\gamma} - 1) + \log_2(e) - \log_2(\mu) - h(S_i)]$

(b) $C_U(\epsilon)$ for the uniform BSTC with service rate μ and support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$ is the smallest among all BSTC with the same service rate and support interval.

[Proof] (a) Let S_i be the service time of the i^{th} bit, and let a_i and d_i be the arrival and the departure time of the i^{th} bit, respectively. Further, let A_i and D_i be the inter-arrival time and the inter-departure time between the $(i-1)^{st}$ bit and the i^{th} bit, respectively, and let W_i be the queue's idle time before the arrival of the i^{th} bit.

An upper bound C_U is the capacity C_{FB} of the timing channel in which there is an additional feedback channel providing the queue size information on the server back to the transmitter, so that the sender has the knowledge of d_{i-1} before deciding a_i. With the feedback information, the sender has full control over W_i and can completely avoid any queueing. Thus, the timing channel is reduced to a sequentially juxtaposed *i.i.d.* channel: $W_i \rightarrow W_i + S_i$. The capacity of this new channel with feedback information is simply

$$C_{FB} = \sup_{\lambda<\mu} \lambda I(W_i; W_i + S_i),$$

where λ is the inter-departure rate ($\lambda = 1/E[D_i]$) and $I(W_i, W_i + S_i) = h(W_i + S_i) - h(S_i)$.

Since $W_i + S_i - (\frac{1}{\mu} - \epsilon) > 0$ and $E[W_i + S_i - (\frac{1}{\mu} - \epsilon)] = \frac{1}{\lambda} - \frac{1}{\mu} + \epsilon$, We have

$$\sup_{W_i > 0} [h(W_i + S_i)] \leq 1 + \ln(\frac{1}{\lambda} - \frac{1}{\mu} + \epsilon) \text{ nats.} \quad \text{Thus,}$$

$$\begin{aligned} C_{FB} &= \sup_{\lambda < \mu} \{\lambda[1 + \ln(\frac{1}{\lambda} - \frac{1}{\mu} + \epsilon) - h(S_i)]\} \\ &= \mu \sup_{\lambda < \mu} \{\frac{\lambda}{\mu}[\ln(\frac{\mu}{\lambda} - 1 + \epsilon\mu) + 1 - \ln(\mu) - h(S_i)] \end{aligned}$$

Let $\gamma = \lambda/\mu$. Define

$$G(\epsilon, \gamma) = \gamma[\log_2(\epsilon\mu + \frac{1}{\gamma} - 1) + \log_2(e) - \log_2(\mu) - h(S_i)]$$

We have an upper bound on the capacity of BSTCs:

$$C_U(\epsilon) = C_{FB} = \mu \sup_{0 < \gamma < 1} G(\epsilon, \gamma) \text{ bits/sec}$$

(b) Since the uniform $U(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$ random variable has the maximum entropy among all random variables with the support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$, $G(\epsilon, \gamma)$ in part (a) for the uniform BSTC is the smallest among all BSTC for each γ. Therefore, $C_U(\epsilon)$ for the uniform BSTC is the smallest among all BSTCs with the same service rate and support interval.

$\blacksquare$

It is apparent that the value of $C_U(\epsilon)$ is dependent on the service time distribution. Next, we will provide two zero-error lower bounds $C_{L,1}(\epsilon)$ and $C_{L,2}(\epsilon)$ on the capacity for BSTCs. Both lower bounds are *independent* of the service time distributions given the support interval.

3.3 Two Lower Bounds on the Capacity of BSTCs

3.3.1 The First Lower Bound

In this section, we will provide a sub-optimal lower bound $C_{L,1}$ on the capacity of BSTC. This lower bound is obtained by using a coding scheme in which the inter-arrival times $A_1, A_2, \cdots$ are *i.i.d.* geometric random variables. We require $A_i \geq \frac{1}{\mu} + \epsilon$

to avoid queuing. Further, the possible values of A_i are spaced 4ϵ apart to allow error-free decoding. More precisely, $A_1, A_2, \cdots$ are *i.i.d.* random variables with the following probability mass function:

$$P\{A_i = (\frac{1}{\mu} + \epsilon) + k(4\epsilon)\} = p_1(1-p_1)^k, \quad k = 0, 1, \cdots$$

Since this encoding scheme does not require prior knowledge of the service time distribution, given the support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$, it yields a *universal* lower bound $C_{L,1}(\epsilon)$ on the capacity of BSTC.

We now state our first lower bound Lemma without proof. The proof is provided in our online technical report [54].

Lemma 1 *Consider a BSTC where the service times $S_1, S_2, \cdots,$ are* i.i.d. *random variables with service rate μ and $P[S_i \in (\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)] = 1$. A zero-error lower bound $C_{L,1}(\epsilon)$ on the capacity of the timing channel is:*

$$C_{L,1}(\epsilon) = \mu \sup_{0 < \gamma < (1+\epsilon\mu)^{-1}} \gamma[H(p_1)/p_1] \textit{ bits/sec}$$

where $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ and

$$p_1 = \frac{4\epsilon\mu}{1/\gamma - 1 + 3\epsilon\mu}.$$

Figure 3.1 shows $C_{L,1}(\epsilon)$ as a function of the load factor $\gamma = \lambda/\mu$ when $\epsilon\mu = 0.01$, along with the upper bounds $C_U(\epsilon)$ for *uniform BSTC* and *Gaussian BSTC*. As shown in this figure, $C_{L,1} = 3.4\mu$ bits/sec, and C_U for the uniform BSTC (4.16μ bits/sec) is smaller than that of the Gaussian BSTC (4.58μ bits/sec). This is expected by Proposition 1(b).

Now, we compare the performance of $C_{L,1}(\epsilon)$ with our upper bound $C_U(\epsilon)$. Denote $\Delta C_1(\epsilon) = C_U(\epsilon) - C_{L,1}(\epsilon)$. We will show that $\Delta C_1(\epsilon)$ for the uniform BSTC is the smallest among all BSTC and $\Delta C_1(\epsilon) = O(1)$ for the uniform and Gaussian BSTC.

Proposition 2 *For BSTC with service rate μ and support interval, $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$, $\Delta C_1(\epsilon)$ satisfies:*

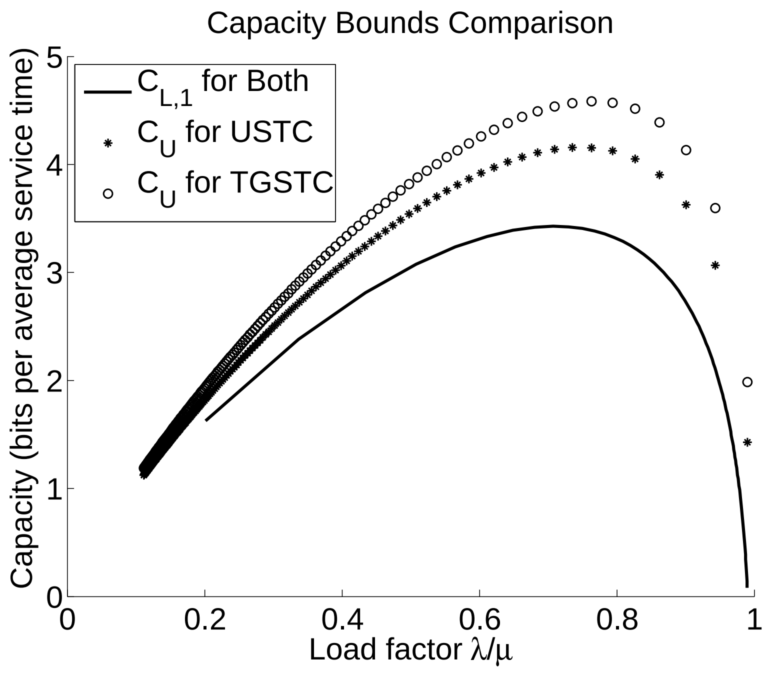

Fig. 3.1. Capacity Lower Bound $C_{L,1}$ compared with C_U for the uniform and Gaussian BSTC when $\epsilon\mu = 0.01$ (in bits per average service time).

(a) $\Delta C_1(\epsilon) < \mu(\log_2(e) + D(S_n||U_{\mu,\epsilon}))$ *bits/sec, where* $D(\cdot||\cdot)$ *is the Kullback-Leibler distance and* $U_{\mu,\epsilon}$ *is the uniform distribution on* $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$*.*

(b) $\Delta C_1(\epsilon)$ *for the uniform BSTC is the smallest such difference between our* $C_U(\epsilon)$ *and* $C_{L,2}(\epsilon)$ *among all BSTCs with the same service rate* μ *and support interval* $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$*.*

[Proof]

(1) We wish to show $\Delta C_1(\epsilon) \leq \mu[\log_2(e) - h(S_n) + \log_2(2\epsilon)]$ bit/sec. By *Proposition 1*,

$C_U(\epsilon) = \mu \sup_{0<\gamma<1} G(\epsilon, \gamma)$ bits/sec, where

$G(\epsilon, \gamma) = \gamma[\log_2(\epsilon\mu + \frac{1}{\gamma} - 1) + \log_2(e) - \log_2(\mu) - h(S_i)]$.

By *Lemma 1*,

$$C_{L,1}(\epsilon) = \mu \sup_{0<\gamma<(1+\epsilon\mu)^{-1}} \gamma[H(p_1)/p_1] \text{ bits/sec},$$

where $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ and
$p_1 = (4\epsilon\mu)(1/\gamma - 1 + 3\epsilon\mu)^{-1}$. Thus,

$$\begin{aligned} \Delta C_1 &= C_U(\epsilon) - C_{L,1}(\epsilon) \\ &< \mu \sup_{0<\gamma<(1+\epsilon\mu)^{-1}} [G(\epsilon,\gamma) - \gamma H(p_1)/p_1] \end{aligned} \tag{3.1}$$

First, express the first term of $G(\epsilon,\gamma)$, $\log_2(\epsilon\mu + 1/\gamma - 1)$, in terms of p_1.

$$p_1 = \frac{4\epsilon\mu}{1/\gamma - 1 + 3\epsilon\mu} \qquad \Rightarrow \epsilon\mu = \frac{(1/\gamma - 1)p_1}{4 - 3p_1}$$

Thus, $\epsilon\mu + 1/\gamma - 1 = (\epsilon\mu)(\frac{4-2p_1}{p_1})$, so that

$$\begin{aligned} & G(\epsilon,\gamma) \\ = \;& \gamma[\log_2(\epsilon\mu + \frac{1}{\gamma} - 1) + \log_2(e) - \log_2(\mu) - h(S_i)] \\ = \;& \gamma\{\log_2[(\epsilon\mu)(\frac{4-2p_1}{p_1})] + \log_2(e) - \log_2(\mu) - h(S_i)\} \\ = \;& \gamma\{\log_2[(\epsilon)(\frac{4-2p_1}{p_1})] + \log_2(e) - h(S_i)\} \end{aligned}$$

Thus,

$$\begin{aligned} & G(\epsilon,\gamma) - \gamma H(p_1)/p_1 \\ = \;& \gamma\{\log_2[(\epsilon)(\frac{4-2p_1}{p_1})] + \log_2(e) - h(S_i)\} - \gamma H(p_1)/p_1 \\ = \;& \gamma[\log_2(2\epsilon(2-p_1)/p_1) + \log_2(e) - h(S_i) \\ & +(\log_2(p_1) + (1-p_1)/p_1 \log_2(1-p_1))] \\ = \;& \gamma[\log_2(e) - h(S_n) + \log_2(2\epsilon)] \\ & +\gamma[\log_2(2-p_1) + (\frac{1-p_1}{p_1})\log_2(1-p_1))] \\ = \;& \gamma[\log_2(e) + D(S_n||U_{\mu,\epsilon})] \\ & +\gamma[\log_2(2-p_1) + (\frac{1-p_1}{p_1})\log_2(1-p_1))] \end{aligned}$$

Since $\log_2(2-p) + \frac{1-p}{p}\log_2(1-p) < 0$ and $0 < \gamma < 1$,

we have $G(\epsilon, \gamma) - \gamma H(p_1)/p_1$

$$< \gamma[\log_2(e) + D(S_n||U_{\mu,\epsilon})] \ < \log_2(e) + D(S_n||U_{\mu,\epsilon})$$

By equation (1), $\Delta C_1(\epsilon) \leq \mu[\log_2(e)) + D(S_n||U_{\mu,\epsilon})]$ bits/sec.

(2) By *Proposition 1* part (b), $C_U(\epsilon)$ the uniform BSTC with service rate μ and support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$ is the smallest among all BSTCs with the same service rate and support interval, and by Lemma 1, $C_{L,1}(\epsilon)$ is independent of the service distribution. Therefore, ΔC_1 for the uniform BSTC is the smallest among all BSTCs with service rate μ and support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$

■

Table 3.1
The Upper and Lower Bounds on the Capacity of uniform BSTC and Gaussian BSTC (in bits per average service time).

	ALL	Uniform	Uniform	Gaussian	Gaussian
$\epsilon\mu$	$C_{L,1}$	C_U	ΔC_1	C_U	ΔC_1
10^{-1}	1.4500	2.0314	0.5814	2.3927	0.9428
10^{-2}	3.4287	4.1582	0.7295	4.5833	1.1547
10^{-3}	5.9081	6.7469	0.8388	7.2127	1.3045

We obtain $\Delta C_1(\epsilon) < \log_2(e)\mu$ bits/sec $\approx 1.447\mu$ bits/sec for uniform BSTCs, and $\Delta C_1(\epsilon) < 2.004\mu$ bits/sec for Gaussian BSTCs by Proposition 2(a).

In Table 3.1, we show the values of $C_{L,1}(\epsilon)$ for BSTCs, and the values of $C_U(\epsilon)$ and $\Delta C_1(\epsilon)$ for the uniform BSTC and the Gaussian BSTC when $\epsilon\mu = 10^{-1}, 10^{-2}$, and 10^{-3}. We can see that $\Delta C_1 < 2\mu$ bits/sec for all ϵ in this Table.

Our first lower bound $C_{L,1}(\epsilon)$ is sub-optimal by using a naive coding scheme with a large (4ϵ) spacing. Nevertheless, it is a good lower bound because it is tight in the sense that for the uniform BSTC, $\Delta C_1(\epsilon) < 1.447$ bits/sec for all ϵ. That is, the *capacity* of the uniform BSTC is $C_{L,1}(\epsilon) + O(1)$. Moreover, it is universal for all BSTC in a given support interval.

We now present our optimal lower bound $C_{L,2}(\epsilon)$ for BSTCs, which is also independent of the service time distribution given the support interval. However, it requires knowledge of absolute timing information at both sender and receiver.

3.3.2 The Second Lower Bound

To derive our second zero-error lower bound $C_{L,2}$, the receiver is required to use more computational power to recover the absolute time. When the absolute time is available to the receiver, we use a *slotted-arrival-time* coding scheme to obtain our lower bound $C_{L,2}(\epsilon)$.

Lemma 2 *Assuming that absolute time information is available to both the sender and the receiver, a zero-error lower bound $C_{L,2}(\epsilon)$ on the capacity of BSTCsde with service rate μ and support interval $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$ is:*

$$C_{L,2}(\epsilon) = \mu \sup_{0<\gamma<(1+(1+2\alpha)\epsilon\mu)^{-1}} \gamma[H(p_2)/p_2] \textit{ bits/sec}$$

where $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$,

$$p_2 = \frac{2\epsilon\mu}{\frac{1}{\gamma} - 1 + (1-2\alpha)\epsilon\mu}, \quad \textit{and } \alpha = \lceil\frac{1+\epsilon\mu}{2\epsilon\mu}\rceil - \frac{1+\epsilon\mu}{2\epsilon\mu}.$$

Proof. When the absolute time information is available, we use *slotted* arrival times with slot size 2ϵ. That is, the arrival time of the i^{th} bit, a_i, is restricted to be on the grid $t = (2\epsilon)k_i, k_i = 0, 1, 2\cdots$.

To see why our scheme works, we start with a simple example of encoding messages by sending only one bit at time a_1. Let d_1 be the departure time of that bit. Assuming the queue is initially empty, we have $d_1 = a_1 + S_1$. In our scheme, the only possible values of a_1 are $(2k)\epsilon, k = 0, 1, 2, \cdots$. Since $S_1 \in (\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$, we have $a_1 = (2k)\epsilon$ if and only if $d_1 \in I_k = (\mu + (2k-1)\epsilon, \mu + (2k+1)\epsilon)$. Moreover, I_k and $I_{k'}$ do not overlap if $k \neq k'$.

To decode a message, the receiver uses slotted departure times with slot size 2ϵ, and the k^{th} slot corresponds to the time interval $I_k = (\frac{1}{\mu} + (2k-1)\epsilon, \frac{1}{\mu} + (2k+1)\epsilon)$. Upon observing a bit departing at time d_1^* in slot k^*, i.e. $d_1^* \in I_{k^*}$, the receiver can recover the arrival time correctly as $a_1^* = (2k^*)\epsilon$.

When we encode messages by transmitting more than one bit, the receiver can decode the message error-free in exactly the same way as transmitting only one bit, as long as there is no queueing at the server. To avoid queueing, simply choose k_i so that $k_i \geq k_{i-1} + \frac{1+\epsilon\mu}{2\epsilon\mu}$ for $i \geq 2$. This condition is equivalent to $a_i - a_{i-1} \geq \frac{1}{\mu} + \epsilon$.

Thus, in our coding scheme, the inter-arrival times $A_1, A_2, \cdots$ must satisfy (1) $A_i = 2k'\epsilon$ and (2) $A_i \geq \frac{1}{\mu} + \epsilon$.

$$\text{Let} \;\; K_0 = \lceil \frac{1+\epsilon\mu}{2\epsilon\mu} \rceil, \;\; \text{and} \;\; \alpha = K_0 - \frac{1+\epsilon\mu}{2\epsilon\mu}, \quad 0 \leq \alpha < 1.$$

Choose $A_1, A_2, \cdots$ to be *i.i.d.* geometric random variables
that satisfy (1) and (2), with probability mass function:

$$P[A_i = K_0(2\epsilon) + k(2\epsilon)] = p_2(1-p_2)^k, \quad k = 0, 1, \cdots$$

Let λ be the departure rate and $\gamma = \lambda/\mu$. We have
$1/\lambda = E[D_i] = E[A_i] = K_0(2\epsilon) + 2\epsilon(\frac{1}{p_2} - 1)$. Thus,

$$p_2 = \frac{2\epsilon\mu}{\frac{1}{\gamma} - 1 + \epsilon\mu - 2\alpha(\epsilon\mu)}$$

Since $I(A_i; D_i) = h(A_i) = H(p_2)/p_2$, we have

$$C \geq \frac{I(A_i; D_i)}{E[D_i]} = \mu[\gamma H(p_2)/p_2]$$

$$\text{for all } \gamma, \text{ such that } \; 0 < \gamma < (1 + (1+2\alpha)(\epsilon\mu))^{-1}.$$

Therefore,

$$C_{L,2}(\epsilon) = \mu \sup_{0<\gamma<(1+(1+2\alpha)\epsilon\mu)^{-1}} \gamma[H(p_2)/p_2] \text{ bits/sec}$$

■

The major difference between the first scheme in Section 3.3.1 and the second scheme is that the timing information is now embedded in the *absolute* timing of each arrival rather than the traditional inter-arrival time. Or equivalently, the timing information is in the inter-arrival time between the i-th bit and the timing origin, rather than in the lapse between the i-th and the $(i-1)$-th bits, so that the noisy component of the service time is kept at 2ϵ rather than 4ϵ, the superposition of the noises from both the i-th and the $(i-1)$-th bits.

The absolute timing (or the timing origin), on the other hand, is generally not available at the receiver end. Nonetheless, by slightly increasing the size of the slots, from 2ϵ to $2\epsilon + \delta$, where δ serves as a guard band, the absolute timing information recovery problem is reduced to a grid realigning problem. The goal of this problem is to find a realignment such that no departure time falls into the guard band δ. With a sufficiently long observation period, the recovery is always possible in probability. By further reducing the size of the guard band δ, we obtain the same lower bound in Lemma 2 as if we have the absolute timing information.

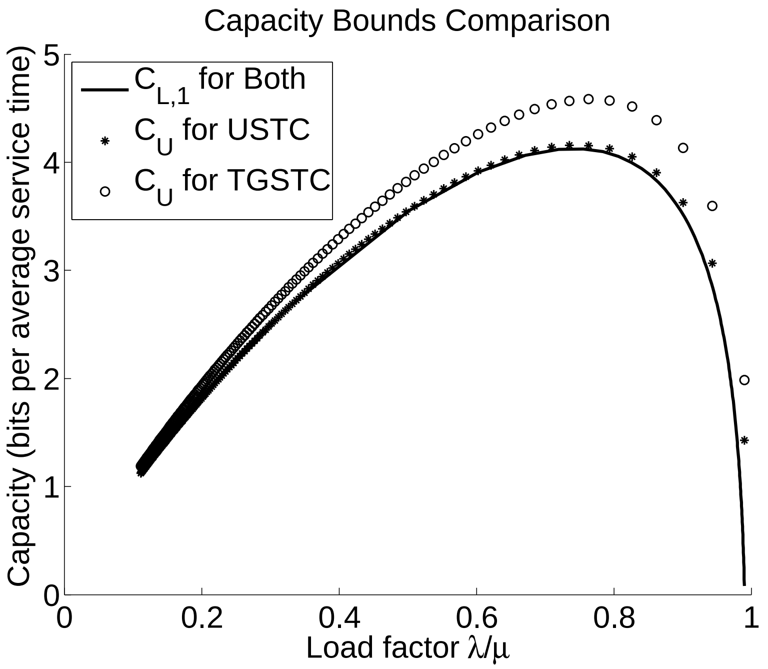

Fig. 3.2. Capacity Lower Bound $C_{L,2}$ compared with C_U for the uniform and Gaussian BSTC when $\epsilon\mu = 0.01$ (in bits per average service time).

Figure 3.2 shows the universal lower bound $C_{L,2}(\epsilon)$ for BSTCs as a function of the load factor $\gamma = \lambda/\mu$ for $\mu\epsilon = 0.01$, along with upper bounds $C_U(\epsilon)$ for *uniform* BSTCs and *Gaussian* BSTCs. As shown in Figure 3.2, $C_U(\epsilon)$ for uniform BSTCs is extremely close to $C_{L,2}(\epsilon)$.

In the next Proposition, we will show that $C_{L,2}(\epsilon)$ is asymptotically optimal in the sense that, $C_U(\epsilon) - C_{L,2}(\epsilon) \to 0$ as $\epsilon \to 0$ for uniform BSTCs.

Proposition 3 *Denote* $\Delta C_2(\epsilon) = C_U(\epsilon) - C_{L,2}(\epsilon)$.
(a) ΔC_2 *for a uniform BSTC is the smallest such difference between our* $C_U(\epsilon)$ *and* $C_{L,2}(\epsilon)$ *among all BSTCs with same service rate* μ *and support interval* $(\frac{1}{\mu} - \epsilon, \frac{1}{\mu} + \epsilon)$.
(b)$\Delta C_2(\epsilon) \to 0$ *as* $\epsilon \to 0$ *for uniform BSTC.*

Proof

(a) Same argument as in the proof of Proposition (2)(b).

(b)We wish to show $\Delta C_2(\epsilon) \to 0$ as $\epsilon \to 0$ for uniform BSTC.

As in the proof of Proposition 2,

$$\begin{aligned}\Delta C_2(\epsilon) &= C_U(\epsilon) - C_{L,2}(\epsilon) \\ &= \mu \sup_{0<\gamma<1} G(\epsilon,\gamma) - \sup_{0<\gamma<(1+\epsilon\mu)^{-1}} [\gamma H(p_2)/p_2]\end{aligned}$$

First, express the first term of $G(\epsilon,\gamma)$, $\log_2(\epsilon\mu + 1/\gamma - 1)$, in terms of p_2. Since

$$p_2 = \frac{2\epsilon\mu}{\frac{1}{\gamma} - 1 + (1-2\alpha)\epsilon\mu} \quad \Rightarrow \epsilon\mu = \frac{(\frac{1}{\gamma}-1)p_2}{2-(1-2\alpha)p_2},$$

$$\text{we have} \quad \epsilon\mu + \frac{1}{\gamma} - 1 = (\frac{1}{\gamma} - 1)(\frac{2+2\alpha p_2}{2-(1-2\alpha)p_2}) = (\epsilon\mu)(\frac{2}{p_2} + 2\alpha).$$

Since $h(S_i) = \log_2(2\epsilon)$ for uniform BSTC, we have

$$\begin{aligned} & G(\epsilon,\gamma) = \gamma[\log_2(\epsilon\mu + \frac{1}{\gamma} - 1) + \log_2(e) - \log_2(\mu) - h(S_i)] \\ = \quad & \gamma\{\log_2[(\epsilon\mu)(\frac{2}{p_2} + 2\alpha)] + \log_2(e) - \log_2(\mu) - \log_2(2\epsilon))\} \\ = \quad & \gamma\{\log_2(\frac{1}{p_2} + \alpha) + \log_2(e)\} \end{aligned}$$

Thus,

$$\begin{aligned} & G(\epsilon,\gamma) - \gamma H(p_2)/p_2 \\ = \quad & \gamma[\log_2(\frac{1}{p_2} + \alpha) + \log_2(e)] - \gamma H(p_2)/p_2 \\ = \quad & \gamma[\log_2(\frac{1}{p_2} + \alpha) + \log_2(e) \\ & +(\log_2(p_2) + \frac{1-p_2}{p_2}\log_2(1-p_2)] \\ = \quad & \gamma[\log_2(e) + (\frac{1-p_2}{p_2})\log_2(1-p_2))] + \gamma\log_2(1+\alpha p_2) \end{aligned}$$

Let $\gamma^* = \gamma^*(\epsilon)$ be the value where $G(\epsilon,\gamma)$ achieves its maximum, i.e. $C_U(\epsilon) = G(\epsilon,\gamma^*)$. The corresponding value of p_2^* sastifies $p_2^* \to 0$ as $\epsilon \to 0$. Thus,

$$\log_2(e) + (\frac{1-p_2^*}{p_2^*})\log_2(1-p_2^*) \to 0 \quad \text{as} \quad \epsilon \to 0,$$

and $\log_2(1+\alpha p_2^*) \to 0$ as $\epsilon \to 0$.

Thus for uniform BSTCs,

$$(G(\epsilon,\gamma^*) - \gamma^* H(p_2^*)/p_2^*) \longrightarrow 0 \quad \text{as} \quad \epsilon \to 0.$$

Therefore, $\Delta C_2(\epsilon) \to 0$ as $\epsilon \to 0$ for uniform BSTCs.

■

Proposition 3 shows that $C_{L,2}(\epsilon)$ is asymptotically tight for the uniform BSTC. By tightness, we mean that when ϵ is small, the *capacity* of the uniform BSTC is $C_{L,2} + o(1)$. Since our lower bound $C_{L,2}(\epsilon)$ is universal for all BSTCs, the uniform BSTC serves a role similar to that of the ESTC in [42]. Namely, when ϵ is small, the uniform BSTC has the smallest capacity among all BSTCs, while the exponential service time has the smallest capacity when considering unbounded service time distribution. Further, Proposition 3(b) does not hold for Gaussian BSTC. We can show that $\Delta C_2(\epsilon) > 0$ as $\epsilon \to 0$ for Gaussian BSTC.

Table 3.2 shows the values of the universal lower bound $C_{L,2}(\epsilon)$ of BSTCs, C_U and ΔC_2 for uniform BSTCs and Gaussian BSTCs, for various values of $\epsilon\mu$. When $\epsilon\mu = 10^{-3}$, $C_{L,2} = 6.7384\mu$; and for the uniform BSTC, $C_U = 6.7469\mu$ and $\Delta C_2 = 0.0086\mu$. Using the two tight bounds, we can infer that the *capacity* of this uniform BSTC is 6.7μ bits/sec.

Table 3.2
The Upper and Lower Bounds on the Capacity of uniform BSTC and Gaussian BSTC (in bits per average service time).

$\epsilon\mu$	ALL $C_{L,2}$	Uniform C_U	Uniform ΔC_2	Gaussian C_U	Gaussian ΔC_2
10^{-1}	1.9106	2.0314	0.1198	2.3927	0.4812
10^{-2}	4.1240	4.1582	0.0342	4.5833	0.4593
10^{-3}	6.7384	6.7469	0.0086	7.2127	0.4743

3.4 Conclusion

We have studied the capacity of timing channels with bounded service times. We have obtained an upper bound, and two universal lower bounds on the capacity of BSTCs. These bounds are shown to be asymptotically tight for uniform BSTCs. An interesting observation that comes about as a by-product of this work is that the uniform BSTC serves a role similar to that of the ESTC in [42], i.e., when ϵ is small, the uniform BSTC has the smallest capacity among all BSTCs.

4. TCP/IP TIMING CHANNELS: THEORY TO IMPLEMENTATION

4.1 Introduction

The Orange Book [57] defines a covert channel to be *"any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy."* A covert timing channel is a type of covert channel in which sensitive information is transmitted by the timing of events. In a multi-level security (MLS) system, covert timing channels can be used by a HIGH process to leak classified information to a LOW process. In a networked environment, it can be used by a program that has access to sensitive information to leak the information through packet inter-transmission times.

Designing and implementing timing channels over a shared network between two distant computers is challenging. Network timing channels are inherently "noisy" due to the delay and jitter in networks, which distort the timing information from the sender when it reaches the receiver. In [58], the authors designed and implemented an IP covert timing channel using an on-off coding scheme, where the reception or absence of a packet within a time interval signals bit 1 or bit 0, respectively. This timing channel achieves a data rate of 16.67 bits/sec between two computers located at two universities with an average round trip time of 31.5 ms.

In TCP/IP networks, end-to-end delays are much larger than the jitter. Information theoretic research shows that the Shannon capacity of timing channels with no jitter is infinite, assuming infinite precision of the system clock ([42, 46–48, 51, 52]), and the capacity can be made very large if the jitter of the underlying channel is very small [54]. Motivated by these theoretical results, we are interested in designing a TCP/IP covert timing channel that significantly improves the current state-of-

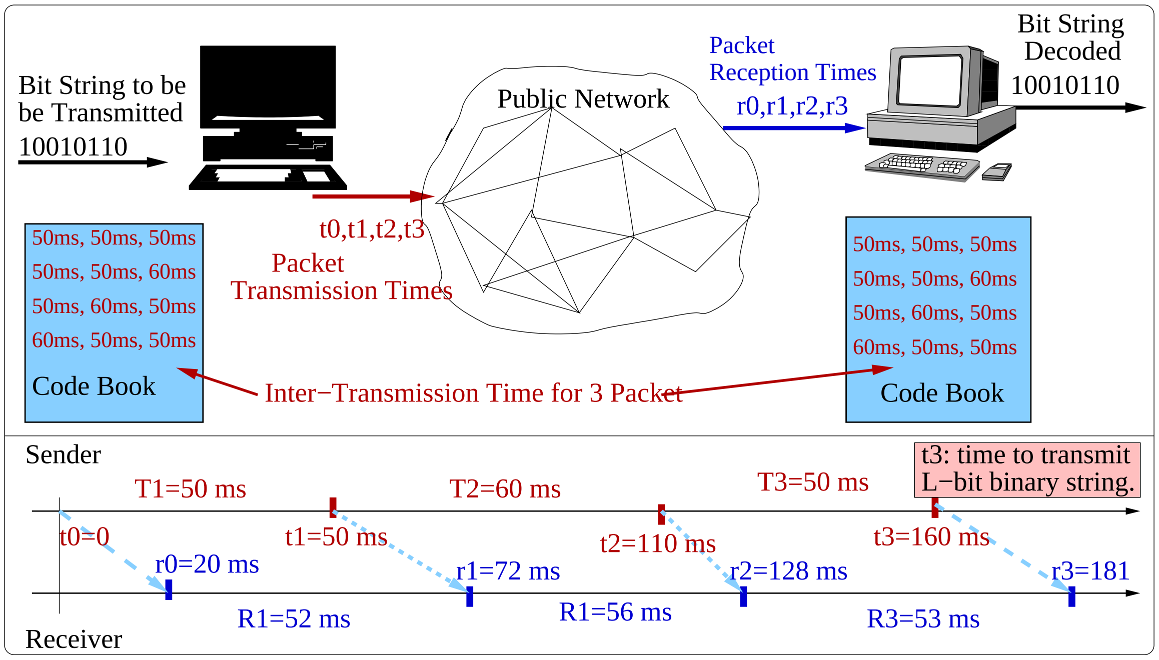

Fig. 4.1. High Level Design Diagram of Covert Timing Channel

the-art data rate [58]. Additionally, our second goal is to design a computationally non-detectable timing channel scheme, assuming that the packet inter-transmission time of regular traffic is independently and identically distributed (*i.i.d.*), but the distribution of the inter-transmission times could be general (e.g., Pareto).

In our design, we use packet *inter*-transmission times (denoted as T_k) to convey information. Figure 4.1 shows a high-level view of our design. A malicious process on the sender side manipulates the inter-transmission times and another malicious process either at the receiver or en route to the receiver can decode the privileged information by observing the inter-reception times. We encode L-bit binary strings in a sequence of n packet inter-transmission times $T_1, T_2, \cdots, T_n$. We call it the "*L-bits to n-packets*" scheme. These n packets are transmitted in *variable* length time intervals. The receiver will then map the n packet inter-reception times $R_1, R_2, \cdots, R_n$ back to an L-bit binary string according to the code book.

We make two novel contributions in this paper (both are also distinct from our prior work [54]). First, we provide a systematic solution of selecting the values of L and n for the *L-bits to n-packets* scheme, so that the data rate of our scheme is near optimal. We implement a TCP/IP timing channel with our *L-bits to n-packets* scheme, and conduct extensive experiments on the PlanetLab environment [85] with five pairs of computers distributed worldwide to show the effect of differing delay and jitter. We demonstrate significant performance improvement (2 to 5 times the covert timing channel data rate) of our scheme over the state-of-the-art [58].

Our second contribution is to systematically develop a computationally non-detectable timing channel scheme, assuming the packet intertransmission time is *i.i.d.* Our design is based on the security of the *cryptographically secure pseudo random number generators* (CSPRNG). The packet inter-transmission times (T_k's) from the proposed timing channel are devised to mimic any legitimate traffic with *i.i.d.* packet inter-transmission time, e.g., the *i.i.d.* Pareto distribution for telent traffic [62]. This allows two parties to communicate at a low data rate (e.g., 5 bits/sec) in a hostile environment such as in battlefield or law enforcement settings while avoiding detection. The proposed non-detectable scheme is also implemented, and experiments are conducted on PlanetLab. The similarity of the traffic patterns of our scheme and the telnet traffic is verified.

The remainder of our paper is organized as follows: In Section 4.2, we review related work. In Section 4.3, we present our system level design and the proposed *L-bits to n-packets* scheme, with discussions on the trade-offs between the data rate and the complexity of our scheme. In Section 4.4, we describe our implementation of covert TCP/IP timing channels and our experimental results. In Section 4.5, we show how to construct a timing channel scheme that is computationally not detectable. We conclude with discussion and future research directions in Section 4.6.

4.2 Related Work

The best achievable data rate of a covert timing channel measures the severity of its threat, and can be obtained by estimating the information theoretic channel capacity [42, 51, 54, 61]. To reduce the throughput of covert timing channels, several methods such as timing channel jammers [46], fuzzy times [59] and network pump [60], have been proposed. Another approach to defend against the usage of covert timing channels is to detect the presence of such usage [56, 58, 81].

In [58], the authors illustrated the threat from IP covert timing channels using a software implementation of covert timing channels over TCP/IP networks. They also developed a detection mechanism for their IP covert timing channels by identifying the regularity of the inter-transmission times. In their design, the sender and receiver agree upon an interval length, say w ms. To signal bit value '1', the sender transmitted a packet in the middle of the time interval; to signal '0', the sender remained silent during that interval. Their scheme achieved a rate of 16.67 b/s with 2% character decoding error ($w = 60$ ms) in an experimental setting where the average RTT between the two hosts is 31.5 ms. Their use of a simple coding scheme, basically an on-off scheme, limits the data rate achievable for the timing channel. Their scheme requires time synchronization between the sender and the receiver in order to correctly decode a message. A constant shift in delay and jitter may have a cascade effect that may cause subsequent bits to be decoded incorrectly.

In [65], the authors built a *Keyboard JitterBug*, a device interposed between the keyboard and the computer, that can leak typed information through a covert network timing channel when a user runs an interactive application. The authors utilized the interactive session and added different delays to the timing sequence of the keystrokes to signal 1 or 0. Their binary encoding scheme allows one bit of information to be transmitted per keypress, and the rate of their timing channel scheme is limited by the user's typing speed. They then used a "4-bit to 1-keypress" encoding scheme to improve the performance. This method uses 16 different delay values to encode the

16 distinct 4-bit binary strings. The timing channel rate being tied to human input is very low, and it is detectable by sophisticated detection algorithms such as [81].

In our proposed scheme, we use packet inter-transmission time to convey information, not just the presence or absence of a packet in a fixed time interval. Our method eliminated the need for absolute time synchronization between the sender and the receiver, and it shares some similarities with the *sparse time base* used for clock synchronizations in distributed systems in [50].

We propose a more general *L-bits to n-packets* scheme that maps binary strings of length L to multiple packet inter-transmission times of size n, which includes both the *on-off* scheme [58] and the keyboard jitter bugs as special cases. We further provide methods for selecting the values of parameters L and n to get higher data rates. Finally, we describe our design of a timing channel scheme that mimics a class of normal traffic to avoid detection.

4.3 Covert Timing Channel Design

4.3.1 System Level Model and Design

Our approach can be applied to a wide variety of communication paradigms to transmit covert information. However, for the sake of illustration, we describe here a high level view (illustrated in Figure 4.1) for communication with TCP/IP. The sender and receiver reside on two distant computers with several routers in between. The sender has access to sensitive information and wants to leak this information to the receiver using the timing of packet transmissions.

We use t_k and r_k to denote the times that the k^{th} packet is transmitted and received, respectively. We use D_k to denote the end-to-end delay of the k^{th} packet, so that $r_k = t_k + D_k$. The delay D_k include the delays from the TCP/IP network and the processing delays on both computers running the timing channel software. It can be expressed as $D_k = D + \epsilon_k$, where D is a constant that represents the average delay, and ϵ_k is a random variable that represents the jitter. In TCP/IP networks, jitter is typically bounded (e.g., $-\epsilon_{max} < \epsilon_k < \epsilon_{max}$), but may not be *i.i.d.*.

The packet inter-transmission (denoted as T_k) between the $(k-1)^{st}$ packet and the k^{th} packet can be expressed as
$T_k = t_k - t_{k-1}$. Likewise, the packet inter-reception time (denoted as R_k) between the $(k-1)^{st}$packet and the k^{th} packet can be expressed as $R_k = r_k - r_{k-1}$. Thus,

$$R_k = T_k + (\epsilon_k - \epsilon_{k-1}) \tag{4.1}$$

Equation (1) models the timing channel, with T_k being the input to the channel and R_k being the noisy output.

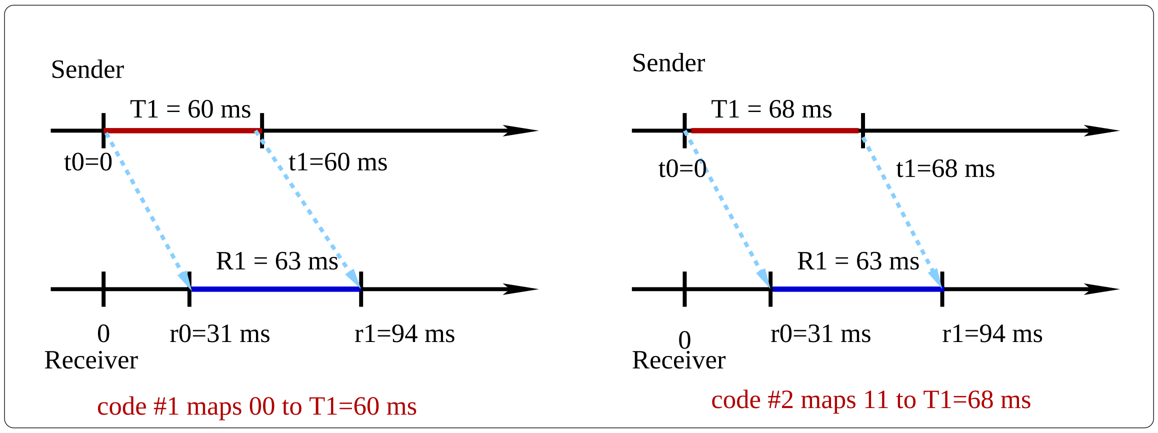

Fig. 4.2. Identical R_1 for two distinct bit strings resulting in decoding error for the timing channel

Let $T_1^{(1)}$ and $T_1^{(2)}$ be any two inter-transmission times representing two distinct binary strings. We require that $|T_1^{(1)} - T_1^{(2)}|$ be large enough so that the corresponding inter-reception times $R_1^{(1)}$ and $R_1^{(2)}$ are *always* distinguishable even with the noisy component $(\epsilon_k - \epsilon_{k-1})$. Figure 4.2 illustrates a scenario when $|T_1^{(1)} - T_1^{(2)}|$ is too small. In this example, $D = 30$ ms and $|\epsilon_k| < 5$ ms for all k. Suppose we encode "00" as $T_1^{(1)} = 60$ ms, and "11" as $T_1^{(2)} = 68$ ms. As shown in the figure, the inter-reception times $R_1^{(1)}$ and $R_1^{(2)}$ for "00" and "11" can be the same, for which it is impossible for the receiver to decide if "00" or "11" was sent.

In our design, we use the parameter δ to denote this minimum difference of $T_1^{(1)}$ and $T_1^{(2)}$, and $\delta > 4\epsilon_{max}$ ensures no confusion for any values of ϵ_k and ϵ_{k-1}. The reason is that when $|\epsilon_k| < \epsilon_{max}$, it follows directly from equation (1) that

$$T_1^{(1)} - 2\epsilon_{max} < R_1^{(1)} < T_1^{(1)} + 2\epsilon_{max} \tag{4.2}$$

$$T_1^{(2)} - 2\epsilon_{max} < R_1^{(2)} < T_1^{(2)} + 2\epsilon_{max} \tag{4.3}$$

If $|T_1^{(1)} - T_1^{(2)}| > 4\epsilon_{max}$, then (2) and (3) guarantees that $R_1^{(1)} \neq R_1^{(2)}$.

Another parameter in our code design is Δ, the minimum value for T_k (i.e., $T_k \geq \Delta$). The intuition for imposing a minimum time Δ between the transmissions of any two consecutive packets is to avoid loss of timing information. If two packets are transmitted too close to each other, they may be queued at the computers running the timing channel software or on the intermediate routers, and queueing could destroy the timing information [42].

Even though a better designed timing channel can achieve a higher data rate, we must also keep in mind that the data rate cannot be arbitrarily large since it is upper bounded by the channel's Shannon capacity, the maximum possible data rate for two parties to communicate reliably. In general, the Shannon capacity of timing channels is not known. However, we show in [54] that among all *i.i.d.* continuous and bounded jitter distributions in $(-\epsilon_{max}, \epsilon_{max})$, the Shannon capacity is the smallest when the jitter distribution is uniform in that interval. In other words, the uniform jitter distribution represents the worst case scenario in terms of channel capacity. A special map of the bit-string to packet inter-transmission time, termed the geometric code is proposed in [54]. We have shown in [54] that the rate of the geometric code comes very close to the true capacity of the timing channel with uniform jitter distribution. It is worth noting that the geometric code is universal that works with all bounded jitter even when the underlying unknown distribution is non *i.i.d.*.

We next describe the geometric code and present a simple realization of the geometric code by the *L-bits to n-packets* scheme. We then evaluate the performance of our *L-bits to n-packets* scheme.

4.3.2 Geometric Codes

The family of geometric codes are those codes with T_i to be *i.i.d* geometric random variables with *probability mass function* :

$$P[T_i = \Delta + k \cdot \delta] = p(1-p)^k, \quad k = 0, 1, 2, \cdots$$

We have shown in [54, Lemma 1] that the data rate of such a geometric code is

$$R(p) = \frac{H(p)}{\Delta \cdot p + \delta \cdot (1-p)}$$

where $H(p) = -p \log_2(p) - (1-p) \log_2(1-p)$, and the achievable data rate for any geometric code is:

$$R^* = \max_{0<p<1} [H(p)/(\Delta \cdot p + \delta \cdot (1-p))].$$

Figure 4.3 shows the data rate $R(p)$ as a function of p for two geometric codes with system parameters (Δ, δ) set to $(50, 10)$ ms and $(50, 5)$ ms. As shown, when $\delta = 5$ ms, $R(p)$ attains its maximum value $R^* = 52$ bits/sec. Likewise, when $\delta = 10$ ms, $R(p)$ attains its maximum value $R^* = 40.56$ bits/sec. It is not surprising that with fixed Δ, the achievable rate R^* is higher when δ is smaller – the smaller δ is, the shorter time it takes to transmit all packets.

4.3.3 L-bits to n-packets Scheme

Our design of the *L-bits to n-packets* scheme is based on insights from geometric codes. In this scheme, each of the *L-bit* binary strings is mapped to a sequence of packet inter-transmission times $(T_1, T_2, \cdots, T_n)$. In our basic scheme, T_i takes values only from the set $E = \{T : \Delta + k \cdot \delta, \quad k = 0, 1, \cdots\}$.

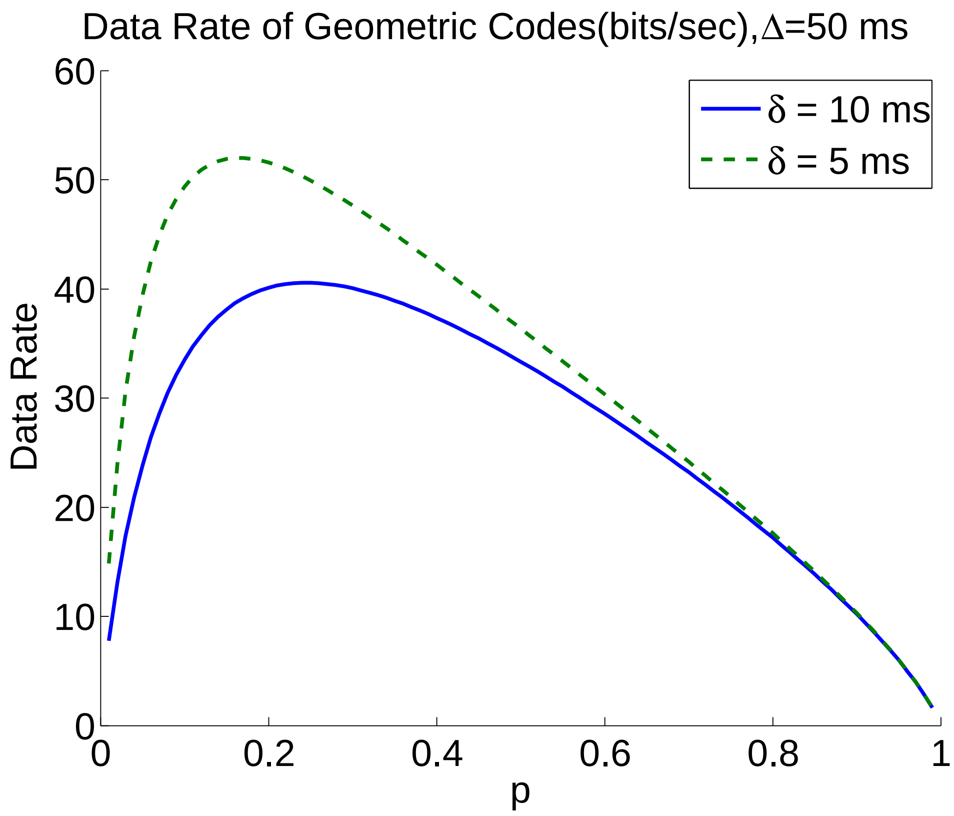

Fig. 4.3. $R(p)$ for geometric codes.

To illustrate our design, we first give examples of a "*2-bit to 1-packet*" scheme and a "*4-bit to 1-packet*" scheme with $\Delta = 60$ ms and $\delta = 20$ ms. (Actually, for the first L bit string, $(n+1)$ packets are needed including the starting packet. But for subsequent L-bit strings, only n packets are needed.) A "*2-bit to 1-packet*" scheme maps bit string "10" in one inter-transmission time $T_1 = 60$ ms. Likewise, it maps bit strings "01", "11", and "00" to T_1=80 ms, 100 ms, and 120 ms, respectively. On average, it takes 90 ms to transmit 2 bits, assuming each bit string is equally likely. So, the data rate is $\frac{2}{0.09} \approx 22$ bits/sec.

Now consider a "*4-bits to 1-packet*" encoding scheme. A total of 16 different values for the inter-transmission times T_1 are needed to represent all the 4-bit binary strings. We can use the following 16 values for the inter-transmission times T_1: 60, 80, 100,

$\cdots$, 340, and 360 ms. On average, it takes 210 ms to transmit 4 bits, and the data rate is $\frac{4}{0.21} \approx 19$ bits/sec.

In these examples, the *2-bits to 1-packet* scheme outperforms the *4-bits to 1-packet* scheme in terms of data rate. It may appear from this example that the data rate of the timing channel monotonically decreases with increasing L. However, the interesting fact our investigation reveals is that the rate is not monotonic with L.

One design challenge is to determine the values for L and n, so that our timing channel achieves a near optimal throughput – close to the data rate of the corresponding geometric code. Thus, given a fixed total packets transmission time t_n ($t_n = \sum_{i=1}^{n} T_i$), we would like to transmit the longest bit strings possible.

To aid our analysis, we introduce another n-dimensional vector $\mathbf{k} = (k_1, k_2, ..., k_n)$ to represent $\mathbf{T} = (T_1, T_2, \cdots, T_n)$, for $T_i = \Delta + k_i \cdot \delta$. We consider a special *L-bits to n-packets* scheme, called an (n, K)*-code*, that satisfies $\sum_{i=1}^{n} k_i \leq K$. Using an (n, K)-code, $t_n \leq n \cdot \Delta + K \cdot \delta$. We have shown that the maximum number of available codewords in an (n, K)-code is $\binom{n+K}{K}$. Therefore, the maximum length of a bit strings that can be mapped to an (n, K)-code is

$$L = \lfloor \log_2 \binom{n+K}{K} \rfloor. \tag{4.4}$$

The data rate $R(n, K)$ of an (n, K)-code with system parameters (Δ, δ) is thus approximately

$$R(n, K) \approx \frac{\log_2 \binom{n+K}{K}}{n \cdot \Delta + \frac{n}{n+1} \cdot K \cdot \delta} \quad \text{bits/sec.} \tag{4.5}$$

A plot of $R(n, K)$ as a function of K is shown in Figure 4.4, for two values of n. In this figure, $(\Delta, \delta) = (50, 10)$ ms, and $n = 3, 5$. As shown, for a fixed value of n, the data rate $R(n, K)$ will initially increase as K increases till it reaches its peak, and it then decreases as K increases. The intuition is that with increasing K, the total number of codewords $\binom{n+K}{K}$ is increasing. Meanwhile, t_n, the time it takes to transmit all n packets, will also increase with K. Initially, the gain in the total number of codewords outpaces the increase in t_n, and we see an increase of the data rate. After a certain point, increase in t_n outpaces the gain in the total number of codes, and we see a decrease of the data rate.

For a fixed value of n, the highest data rate using (n, K)-codes with system parameters (Δ, δ) is approximately

$$R^*(n) \approx \max_{K \geq 0} \frac{\log_2 \binom{n+K}{K}}{(n \cdot \Delta + \frac{n}{n+1} \cdot K \cdot \delta)} \quad \text{bits/sec.} \tag{4.6}$$

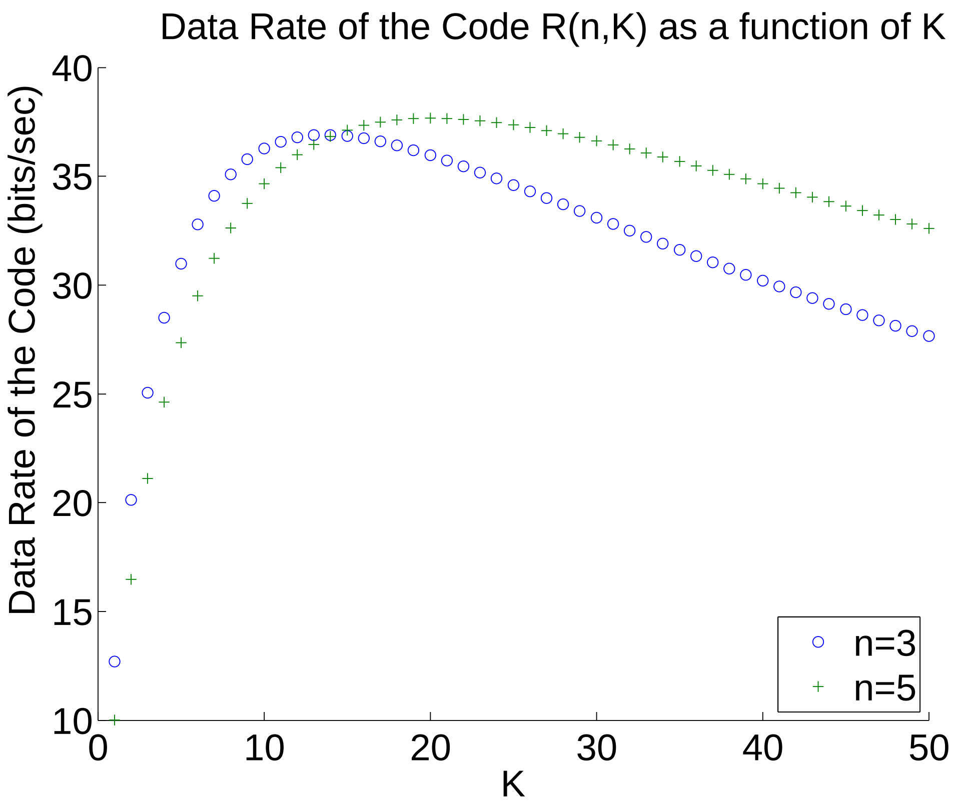

Fig. 4.4. $R(n, K)$ for $\Delta = 50$ ms, $\delta = 10$ ms.

In the $n = 3$ case, the $(3, K)$-code achieves its highest data rate $R^*(3) = 36.96$ bits/sec when $K = 13$. The optimal value of L can be calculated using formula (4). The total number of codewords is $\binom{16}{13} = 560$, and $L = 9$. Thus, when $n = 3$, a *9-bits to 3-packets* gives us the best data rate. Likewise, when $n = 5$, a *15-bits to 5-packets* scheme yields the highest data rate of 37.73 bits/sec.

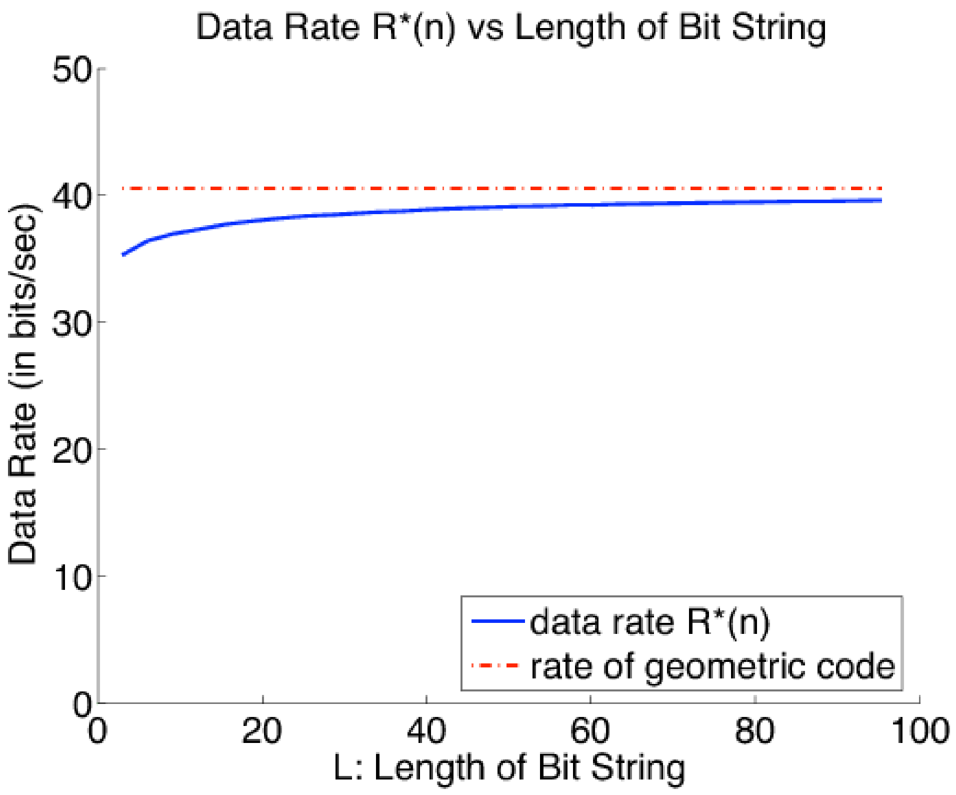

Fig. 4.5. $R^*(n)$ for $\Delta = 50$ ms, $\delta = 10$ ms.

Figure 4.5 shows the optimal data rate $R^*(n)$ as a function of L for the same system parameter $(\Delta, \delta) = (50, 10)$ (ms), along with the rate of the corresponding geometric code. In general, $R^*(n)$ increases as L increases. When L is large, $R^*(n)$ is very close to the geometric code rate. However, the complexity of the codes also increases as L increases. As shown in this figure, in order to gain a small amount of data rate $R^*(n)$, we must increase L drastically. For instance, using a *9-bits to 3-packets* scheme yields a rate of 37 bits/sec, while to achieve a data rate of 39 bits/sec, we need to use a *66-bits to 32-packets* scheme. The latter is much more expensive in terms of storage for the code book and processing for encoding and decoding, since 2^{66} codewords will have to be stored and searched for. However it only offers very little gain in data rate (2 bits/sec).

4.4 Experimental Results

Based on our design, we have developed a covert timing channel software running over TCP/IP networks. Our software is implemented in Java, consisting of a server program and a client program that act as the sender and the receiver, respectively. The sender controls the TCP packet inter-transmission time by using $sleep(T)$, where T is the desired time (in milliseconds) between two packets being transmitted. The accuracy of the $sleep(T)$ function on our system is 1 ms. The receiver passively collects the TCP packet reception times, and uses the shared code book to decode the message. It is a one-way channel in that the sender does not receive feedback from the receiver regarding when the packet is received or whether it is decoded correctly. This limits the performance of the timing channel but increases the difficulty of detection. In our implementation, we choose an *8-bits to 3-packets* scheme for simplicity and efficiency.

As mentioned in our design, our timing channel does not require time synchronization between the sender and the receiver, which makes it attractive for an open network like the Internet. Moreover, the errors occurring earlier will not affect the decoding capability of messages sent later because of independent decoding of each L-bit string. This is in contrast to [58], where a packet delay will cause subsequent bits to be erroneously decoded.

We conducted our experiments using the PlanetLab environment. We ran our covert timing channel software on five pairs of computers. The senders are hosts at Purdue University, and the receivers are PlanetLab nodes located at Beijing Tsinghua University, Technical University of Madrid, University of Zurich, Stanford University and Princeton University. These five pairs are chosen to represent a wide range of Round Trip Times between the senders and receivers. At the receiver side, we use the packets captured by *tcpdump* to decode the timing channel message.

In a single experiment, the sender leaks the information obtained from a text file of 1336 ascii characters to the receiver via our covert timing channel. A set of

Table 4.1
Summary of Decoding Error for the Timing Channel Experiments

Δ	δ	data rate	Princeton		Stanford		Zurich		Madrid		Tsinghua	
(ms)	ms	(bits/sec)	mean(%)	stdev (%)	mean(%)	stdev (%)	mean(%)	stdev (%)	mean(%)	stdev (%)	mean(%)	stdev (%)
50	10	36.85	0.82	0.12	4.27	1.70	3.01	0.34	3.74	1.59	5.51	0.70
50	5	42.92	6.15	3.10	12.19	3.73	4.68	0.52	9.94	7.24	5.88	1.37
40	10	42.75	0.82	0.11	3.10	1.02	3.93	0.75	4.41	1.37	5.19	0.99
40	5	51.14	5.12	1.88	11.51	2.88	4.66	1.02	7.03	5.38	5.06	0.76
30	10	50.90	1.46	0.50	4.49	0.51	3.96	0.48	4.27	1.48	5.53	1.18
30	5	63.24	5.00	1.24	10.41	1.86	4.63	0.97	7.06	4.76	4.44	0.69
20	10	62.87	2.59	0.55	5.18	0.92	4.48	0.85	6.18	4.52	5.03	0.54
20	5	84.15	5.72	1.47	9.20	1.65	3.95	0.96	6.78	5.69	4.39	0.73
10	10	82.21	4.06	1.00	5.96	0.85	5.33	0.89	6.52	3.18	5.81	0.93
10	5	124.28	6.16	1.49	9.69	2.45	4.45	0.82	11.58	11.00	5.29	1.05
		Average RTT (ms)		39.96		67.17		135.65		155.09		272.81

experiments consists of 10 such experiments with system parameters (Δ, δ): $\Delta = 10, 20, 30, 40, 50$ (ms) and $\delta = 5, 10$ (ms). We ran the set of 10 experiments daily between these five pairs of sender and receiver during morning hours (EST) for 10 days.

Table 4.1 summarizes our results of these experiments. In the table, we provide the average and the standard deviation of the character decoding error of our experiments. If one decoded character does not match the transmitted character, it is counted as one decoding error. In our *8-bits to 3-packets* scheme, if one of the three packet inter-reception times deviates too much from the corresponding inter-transmission time, it will result in one character decoding error (1 character is represented by 8 bits). Thus, a 1% of packet inter-reception time error could result in a 3% character decoding error. In addition to the results on error rates, the actual data rate for all the system parameters (Δ, δ), and the average RTT time between the pair just before our experiments are also shown in this Table. Two general trends are observable. As Δ or δ decreases, the data rate and the error rate increase. The data decoding errors are caused when the network jitter exceeds the assumed bound on the jitter. Note that whether the jitter is *i.i.d.* or not, does not affect the performance of the system as our *L-bit to n-packet* scheme is error-free for all bounded jitter. TCP can recover from packet loss, but retransmitting missing packets could result in decoding errors because of the jitter exceeding the assumed bound.

The setting in our experiments between Purdue and Priceton is comparable to that of [58]. As shown, when $\Delta = 40$ ms, $\delta = 10$ ms, the average decoding error rate between Purdue and Princeton is only 0.82%. The data rate of this timing channel is 42.75 bits/sec, which is more than twice the rate (16.76 bits/sec) in [58], while achieving higher accuracy (their error was 2%). When $\Delta = 10$ ms, $\delta = 10$ ms, the average decoding error between Purdue and Princeton is 4.06% and the standard deviation of the decoding error is 1.00%. In this case, we can achieve a rate of 82.21 bits/sec, which is five times the rate of [58].

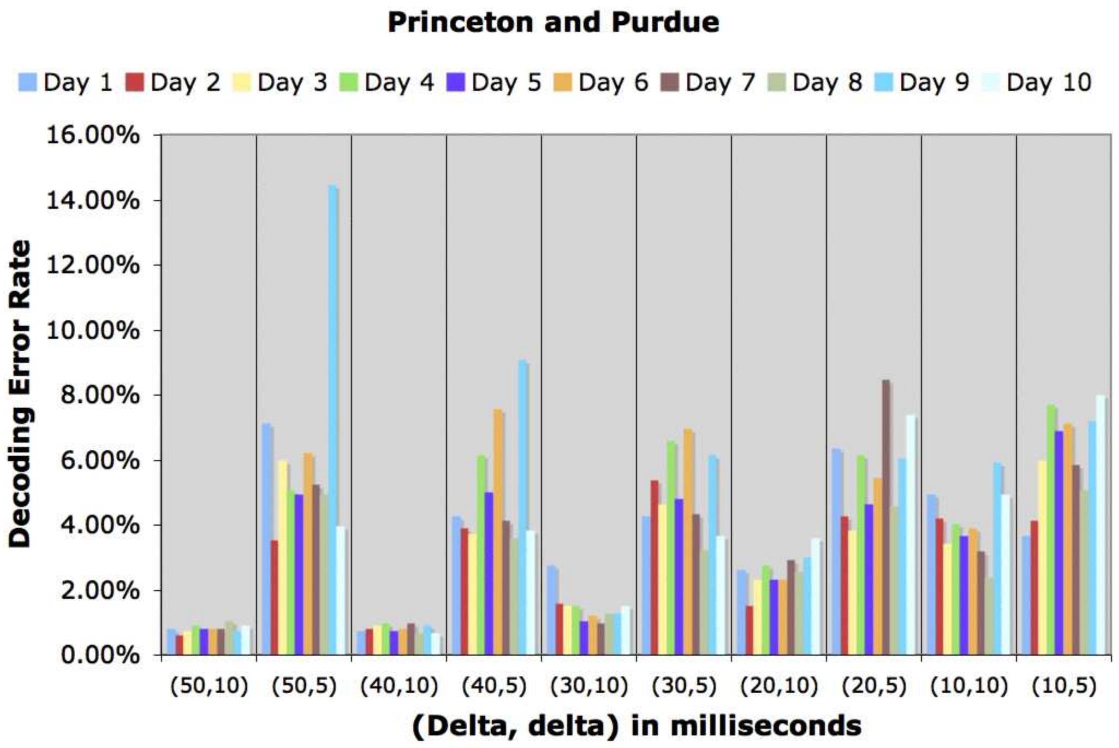

Fig. 4.6. Daily decoding errors for the covert channel between Purdue and Princeton

Figure 4.6 provides a detailed view of our daily experimental results between Princeton and Purdue. We notice there is an error spike (14%) on day 9 when $\Delta = 50$ ms, $\delta = 5$ ms. This could either be due to large variations in packet delays or packet losses on the network. We examined our log and compared the packet inter-transmission times T_i and inter-reception times R_i. We found the error is caused by the jitter of the network. In order to decode correctly, the combined jitter $|R_i - T_i|$ must be less than 2.5 ms for $\delta = 5ms$. In this experiment, 4.26% of packets have a combined jitter $|R_i - T_i| = 3ms$. and the jitter happens in random places. Since 3 packets map to 1 character, these 4.26% jitter results in 12.78% overall character decoding error. This also explains why the error rate is so small when $\delta = 10$, as it can tolerate combined jitter under 5 ms. The histogram of the combined jitter for day 9 is shown in Figure 4.4.

Our experiments with a receiver located outside the US also yielded good results. From Zurich and Purdue, all but one of the average decoding error are less than 5%. When $\Delta = 50$ ms and $\delta = 10$ ms, the average decoding error rate is only 3.01% and the data rate is nearly 37 bits/sec. Daily experimental results are illustrated in Figure 4.7.

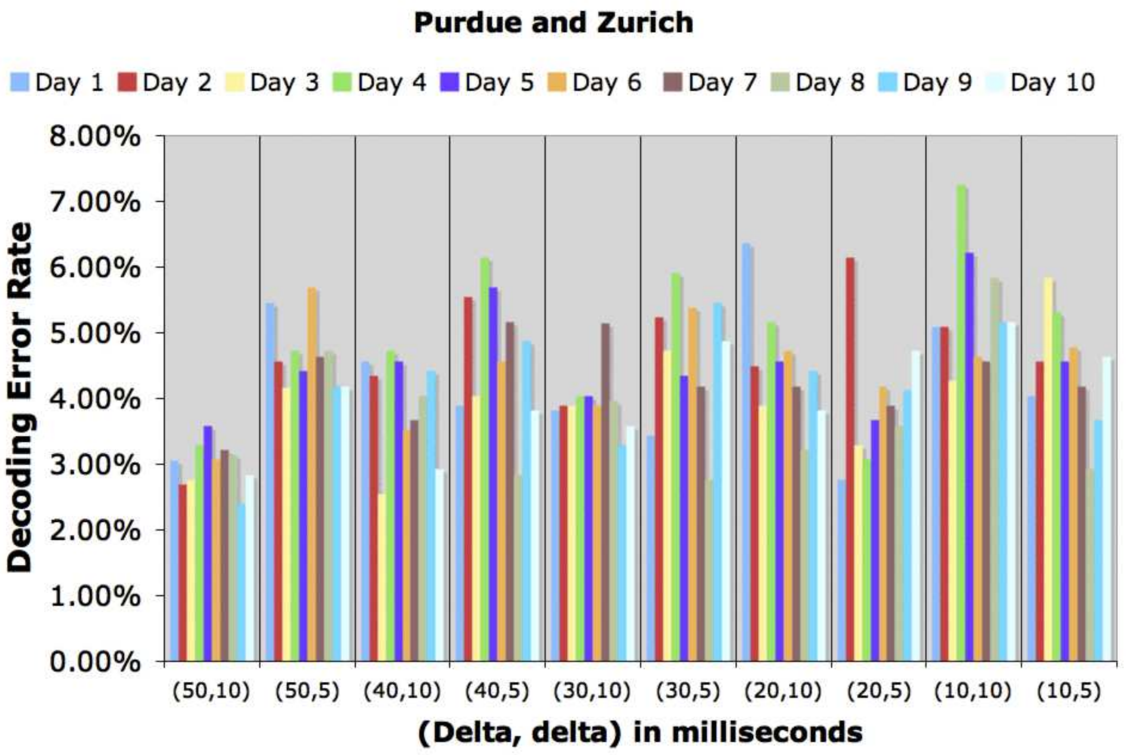

Fig. 4.7. Daily decoding errors for the covert channel between Purdue and Zurich

In addition to the 10 daily experiments we ran between the five pairs, we also ran the timing channel between Princeton and Purdue during various times of the day. We found that the network is more congested during the afternoon hours. The RTT can vary from 63 ms to 108 ms, and the combined jitter spreads more widely ranging from -50 ms to 120 ms. The decoding error thus increases to between 6% to 7%. The histogram of the combined jitter during this time is shown in Figure 4.9. In contrast to Figure 4.4 where the combined jitter is concentrated around 0, the combined jitter during busy hours spreads much more widely, and can range from -50 ms to 120 ms.

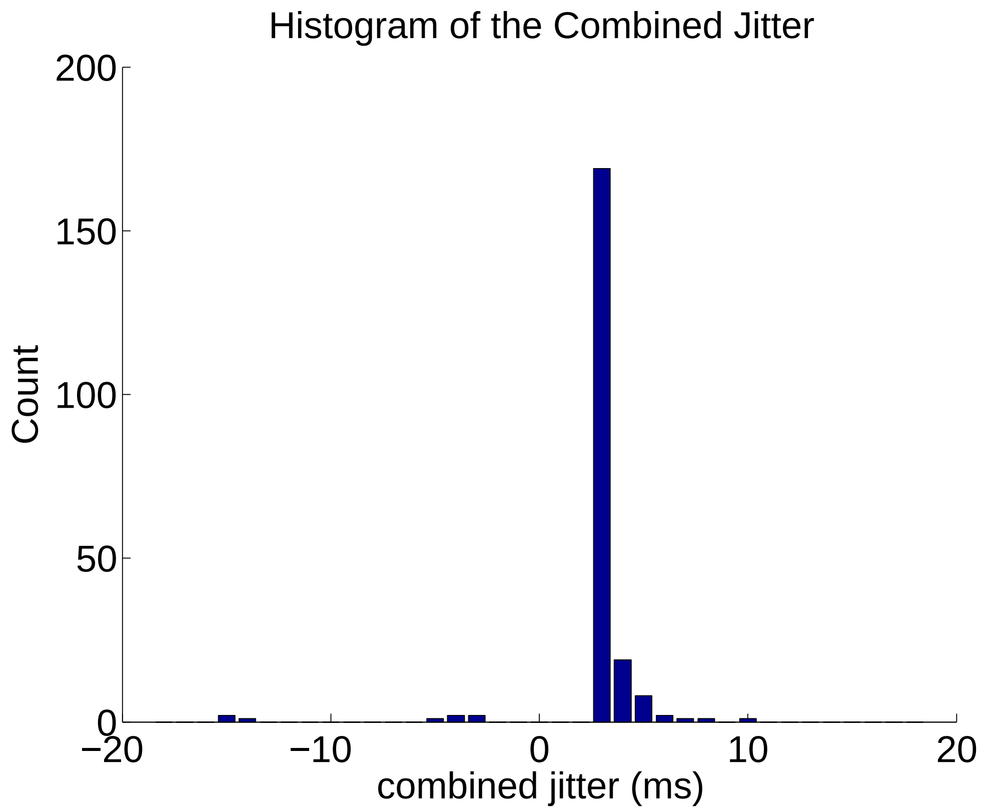

Fig. 4.8. Combined Jitter during Normal Time (Excluding $|R_i - T_i| < 3$)

In these experiments, we demonstrated that our *L-bits to n-packets* scheme achieves good data rates with low error rate under various network conditions. However, this timing channel can also be easily detected, as the inter-transmission times T_i for our basic scheme are aligned on a grid of δ ms. One simple randomized scheme is for the sender and receiver to generate a common pseudo-random sequence $\{v_k, k = 1, 2, \cdots\}$ using a shared seed, v_k is uniformly distributed in $(0, \delta)$. The sender adds the pseudo-random number v_k to the inter-transmission time T_k from the basic code. So, the

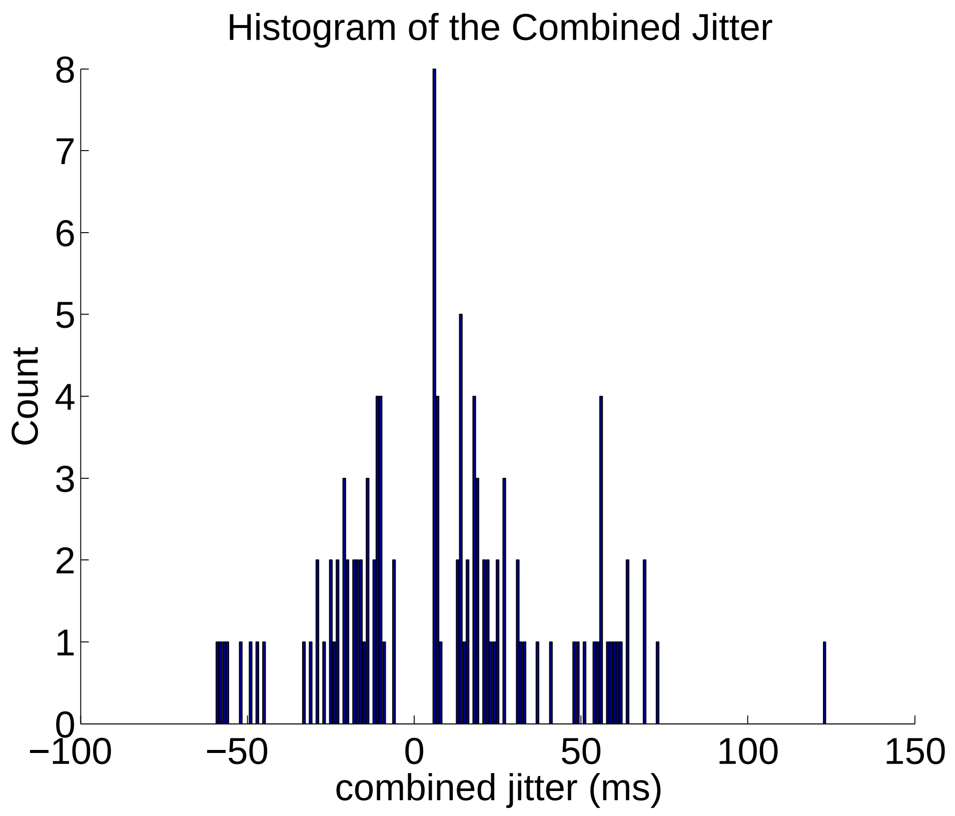

Fig. 4.9. Combined Jitter during Busy Time (Excluding $|R_i - T_i| < 5$)

Fig. 4.10. Combined Jitter during Normal and Busy Time

inter-transmission time for this scheme is $T_k^{rand} = T_k + v_k$. Since the receiver knows the exact value of v_k for all k, it can decode just as in our basic scheme.

However, the above randomized timing channel can still be detected since the probability distribution of T_k^{rand} may not match any legitimate traffic pattern. In the next section, we will introduce a new scheme that allows the timing channel traffic to mimic a given legitimate traffic pattern. Moreover, we will show analytically that it is computationally impossible to detect our proposed covert channel usage using only the first order statistics of the packet inter-transmission time.

4.5 Non-detectable Timing Channel

The design goal of our non-detectable timing channel is for the timing channel traffic to be computationally indistinguishable from a class of legitimate traffic whose packet inter-transmission times are *i.i.d.*, while assuming bounded jitter. Telnet traffic is an example of such traffic, as its inter-transmission times can be modeled by an *i.i.d.* Pareto distribution [62].

A pseudo-random bit generator is called *secure* if an adversary cannot do better than random guessing at the next bit in the sequence from the prefix of the sequence. It has been proved that if a generator passes the next bit test, it will pass *all polynomial-time* statistical tests (Theorem 3.10 in [63]). Cryptographically secure pseudo random number generators (CSPRNG) such as Blum-Blum-Shub, Rabin, and RSA are provably secure PRNG. That is, they are able to generate pseudo random numbers that are computationally indistinguishable from true random numbers. On the contrary, *linear feedback shift registers*, a classical PRNG, is well known to be insecure. A thorough discussion on the theory of computational indistinguishability can be found in [63, 64].

Our non-detectable timing channel scheme is illustrated in Figure 4.11. In what follows, we will use an *8-bits to 2-packets* example to explain this scheme. In this example, an 8-bit ASCII character will be mapped into 2 packet inter-transmission times T_1, T_2 in three step. The shared code book contains the one-to-one mapping of 8-bit binary strings to two-dimensional vectors $(\frac{k_1}{16}, \frac{k_2}{16})$, where k_1 and k_2 are integers between 0 and 15. These 256 vectors $(\frac{k_1}{16}, \frac{k_2}{16})$ are sufficient to accommodate all the 8-bit binary strings. Unlike our first scheme, the vector $(\frac{k_1}{16}, \frac{k_2}{16})$ does not directly correspond to any inter-transmission time, which will be obtained later.

Suppose the sender wishes to transmit a message consisting of a sequence of n characters $\mathbf{msg} = \{c_1, c_2, \cdots, c_n\}$. The first step of our scheme is to look up the codeword for each character in the message. We use (x_{2k-1}, x_{2k}) to denote the codeword

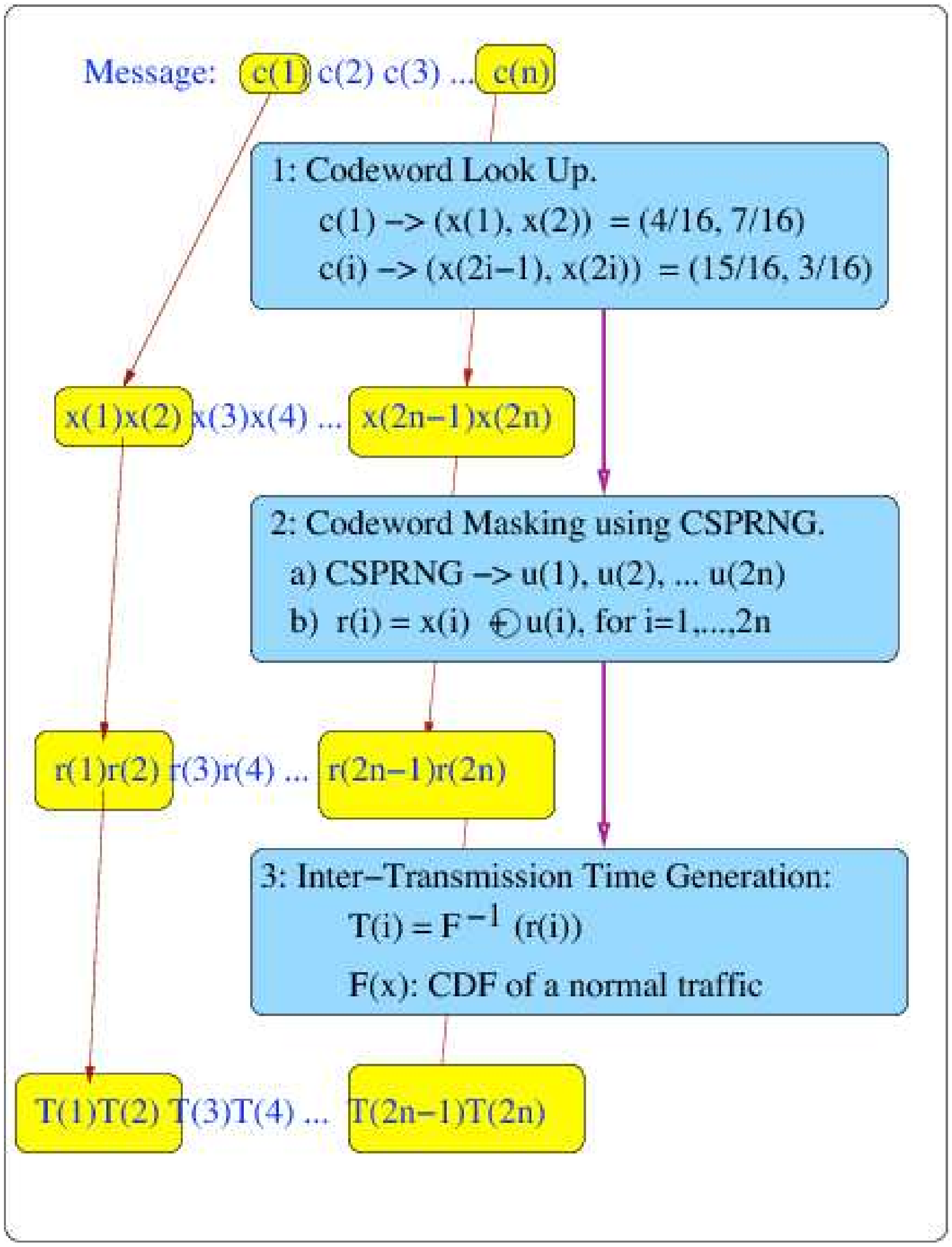
Message: c(1) c(2) c(3) ... c(n)
1: Codeword Look Up.
c(1) –> (x(1), x(2)) = (4/16, 7/16)
c(i) –> (x(2i–1), x(2i)) = (15/16, 3/16)
x(1)x(2) x(3)x(4) ... x(2n–1)x(2n)
2: Codeword Masking using CSPRNG.
a) CSPRNG –> u(1), u(2), ... u(2n)
b) r(i) = x(i) ⊕ u(i), for i=1,...,2n
r(1)r(2) r(3)r(4) ... r(2n–1)r(2n)
3: Inter–Transmission Time Generation:
T(i) = F⁻¹ (r(i))
F(x): CDF of a normal traffic
T(1)T(2) T(3)T(4) ... T(2n–1)T(2n)

Fig. 4.11. Non-Detectable Scheme (Sender)

for character c_k. At the end of the first step, the message **msg** is transformed to a sequence of numbers $\mathbf{x} = \{x_1, x_2, \cdots, x_{2n-1}, x_{2n}\}$.

In the second step, we use a CSPRNG to generate a sequence of pseudo uniform (0,1) random numbers $\mathbf{u} = u_1, u_2, \cdots, u_{2n-1}, u_{2n}$. The seed used by CSPRNG is shared between the sender and receiver, but not with the detector of the covert timing traffic. We then *mask* the sequence $\mathbf{x}$ with $\mathbf{u}$ to obtain a new sequence of numbers $\mathbf{r} = r_1, r_2, \cdots, r_{2n-1}, r_{2n}$ by setting $r_k = x_k \oplus u_k \overset{\Delta}{=} x_k + u_k \bmod 1$. The *masking* can be thought of as the well-known *one-time pad* encryption technique operating on $\mathbf{x}$.

In the last step, we set $T_k = F^{-1}(r_k)$, where $F(x)$ is the given *c.d.f.* of the packet inter-transmission time of legitimate traffic. We use the sequence $T_1, T_2, \cdots, T_{2n}$ as the packet inter-transmission times for message $\{c_1, c_2, \cdots, c_n\}$. It can be shown that without knowing the seed, the sequence $T_1, T_2, \cdots, T_{2n}$ is computationally indistinguishable from a sequence of true *i.i.d.* random variables with *c.d.f.* $F(x)$, i.e., from legitimate traffic.

This computational indistinguishability can be proved using the framework of [63, 64]. The basic idea is the use of *proof by contradiction.* Suppose the following statement "$\mathbf{T}$ *is computationally indistinguishable from a sequence of true* i.i.d. *random variables with* c.d.f. $F(x)$." is **not true**. Then, we can find a polynomial time algorithm Q that can tell that the sequence $\mathbf{T}$ is not a sequence of true *i.i.d.* random variables with *c.d.f.* $F(x)$. Since $r_k = F(T_k)$ and $u_k = r_k - x_k \bmod 1$, we have $u_k = F(T_k) - x_k \bmod 1$. Then, we can construct another polynomial time algorithm Q* based on Q to tell that $u_1, u_2, \cdots, u_{2n}$ is not from a true *i.i.d.* uniform $(0, 1)$ random variables, which contradicts the construction that $u_1, u_2, \cdots$ is generated by a CSPRNG. Therefore, the statement *the sequence* $\mathbf{T}$ *is indeed computationally indistinguishable from a sequence of true* i.i.d. *random variables with* c.d.f. $F(x)$ must be true.

Another feature of our scheme is that it can mimic different traffic patterns using the same code book. Only the *c.d.f.* F(x) for the desired traffic pattern is needed in the last step to obtain $T_1, T_2, \cdots$ by setting $T_i = F^{-1}(r_i)$. This allows the sender and receiver to adapt to various traffic patterns easily. For instance, when the normal traffic pattern changes, the sender only needs to determine the *c.d.f.* of the distribution of the normal traffic, and it can use the new *c.d.f.* to map $\mathbf{r}$ to the desired packet inter-transmission times without changing the existing code book. Moreover, adaptation does not require any handshake between the sender and receiver, for the receiver can independently compute the *c.d.f.* using the traffic pattern of the inter-packet reception times.

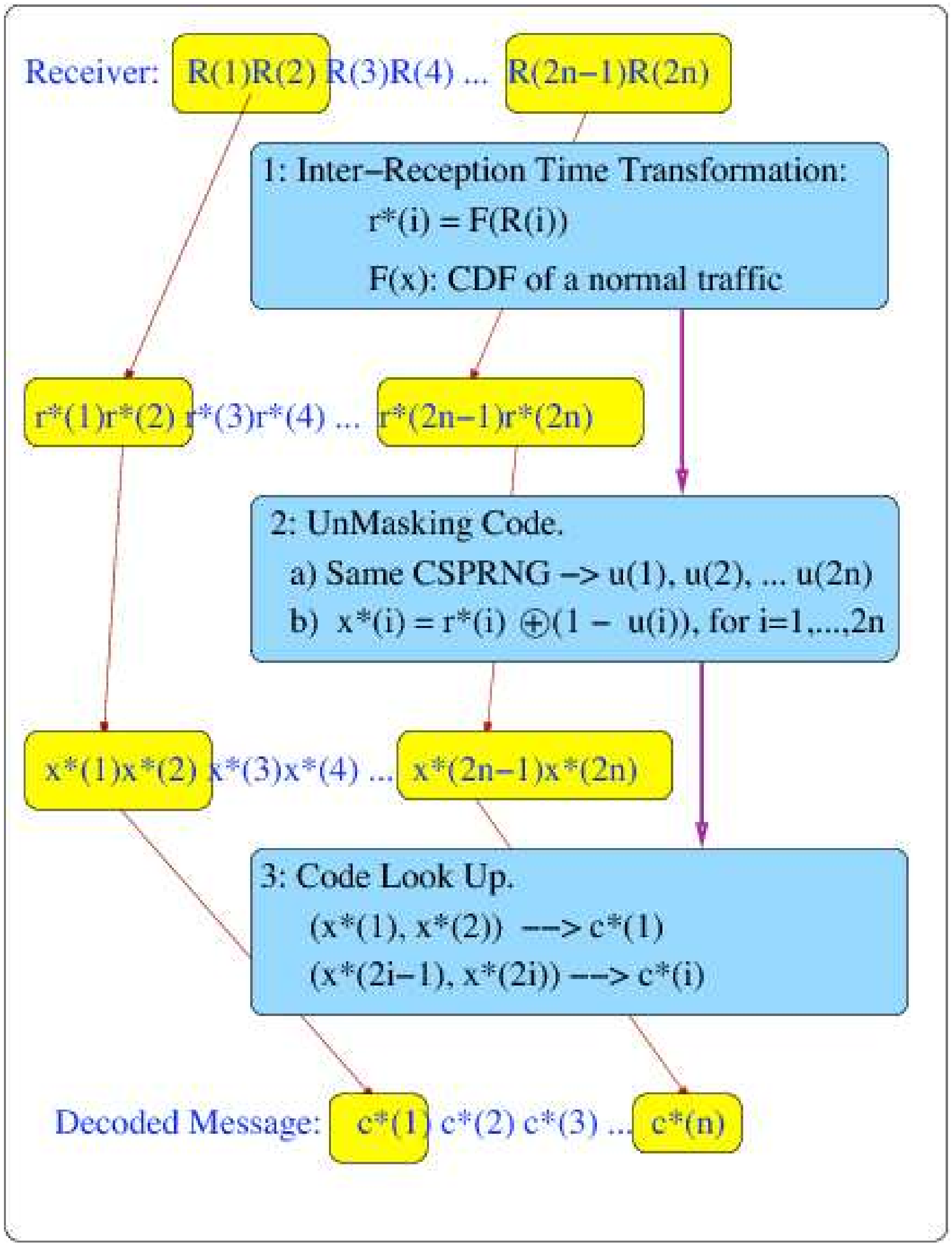
Receiver: R(1)R(2) R(3)R(4) ... R(2n−1)R(2n)
1: Inter−Reception Time Transformation:
r*(i) = F(R(i))
F(x): CDF of a normal traffic
r*(1)r*(2) r*(3)r*(4) ... r*(2n−1)r*(2n)
2: UnMasking Code.
a) Same CSPRNG −> u(1), u(2), ... u(2n)
b) x*(i) = r*(i) ⊕(1 − u(i)), for i=1,...,2n
x*(1)x*(2) x*(3)x*(4) ... x*(2n−1)x*(2n)
3: Code Look Up.
(x*(1), x*(2)) −−> c*(1)
(x*(2i−1), x*(2i)) −−> c*(i)
Decoded Message: c*(1) c*(2) c*(3) ... c*(n)

Fig. 4.12. Message Recovery (Receiver)

The procedure for recovering the message at the receiver is simply the reverse of the sender scheme, and is shown in Figure 4.12. Let $R_1, R_2, \cdots, R_{2n}$ be the packet inter-reception times. We first calculate $x_i^* = F(R_i) \oplus (1 - u_i)$ as depicted in step 1 and 2. In the last step, we first round x_i^* to the nearest value of $\frac{k}{16}$, denoted as x_i^d ($x_i^d = \lfloor 16 \cdot x_i^* + 0.5 \rfloor / 16$). We then decode (x_{2k}^d, x_{2k+1}^d) to character c_k^d by looking up the code book. The entire recovered message is then $c_1^d, c_2^d \cdots, c_n^d$.

The value of network jitter ϵ_i's and the *c.d.f.* $F(x)$ of the legitimate traffic affect the decoding error, although the jitter does not need to be *i.i.d.*. Recall that $R_i = T_i + (\epsilon_i - \epsilon_{i-1})$, where $\epsilon_i - \epsilon_{i-1}$ signifies the combined jitter in the network and will be denoted as $\epsilon_i^{(c)}$. In this example, the receiver can decode correctly if $|\epsilon_i^{(c)}| < 1/(32 \cdot \sup |F'(x)|)$, where $F'(x)$ is the first order derivative of $F(x)$. This is because, $F(R_i) = F(T_i + \epsilon_i^{(c)}) = F(T_i) + F'(t^*) \cdot \epsilon_i^{(c)} = r_i + F'(t^*) \cdot \epsilon_i^{(c)}$, where $t^* \in (T_i, T_i + \epsilon_i^{(c)})$. Since $x_i = r_i \oplus (1 - u_i)$, we have $x_i^* = F(R_i) \oplus (1 - u_i) = x_i + F'(t^*) \cdot \epsilon_i^{(c)}$. Thus, $|x_i^* - x_i| < 1/32$, if $|\epsilon_i^{(c)}| < 1/(32 \cdot \sup |F'(x)|)$. This allows correct decoding (i.e. $x_i^d = x_i$), since x_i^d is the value of x_i^* rounded to the nearest $k/16$.

The example of *8-bit to 2-packet* scheme is readily generalized to an *L-bit to n-packet* scheme. The code book contains a one-to-one mapping of L-bit binary strings to n-dimensional vectors $(\frac{k_1}{K}, \cdots, \frac{k_n}{K})$, where K is a positive integer and $k_1, k_2, \cdots, k_n$ are non-negative integers smaller than K. Similar to the analysis for the above *8-bit to 2-packet* example, the value for K can be conservatively estimated as $K < ((\sup_x F'(x)) \cdot \epsilon_{max})^{-1}$ for correct decoding.

To demonstrate our non-detectable timing channel, we implemented an 8-bit to 2-packet scheme in which the timing channel traffic mimics the telnet traffic pattern. We use Java's *SecureRandom* class for the generation of cryptographically secure pseudo random numbers.

A Pareto distribution has a *c.d.f.*:

$$F(x) = P[X \leq x] = 1 - (\alpha/x)^\beta, \; x > \alpha, \alpha, \beta > 0$$

The inverse function of $F(x)$ needed in the step 3 is:

$$F^{-1}(x) = \alpha(\frac{1}{1-x})^{1/\beta},\ 0 < x < 1$$

In our experiments, we use parameter $\alpha = 100ms$ and the shape parameter $\beta = 0.95$ as in [62]. The receiver is a PlanetLab node at Princeton University, and the sender is a host at Purdue University. We sent the same text file as in the previous experiments over this non-detectable timing channel. Our experiments were conducted during peak time when the RTT varies from 39.8 ms to 63.5 ms. The data rate is approximately 5 bits/sec, and the decoding error is only 1%. Figure 4.13 illustrates the distribution of the packet inter-transmission times from this experiment, along with Pareto and geometric distributions. We observe that visually the resulting traffic is similar to Pareto but distinct from geometrically distributed traffic. As the distribution of the inter-transmission time matches that of a given distribution, our scheme guarantees undetectability for any detector using only the first order statistics of the traffic. It is worth noting that the same concept can be applied to mimic traffic up to the second order (or even higher order) statistics. We are currently investigating efficient implementation that establishes the indistinguishability for higher order state-of-the-art detectors. For example, a recent covert channel detection scheme [81] uses entropy changes in correlated traffic to detect covert timing channels. Their detection method is based on the observation "that a covert timing channel cannot be created without causing some effects on the entropy of the original process". As the authors pointed out, this observation does not apply to *i.i.d.* processes as in our setting. An interesting future research direction is to derive a model for some normal traffic with correlated inter-packet transmission times, and to design a covert timing channel that mimics the correlated traffic.

4.6 Conclusion

We have designed a robust *L-bits to n-packets* scheme for communication using timing channels. The data rate of our scheme is close to the theoretical upper bound

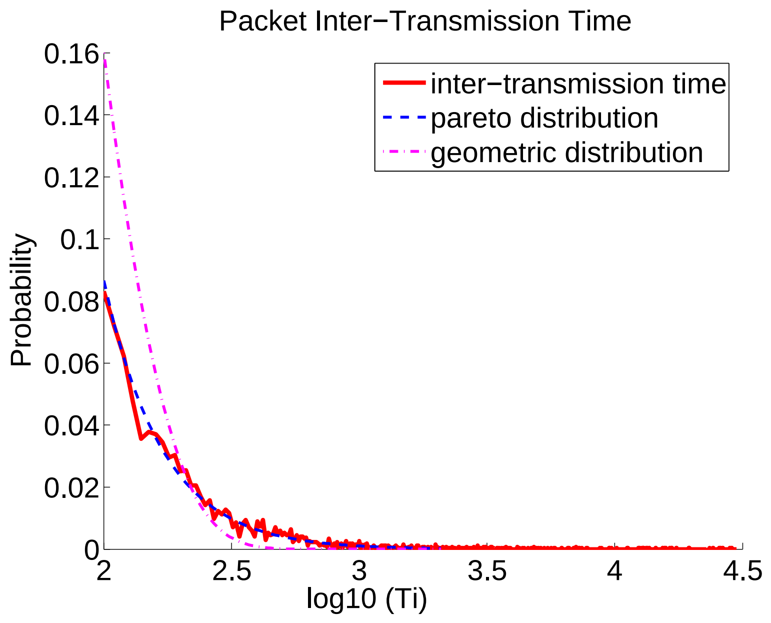

Fig. 4.13. Probability Distribution of T_i.

– the achievable rate of the geometric codes. We have implemented our scheme and have conducted extensive experiments on the PlanetLab nodes and found that our scheme achieves between two to five times the data rate of the previous state-of-the-art. In local networks with greater control over timing, one can significantly improve the achieved data rate. Thus, the data leakage rate can be much higher if the receiver is planted closer to the sender. We have also designed a computationally non-detectable timing channel scheme so that the distribution of the inter-transmission times generated by our timing channel mimics any *i.i.d.* traffic pattern. The non-detectable scheme results in a drop of the data rate; however it is still able to achieve a rate of 5 bits/sec for telnet traffic with only 1% decoding error even during peak time. This suggests that TCP/IP timing channels can be far stealthier than previously thought possible.

There are several interesting future directions for this work. One is to investigate the effect of jammers when additional jitter is added to the TCP/IP flow. Another is to design a computationally non-detectable covert timing channel that mimics correlated traffic.

4.7 Acknowledgment

We thank Mr. Pablo Navarrete and Professor Edward Coyle for assisting us with our initial experiments. We thank Professor Ninghui Li for very helpful discussions on computational indistinguishability. This work has been supported in part by ARO No. W911NF-08-1-0238, NSF grants CNS-0523249 and CNS-0712846.

5. CAMOUFLAGING TIMING CHANNELS IN WEB TRAFFIC

5.1 Introduction

Due to the rapid growth of Internet and Web based applications, covert communication over the Internet has received increased attention from industry and the research community. Since traditional firewalls do not usually exam packet inter-arrival times, covert timing channels could be an attractive way for confidential communication between untrusted systems. With the emergence of new designs of network timing channels, detection methods attempt to differentiate these timing channels from legitimate traffic. The question that we ask us is whether we design network timing channels that behave like legitimate traffic and that are robust to the timing noise characteristic on the Internet.

In our prior work, we have presented a computationally non-detectable timing channel for mimicking legitimate *i.i.d.* traffic [82] . The marginal distribution of the packet inter arrival times from this timing channel can be any probability distribution. However, these *i.i.d.* timing channels cannot mimic traffic that is auto-correlated.

Extensive research has shown that aggregated traffic is self similar and long range dependent (LRD). For instance, the TCP connection start time for HTTP traffic, the most observed traffic on the Internet [86], is shown to be LRD and non-stationary [67]. Our goal is to design a covert timing channel that can mimic LRD traffic, such as HTTP traffic. In particular, we would like our timing channel traffic not only to have the desired marginal distribution, but the same autocorrelation function (equivalently, same long range dependence) as legitimate traffic trace data.

To achieve this goal, we first analyze traffic trace data for HTTP traffic from 2009 available from CAIDA [1]. We use the insights from these traffic traces to revive and expand a prior statistical model for HTTP traffic [67] to better fit current data. Our model uses a Fractional Auto-Regressive Integrated Moving Average (FARIMA) time series to capture LRD behavior and also matches the marginal distribution of the data. We then create covert timing channel traffic by embedding covert information in this model without disturbing any first- or second-order statistical behavior of the legitimate traffic. We implement the LRD timing channel and conduct experiments with geographically distributed senders and receivers on PlanetLab. Our experimental results indicate that the new timing channel traffic is statistically indistinguishable from legitimate traffic, and our tests confirm it evades the best available detection methods.

One scenario of using this timing channel is when a sender resides in a country with tight censorship. She is able to manipulate the inter-transmission times of aggregated HTTP traffic within her organization to communicate in a covert manner with a receiver who is outside the geographical boundary of the country and where there are no such censorship laws. Since HTTP traffic has a high volume in most settings, the sender can achieve reasonable rates of covert communication. The receiver intercepts the traffic en-route to the final web servers, observes the inter-reception times, and decodes the privileged information from them using a code-book that the sender and the receiver have agreed to *a priori*.

The remainder of our paper is organized as follows: In Section 5.2, we review related work on network timing channels and long range dependent traffic. In Section 5.3, we present our analysis on CAIDA data and updated traffic models. In Section 5.4, we describe our design and implementation of covert HTTP timing channels. Our experimental results are described in Section 5.5. We conclude with discussion and future research directions in Section 5.6.

[1]Support for CAIDA's Internet traces is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA Members.

5.2 Related Work

Past research on network timing channels has investigated channel capacity, schemes for reducing the capacity, designs of network timing channels, and detection schemes to identify the existence and usage of timing channels.

Early implementation of an on/off timing channel [58] demonstrated the feasibility of leaking information by a network covert timing channel. In [65], the authors built a Keyboard JitterBug, a device interposed between the keyboard and the computer, that can leak typed information through a covert network timing channel when a user runs an interactive application such as *ssh*. We designed a timing channel [82] that maps L-bits strings to n packet inter transmission times.This design includes both the on-off scheme and the keyboard jitter bugs as special cases. It significantly improved the data rate of the timing channel. In fact, the data rate of this scheme is close to the theoretical upper bound – the achievable rate of the geometric codes.

Given the threat of clandestinely leaking information by network covert timing channels, researchers found ways to detect them [56,58,81]. However, network timing channels can be very surreptitious. In [82], we constructed a computationally non-detectable timing channel, mimicking any *i.i.d.* legitimate traffic patterns. The inter-transmission times from telnet traffic are shown to be *i.i.d.* from a Pareto distribution [62]. When used to imitate telnet traffic, our timing channel can evade detections entirely. In spite of the strong non-detectability property, the usage of this timing channel is limited to imitating *i.i.d.* legitimate traffic. It cannot be used to imitate correlated traffic such as HTTP traffic, which represents more than 50% of Internet traffic today. It is well-established that HTTP traffic is non-stationary and long range dependent (LRD) [67, 68]. LRD means the autocorrelations are positive and decay slowly, thus are not summable. The focus of this work is to construct timing channels that emulate the long range dependence of HTTP traffic.

The discovery of long range dependence and self-similarity of Internet traffic ([76]) had profound effect on our understanding of network traffic. Many papers on long range dependent and self similar network traffic have been published. Readers are referred to [73] for a comprehensive overview.

Mathematical models and simulations are essential for network performance evaluation, traffic controls, and resource provisioning. Synthesized long range dependent traffic is required as an input process for simulations in order to reflect the reality that many traffic variables are long-range persistent. Fractional Gaussian noise (FGN) and fractional ARIMA (FARIMA) processes are the two most widely-used input processes for network simulations. Cleveland *et al.* proposed a stochastic model well suited for TCP start time for HTTP traffic [67]. The authors analyzed 23 million TCP connections organized into 10704 blocks of approximately 15 minutes each. They used a FARIMA sequence to capture the long range persistence, and the model produces synthetic traffic stochastically similar to that from the actual wire of an Internet link.

A fast Fourier transform method for synthesizing approximate sample paths for Fractional Gaussian Noise (FGN) is proposed in [74]. A summary of other methods for generating realistic network traffic can be found in [74].

Synthesizing a realistic traffic trace is evidently a crucial part of constructing timing channel traffic to evade detection. Unlike traditional modeling and synthetic trace generation, the synthetic traffic for our timing channel must meet more rigorous requirements: 1) be statistically indistinguishable (not just similar) from the underlying HTTP traffic to avoid detection; and 2) be able to transmit covert information through timing, and of course, be decodable by the receiver.

Next, we will present a stochastic model for HTTP new connection inter arrival times based on packets trace data collected by CAIDA in 2009.

5.3 Model for A LRD Timing Channel

The goal of this research is to design a network timing channel that can be hidden within HTTP traffic. *Persistent connections* are adopted in HTTP/1.1, that is, a single TCP connection is created and reused for multiple HTTP request/response interactions. Our timing channel will use the new TCP connection inter-arrival times, not HTTP packet inter-arrival times, to carry covert information. Due to the network jitters distorting timing information, larger inter-arrival times are more resilient to decoding errors. Therefore, the TCP connection inter-arrival times, larger than HTTP packet arrival times, are more reliable for transmitting covert information. The tradeoff of the enhanced robustness is the corresponding throughput reduction when compared to the packet inter-arrival time schemes in [58, 65, 82].

We use the packet traces collected by CAIDA in March 2009 as a baseline for comparing our covert timing traffic and legitimate traffic. This dataset contains anonymized passive traffic traces from CAIDA's Equinix-Chicago and Equinix-Sanjose monitors on OC192 Internet backbone links. The Equinix-Chicago Internet data collection monitor is located at an Equinix datacenter in Chicago, IL, and is connected to an OC192 backbone link (9953 Mbps) of a Tier1 ISP between Chicago, IL and Seattle, WA.

The original data set from Equinix-Chicago direction A contains approximately 15 GB of compressed data. We first extract the new TCP connection packets for HTTP from the dataset. The resulting data is only about 1% of the original data. Since the attack scenario under consideration is a compromised edge router of an enterprise network leaking information via timing channels, we further partition the new TCP connection packet trace into subnets according to their 8 bits network prefix. Within each subnet, we divide the one-hour trace into four 15-minutes intervals as in [67].

5.3.1 Limitation of the Existing Model for HTTP Traffic

A statistical model [67] for TCP new connection times was developed, based on traffic traces collected at Bell Labs between 1998 and 2000. The authors conducted extensive empirical as well as in-depth theoretical studies of 23 million TCP connections collected at Bell Labs between 1998 and 2000. They concluded that TCP start times for HTTP are nonstationary and LRD, the marginal distribution of the inter-arrival times is approximately Weibull, and the autocorrelation of the log inter-arrival times is modeled by adding white noise to a FARIMA time series.

FARIMA models are generalizations of *Autoregressive Integrated Moving Average* ARMA model by allowing fractional values d in the degree of difference [69]. It is commonly used to model long range dependent behavior. The FARIMA series s_j can be generated from equation (1):

$$(I - B)^d s_j = \epsilon_j + \epsilon_{j-1} \tag{5.1}$$

where B is the backward shift operator ($Bs_j = s_{j-1}$) and ϵ_i are *i.i.d.* Gaussian random variables with mean 0 and variance σ_ϵ^2. Their model is developed for data with Hurst parameters around 0.75, and they used a fixed value 0.25 for the degree of difference d. The Hurst parameters for the Bell Lab data are approximately 0.75, and the relationship between H and d is $H = d + 0.5$.

One advantage of this model is that only one parameter, r, the rate of new TCP connection arrival times, is needed to generate the traffic trace. In their model, the parameters (α and λ) for a Weibull distribution are functions of the connection rate r. The corresponding *c.d.f.* of the marginal distribution of the new connection inter-arrival times for is:

$$F(t) = 1 - e^{-(t/\alpha(r))^{\lambda(r)}}, t \geq 0 \tag{5.2}$$

The Internet has changed tremendously in the last decade. It is not surprising that this model does not fit current CAIDA data well. For instance, we select a dataset consisting of a 15-minute block of CAIDA traffic trace from a subnet, and compare it with the model in [67]. The load is calculated as $r = 0.6387$ connection/seconds for this data. We then create a synthetic dataset according to the model in [67] using $r = 0.6387c/s$.

Figure 5.1 compares the empirical cumulative distribution functions from the CAIDA trace and the trace generated according to the model in [67]. Although it may appear that their empirical *cdfs* are very close to each other, the two-sample Kolmogorov-Smirnov test rejects the null hypothesis that these two samples are drawn from the same distribution at level 0.05. The maximum distance between the two empirical distributions is 0.2558, and the p-value is $2.8 \cdot 10^{-27}$. This conclusion is further confirmed visually by the Weibull plots in Figure 5.2. While the marginal distribution of CAIDA's new TCP connection times is approximately Weibull, it does not come from the same Weibull distribution as the model in [67].

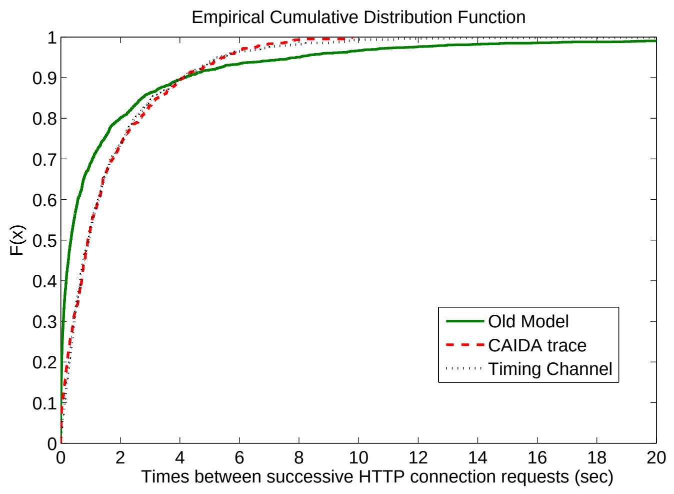

Fig. 5.1. Comparison of Empirical cdf's

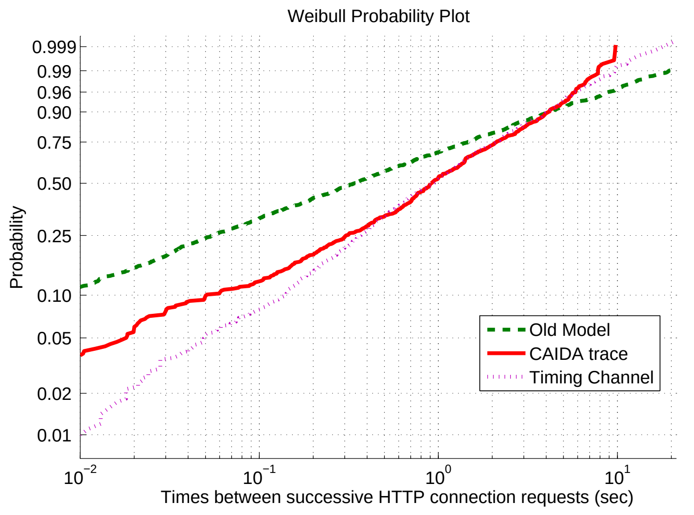

Fig. 5.2. Fragment of Weibull Plots

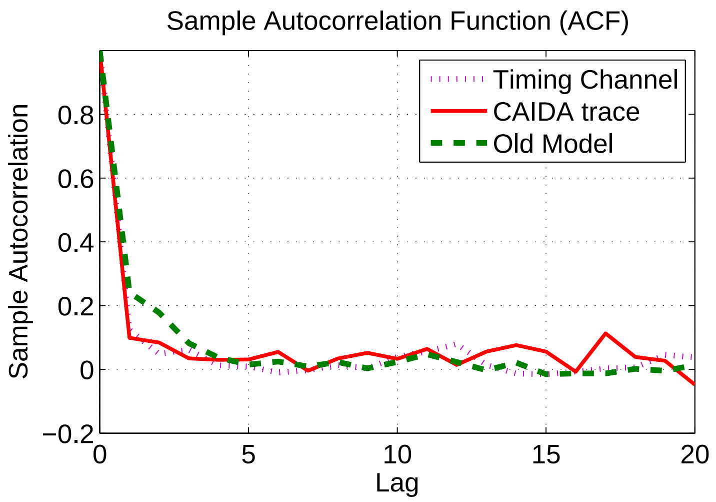

Fig. 5.3. Sample Autocorrelation Functions

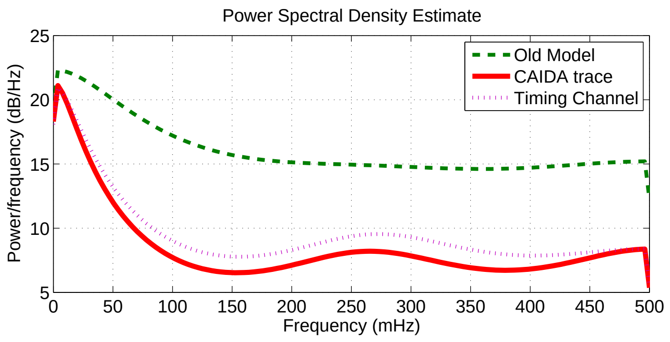

Fig. 5.4. Power Spectrum Density Estimates

In addition to the mismatch of the marginal distribution between the CAIDA data and the model, the second order statistics from the two data sets also differ. The Hurst parameter for the log inter-arrival times in the CAIDA data set is 0.61, while the model is developed for data with Hurst parameters around 0.75. In the existing model, no calculation of the Hurst parameter value is done based on the data. Instead, the value of the Hurst parameter is fixed as 0.75. Another misfit is the autocorrelation function. Since it is difficult to discern the differences from the autocorrelation plots in Figure 5.3, we plot the power spectrum density (PSD) estimates in Figure 5.4. It shows significant difference between the PSD estimates of the CAIDA data and of model in [67]. The PSD and the ACF of a time series form a Fouries transform pair, i.e. the PSD is the Fourier transform of the autocorrelation function of the time series, assuming the time series is wide sense stationary. The difference in PSD suggests different second order statistics.

The discrepancy described above indicates the need of updating the model for HTTP traffic in order to create timing channels that can hide in today's HTTP traffic.

5.3.2 New Model for HTTP Traffic

We first design a new model that expands the model in [67], so that it can model traffic with Hurst parameters other than 0.75. In our model, the scale and shape parameters for Weibull distribution, denoted by α and λ respectively, are estimated directly from the data, not computed from the load parameter r, to better fit a particular trace data.

Table 5.1 contains the notations we use in our model[2]. The TCP new connection inter-arrival times are denoted as $t_j, j = 1, \cdots, n$. Since t_j can vary by several orders of magnitude, $l_j = \log_2(t_j)$ is used for model fitting as in [67]. The marginal distribution of the $t_i's$ is approximately Weibull, with shape parameter $\lambda > 0$ and scale parameter $\alpha > 0$. Its cumulative distribution function (*c.d.f.*) is then:

$$F(t) = 1 - e^{-(t/\alpha)^{\lambda}}, t \geq 0; \alpha, \lambda > 0 \tag{5.3}$$

The pseudocode for generating synthetic traces using our model is shown in *Algorithm 3.1 NewModel*, There are four *input* parameters in our synthetic trace generation model: α, λ, H, and ρ_1. Here, α and λ are the scale and shape parameters of the Weibull distribution, H is the Hurst parameter of $l_j = \log_2(t_j)$, and ρ_1 is the autocorrelation of l_j at lag 1. The values of all four parameters are estimated directly from the data trace.

[2]we use the same notation as [67] whenever possible in our new model

In the first 10 steps, we calculate all the parameters needed for synthetic trace generation. We will show the derivation of the values of these parameters shortly as we build our model. Steps 11 to 15 in *for loop* are the main component for generating n data points.

The *output* of the algorithm is simply an array that contains the sequence of inter-arrival times. They are statistically indistinguishable from today's HTTP traffic. As shown in Figures 1 to 4, the inter-arrival times from our new model match the marginal distribution and second order statistics of the real traffic trace. This new model will later be used to generate covert timing channel traffic, which will be shown in Section 5.4.

The first step, $d = H - 0.5$ calculates the degree of difference in the FARIMA model. The Euler constant $\gamma = 0.5772$ is set in step 2, and it is used in step 3 for calculating the mean of $l_j = \log_2(t_j)$. The variance of l_j is calculated in step 4 using the Weibull shape parameter λ. The variance of s_j (denoted as σ_s^2) is calculated in step 5, and it is used in step 11 for FARIMA series generation. The value of σ_n^2 is calculated in step 6, and is used, along with the value of μ_l from step 3, to generate an *i.i.d.* Log-Weibull sequence in step 12. We will show how the formulas for σ_s^2 and σ_n^2 are developed shortly. The step 7 calculates the load r of the TCP new connections as $r = 1/E[t_j]$, and $E[t_j] = \Gamma(\alpha(1 + 1/\lambda))$. Note, $\Gamma(\cdot)$ is the Gamma function defined as: $\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$. The load r is then used in steps 8 through 9 to calculate parameters b_0, b_1, and b_2, which are used to obtain l_j from v_j in step 14.

Algorithm 5.3.1: NEWMODEL($\alpha, \lambda, H, \rho_1$)

```
[1]$d = H - 0.5$
[2]$\gamma \leftarrow 0.5772$          //Euler Constant
[3]$\mu_l = \log_2(\alpha) - \gamma \log_2(e)/\lambda$,
[4]$\sigma_l^2 = \pi^2 \log_2^2(e)/6\lambda^2$
[5]$\sigma_s^2 = \sigma_l^2 \rho_1 (2-d)/(1+d)$
[6]$\sigma_n^2 = \sigma_l^2 - \sigma_s^2$
[7]$r = 1/(\alpha\Gamma(1+1/\lambda))$
[8]$b_0 = 0.7 - e^{-0.7088-0.05857r}$
[9]$b_1 = 1 - e^{-1.6301-0.06399r}$
[10]$b_2 = -e^{-4.1896-0.06254r}$
for $j \leftarrow 1$ to $n$
  do {
    [11]  $s[j] \leftarrow$ FARIMA sequence with variance $\sigma_s^2$
    [12]  $n[j] \leftarrow$ $i.i.d.$Log-Weibull $(\mu_l, \sigma_n^2)$ sequence
    [13]  $v[j] = s[j] + n[j]$
    [14]  $l[j] = b_0 + b_1 v[j] + b_2 v^2[j]$
    [15]  $t[j] = 2^{l[j]}$
  }
return $(t)$
```

We now will explain the two key components, the FARIMA sequence (step 11) and the *i.i.d.* random sequence with a Log-Weibull distribution [3] (step 12) in our algorithm. We will show how we use these two sequences to build a model that matches the first and second order statistics of a data trace.

A FARIMA series is commonly used to model long range dependent behavior [69]. It is a generalization of ARMA model by allowing fractional values d in the degree of difference. The LRD series s_j can be generated from equation (1):

$$(I - B)^d s_j = \epsilon_j + \epsilon_{j-1} \tag{5.4}$$

[3]Log-Weibul is a type of extreme-value distributions.

where B is the backward shift operator ($Bs_j = s_{j-1}$) and ϵ_i are *i.i.d.* Gaussian random variables with mean 0 and variance σ_ϵ^2.

The long range dependence property of t_j is well modeled by Eq (1) [67, 69]. The advantage of FARIMA models is that it can capture the LRD using only one parameter – Hurst Parameter H. The degree of difference d is $d = H - 0.5$. In the earlier model [67], d is fixed to 0.25; That model fitted the Bell Lab data well since the Hurst parameters from those data are approximately 0.75. When a traffic trace has a significantly lower or higher Hurst parameter, like recent CAIDA data, the old model is no longer appropriate. One of our contribution is to model LRD traffic with wide range of Hurst parameters, we allow d to be in $(0, 0.5)$, corresponding to Hurst parameters in $(0.5, 1)$.

The *i.i.d.* Log-Weibull sequence $\{n_j\}$ is added to s_j, in an attempt to capture the first order and second order statistics of l_j. This method was first proposed in [67]. The random sequence $\{n_j\}$ and $\{s_j\}$ are not correlated, i.e. $cov(n_i, s_j) = 0$ for all i, j. Recall that $l_j = \log_2(t_j)$, and t_j is Weibull(α, λ). Thus l_j has a Log-Weibull distribution with mean μ_l and variance σ_l^2. The values of μ_l and σ_l^2 can be expressed in terms of the Weibull parameters λ and α: $\mu_l = \log_2(\alpha) - \gamma \log_2(e)/\lambda$, and $\sigma_l^2 = \pi^2 \log_2^2(e)/6\lambda^2$.

Our goal is to obtain l_j from s_j and n_j. For the ease of expositions, we use an intermediate variable v_j, and denote $v_j = s_j + n_j$. The goal is to have v_j statistically as close to l_j as possible. Thus, we design s_j and n_j, so that v_j satisfies $E[v_j] = \mu_l$ and $var[v_j] = \sigma_l^2$. In addition, v_j retains the same second order statistics of l_j. The Hurst parameter H and the autocorrelation of l_j at lag one, ρ_1, are obtained from the data.

We calculate the autocorrelation function $a_s(k)$ for s_i according to [69], and obtain:

$$a_s(k) = a_x(k) \cdot \frac{2k^2(1-d) - (1-d)^2}{k^2 - (1-d)^2}$$

where

$$a_x(k) = \frac{d(1+d)\cdots(k-1+d)}{(1-d)(2-d)\cdots(k-d)}$$

Table 5.1
List of Notations

Symbol	Explaination
t_j	HTTP connection inter-arrival times
α, λ	Weibull scale and shape parameters
$r = 1/E[t_j]$	new TCP connection rate
l_j	log scale of $t_j : l_j = \log_2(t_j)$
ρ_1	autocorrelation of l_j at lag one
H	Hurst Parameter of l_j
d	degree of difference in FARIMA model $d = H - 0.5$
s_j	FARIMA series: $(I - B)^d s_j = \epsilon_j + \epsilon_{j-1}$ ϵ_j are *i.i.d.* Gaussian$(0, \sigma_\epsilon^2)$
n_j	*i.i.d.* Log-Weibull (μ_l, σ_n^2) random variables uncorrelated with s_j
v_j	intermediate random sequence to model l_j: $v_j = s_j + n_j$

In particular, the autocorrelation of s_j at lag one is

$$a_s(1) = \frac{1+d}{2-d}$$

Since $v_j = s_j + n_j$ and $var(v_j) = \sigma_l^2$, we have

$$\sigma_l^2 = \sigma_s^2 + \sigma_n^2$$

Define

$$\theta = \frac{\rho_1}{a_s(1)} = \rho_1 \frac{2-d}{1+d},$$

where ρ_1 is the autocorrelation of l_j at lag one, an input parameter to our new model. Then, $\theta = \sigma_s^2/\sigma_l^2$, so that $\sigma_s^2 = \theta\sigma_l^2$.

Also we obtain the value of the variance of s_j using results in [69],

$$\sigma_s^2 = \frac{2}{1-d} \cdot \frac{\Gamma(1-2d)}{\Gamma^2(1-d)} \cdot \sigma_\epsilon^2$$

so that,

$$\sigma_\epsilon^2 = \frac{\theta(1-d)}{2} \cdot \frac{\Gamma^2(1-d)}{\Gamma(1-2d)} \cdot \sigma_l^2$$

The value of σ_ϵ^2 is used for generating the FARIMA sequence s_j defined by equation (1), which is used in step 11 in our model.

The parameters μ_l and σ_n^2 are used for generating *i.i.d.* Log-Weibull random sequence n_j in step 12, where

$$\sigma_n^2 = (1-\theta) \cdot \sigma_l^2$$

Even though v_j has the desired second order statistics, and satisfies $E(v_j) = \mu_l$ and $var(v_j) = \sigma_l^2$, its marginal distribution is not the desired Log-Weibull distribution. Therefore, to obtain l_j from v_j with the desired Log-Weibull distribution, we apply the following transformation:

$$l_j = b_0(r) + b_1(r)v_j + b_2(r)v_j^2$$

is used to obtain l_j in step 14, so that l_j has the desired Log-Weibull distribution. The values of b_0, b_1, and b_2 are computed in steps 8 to 10, according to equations (17), (18), and (19) on page 168 of [67], We made some adjustment to b_0 for a better fit. Note that b_0, b_1, and b_2 incoporate the effect of load on the inter-arrival times.

We generated synthetic traces according to our new model, and compare them with the real data and the existing model in [67]. The input parameters for our new model are estimated directly from the real data set. The value of the Hurst parameter H is estimated using R/S method [4]. Other methods and tools for Hurst parameter estimation can be found in [78]. The value of ρ_1 is estimated using a *Matlab* function *autocorr*. The Weibull parameters (α, λ) are estimated using the *Matlab* function *wblfit*. The CAIDA data set we used to generate figures 5.1 to 5.4 has the following parameter values: $\alpha = 1.36, \lambda = 0.76, H = 0.65, \rho_1 = 0.18$. The load r is 0.64 c/s.

Figure 5.1 compares the empirical cumulative distribution functions from our model, the model in [67] and the CAIDA trace. In this figure, the empirical *cdf* of data from our model almost follow that of the CAIDA trace exactly. The Weibull plots in Figure 5.2 are also very close between our model and the data. There two figures demonstrate that the marginal distribution from our model is a much closer match to the real data than the existing model.

The second order statistics also match well between our model and the data. Autocorrelation plots are in Figure 5.3, and the power spectrum density (PSD) estimates are in Figure 5.4. Figure 5.4 shows the PSD estimates of the CAIDA data is much closer to that of our model than that of the existing model. These two figures show that the second order statistics from our model also matches the real data better than the existing model.

The fundamental reasons behind the better match is that we use H and ρ_1 in our model, in addition to the load(r) and that we measured these values directly from recent real data.

[4]R/S method is also known as the rescaled adjusted range statistics method.

5.4 Design of HTTP Timing Channel

As we have seen, our model can be used to generate synthetic data that is statistically indistinguishable from the real trace. If we can embed covert information in our model while maintaining the statistical properties, a detection-resistant covert timing channel can be created. In what follows, we will explain how we incorporate this model in our HTTP timing channel design.

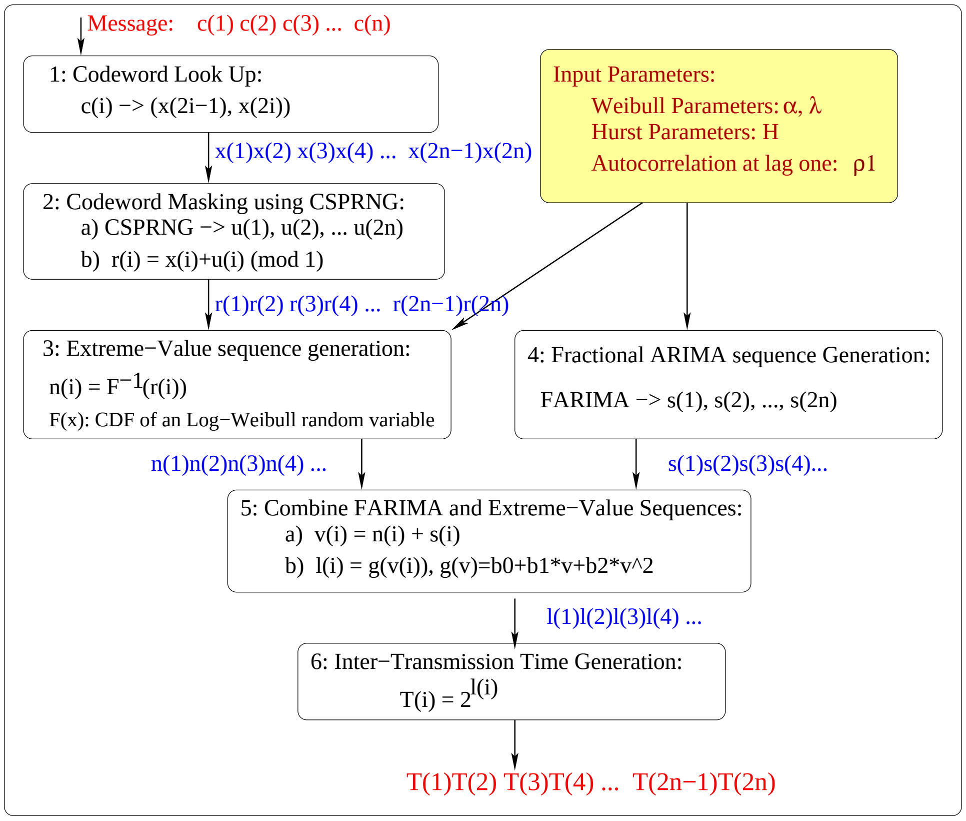
Message: c(1) c(2) c(3) ... c(n)
1: Codeword Look Up:
c(i) −> (x(2i−1), x(2i))
x(1)x(2) x(3)x(4) ... x(2n−1)x(2n)
Input Parameters:
Weibull Parameters: α, λ
Hurst Parameters: H
Autocorrelation at lag one: ρ1
2: Codeword Masking using CSPRNG:
a) CSPRNG −> u(1), u(2), ... u(2n)
b) r(i) = x(i)+u(i) (mod 1)
r(1)r(2) r(3)r(4) ... r(2n−1)r(2n)
3: Extreme−Value sequence generation:
n(i) = F−1(r(i))
F(x): CDF of an Log−Weibull random variable
4: Fractional ARIMA sequence Generation:
FARIMA −> s(1), s(2), ..., s(2n)
n(1)n(2)n(3)n(4) ...
s(1)s(2)s(3)s(4)...
5: Combine FARIMA and Extreme−Value Sequences:
a) v(i) = n(i) + s(i)
b) l(i) = g(v(i)), g(v)=b0+b1*v+b2*v^2
l(1)l(2)l(3)l(4) ...
6: Inter−Transmission Time Generation:
T(i) = 2l(i)
T(1)T(2) T(3)T(4) ... T(2n−1)T(2n)

Fig. 5.5. Encoder

The encoder of our HTTP timing channel is detailed in Figure 5.5. A single 8-bit ASCII character c_i will be mapped to two inter-arrival times T_{2i-1}, T_{2i} by this encoder. A message, consisting of a sequence of 8-bit ASCII characters $c_1, c_2, \cdots, c_n$, is encoded in a sequence of TCP new connection inter-arrival times $T_1, T_2, \cdots, T_{2n}$

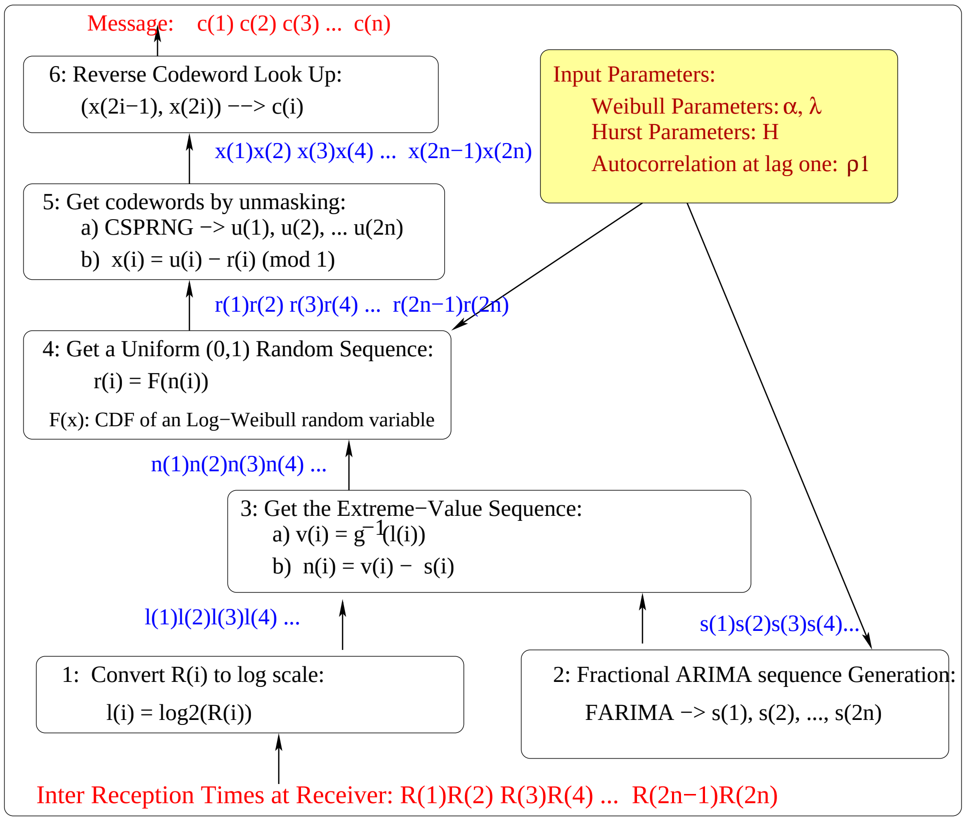
Message: c(1) c(2) c(3) ... c(n)
6: Reverse Codeword Look Up: (x(2i−1), x(2i)) −−> c(i)
x(1)x(2) x(3)x(4) ... x(2n−1)x(2n)
Input Parameters: Weibull Parameters: α, λ Hurst Parameters: H Autocorrelation at lag one: ρ1
5: Get codewords by unmasking: a) CSPRNG −> u(1), u(2), ... u(2n) b) x(i) = u(i) − r(i) (mod 1)
r(1)r(2) r(3)r(4) ... r(2n−1)r(2n)
4: Get a Uniform (0,1) Random Sequence: r(i) = F(n(i)) F(x): CDF of an Log−Weibull random variable
n(1)n(2)n(3)n(4) ...
3: Get the Extreme−Value Sequence: a) v(i) = g−1(l(i)) b) n(i) = v(i) − s(i)
l(1)l(2)l(3)l(4) ...
s(1)s(2)s(3)s(4)...
1: Convert R(i) to log scale: l(i) = log2(R(i))
2: Fractional ARIMA sequence Generation: FARIMA −> s(1), s(2), ..., s(2n)
Inter Reception Times at Receiver: R(1)R(2) R(3)R(4) ... R(2n−1)R(2n)

Fig. 5.6. Decoder

which have the same marginal distribution and autocorrelations as a legitimate HTTP traffic trace.

The covert message is implanted in the *i.i.d.* Log-Weibull random sequence n_i. The sender and receiver share a code book, a one-to-one mapping of 8-bit binary strings to two-dimensional vectors $(\frac{k_1}{16}, \frac{k_2}{16})$, where k_1 and k_2 are integers between 0 and 15.

The first step of our scheme is to look up the codeword for each character in the message. We use (x_{2k-1}, x_{2k}) to denote the codeword for character c_k. At the end of the first step, the message **msg** is transformed to a sequence of numbers $\mathbf{x} = \{x_1, x_2, \cdots, x_{2n-1}, x_{2n}\}$.

In the second step, we use a Cryptographic Secure Pseudo Random Number Generator (CSPRNG) to generate a sequence of pseudo uniform (0,1) random numbers

$\mathbf{u} = u_1, u_2, \cdots, u_{2n-1}, u_{2n}$. The seed used by CSPRNG is shared between the sender and receiver, but not with the detector of the covert timing traffic. We then *mask* the sequence $\mathbf{x}$ with $\mathbf{u}$ to obtain a new sequence of numbers $\mathbf{r} = r_1, r_2, \cdots, r_{2n-1}, r_{2n}$ by setting

$$r_k = x_k \oplus u_k \overset{\Delta}{=} (x_k + u_k) \bmod 1.$$

In the third step, we create an *i.i.d.* Log-Weibull random sequence $\{n_k, k = 1, 2 \cdots\}$ by setting $n_k = F^{-1}(r_k)$, where $F(x)$ is the *c.d.f.* of a Log-Weibull random variable with mean μ_l and variance σ_n^2. This step accomplishes the goal specified in step 12 of our Algorithm 3.1 (in Section 5.3), that is to generate an *i.i.d.* Log-Weibull random sequence $\{n_j\}$. Additionally, the third step in our design also embeds the covert information in $\{n_j\}$. This sequence $\{n_i\}$ will then be added to a fractional ARIMA sequence $\{s_1, s_2, \cdots\}$ generated in step 4. This fractional ARIMA sequence has the same Hurst parameter as the trace data.

In the last two steps, the fractional ARIMA and the *i.i.d.* Log-Weibull sequence are joined together, and transformed to the inter-arrival time $T_1, T_2, \cdots, T_{2n}$ according to our model introduced in Section 5.3. The sender then initiates new HTTP connection times according to the values of $T_1, T_2, \cdots, T_{2n}$.

The sender and the receiver share the following using an auxiliary channel prior to initiating the covert communication:

- Code Book: it contains the mapping from 8-bit characters to two-dimensional vectors $(x_1, x_2) = (k_1/16, k_2/16)$, where k_i are integers between 0 and 15, inclusive.

- Traffic Model Parameters α, λ, H, and ρ_1: The underlying assumption is that the sender does the determination of model parameters for legitimate traffic that is similar in characteristic to the legitimate traffic in which he will embed the covert information. For example, the daytime traffic over different weekdays may be statistically similar.

- Seed for CSPRNG: it is used to generate a common CSPRNG sequence.

- Seed for FARIMA series: it is used to generate a common FARIMA sequence.

The procedure for recovering the message at the receiver is simply the reverse of the sender scheme, as illustrated in Figure 5.4. After the receiver get inter-arrival times R_i, it will execute the following tasks:

- 1. Convert R_i to log scale: $l_i = \log_2(R_i)$
- 2. Generate s_i from the FARIMA model using the same seed as the sender, so that the resulting series is identical to that used by the sender.
- 3 a) Obtain the intermediate sequence v_i: $v_i = g^{-1}(l_i)$

  3 b) Obtain the Log-Weibull sequence n_i: $n_i = v_i - s_i$
- 4. Transform the Log-Weibull sequence n_i to a random sequence r_i: $r_i = F(n_i)$, where $F(\cdot)$ is the *cdf* of Log-Weibull distribution
- 5 a). Generate u_i from the CSPRNG using the same seed as the sender

  5 b). Obtain the codeword x_i: $x_i = (r_i - u_i) \bmod 1$
- 6. Get character c_k from x_i, x_{i+1} using codebook

At the receiver, the inter-arrival times observed are $R_1, R_2, \cdots, R_{2n}$. These are a distorted version of sender's inter-arrival times $T_1, T_2, \cdots, T_{2n}$. $R_i = T_i + \epsilon_i$, where ϵ_i's are network jitters.

Our initial experiments show most decoding errors occur when the inter-arrival times are small. We conducted more rigorous error analysis and proved that the smaller the inter-arrival time, the less jitter it can tolerate. The detailed error analysis can be found in section 5.4.1. Through our experiments and error analysis, we found when the inter arrival times is greater than 100 ms, jitters has much less impact on decoding errors.

Based on this observation, our method of reducing the decoding error is that if a codeword x^* is encoded with a small inter-arrival time, we will re-encode the same character x^* using the next values of CSPRNG and FARIMA sequence until the inter-arrival time T obtained is larger than 100 ms. Note that, using our encoder, the same codeword can be mapped to different inter-arrival times. We will transmit all the inter-arrival times obtained through the encoding process illustrated in Figure 5.5, including the small inter-arrival times that are less than 100 ms so that the desired statistical properties of the traffic is not disturbed. The receiver will record all the inter-reception times, but the decoder will discard the small inter-reception times when recovering the covert information.

In our encoding scheme, it is important that an error in one character does not ripple over to subsequent characters and is contained. In our basic timing channel design, each character is represented by two inter-arrival times. Due to "re-encoding" in our error correction, a character could be mapped to three or more inter-arrival times. In order to contain the character decoding error, we require even number of inter-arrival times (including small inter-arrival times) to represent one character. This design guarantees that even if the decoder mistakenly discards a "small" inter-reception time or accepted a "larger" inter-reception time, it can always start at the right positions to decode characters.

5.4.1 Analysis of Decoding Errors

In Section 5.4, we presented our design of a covert timing channel that can mimic LRD traffic, based on a model we proposed in Section 5.3. The design of our timing channel encoder that converts a character c_k to two inter-arrival times T_{2k-1}, T_{2k} (for $k = 1, 2, \cdots, n$), is illustrated in Figure 5.5. Due to network jitters, the inter-reception times R_{2k-1}, R_{2k} obtained by the receiver are slightly distorted from T_{2k-1}, T_{2k}, and $R_j = T_j + \epsilon_j$ (ϵ_j are network jitters).

The decoder in Figure 5.4 uses $R_1, R_2, \cdots, R_{2n}$ to recover the covert message. Our initial experiments show most decoding errors occur when the inter-arrival times T_j

are small. Here, we present a more rigorous error analysis and show that the smaller the inter-arrival time, the less jitter it can tolerate.

Let $l_j^D = \log_2(R_j) = log_2(T_j + \epsilon_j)$, and recall that $l_j = \log_2(T_j)$. We have

$$\Delta l_j = l_j^D - l_j = \log_2(T_j + \epsilon_j) - \log_2(T_j) \tag{5.5}$$

By the mean value theorem, there is a $T^* \in (T_j, T_j + \epsilon_j)$ such that

$$\log_2(T_j + \epsilon_j) - \log_2(T_j) = \frac{\epsilon_j}{\ln(2)T^*} \tag{5.6}$$

By Equations (5) and (6), we have

$$\Delta l_j = \frac{\epsilon_j}{\ln(2)T^*} \tag{5.7}$$

Recall that $\{n_j\}$ is an *i.i.d.* Log-Weibull random sequence, and $\{s_j\}$ is a a LRD FARIMA sequence. In our encoder, we used an intermediate random variable $v_j = n_j + s_j$ to obtain the Log-Weibull distributed l_j, using $l_j = b_0 + b_1 \cdot v_j + b_2 \cdot v_j^2$. Let v_j^D satisfies $l_j^D = b_0 + b_1 \cdot v_j^D + b_2 \cdot (v_j^D)^2$. and $\Delta v_j = v_j^D - v_j$. Then, we have $\Delta l_j = b_1 \Delta v_j + b_2 \cdot (v_j + v_j^D) \cdot \Delta v_j$

Based on the values of b_1 and b_2, calculated according to [67], $\Delta l_j \approx \Delta v_j$.

Dente $n_j^D = v_j^D - s_j$ in step 3b of the decoder in Figure 5.4. Recall that $\{s_j\}$ is shared between the encoder and decoder since they share the seed for generating the sequence, and $v_j = n_j + s_j$. We have,

$$\Delta l_j \approx \Delta n_j = n_j^D - n_j \tag{5.8}$$

In step 3 of Figure 5.5, $n_j = F^{-1}(r_j)$, where $F^{-1}(x)$ is the inverse function of *c.d.f.* of the Log-Weibull distribution. Therefore,

$$\Delta n_j = -b \ln(-\ln(r_j^D)) + b \ln(-\ln(r_j)), \tag{5.9}$$

where $b = -1/(\ln(2) \cdot \lambda)$, and λ is the shape parameter of the Weibull distribution.

Note $r_j = u_j + x_j \pmod 1 \; = u_j \oplus x_j$ from step 2b of Figure 5.5. By equations (7), (8), and (9), we have

$$\frac{-\epsilon}{ln(2) \cdot b \cdot T_j} = \ln(-\ln(u_j \oplus x_j^D)) - \ln(-\ln(u_j \oplus x_j)) \tag{5.10}$$

Thus,

$$\frac{-\epsilon}{ln(2) \cdot b \cdot T_j} = \ln \frac{\ln(u_j \oplus x_j^D)}{\ln(u_j \oplus x_j)} \tag{5.11}$$

So that,

$$\frac{\ln(u_j \oplus x_j^D)}{\ln(u_j \oplus x_j)} = \exp\{\frac{-\epsilon}{ln(2) \cdot b \cdot T_j}\} \tag{5.12}$$

Denote

$$\beta = \exp\{\frac{\epsilon_j}{\ln(2) \cdot b \cdot T_j}\} \tag{5.13}$$

We have

$$u_j \oplus x_j^D = ((u_j \oplus x_j)^\beta = r_j^\beta$$

$$\Delta x_j = x_j^D - x_j = (x_j^D \oplus u_j) - (x_j \oplus u_j) = r_j^\beta - r_j,$$

In order for our decode to decode correctly, we would like to have $|\Delta x_j| < 1/32$. When $0.92 < \beta < 1.08$, we have $|\Delta x_j| < 1/32$ regardless of the values of u_j.

By equation (13), we have

$$|\frac{\epsilon_j}{\ln(2) \cdot b \cdot T_j}| = |\ln(\beta)| \tag{5.14}$$

Thus,

$$|T_j| = |\frac{\epsilon_j}{\ln(2) \cdot b \cdot \ln(\beta)}| \tag{5.15}$$

Since $b = -1/\ln(2) \cdot \lambda$, we have

$$|T_j| = |\frac{\lambda \epsilon_j}{\ln(\beta)}| \tag{5.16}$$

We can see from Equation (16) that if T_j is too small, $\ln(\beta)$ will be too large, causing β to be outside the interval (0.92, 1.08) for correct decoding. Using the Weibull parameter $\lambda = 0.9$ in our data 1 from Subnet 2 (Table 5.2, Section 5), and choosing $\beta = 1.08$, or 0.92, the worst case scenario, we have $1/\ln(\beta) \approx 12$, and Equation(16) gives

$$|T_j| > (12\lambda)\epsilon_j > 10\epsilon_j \tag{5.17}$$

If the maximum jitter in the network is 10 ms, We will not have decoding error when $T_j > 100$ ms. Our analysis show that small T_j has less tolerance for network jitters, and can cause decoding errors. Our proposed solution is to add redundancy and if a character is encoded with a small T value, we will re-encode this character until it is encoded with a larger T value.

5.5 Experiments

We implemented this covert timing channel design in Java, using a client/server architecture. The sender injects the covert information into the *i.i.d.* Log-Weibull series, and obtains the desired inter-arrival times T_i according to our design in Figure 5.5. It controls the inter-transmission time by using $Thread.sleep(T_i)$ (T_i is in milliseconds). The accuracy of the $Thread.sleep(T)$ method is 1 ms. The receiver passively collects the TCP packet reception times and decodes the message by extracting the covert information from the inter-reception times R_i, according to the design in Figure 5.4. There is no feedback from receiver to sender regarding when the packet is received or whether it is decoded correctly.

We conducted our experiments on two pairs of computers using the PlanetLab environment. The receivers are hosts at Purdue University, and the senders are PlanetLab nodes located at Princeton University and Stanford University. The average RTT between Purdue and Princeton is approximately 39.5 ms, and the average RTT between Purdue and Stanford is approximately 73.5 ms. The average RTT times for both pairs remained stable during the course of ten hour experiments through the day.

In our experiments, we used parameters estimated from two subnets. Each subnet has four data sets, each containing about 15-minutes traffic trace. The data was collected by CAIDA in March 2009. Further details of the data set have been presented in Section 5.3. We will verify through our experiments if our covert timing channel traffic is statistically indistinguishable from these real data, and if it can avoid detection.

The values of input parameters for the eight data sets, α, λ, H and ρ_1, all estimated directly from these trace data, are listed in Table 5.2. We create 8 timing channels corresponding to the 8 traces. The sender sends a text file of 410 characters over the covert channels mimicking subnet 1, and a text file of 1100 characters over the covert channel mimicking subnet 2. We will see shortly that the timing channels mimicking subnet 2 has a higher data rate than subnet 1. We ran a set of the eight timing channels from Princeton to Purdue. The inter-arrival times from each of these eight timing channels are later used to test if these timing channels can avoid the best available detection schemes, such as the entropy based detection and the Kolmogorov-Smirnov Test.

Additionally, we ran two timing channels hourly over the course of a day on two pairs of hosts to see how network conditions impact the decoding error. The first timing channel runs from Stanford University to Purdue University, mimicking the first subnet. The second timing channel runs from Princeton University to Purdue University, mimicking the second subnet. We found the decoding error ranges from 2.9% to 6%. The data rate for the covert channels mimicking the first subnet's traffic is approximately 2 bits/sec; the data rate for the second covert channel is approximately 6 bits/sec.

The data rate for our covert timing channel is largely determined by $E[t_j]$, the mean value of the consecutive new HTTP connection request times. If one character (8-bits) is encoded in two inter-arrival times, the data rate of our timing channel is approximately $8/(2E[t_j]) = 4r$ b/s. Because of the use of error-correction in our timing channel, small inter-arrival times do not carry information, and one character could be mapped to $2N$ inter-arrival times, where N is an integer and $N \geq 1$. This reduces the actual data rate to be less than the maximum achievable rate of $4r$ b/s. As shown in Table 5.2, $r = 2.12c/s$ for data 3 in subnet 2. The data rate of this timing

Table 5.2
Parameter Values for Two SubNets

SubNet 1 Parameters	data 1	data 2	data 3	data 4
α	1.50	1.34	1.20	1.36
λ	0.77	0.74	0.73	0.76
H	0.61	0.52	0.81	0.65
ρ_1	0.12	0.14	0.23	0.18
r (c/s)	0.57	0.62	0.70	0.64
SubNet 2 Parameters	data 1	data 2	data 3	data 4
α	0.51	0.43	0.44	0.45
λ	0.90	0.85	0.87	0.91
H	0.57	0.56	0.59	0.70
ρ_1	0.08	0.10	0.12	0.09
r (c/s)	1.87	2.15	2.12	2.11

channel is 6 bits/sec, less than $4r$ b/s. Although the data rates are not very high, we would caution the reader not to underestimate the potential security risks. Since these timing channels mimic non-stationary LRD HTTP traffic, it can potentially be used long-term without detection. Further, often systems have small-sized private data items.

We compare the traffic trace from our timing channels and the real data. Figures 5.7 – 5.10 compare the first order and the second order statistics of the data trace from our timing channel with its corresponding real data trace (data 1 of subnet 1 is used[5]) . Figure 5.7 shows that the empirical distributions of the two traces are very close to each other. In fact, the maximum distance between these two empirical distributions using the Kolmogrov-Smirnov test is only 0.066. Figure 5.9 shows the PSD estimates of the covert traffic and the legitimate traffic, and they are very close to each other. The closeness of the PSD indicates the closeness of their autocorrelations functions.

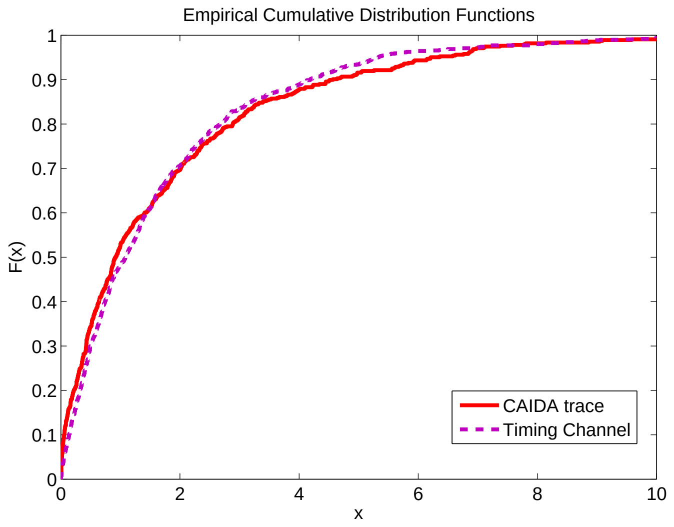

Fig. 5.7. Empirical *cdf* of the inter-arrival times from our covert channel and CAIDA data (Subnet 1, data 1)

[5]Data 4 of subnet 1 was used in Section 3.

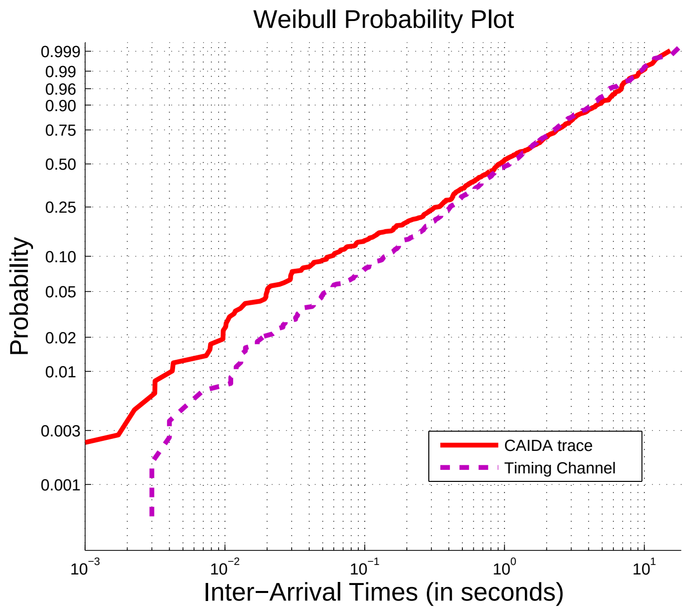

Fig. 5.8. Weibull probability plots

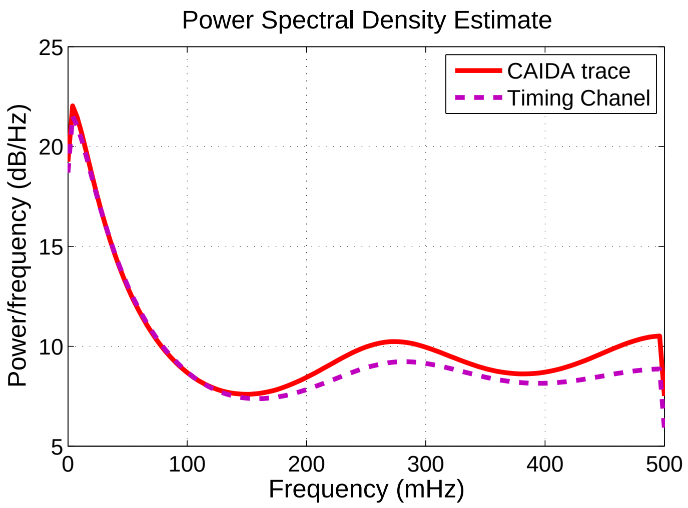

Fig. 5.9. PSD Estimates of the log inter-arrival times from our timing channel and CAIDA data (Subnet 1, data 1)

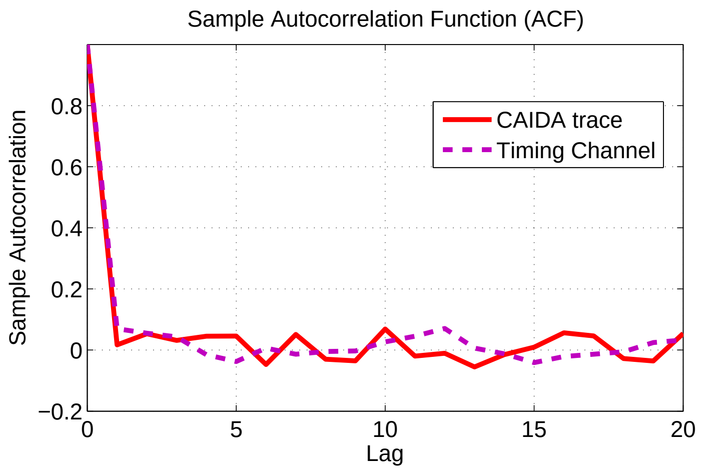

Fig. 5.10. Sample Autocorrelation Functions

Next, we evaluate how well our timing channel can evade current detection methods. First, we conduct two sample Kolmogorov-Smirnov tests (KS test) on each covert traffic and real data pair. The two sample KS test uses the maximum distance between two empirical distributions, $KS\_STAT = \max(|F1(x) - F2(x)|)$, where $F1(x)$ and $F2(x)$ are empirical distributions of the real data and the covert channel data. Table 5.3 shows the values of KS_STAT and corresponding p-values for each pair.

Table 5.3
Kolmogorov-Smirnov Test

SubNet 1	data 1	data 2	data 3	data 4
KS-STAT	0.0657	0.0845	0.0637	0.0516
p-value	0.096	0.012	0.0863	0.2678
detect	no	no	no	no
SubNet 2	data 1	data 2	data 3	data 4
KS-STAT	0.0402	0.0442	0.0466	0.0513
p-value	0.2088	0.1088	0.08	0.043
detect	no	no	no	no

When the allowable false alarm level is set to 1% as in [81], none of our timing channel traffic can be detected since all the p-values are greater than 1%.

The regularity test proposed in [58] checks if the variance of the inter-arrival times is relatively constant, and the traffic is flagged as a covert timing channel if the variance remains constant. Our timing channel is designed to mimic the non-stationary LRD HTTP traffic. More specifically, our timing channel traffic patterns match the legitimate HTTP traffic patterns in each small time interval (say 15 minutes). The marginal distributions of our timing channels match that of the changing legitimate HTTP traffic patterns. Therefore, the variance of the inter-arrival times of our timing channels change the same way as the legitimate HTTP traffic patterns change. Thus, by design, our timing channel can evade the regularity test independent of the threshold used in the test.

A recent detection scheme based on entropy is proposed in [81], where data is first partitioned into Q equiprobable bins according to a desired cumulative distribution function $F(x)$. The bin numbers y for the data value x can be calculated using $y = \lfloor F(x) * Q \rfloor$, and then used for the entropy and the corrected conditional entropy computation. By using a single *cdf* for legitimate traffic, they implicitly assume that the legitimate traffic is stationary. The *c.d.f.* of a non-stationary traffic can change over time, so their detection method does not apply directly to timing channels mimicking non-stationary traffic.

Their recommended value for fine binning $Q = 65536$ also fails to work for small data sets with less than 65536 data points. The reason is that legitimate traffic is expected to be uniformly distributed among the Q bins, thus having entropy value near 16 when $Q = 65536$. Traffic with entropy values less than 15.12 in [81] is flagged as covert traffic. Consider legitimate traffic that follows exactly the desired *c.d.f*, but only has 2000 inter-arrival times. Since the maximum entropy for any data sets with 2000 data points is $\log_2(2000) = 10.97$, the entropy for this legitimate traffic is far less than the desired entropy 16, and will be falsely flagged as covert traffic.

In each of our 15-minute data sets from subnet 1, there are approximately 600 HTTP connections. It is appropriate to use $Q = 128$ for fine binning. We use $Q = 5$ to calculate CCE values as in [81]. Table 5.4 and 5.5 show the entropy values of covert traffic mimicking subnet 1 and subnet 2 respectively, compared with the legitimate traffic and the training data. Table 5.6 and 5.7 show the CCE values of covert traffic mimicking subnet 1 and subnet 2 respectively, compared with legitimate traffic and training data. The training data are the CAIDA data set that was used to obtain the model parameters for the covert timing channels. The p-values are calculated for each entropy or CCE value using T-test. The T-test was applied to determine if the traffic coming from the covert channel differs significantly from the "normal" traffic, where normal traffic included the training and the legitimate traffic traces. As shown in these tables, all the 8 covert timing channels evade entropy and CCE tests, even if the allowable false alarm level is 5%.

5.6 Conclusion

Internet traffic has often been show to display LRD characteristics. Thus, traditional covert channel schemes can easily be detected by comparing their traffic

Table 5.4
Entropy Values for Covert Traffic 1

traffic type	data 1	data 2	data 3	data 4
training data	6.81	6.80	6.85	6.82
legit	6.56	6.50	6.50	6.51
covert 1	6.85	6.84	6.87	6.89
p-value	0.33	0.36	0.28	0.24
detect?	no	no	no	no

Table 5.5
Entropy Values for Covert Traffic 2

traffic type	data 1	data 2	data 3	data 4
training data	6.93	6.94	6.95	6.95
legit	6.29	6.38	6.44	6.34
covert 2	6.86	6.88	6.90	6.87
p-value	0.55	0.52	0.48	0.53
detect?	no	no	no	no

Table 5.6
CCE Values for Covert Traffic 1

traffic type	data 1	data 2	data 3	data 4
training data	2.21	2.19	2.17	2.21
legit	1.84	1.81	1.82	1.84
covert 1	2.22	2.23	2.18	2.21
p-value	0.35	0.33	0.44	0.37
detect?	no	no	no	no

Table 5.7
CCE Values for Covert Traffic 2

traffic type	data 1	data 2	data 3	data 4
training data	2.26	2.21	2.22	2.22
legit	1.75	1.85	1.88	1.80
covert 2	2.25	2.25	2.24	2.22
p-value	0.37	0.37	0.39	0.43
detect?	no	no	no	no

characteristics with Internet traffics. T o overcome this problem, we have designed a covert timing channel scheme that can mimic legitimate traffics displaying LRD property. We show that our covert timing channel can be hidden in the Web traffic, the most observed traffic on Internet today. We used the HTTP new connection inter-arrival times to carry the covert information. We found that the marginal distribution and autocorrelation functions of the inter-arrival times from our covert timing channel matched closely with that from recent traces of real traffic.

We implemented our design and have conducted extensive experiments on the PlanetLab nodes and verified the close match of our covert traffic with the real the data. Further, our experiments show that our our covert timing channels evade the current best available detection methods. The data rates of our covert channels range from 2 to 6 bits/sec, and decoding errors range from 3% to 6%.

There are several interesting future directions for this work. One is to develop timing channels for short range dependent (SRD) traffic; and the other is to design a covert timing channel to mimic other commonly used traffics, such as peer-to-peer traffic.

LIST OF REFERENCES

LIST OF REFERENCES

[1] CAIDA, “CAIDA Analysis of Code-Red,,” *http://www.caida.org/analysis/security/code-red/*.

[2] NLANR “National Laboratory for Applied Network Research Project,,” *http://www.nlanr.net/*.

[3] USA Today News, “The Cost of Code Red: $1.2 billion, ” *http://www.usatoday.com/tech/news/2001-08-01-code-red-costs.htm*.

[4] H. Andersson and T. Britton, “Stochastic Epidemic Models and Their Statistical Analysis,” *Lecture Notes in Statistics*, vol 151, *Springer*.

[5] V. H. Berk, R. S. Gray, and G. Bakos, “Using Sensor Networks and Data Fusion for Early Detection of Active Worms,” *In Proceedings of AeroSense 2003: SPIE’s 17th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls,* Orlando, Florida, April 2003.

[6] Z. Chen, L. Gao, and K. Kwiat, “Modeling the Spread of Active Worms,” *in Proc. of IEEE INFOCOM 2003*, pp. 1890-1900.

[7] P. C. Consul, “Generalized Poisson Distributions, Properties and Applications,” *STATISTICS: textboods and monographs*, vol 99, *Marcel Dekker, Inc.*

[8] D. J. Daley and J. Gani, “Epidemic Modelling, An Introduction,” *Cambridge University Press*, 1999.

[9] A. Ganesh, L. Massoulie, and D. Towsley “The Effect of Network Topology on the Spread of Epidemics,” *in Proc. of IEEE INFOCOM 2005*.

[10] A. Ganesh, D. Gunawardena, P. Key, L. Massoulie, and J. Scott “Efficient Quarantining of Scanning Worms: Optimal Detection and Coordination,” *in Proc. of IEEE INFOCOM 2006*.

[11] D. Dagon, X. Qin, G. Gu, W. Lee, J. B. Grizzard, J. G. Levine, and H. L. Owen “HoneyStat: Local Worm Detectiong Using Honeypots,” *in RAID* pp 39 - 58, 2004.

[12] Foresount, “WormScout,” *http://www.foresout.com/*.

[13] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan “Fast Portscan Detection Using Sequential Hypothesis Testing,” *in IEEE Symposium on Security and Privacy,* pp 211 - 225, 2004.

[14] S. Karlin and H. M. Taylor, “A First Course in Stochastic Processes, Second Edition,” *Academic Press*, 1975.

[15] J.O. Kephart and S. R. White, "Directed-graph Epidemiological Models of Computer Viruses," *in Proc. of the IEEE Symposium on Security and Privacy*, 1991.

[16] J.O. Kephart D. M. Chess and S. R. White, "Computers and Epidemiology," *in IEEE Spectrum*, 1993.

[17] J.O. Kephart and S. R. White, "Measuring and Modeling Computer Virus Prevalence," *in Proc. of the IEEE Symposium on Security and Privacy*, 1993.

[18] LaBrea, "LaBrea Technologies," *http://www.labreatechnologies.com/.*

[19] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray, "Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing," *in ACM WORM'03*, pp.24–33.

[20] Mirage Networks, *http://www.miragenetworks.com/.*

[21] D. Moore, C. Shannon, and J. Brown, "Code-Red: A Case Study on The Spread and Victims of an Internet Worm," *in Proc. 2nd ACM Internet Measurement Workshop*, 2002.

[22] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *in IEEE Security and Privacy*, 1(4):33-39, July 2003.

[23] D. Moore, C. Shannon, G. M. Voelker and S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," *in Proc. IEEE INFOCOM 2003.*

[24] V. Paxon "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks, (Amsterdam, Netherlands:1999), 31(23-24):2435-2463, 1999.*

[25] S. Ross "Stochastic Processes, 2nd edition" *John Wiley & Sons, Inc*, 1996.

[26] S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time," *in Proc. 11th USENIX Security Symposium* (SEC 02), Usenix Assoc., 2002, pp. 149–167.

[27] M. M. Williamson "Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code," *in Proc. ACSAC Security Conference*, 2002.

[28] C. Wong, C. Wang, D. Song, S. Bielski, G. R.Ganger "Dynamic Quarantine of Internet Worms," *The IEEE International Conference on Dependable Systems and Networks*, 2004.

[29] C. C. Zou, W. Gong and D. Towsley, "Code Red Worm Propagation Modeling and Analysis," *in 9th ACM Symposium on Computer and Communication Security*, pp. 138–147, 2002.

[30] C. C. Zou, L. Gao, W. Gong and D. Towsley, "Monitoring and Early Warning for Internet Worms," *in 10th ACM Symposium on Computer and Communication Security*, pp.190–199, 2003.

[31] C. C. Zou, W. Gong, and D. Towsley, "Worm Propagation Modeling and Analysis under Dynamic Quarantine Defense," *ACM WORM'03*, pp. 51–60, 2003.

[32] Computer Economics "2001 Economic Impact of Malicious Code Attacks," *http://www.computereconomics.com/cei/press/pr92101.html.*

[33] LBL-CONN-7 "Thirty Days' Wide-Area TCP Connections," *http://ita.ee.lbl.gov/html/contrib/LBL-CONN-7.html.*

[34] H. W. Hethcote, "The Mathematics of Infectious Diseases," *In SIAM Review, vol. 42, no. 4, pp. 599-653,* 2000.

[35] K. Rohloff and T. Basar "Stochastic Behavior of Random Constant Scanning Worms," *In Proc. ICCCN 2005* San Diego, OCT 2005

[36] K. Rohloff and T. Basar "The Detection of RCS Worm Epidemics," *In Proc. WORM 2005* Fairfax, VA, NOV 2005

[37] S. E. Schechter, J. Jung, and A. W. Berger "Fast Detection of Scanning Worm Infection," *7th International Symposium on Recent Advances in Intrusion Detection,* September 2004.

[38] S. Sellke, N. Shroff, and S. Bagchi "Modeling and Automated Containment of Worms," *The IEEE International Conference on Dependable Systems and Networks* pp. 528 - 537, 2005

[39] E. Skoudis "Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses," *Prentice Hall*, 2002.

[40] N. Weaver, S. Staniford, and R. Cunningham "A Taxonomy of Computer Worms," *ACM WORM'03* 2003.

[41] N. Weaver, S. Staniford, and V. Paxson "Very Fast Containment of Scanning Worms," *13th USENIX Security Symposium,* pp. 29 – 44, August 2004.

[42] N. Anantharam and S. Verdu, "Bits through queues," *IEEE Trans. Inform. Theory,* vol. 42, pp. 4 – 18, Jan. 1996

[43] R. Sundaresan and S. Verdu "Robust decoding for timing channels," *IEEE Trans. Inform. Theory,* vol. 46, pp. 405 – 419, Mar. 2000

[44] R. Sundaresan and S. Verdu "Sequential decoding for the exponential server timing channel," *IEEE Trans. Inform. Theory,* vol. 46, pp. 405 – 419, Mar. 2000

[45] R. Sundaresan and S. Verdu "Capacity of Queues Via Point-Process Channels," *IEEE Trans. Inform. Theory,* vol. 52, pp. 2697 – 2709, June 2006

[46] J. Giles and B. Hajek "An Information-theoretic and game-theoretic study of timing channels," *IEEE Trans. on Inform. Theory,* VOL. 48, pp. 2455 – 2477, Mar. 2002

[47] X. Liu and R. Srikant "The timing capacity of single-server queues with multiple flows," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* 2004

[48] A. B. Wagner and V. Anantharam "Zero-reate reliability of the exponetial-server timing channel," *IEEE Trans. on Inform. Theory,* VOL 52, No 2, pp. 447 – 465, Feb. 2005

[49] P. Mimčilović "Mismatch Decoding of a Compound Timing Channel," *Forty-Fourth Annual Allerton Conference on Communication, Control, and Computing*, Sep. 2006

[50] H. Kopetz and G. Bauer "The time-triggered architecture," *Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software*, Jan 2003

[51] A. Bedekar and M. Azizoglu "The information-theoretic capacity of discrete-time queues," *IEEE Trans. Inform. Theory,* vol. 44, pp. 446 – 461, Mar. 1998

[52] J. A. Thomas "On the Shannon capacity of discrete time queues," *IEEE Int. Symp. Inform. Theory*, pp. 333, July 1997

[53] S. H. Sellke, N. B. Shroff, S. Bagchi, and C. C. Wang "Timing Channel Capacity for Uniform and Gaussian Servers," *Forty-Fourth Annual Allerton Conference on Communication, Control, and Computing*, 2006

[54] S. H. Sellke, C. C. Wang, N. B. Shroff, and S. Bagchi "Capacity Bounds on Timing Channels with Bounded Service Times," http://www.stat.purdue.edu/∼ssellke/TimingChannels.pdf

[55] T. Cover and J. Thomas *Elements of Information Theory*, *New York: Wiley,* 1991

[56] V. Berk, A. Giani, and G. Cybenko "Detection of covert channel encoding in network packet delays," *Technical Report, TR2005-536, Dartmouth College, 2005*

[57] U.S. Department of Defense. "Trusted computer system evaluation. The Orange Book," *DoD 5200.28-STD Washington: GPO:1985*, 1985

[58] S. Cabuk, C. E. Brodley, and C. Shields "IP covert timing channels: design and detection," *Proceedings of 11th ACM conf. Computer and communication security*, pp. 178 – 187, New York, 2004

[59] W. M. Hu "Reducing Timing Channels with Fuzzy Time," *in Proceedings of the IEEE Symposium in Security and Privacy*, May. 1991

[60] M. H. Kang, I. S. Moskowitz, and D. C. Lee "A network version of the pump," *in Proceedings of the IEEE Symposium in Security and Privacy*, May. 1995

[61] I. S. Moskowitz and A. R. Miller "Simple timing channels," *Proceedings of IEEE Computer Society Symposim on Research in Security and Privacy*, pp. 56 – 64, Oakland, CA, 1994

[62] V. Paxson and S. Floyd "Wide-area traffic: the failure of Poisson modeling," *IEEE tran. Networking*, May 1991

[63] S. Goldwasser and M. Bellare "Lecture Notes on Cryptograpy," Available: *http://www-cse.ucsd.edu/ mihir/papers/gb.html*

[64] O. Goldreich *The Foundations of Cryptography*, *Cambridge University Press*, 2001

[65] G. Shah, A. Molina and M. Blaze "Keyboard and covert channels," *USENIX*, pp. 56 – 64, Oakland, CA, 2006

[66] J. C. Wray "An analysis of covert timing channels," *Proceedings of IEEE Computer Society Symposim on Research in Security and Privacy*, May 1991

[67] W. S. Cleveland, D. Lin, and D. X. Sun, "IP Packet Generation: Statistical Models for TCP Start Times Based on Connection-Rate Superposition," *in Proceedings of ACM SIGMETRICS, 2000*

[68] J. Cao, W. S. Clevland, D. Lin, and D .X .Sun "On the Nonstationarity of Internet Traffic," *in Proceedings of ACM SIGMETRICS, 2001*

[69] J. R. M. Hosking, *Fractional Differencing*, in Biometrika, 1981

[70] J. R. M. Hosking, *Modeling Persistence in Hydrological Time Series Using Fractional Differencing*, Water Resources Res., Vol. 20, No. 12, pp 1898-1980, 1984

[71] L. J. De La Cruz, *Self-Similar Traffic Generation using a fractional ARIMA model: Application to the VBR MPEG video traffic*

[72] M. W. Garrett and W. Willinger, *Analysis, Modeling and Generation of Self-Similar VBR Video Traffic*, in ACM SIGCOMM 1994

[73] K. Park and W. Willinger, *Self-Similar Network Traffic: An Overview*, Book Chapter in Self-Similar Network Traffic and Performance Evaluation, John Wiley & Sons, Inc, 2002

[74] V. Paxson, *Fast, Approximate Synthesis of Fractional Gaussian Noise for Generating Self-Similar Network Traffic*, Computer Communication Review 27(5), pp. 5-8, Oct. 1997.

[75] J. Lee and M. Gupta, *A New Traffic Model for current User web browsing behavior*

[76] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, *On the Self-Similar Nature of Ethernet Traffic*, in IEEE Transactions on Networking, Feb 1994

[77] T. Karagiannis, M. Molle, and M. Faloutsos, *Long-Range Dependence: Ten Years of Internet Traffic Modeling*, in IEEE Internet Computing, 2004

[78] T. Karagiannis and M. Faloutsos, *SELFIS: A Tool for Self-Similarity and Long-Range Dependence Analysis*, in 1st workshop on Fractals and Self-Similarity in Data Mining: Issues and Approaches (in KDD) Edmonton, Canada, July 23, 2002

[79] T. Karagiannis, M. Faloutsos, and M. Molle, *A User-Friendly Self-Similar Analysis Tool*, in Special Section on Tools and Technologies for Networking Research and Education, ACM SIGCOMM Computer Communication Review, 2003

[80] T. Karagiannis, *SELFIS: A Short Tutorial*, available at http:http://www.cs.ucr.edu/ tkarag/papers/SELFIS-Tutorial.pdf

[81] S. Gianvecchio and H. Wang "Detecting Covert Timing Channels: An Entropy-Based Approach," *Proceedings of 14th ACM conf. Computer and communication security*, 2007

[82] S. H. Sellke, C. C. Wang, S. Bagchi, and N. B. Shroff, "Covert TCP/IP Timing Channels: Theory to Implementation," *IEEE INFOCOM,* 2009

[83] S. H. Sellke, C. C. Wang, S. Bagchi, and N. B. Shroff, "Camouflage Timing Channels in Web Traffic," Purdue University ECE Technical Report, 2009, available at http://docs.lib.purdue.edu/ecetr/

[84] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson "Self-Similarity Through High-Variability: Statistical Analysis of Ethernet LAN Traffic at the Source Level," *in IEEE/ACM Transactions on Networking*, 1997

[85] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Petterson, M. Wawrzoniak, and M. Bowman "Planetlab: an overlay testbed for broad-coverage services," *in SIGCOMM Comput. Commun. Rev. 33,3*, (2003), 3-12

[86] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, and M. Karir "ATLAS Internet Observatory 2009 Annual Report," *available at http://www.nanog.org/meetings/nanog47/presentations/Monday/ Labovitz_ObserveReport_N47_Mon.pdf*

[87] Colby Walsworth, Emile Aben, kc claffy, Dan Andersen, "The CAIDA Anonymized 2009 Internet Traces DataSet,"
available at http://www.caida.org/data/passive/passive_2009_dataset.xml

VITA

VITA

Sarah Sellke was born in Tianjin, China. She studied Applied Mathematics at Tsinghua University in Beijing, China. She received her master's degrees in Mathematics and Electrical Engineering, both from Purdue University in West Lafayette. Sarah started to pursue a Ph.D. in the School of Electrical and Computer Engineering at Purdue University in fall 2003. She has been conducting research on network security under the guidance of Professors Saurabh Bagchi and Ness B. Shroff. Her research stressed both the theoretical and analytical aspect of the network security challengs, as well as practical solutions and implementations. Sarah was a recepient of a Purdue Ross fellowship from 2003 to 2004, and an NSF graduate research fellowship from 2004 to 2007.

Aside from her dissertation research, Sarah has been actively involved in the community. She served as a co-coordinator for the Stanford Educational Program for the Gifted Youth in West Lafayette, a volunteer for MATHCOUNTS competitions, a Science Bowl Coach at Burnett Creek Elementary School, and a coordinator of classroom parent volunteers at Klondike Elementary School.