PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis Acceptance

This is to certify that the thesis prepared

By Serdar Cabuk

Entitled

Network Covert Channels: Design, Analysis, Detection, and Elimination

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Final examining committee members

E. H. Spafford (Co-Chair)

C. E. Brodley (CoChair)

C. W. Clifton

N. B. Shroff

Approved by Major Professor(s): E. H. Spafford (Co-Chair)

C. E. Brodley (CoChair)

Approved by Head of Graduate Program: V. Balakrishnan

Date of Graduate Program Head's Approval: 12/06/2006

NETWORK COVERT CHANNELS:

DESIGN, ANALYSIS, DETECTION, AND ELIMINATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Serdar Cabuk

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2006

Purdue University

West Lafayette, Indiana

Dedicated to Pinar, Sema, and Mustafa Cabuk, for their love and support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABBREVIATIONS

ACK	Acknowledgment
AODV	Ad-hoc on Demand Distance Vector
ASCII	American Standard Code for Information Exchange
BMC	Binary-matching Channel
CC	Common Criteria
CDM	Compression-based Distance Measure
CFT	Covert Flow Trees
CPU	Central Processing Unit
DDOS	Distributed Denial of Service
EAL	Evaluation Assurance Level
ECC	Error-correcting Code
EMC	Exact-matching Channel
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
HTTP	Hyper-text Transfer Protocol
IA	Inter-arrival
ICMP	Internet Control Message Protocol
ID	Identification
IDI	Index of Dispersion for Intervals
IP	Internet Protocol
IPSEC	Secure Internet Protocol
ISN	Initial Sequence Number
KVM	Kernel Virtual Machine

LAN	Local Area Network
MAC	Media Access Control
MLS	Multi-level Secure
MSC	Message Sequence Chart
NCSC	National Computer Security Center
NRL	Naval Research Laboratory
NZIX	New Zealand Internet Exchange
OSI	Open System Interconnection
PLL	Phased-locked Loop
RMC	Rule-matching Channel
RTT	Round-trip Time
SCC	Simple Covert Channel
SMC	Small Message Criterion
SOF	Start of Frame
SRM	Shared Resource Methodology
SSH	Secure Socket Shell
SVM	Support Vector Machines
TCP	Transmission Control Protocol
TCSEC	Trusted Computer System Evaluation Criteria
TRCC	Time-replay Covert Channels
UDP	User Datagram Protocol
WAND	Waikato Applied Network Dynamics Group
WWW	World Wide Web

ABSTRACT

Cabuk, Serdar Ph.D., Purdue University, December, 2006. Network Covert Channels: Design, Analysis, Detection, and Elimination. Major Professors: Carla E. Brodley and Eugene H. Spafford.

Indirect communication channels have been effectively employed in the communications world to bypass mechanisms that do not permit direct communication between unauthorized parties. Such covert channels emerge as a threat to information-sensitive systems in which leakage to unauthorized parties may be unacceptable (e.g., military systems). In this dissertation, we show that traffic analysis can counter traditional event-based covert channels, which do not employ any additional scheme to obfuscate the channel further. For these channels, we introduce effective noiseless and noisy covert channel detection mechanisms that capture the anomalous traffic patterns. However, because a motivated user can potentially hide the channel further, we introduce a new family of covert channels that do not produce such anomaly. These IP time-replay covert channels transmit covert messages by adjusting packet timings consistent with inter-arrival time sequences that are extracts from recently recorded normal sequences. Under certain assumptions and lowered data rates, these channels generate output sequences that are equal in distribution to normal sequences allowing them to by-pass traffic anomaly detection schemes that are based on distribution analysis. Additionally, we illustrate that these channels can potentially survive channel elimination schemes such as jammers and network data pumps with lowered data rates. Thus, we discuss two types of transformations on packet inter-arrival times to increase the efficacy of existing elimination schemes.

1. INTRODUCTION

Consider a Multi-level Secure (MLS) system in which a malicious user Alice wishes to leak sensitive information to Eve while sending a legitimate message to another user Bob. Additionally, suppose that Alice and Bob have HIGH security clearances, and Eve has a LOW security clearance. In systems where the security policy physically separates Alice's and Eve's resources, this leakage is not possible and the resulting system is called a *total isolation* system [1]. However, this alternative is impractical because of under-utilization of resources. A more common approach is to allow resource sharing between users with different security levels and to restrict access to shared resources using access control schemes. In our research, we assume that a Bell-LaPadula access control scheme is active in the system with HIGH and LOW security levels [2]. Even though this scheme controls direct access to shared resources, it enables Alice and Eve to establish indirect communication channels – also known as covert channels. A *covert channel* is a communication channel that violates a security policy by using shared resources in ways for which they were not initially designed. This is a different type of channel than the subliminal channel described in Simmons' *prisoner's problem* [3], which assumes a direct channel between Alice and Eve, and is inherently not allowed in an MLS system. Covert channels arise in systems for which direct access is disallowed by policy. Therefore, covert channels are a related but different information hiding construct as compared to other information hiding techniques such as cryptography and steganography.

Traditionally, covert channels are characterized into two types: storage and timing channels. A *storage covert channel* "involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process" [4]. A *timing covert channel* involves a sender process that "signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process" [4]. Covert channels can be further classified depending on the type of the system in which they are used. A *network covert channel* is a covert channel in which the shared medium is the network environment (e.g., transmission lines, firewalls, routers, etc.). Accordingly, a *single system covert channel* uses the shared resources within a single computer (e.g., files, hardware resources, operating system components, etc.).

A crucial step in covert channel design is to find and exploit a shared resource that is unlikely to be used as a communication medium by design. In principle, the less obvious the shared resource, the more stealthy the covert channel. For example:

1. The *file-lock covert channel* [5] exploits systems with shared file systems by using file description tables that indicate a file's locked/unlocked status as the shared resource.

2. The *disk-arm covert channel* [6, 7] exploits KVM/370 systems with a security kernel and a shared read-only disk drive by using the position of the disk arm as the shared resource.

3. The *bus-contention covert channel* [8] exploits multi-processor systems running on a single shared bus by using the busy/idle status of the bus as the shared resource.

Little emphasis has been given to methods for hiding the traffic generated by covert channels – the secrecy of a covert channel mostly relies on the secrecy of knowing the shared resource. Without additional techniques in place to hide the channel better, we argue that once this shared resource is identified, the traffic generated by an event-based covert channel can be distinguished from the traffic generated by legitimate channels in the system. To see this, suppose that Alice and Eve use a shared resource to communicate via a covert channel. To signal one bit of information to Eve, Alice generates an *event* that alters the state of this shared variable. As an example, two events that alter the file description table in the file-lock channel are `lock_file` and `unlock_file`. To transmit a series of signals to Eve, Alice subsequently generates the corresponding events each separated by a *timing interval* that is either constant (i.e., a storage channel) or picked from a small set that is used repetitively (i.e., a timing channel). The resulting sequence is an event sequence that is composed of 2-tuples (E, t), where E is an event and t is a point in time. In the first part of this dissertation, we argue that event sequences generated by an unhidden event-based covert channel can be distinguished from event sequences generated by legitimate channels because the former shows a more regular pattern caused by the encoding scheme that generates repeating inter-event times. Hence, if the shared resource that the covert channel uses is identified, the channel can be *detected* by examining the event traffic it generates. In the second part of the dissertation, we illustrate ways to hide the channel further by replaying event sequences generated by legitimate channels to create network covert channels that are virtually undetectable through traffic distribution analysis. To counter these channels for which our detection schemes fail, we present ways to eliminate and/or rate-limit them.

1.1 Research Objectives and Contributions

Our research addresses event-based network covert channels that arise in distributed TCP/IP MLS systems even when the transmission lines between network nodes are controlled. In this dissertation, we show that traffic analysis can counter traditional event-based covert channels, which do not employ any additional scheme to obfuscate the channel further, although rate-limited and more complex channels can be indistinguishable from normal channels through traffic distribution analysis. To prove this statement, we provide answers to two questions: How can we design network covert channels using the TCP/IP protocol that leak information from Alice to Eve? How can the system detect and eliminate such leakage? In particular,

1. We present and implement an IP covert channel prototype using a novel design in which we use IP packet timings to transmit covert information over the network effectively.

2. We introduce novel detection measures that effectively detect both noiseless and noisy IP covert channels (and all event-based covert channels in general) using traffic analysis and force malicious users to either design more complex channels and/or rate-limit the channel to avoid detection.

3. We introduce time-replay covert channels (TRCC) that are a covert channel family that hides event-based channels, and argue that TRCCs are virtually undetectable through distribution analysis under certain assumptions.

4. We discuss prevention and elimination techniques for time-replay covert channels that aid current elimination schemes in stopping these channels.

Additionally, we provide simple accuracy and bandwidth analysis for both unhidden and hidden IP covert channels and investigate the efficacy of detection and elimination techniques through an experimental study. Next, we detail each contribution further.

1.1.1 Designing Unhidden Network Covert Channels

Our research initially focused on designing event-based storage and timing covert channels which we call IP simple covert channels (SCC). Our design of IP SCCs use the packet arrival as the event and employ the inter-arrival times between the IP packets to convey information in distributed MLS systems. While simple in concept, there proved to be some non-obvious issues in creating the channel and designing the software. One subtle issue is the synchronization of Alice and Eve's event clocks to guarantee channel accuracy. Because IP packets offer no guarantees on the time of packet deliveries, we employ additional schemes to preserve synchronization and resynchronize the channel as needed. In the presence of these factors that introduce noise into the covert channel, we assess the efficacy of IP SCCs in terms of bit/character accuracy (the number of bits/characters transmitted correctly divided by the total number of bits/characters) and channel bandwidth. To do so, we perform basic accuracy and bandwidth analysis for both noiseless and noisy IP SCCs. We show that two factors affect channel efficacy: *Contention noise*, which is the amount of non-covert traffic Eve observes in the covert channel and can potentially reduce channel accuracy [9], and *clock skew*, which is the amount of jitter in the network and can potentially result in the loss of synchronization between Alice's and Eve's event clocks.

1.1.2 Countering Unhidden Network Covert Channels

To counter these channels, we investigate IP SCC detection, elimination, and rate-limiting techniques and introduce methods to detect both noiseless and noisy IP SCCs. Our analysis of the behavior of IP SCCs illustrates that both storage and timing channels show some type of regularity in packet inter-arrival times. This is an expected behavior for storage channels, because they have a fixed timing interval to send/not send the packets. In contrast, timing channels do not use constant timing intervals. However, in a straightforward implementation, a symbol is sent using only a limited number of these timing intervals and these intervals repeat over time. Hence, the packet inter-arrival times for both storage and timing IP SCCs are repetitive. For example, the inter-arrival times for a storage covert channel in Figure 1.1(a) illustrate this regularity as compared to a sample WWW trace (Figure 1.1(b)) taken from the NZIX-II dataset [10], in which no such regularity can be deduced. For such legitimate channels, user and network affects can potentially be observed on the traffic pattern on a longer window. However, we claim that, in most cases (e.g., except streaming traffic), legitimate traffic patterns are much less repetitive than IP SCC traffic.

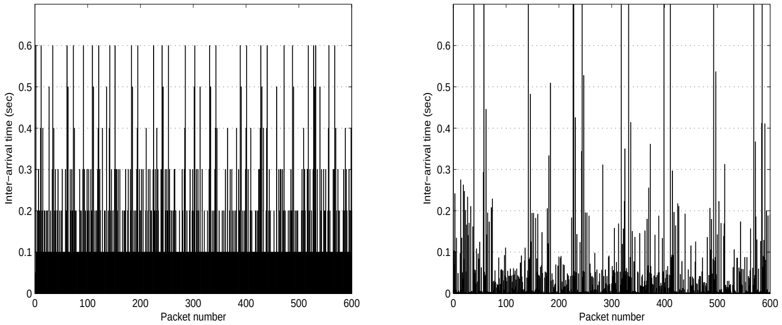


Fig. 1.1. Inter-arrival times for different traffic types. (a) A simulated IP storage SCC with $\tau = 0.2$ secs (no clock skew). (b) An example WWW traffic.

To detect this type of regularity, we introduce two measures that can be used for online (i.e., real-time) detection and evaluate their capabilities in an experimental study using our covert channel software in a real-world scenario. Further, we show that our detection measures fail for noisy covert channels for noise levels higher than 10%. To counter this deficiency, we investigate effective yet computationally expensive search mechanisms to locate the hidden covert channels locally for which the global measures fail. An important observation here is that this regularity characteristic is not limited solely to the channels we designed but to all event-based covert channels. Hence, our detection measures can be employed not only for IP SCCs but for any channel that is based on event timings.

1.1.3 Designing Hidden Network Covert Channels

Despite the characteristic regularity of IP SCCs and event-based channel channels in general, more sophisticated schemes that better hide the channel can be devised. In the second part of our research, we introduce Time-Replay Covert Channels (TRCC) which are specifically designed to hide covert traffic by adjusting packet timings consistent with inter-arrival time sequences that are extracts from recently recorded normal sequences under similar network conditions. Such time-replay channels represent a family of covert channels rather than a particular channel or a channel type. We first detail the channel model for IP TRCCs that use network packet timings to transmit data and hide the covert channel within legitimate IP traffic. We illustrate that using a simple IP channel, called the binary-matching channel, Alice and Eve can exchange messages effectively over the network with high accuracy and secrecy. Further, we investigate the channel efficacy of IP TRCCs and assess it in terms of bit/character accuracy (the number of bits/characters transmitted correctly divided by the total number of bits/characters) and channel bandwidth. We observe that in addition to contention noise and clock skew as in IP SCCs, a third factor, *sequence selection* (i.e., which sequence the time-replay covert channel uses) plays a major role in channel accuracy and bandwidth.

1.1.4 Countering Hidden Network Covert Channels

To counter IP TRCCs, we investigate covert channel detection, elimination, and prevention techniques. We first investigate online (i.e., real-time) detection and elimination schemes that are applicable to IP time-replay covert channels and the conditions under which these schemes are effective. Because IP TRCCs use replays of inter-arrival sequences extracted from legitimate sources, the traffic pattern shows no regularity as in the case of IP SCCs and the regularity-based detection measures do not apply to these channels. Further, we show that a sophisticated attacker can potentially devise and utilize covert time sequences that are equal in distribution to normal sequences that were generated by legitimate channels. Thereby creating replay covert channels that are undetectable through analyzing localized distribution of inter-arrival times. To deal with these channels, we provide a discussion on how the existing covert channel elimination schemes need to be revised to eliminate the new class of covert channels we introduce. To do so, we focus on two covert channel elimination schemes, jamming [11] and the network pump [12], and provide a discussion on their effectiveness on IP TRCCs. We show that the binary-matching channel may survive these elimination schemes with a slightly lowered data rate. (The actual amount by which the data rate is reduced depends on the amount of added average delay to the packets.) We argue that to eliminate these channels, elimination schemes need to transform packet timings more intelligently to remove the associations between the timing values and symbols.

1.2 Significance of the Work

Handling covert channels, that is identifying, analyzing, limiting, and eliminating these channels, is particularly important for multi-level secure (MLS) systems, in which processes may leak information to other processes with a lower classification level via the use of shared resources [13]. Indeed, the original (but now obsolete[1]) evaluation criteria for trusted computer systems (TCSEC: 1983–1999 – *The Orange Book*) included requirements to analyze covert channels in terms of their bandwidth and to develop policies to monitor and maintain their bandwidth below maximum acceptable levels [4]. TCSEC required storage channel analysis for a B2 system, and a complete analysis for B3 and higher assurance level systems. These requirements were also carried to the new version of evaluation criteria: the Common Criteria (CC: 1998–Present) [14]. CC requires covert channel analysis for an EAL5 assurance level as well as a systematic search for covert channels for EAL6 and higher assurance levels.

Despite their importance in secure systems, auditing and detecting covert channels have received less attention compared to other covert channel handling techniques. Furthermore, covert channel detection still remains a novel area often confused with covert channel identification (as in [15, 16]). The identification problem is to discover the potential covert channels that can be realized in the analyzed system. In contrast, covert channel detection mechanisms are similar to intrusion detection systems. The

[1] Replaced by Common Criteria (CC) in 1999.

essential task in covert channel detection is to detect anomalous traffic patterns that can potentially signal the use of a covert channel. In addition to this, an ideal detection scheme would be the one that identifies the communicating parties, the information transferred, and the channel capacity.

Network covert channels are particularly important when the receiver of the (direct) channel (i.e., Bob) is not in the same system as the sender (i.e., Alice) and the access is controlled between the two. These channels pose a threat to distributed systems in which sensitive information is stored. This is because once an attacker (i.e., Eve) implants a back-door into one of these systems, she can now steal information by leaking it through the covert channel that can be established using the back-door program. Moreover, even without the presence of this back-door, Eve can monitor the traffic between Alice and Bob to gather information Alice is leaking. Without these channels, this scenario is not possible because direct access to the untrusted parties (e.g., the nodes outside the trusted network), including Eve, is controlled by policy.

Detection is a common practice in secure systems to monitor malicious activity [14]. Detecting covert channels is desirable for three reasons:

1. Detection provides a mechanism to discourage the use of these channels and may work as a deterrent [5].

2. Most covert channel identification systems need input from system analysts (i.e., to specify the shared resources). As a result of human error, a number of covert channels may remain unidentified. Detection can help record the activity of these channels.

3. Covert channel elimination can be costly for high performance systems [17]. In this case, allowing these channels to exist but monitoring their activity is crucial.

The following type of systems can potentially benefit from covert channel detection and elimination techniques:

High-assurance Systems: Covert channel elimination schemes usually have a negative performance impact on the system [17]. In high-assurance systems (e.g., real time systems) these schemes are not practical. Therefore, allowing these channels to exist, but monitoring their activity, is desirable both to deter the usage of such channels, and to detect the actively running ones in the system.

Military Systems: In a military system, identifying spying users or processes are as important as limiting them. Such spy networks are potentially dangerous and identifying their users can enable authorities take further actions against such users. This way, potential future actions of these users may also be prevented.

As a final note, in both applications, the detection mechanisms can potentially run at the front end of the elimination protocols. This way, the elimination schemes may be activated only if there is a suspicious activity in the network. This architecture helps reduce the negative performance impact of the elimination schemes in addition to enabling authorities to identify the users of the covert channel.

1.3 Dissertation Outline

In this dissertation, we first provide a complete background on covert channel research in Chapter 2. In Chapter 3, we introduce the IP simple covert channel, which is a trivial (i.e., unhidden) covert channel implementation that uses packet timings to convey secret information. In the same chapter, we investigate detection schemes that counter these channels and report the results of our experimental study on detection efficacy. In Chapter 4, we introduce a novel covert channel family, time-replay covert channels, that hide the covert channel using replays of legitimate traffic traces. These channels are resilient against the detection methods that we introduce in Chapter 3 and to counter them, we discuss prevention and elimination techniques specifically adapted to disrupt the channel activity in the same chapter. We present our conclusions in Chapter 5 and provide insight on future directions one could take as a continuation to this work.

2. BACKGROUND AND RELATED WORK

Computer resources are scarce; so much so that dedicating these resources to individual entities is not a common practice in wide-area systems because of the cost and under-utilization of the resources. Today, the general trend is to maximize the utilization of these resources even in single systems through technologies such as virtualization [18] and grid-computing. As a result, we have systems in which the resources are utilized among many entities with different sensitivity levels. However, the sharing of these resources creates potential unauthorized communication channels between these entities, such as covert communication channels. A covert channel finds indirect ways to exploit and use the shared resources to establish stealth communication channels. These channels are then used to leak sensitive information to unauthorized parties. To limit these channels in secure systems, resource sharing between different security levels needs to be well-defined and controlled. The resulting secure system with controlled resource sharing is called a *multi-level secure* (MLS) system. More precisely, an MLS system is a computer environment in which a number of users with different security levels share computer resources. The access to these resources, most importantly read and write permission, is often controlled using access control mechanisms.

In this chapter, we give a brief introduction to the confinement problem and a complete background on covert channels in MLS systems. Since Lampson's identification of covert and storage channels, research in this area has focused on creating, identifying, analyzing, eliminating, limiting, auditing, and detecting these channels. The first step in covert channel handling is to search for potential covert channels and identify them. Once these channels are identified, their capabilities must be analyzed to assess the risk they pose to the system. Typically, an assessment calculates the channel capacity. This analysis is particularly important because closing covert channels can be costly; thus, we need to focus merely on high-capacity channels. After these potentially dangerous channels are identified, eliminating and limiting these channels must be considered. An easy way to eliminate all covert channels and move to a total isolation system would be to disallow the sharing of all resources between different security levels. When this scenario is not practical, the elimination methods mimic this idea to provide means of isolation. Finally, less attention has been paid to auditing and detecting covert channels. In the literature, the detection problem is often confused with the identification problem (as in [15, 16]). The identification problem is to discover the potential covert channels that can be realized in the analyzed environment, whereas the detection problem is to detect an actively running covert channel. Next, we provide details on the confinement problem and covert communication channels.

2.1 Confinement and Information Flow

To design access control schemes in MLS systems, the goal is to confine users to their security levels and regulate their interactions with other users and resources as much as possible. This problem is called the *confinement problem*, which was first identified by Lampson [1]. As defined there, the problem is how to prevent a *service* leaking information to its *owner* while serving a *customer*. In [1], Lampson illustrates several examples to leak information, which he identifies as legitimate, storage, and covert channels.

Lipner argues that Lampson's storage and legitimate channels can be closed using access control mechanisms, but closing the covert channel would be too costly [19] (e.g., dedicating one CPU to each user). Instead of dedicating separate resources to each user, a proposed solution to the confinement problem is to virtually separate the users. One way to do this is by implementing virtual machines. A *virtual machine* is a simulator program that mimics a physical computer(see [20]) and provides a virtual separation of system resources made available to each user. One such example is the

KVM/370 system [21]. This security kernel provides virtual machines to its users and each virtual machine is given the same security clearance as its user. No direct communication is allowed between different "machines" if they belong to different security levels. Another solution to the confinement problem is to use *sand-boxing* to confine the process' actions and information flow models to control the flow of information between the users (e.g., [22]).

Nevertheless, it has been shown that none of these traditional techniques completely counter the covert channel threat. For example, both access control schemes and virtual machines provide a sense of separation of different security levels by restricting the direct access between these levels and can be thought of as mimicking a *total isolation* system [1]. However, neither model is secure against indirect communication methods (i.e., covert channels). Lipner suggests that Lampson's storage channels can be eliminated using access control schemes. However, to do so, each resource in the system needs to be carefully evaluated to determine if it can be used as a communication medium [23].

In our research, we assume that the system is secured for direct access using access control mechanisms to protect the system policies. The Bell-LaPadula model is a well known example, which combines mandatory and discretionary access controls [2]. In this model, the users are seen as *subjects* and each subject is given a security clearance. Each resource is an *object* and has a security classification. Without loss of generality, we assume that the system has HIGH and LOW clearance and classification levels for subjects and objects, respectively. The model works on two principles: a LOW clearance subject cannot read a HIGH classified object (read-up), and a HIGH clearance subject cannot write to a LOW classified object (write-down) [2].

In our scenario, the covert channel *sender* (Alice) and *receiver* (Eve) are subjects with HIGH and LOW security clearances, respectively. The shared resource is an object with HIGH security classification. Therefore, Eve cannot read from this resource directly because this is disallowed by policy. The receiver may be an automated process (i.e., a back-door) leaking information to its master[1]. A *trojan horse* is one such program that performs back-door activity to steal information. These programs usually arrive at the user environment through legitimate channels (i.e., through e-mail) and their installation usually does not violate system policies (i.e., no read-up or write-down is needed).

2.2 Covert Communication Channels

2.2.1 Channel Definition and Types

A *covert channel* is a general term used to define a mechanism that can be used to violate a security policy by allowing sensitive information to leak to an unauthorized party. To bypass the access control schemes in MLS systems, covert channels use the shared resources in ways not intended in their design. Examples of covert channels were first identified as a part of the confinement problem [1]. In such examples, the servicing process (i.e., the sender) leaks information shared between the customer and the service to the service owner (i.e., the receiver) by modifying the file access and processor usage. The three classic examples are the file-lock [5], the disk-arm [6, 7], and the bus-contention [8] covert channels. For example, the *file-lock channel* is implemented by a sender process locking a shared file to send a *one*, and releasing it to send a *zero* [5]. The receiver process then tries to access the file. It records a *one* if it can access the file, and a *zero* otherwise. Another example is the *disk-arm channel* identified in the KVM/370 security kernel of the systems with a shared read-only disk drive [6, 7]. In this channel, the receiver process uses the direction of the disk head to observe the sender process' actions. To send a *one*, the sender process seeks to the inner cylinders on the disk, and to send a *zero*, it seeks to the outer cylinders. The receiver process then issues two reads: one from the inner cylinder and one from the outer. Depending on which read completes first, the receiver decides

[1] In highly controlled environments where all communication channels are monitored, Alice may also want to leak information covertly to avoid detection.

whether the sender has issued a *one* or *zero*. The disk-arm channel is an example showing that virtually separating the processes, as in KVM/370, is not sufficient for total isolation. Hu [8] introduced the *bus-contention channel*, which exploits systems with multi-processors running on a single bus. To send a *one*, the sender process issues several CPU calls saturating the bus. To send a *zero*, it stays idle. The receiver process continuously generates CPU calls and records a *zero* if the responses are faster than a pre-specified threshold, and a *one* otherwise.

Covert channels can be characterized in variety of different ways: depending on the type of shared resource used (storage or timing), the noise level (noiseless or noisy), the type of the system they are found in (single or distributed), and the participation of the source (active or passive).

Storage versus Timing Covert Channels

A *covert storage channel* "involves the direct or indirect writing of a storage location by one process and the direct and indirect reading of the storage location by another process" [4]. A *timing covert channel* involves a sender process that "signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process" [4]. Our use of terminology is the same as that found in Lampson [1].

The classification of covert channels into storage and timing is not clear cut and different definitions exist in the literature. In reality, this classification is only useful when the channel capacity is calculated. In capacity analysis, we need to make a distinction between storage and timing channels. This is because in storage channels, the time to send each symbol is constant, whereas in timing channels the time varies each time the symbol is sent (see Section 2.2.3).

An alternative covert channel definition was given by Kemmerer [23] as a list of conditions for storage and timing channels. A covert channel is a storage channel if the following conditions are met:

1. The sending and receiving processes must be able to access to a common attribute of a shared resource.
2. The sender must be able to force this shared attribute to change.
3. The receiver must be able to recognize this change.
4. The sender and receiver must be able to initiate this conversation (i.e., using a lower bandwidth channel).

According to these criteria, the disk-arm channel discussed above is a storage channel. In this example, the receiver and the sender processes share a common attribute (i.e., the cylinder head) of a shared resource (i.e., the hard drive); the sender can force this attribute to change (i.e., by moving the head); and the receiver can recognize this change (i.e., by examining the read finish time). Additionally, the processes need to negotiate on a start time, which can be accomplished by using another low-bandwidth channel. Similarly, the file-lock channel is also a storage channel.

Similar criteria for timing channels are:

1. The sending and receiving processes must have access to a common attribute of a shared resource.
2. The sender and receiver must share a *reference clock*.
3. The sender must be able to modulate the receiver's response time for detecting a change in this attribute.
4. The sender and receiver must be able to initiate this conversation (i.e., using a lower bandwidth channel).

The bus-contention channel presented above is a timing channel. Timing channels require access to a reference clock shared between the sender and receiver. In most systems, the system clock is shared between different security levels and can serve as a reference clock. If no such global clock is available, the receiver can still monitor the number of instructions generated by the CPU to measure the progress of time [24]. Assuming that the parties share a reference clock in the bus-contention channel, the sender and receiver both have access to a shared resource (i.e., the system bus); and the sender process can modulate the receiver's response time (i.e., by congesting the bus or staying idle). And lastly, the parties need to negotiate on a start time (e.g., by using another low-bandwidth channel).

Observe that, according to this criteria, the disk-arm channel discussed above may also be classified as a timing channel. This is because the sender modulates the response time for the receiver process by positioning the disk head on different cylinder positions. This is exactly the condition required in the third Kemmerer criterion for timing channels. Because of this property, the disk-arm channel is sometimes referred to as a timing channel with a storage exploitation [24, 25]. However, this is not true for the bus-contention channel, as it uses a *logical* shared resource (the busy/idle status of the bus).

The disk-arm channel example shows that the above criteria for channel classification by Kemmerer is ambiguous. This ambiguity can be eliminated by defining *covert hybrid channels*[2], which use shared resources with both timing and storage characteristics. A non-ambiguous definition is given by Moskowitz et al [17], in which the storage channel is defined as a channel "where the output alphabet consist of different responses all taking the same time to be transmitted". A timing channel is thus a channel "where the output alphabet is made up of different time values corresponding to the same response". According to this definition, both disk-arm and bus-contention channels are timing channels. The file-lock channel is a storage channel. In our research, we use Moskowitz's definition to classify our covert channels.

[2] This is our terminology. These channels are sometimes referred as *mix channels* [17].

As a final note, we need to emphasize that none of the channel definitions above explicitly specify how long the receiving process needs to observe the channel for one bit of information. For example, to send a bit stream `001` using the file-lock channel, the sender process needs to release-release-lock the file in that order. However, without specifying a *timing interval* (τ), it is not possible for the receiving process to observe two consecutive zeros. Similarly, the receiver cannot distinguish consecutive ones. However, this is only a problem for storage channels, because in timing channels the timing of the events is defined implicitly. Moreover, the above is true for all storage channels except those that do not use repeating symbols to send information. For most storage channels the sender uses the same events (i.e., locking a file) to output the same output symbols. These symbols belong to the alphabet used in encoding the messages. Unless this encoding is chosen such that no symbol repeats consecutively, there must be a pre-specified interval that the receiver process observes, and, for example, records two ones if the file remains locked for two intervals. However, this alternative solution reduces the message space substantially. For example, given an alphabet consisting of n different symbols and messages of m length, the number of different messages is limited to $n(n-1)^{m-1}$ instead of n^m. This suggests that storage channels have a 5^{th} criterion:

5. Unless a special encoding is chosen to avoid consecutive symbols, the sender and receiver must be able to pre-negotiate on a *timing interval* τ during which the receiving process will observe the change in the channel.

In this framework, the file-lock channel receiver observes the file for the duration of the timing interval to decide on the bit the HIGH sender has intended to send. If it decides the file is locked, it records a *one*, idles for the rest of the interval, and restarts observing the file in the next interval. Otherwise, it records a *zero* and the rest is the same as the *one* case.

Two other variations on covert channels have been introduced in the literature. The *covert sorting channel* puts emphasis on the ordering of the events observed in a timing interval [26, 27]. In sorting channels, the covert information is transmitted by the order of a set of events. For example, if the receiver is monitoring n events given a timing interval, then the occurrence of n events in a particular order transmits one message from message space of size $n!$. Another variation is the *covert counting channel*, which transmits information by the number of occurrences of a single set of events [28].

Noisy versus Noiseless Covert Channels

If a shared resource is used not only by the covert channel parties but also by other legitimate users, the resulting channel is a noisy communication channel. Specifically, a *noisy covert channel* is a channel in which both normal and covert traffic are observed. In contrast, in a *noiseless covert channel*, the shared resource is solely used by the parties involved in the covert communication. The channel noise described here is not *signal noise*, but is *contention noise* caused by the contention for the shared resources [9]. Noise is a degrading factor in covert channel accuracy. For example, in the file-lock channel, the shared file may also be locked by other legitimate processes. This deteriorates the channel accuracy, because the LOW process can no longer monitor the HIGH process' activity alone. The effects of noise in covert channels have been analyzed to estimate channel capacity. Introducing noise into covert channels has also been illustrated as a way limiting covert channel bandwidth. We present these techniques in Section 2.2.4.

Network Covert Channels

A *network covert channel* is a covert channel in which the shared medium is the network environment (i.e., the transmission lines, firewalls, routers, etc.). File-lock, disk-arm, and bus-contention channels are all single system covert channels. This is because the resources shared by these channels all belong to a central computer environment. Initial research in covert channels focused on single systems. However, these results can be easily extended to include network channels [29].

Network covert channels have been used by attackers to communicate with compromised hosts (i.e., zombies), particularly in distributed denial of service (DDOS) attacks [30]. Many tools exist for setting up network covert channels using a variety of protocols including TCP, IP, HTTP, ICMP, AODV, and MAC protocols [26,27,31–44]. The common practice in most of these examples is to encapsulate covert channels in legitimate protocols to bypass firewalls and content filters.

The data section of packets is the easiest place to convey covert information, because of its large size and because it is relatively unstructured compared to headers. Modifying the packet payload, which is used in steganography to hide data, is outside the scope of our research. Unused header fields that are either designed for future protocol improvements or in general go unchecked by firewalls and network intrusion detection devices may convey information in the form of a covert channel. Many covert channels have been illustrated in TCP and IP protocols using packet header fields [26, 27, 31, 32, 34–37, 39, 41, 43, 44]. Rowland [44] illustrates three examples of covert storage channels. The first encodes the messages into the identification field (ID) of an IP header. The second embeds the messages into the initial sequence number (ISN) field of a TCP header. And the third uses the TCP ACK sequence number field. An improved version of Rowland's TCP ISN covert channel, *Nushu*, is presented in [43]. The main difference between Rowland's TCP ISN channel and Nushu is that Nushu encrypts the TCP ISN field to make it look like a random number to better obscure the channel. A smart attacker can even devise means to use some of the header fields that *do* fall under scrutiny, such as the IP checksum field. Abad [31] illustrates that the sender can send any message using hash collision in the checksum field. Sorting covert channels have been devised for the TCP/IP protocol, in which it is the ordering of the packets that transmits the covert information to the receiver side [26, 27]. However, for this channel to work, the receiver needs to know the correct ordering of the packets. Therefore, these channels require the use of the

IPSEC protocol, which provides the original order in which the packets were sent. Murdoch et al [40] argue that the previous efforts to hide covert channels in packet header fields alters the behavior of these fields. They claim that this change can be detected and introduce *Lathra*, a covert channel that uses the TCP ISN field, but tries to mirror the ISN generation process to avoid detection.

TCP and IP are not the only protocols used for covert channel exploitation. *Reverse WWW Shell* is a covert shell that uses HTTP request parameters to leak information [39]. In this channel, the sender issues a HTTP `GET` command with the specified parameters, bypasses the HTTP-only firewall, and leaks the information to the receiver side. Another HTTP covert channel was illustrated by Bauer [32] for privacy purposes. In this channel the sender wants to remain *unlinkable*, hence it redirects the communication via an unwitting client's browser. The author shows that this is possible using redirects, cookies, active content, etc. ICMP packets have also been used to host covert shells. The *Loki* covert shell runs on ICMP and uses the data portions of the `ICMP_ECHO` packet to send a command to a remote receiver [34,35]. The receiver back-door then executes this command on the victim computer, and returns the result embedded in the data portion of the `ICMP_ECHOREPLY` packet. A list of similar covert shells developed by the black hat community can be found in [37].

Other interesting examples of covert channels were illustrated in the OSI protocol stack [42], in LAN networks [38, 45], and in ad-hoc wireless networks [33]. In LAN networks, the destination ID, the packet length, the time between successive transmissions [42], and the number of collisions in a resolution period (in MAC protocol) [45] were used. In ad-hoc networks, Li et al [33] identify four covert channels in AODV routing protocol. The authors consider the timing between the route requests, the source sequence number, the lifetime field, and the destination ID.

At this point, we need to emphasize that although the above channel examples are referred to as "network covert channels" in the literature, they rely on *information hiding* principles [25]. That is, these channels use resources (i.e., packet headers) that are directly accessible by the communicating parties, unlike the covert channels in

MLS systems. *The network channels we introduce in our work are covert channels because we do not assume direct access to the packets transmitted over the network.*

Active versus Passive Covert Channels

In the literature, covert channels are usually referred to as communication channels with an active sender and a receiver. The sender of this channel is usually a malicious software component actively leaking information. However, the source of this communication channel does not need to be active for a covert channel to exist and a LOW receiver can still gain bits of information even if the HIGH source does not actively leak information. Therefore, we make a distinction between active and passive sources. An *active-source covert channel* is the one with a trojaned source actively leaking information. A *passive-source covert channel* exists with no participating source. Covert channels described in the literature are usually active source channels. However, this may lead to the false implication that a system free of malicious software is also free of covert channels. In fact, it is a popular practice for password crackers to observe the running time of password checking algorithms to extract information. The resulting communication channel in which the sender is the algorithm and the receiver is the cracker is a passive-source covert timing channel.

Finally, our research does not focus on the following information hiding techniques. The *subliminal channel* was first introduced in [3] as a solution to the *prisoner's problem.* Simmons describes this problem as follows: Two prisoners are placed in two far cells and want to coordinate an escape plan. Exchanging messages is allowed, however the messages are open to the guardian. Moreover, the guardian may introduce fraudulent messages to deceive them. Therefore, they need to authenticate the messages. The channel the prisoners establish to deceive the guardian is called the subliminal channel. These channels are considered as a special type of covert channels, and cryptographic protocols are used to implement them [46, 47]. However, they are different from Lampson's definition of covert channels in MLS systems because direct access to the shared resource is allowed.

Second, steganographic techniques were illustrated to hide information into a directly accessible resource (e.g., a picture) in ways to prevent detection (e.g., undetectable to human eye). Detection of these concealed messages has also received significant attention (see [48]). Similar to subliminal channels, steganoraphic protocols work on information hiding principles, hence again are different from Lampson's definition of covert channels in MLS systems.

2.2.2 Channel Identification

Several methods were proposed for identifying covert channels in MLS systems [16, 23, 49–52]. In one of the early examples, Denning et al [49] use lattice models to identify storage channels by determining insecure information flows between the subjects. They particularly focus on identifying the *implicit flows*, which are not obvious from analyzing the direct (explicit) flows. The following techniques all identify these implicit flows between the resource attributes and the primitive operations by using different data structures (e.g., resource matrices, flow trees, messaging charts). The resource matrix was the first such data structure used to visually identify covert storage channels [23]. In this shared resource methodology (SRM), the attributes and operations are the rows and columns of a matrix, respectively. The relation between these attributes and operations are then represented in the matrix cells. Moreover, the implicit flows are identified by taking a form of transitive closure of the matrix. In [23], Kemmerer et al argue that timing channels can be represented with this matrix by including the system time as an attribute. But this methodology was mainly designed for storage channels and extending it to include the timing channels does not guarantee the identification of all timing channels in the system. A less efficient but more effective data structure is the covert flow tree (CFT). CFT is a data structure that is used in goal-directed deductive reasoning. Porras et al [50] use the same goal-oriented approach using goals such as "Covert Storage Channel via Attribute A", and create CFT trees for each goal. Traversing this tree tells us whether this goal is achievable or not. In comparison to SRM methodology, CFT trees provide a more natural but less efficient technique to analyze the storage channels with multiple resource exploitation.

Both the SRM and CFT methods were initially designed for single systems but can be extended to network channels. A more natural way to represent the interactions in network channels is by using message sequence charts (MSC). Helouet et al [16] illustrate how to use MSCs to identify storage channels at the requirements level of a software development cycle, however they do not provide analysis of how MSCs can be used to detect implicit flows that use multiple attributes to leak information. Tsai et al [51] argue that all of the above methods are high-level analysis techniques and the covert channels identified at these levels (e.g., at the requirements level) may not represent all channels that might occur in the source level. The authors provide a detailed method that can be applied at the source-level which includes the identification of implicit flows by analyzing the direct and indirect visibility of the attributes. However, both SRM and CFT are generic methods and are not restricted to a high-level (e.g., requirement level) analysis, although using these methods for source-level analysis may be inefficient (especially CFT). Lastly, a technique similar to SRM is used in [52]. In this method, covert channels with the same sender and receiver are aggregated. This is particularly useful when the communicating parties use more than one medium to communicate covertly in an attempt to create seemingly unimportant low bandwidth channels.

Except for the SRM methodology, none of the methods presented above focus on timing channels. Additionally, most of these techniques depend on the system analyst's input to the identification algorithm. Therefore, in all of the techniques the methods require manually identifying all of the attributes of all shared resources in the system. Covert channels exploit an attribute of a resource that is not initially thought to be used as a communication medium (e.g., the disk cylinder head). Therefore, it is possible that one of these attributes will be overlooked and a covert channel can be established later by exploiting this attribute.

2.2.3 Channel Analysis

The second step in covert channel handling is the analysis of each identified channel to assess the threat level of each covert channel. Essentially, the amount of information that leaks through these channels needs to be calculated to identify high-capacity channels for closure or rate limitation. Tsai et al [53] analyze covert storage channel bandwidth using Markov models applied to both noiseless and noisy channels. The bandwidth for both cases is presented in a closed form. However, the assumptions required by a Markov model are not realistic for real systems. A more general approach to covert channel analysis is to treat these channels as legitimate communication channels and apply the results from information theory (i.e., Shannon's notion of channel capacity [54]). This idea was first used by Millen [55] which inspired the more rigorous analysis presented later [9, 17, 56–59]. In the covert channel analysis guide published by the National Computer Security Center (NCSC), the analysis by Tsai et al [53] (the Markov model) and by Millen [55] (the maximum information rate) are compared. Although they found the results comparable, the guide favors Millen's analysis because it defines a more realistic scenario of covert channel use [5].

In information theory, the capacity of a channel is defined as the maximum information rate. For noiseless channels, capacity C can be defined as (using a similar notation as in [55]),

$$C = \lim_{n \to \infty} \left(\frac{log(N(t))}{t}\right) \tag{2.1}$$

where $N(t)$ is the number of possible messages that take t time. Millen [55] uses this definition to calculate the capacity of noiseless channels using finite-state diagrams. Later, Moskowitz et al [56] prove a theorem that provides a solution to this asymptotic definition of capacity for *simple timing channels* in general. As illustrated in that paper, another way to define capacity is through *mutual information.* Let X and Y be two random variables representing the channel's input and output characteristics, respectively. The amount of mutual information shared between the parties is denoted by $I(X, Y)$. For noisy channels, I can be calculated in terms of entropy as follows,

$$I(X, Y) = H(X) - H(X|Y) \tag{2.2}$$

where $H(.)$ denotes the entropy and $H(.|.)$ denotes the conditional entropy, or *equivocation.* The sender and receiver are said to be *non-interfering* if $I(X, Y) = 0$. Moreover, if the channel is noiseless, then $I(X, Y) = I(X)$. For timing channels, the mutual information per tick (a tick is a unit measure of time) is given by,

$$I_t = \frac{I(X, Y)}{E(T)} \tag{2.3}$$

where $E(T)$ is the expected value of time for a symbol to be transmitted. For storage channels, $E(T)$ is the timing interval τ. For noiseless channels as in [55] and [56],

$$I_t = \frac{I(X)}{E(T)} \tag{2.4}$$

Finally, the capacity of the channel is simply the maximum value of I_t, maximized over the distributions of X,

$$C = max(I_t) \tag{2.5}$$

The asymptotic and mutual information analysis of noiseless channel capacity are equivalent. However, only mutual information analysis is easily extendable to noisy channels. Costich et al [57] use mutual information to calculate the capacity of a noisy storage channel that exists in the two-phase commit protocol used in MLS transaction-based database systems (see [60]). In this channel, the HIGH sender passes information to the LOW receiver by either aborting (to send a *zero*) or committing (to send a *one*). The channel is a noisy channel, because the other parties can participate in the commit or abort decision and can affect the output observed by the receiver[3]. In this channel, if the sender inputs a *zero*, the output is *zero* independent of other users' inputs. In contrast, if the input is a *one*, there is a certain probability for the receiver process to observe a *one*. The resulting communication channel is called a *Z-Channel*. A Z-channel is a type of noisy communication channel, in which one of the symbols is transmitted perfectly because it is independent of the noise (e.g., the *zero* symbol in the two-phase commit protocol) [61]. The capacity of this particular Z-channel is then calculated using the equations (2.2), (2.3), and (2.5) from above. Detailed analysis of Z-channels can be found in [62].

Son et al [59] further analyze covert channels in MLS database systems by applying the mutual information principles from above to calculate the covert channel capacity in secure two-phase locking in database systems, with the added requirement of real-time processing. Mutual information is also used to calculate the capacity of noisy timing channels [9,58]. The major assumption in these analysis is that whenever there is a contention for resources, the output signal is modeled to arrive exponentially distributed with parameter λ. The capacity is then expressed using λ and p, where p is the probability of the user sending a *zero*.

Shannon's techniques for calculating capacity works only for *finite-state* communication channels [54]. *Infinite-state* covert channels cannot be analyzed using the information theoretic results from above. Shieh [63] presents examples of storage channels that belong to this category, and proposes an encoding technique by which they can determine the capacity of infinite-state channels.

Covert channel guidelines often give a threshold for acceptable covert communication (e.g., 100 bits per second in the NCSC guideline [5]). However, this threshold really depends on the system under question (e.g., depending on the importance of

[3] A transaction is committed if all of the participating parties commit individually. It is aborted if at least one party aborts or timeouts.

the data [63]). Moreover, capacity is not the only measure that can be used to identify the importance of the channel. Indeed, because capacity is an asymptotic measure, it may be misleading. For example, Moskowitz et al [17] illustrate that a zero-capacity covert channel can be used to send a short but important message. To deal with these types of channels, the authors introduce small message criterion (SMC), which can be used in conjunction with channel capacity. SMC focuses on short messages and gives guidelines for what level of covert transmission will be tolerated.

In this dissertation, our goal is not to provide a full analysis of channel capacity but to evaluate the efficacy of the IP covert channels we introduce. That is, we would like to know how much information Alice can leak to Eve under the given channel setup and how we can improve on this by improving the channel design. Therefore, we do not employ the capacity analysis techniques presented above but provide simple bandwidth analysis for our IP covert channels.

2.2.4 Elimination and Limiting

The next step is to eliminate or rate-limit high-capacity covert channels. Intuitively, covert channel capacity can be limited by adjusting the parameters that appear in the capacity formula. For example, in the two-phase commit channel, the capacity is inversely proportional to the number of other participants (n) and the probability of aborting (p) [57]. Therefore, the capacity can be reduced by increasing the value of either parameter. Similarly, parameter adjustments can be done in other covert channels analyzed above. However, most of these adjustments work against the system performance. For example, Moskowitz et al [17] argue that adjusting n and p to limit the two-phase commit channel substantially degrades the overall system performance. They conjecture that eliminating covert channels in high-assurance systems is not easy and thus more sophisticated elimination schemes are needed.

Different approaches for limiting covert channels are illustrated. One possible way is to introduce contention noise into the channel and separate the sender and receiver's system clock. Hu [8] describes one such approach using *fuzzy time*. In this scheme, random interrupts generate random clock shifts to disrupt the system clock in different security levels. As a result, the sender and receiver of the covert channel no longer share a reference clock; Hence, they cannot establish a channel using the system clock as the shared resource. Trostle [64] improve this scheme by further separating the processes with different security levels. Gray [65, 66] analyzes the channel capacity with fuzzy time using the standard definition of discrete memoryless channels. Further, Gray [28] presents another technique called the *probabilistic partitioning* for limiting covert timing channels. This technique allows processes to share accurate reference clocks and can be used to close the bus-contention channel completely. In this scheme, the bus controller program randomly switches between the *secure* mode (in which the bus is partitioned between different security levels) and the *insecure* mode (in which covert channels can exist). Lastly, Giles et al [11] discuss the use of jamming devices to disrupt the timing of the events in the system. They additionally provide game-theoretic analysis of jammed channels to calculate the channel capacity and the efficacy of the jammer.

Another way to rate-limit covert channels but still assure a certain level of system performance is to use a *data pump*. A data pump, also called the NRL pump, is a store-and-forward buffer that controls the data exchange between different security levels [67, 68]. The pump 1) stores a message sent by the LOW process, 2) ACKs the LOW process upon the receipt of the message, 3) forwards the message to the HIGH process, and 4) waits for the ACK response from the HIGH process. If the HIGH process' ACK rate is slower than LOW's message input rate, a covert channel can be established using the data pump as follows: The LOW process first sends back-to-back messages to the pump until it no longer receives ACK messages. The LOW process thus *blocks* the pump. To send a *one*, HIGH removes one of the queued messages from the pump. To send a *zero*, it stays idle. The LOW process constantly monitors a timing interval τ. If it receives an ACK during this time, it records a *one*. Otherwise, it records a *zero*. This channel does not exist if HIGH's ACK rate is faster than LOW's message input rate because in this scenario, LOW cannot fill the buffer to create the channel. The solution to the above covert channel is to add probabilistic delay to slow down LOW and reduce LOW's message rate down to HIGH's ACK rate [67–69]. Montrose et al [70] present an implementation of the data pump called the *event driven pump*. In [12, 71], a network version of the NRL pump is introduced. Different from the basic pump, the network pump serves multiple network users and provides additional features. One such feature is *fairness* which is a scheme used to provide fair load cuts when some of the network load needs to be cut by the pump because of system limitations.

In this dissertation, we illustrate that the type of channels we introduce can potentially survive channel elimination schemes such as jammers [11] and network pumps [12] with lowered data rates. We then discuss two types of transformations on packet inter-arrival times to increase the efficacy of these elimination schemes.

Finally, an effective way to eliminate most TCP/IP covert shells (i.e., information hiding channels that use packet headers) is through traffic normalizers [72, 73], which modify both incoming and outgoing packets by standardizing fields that are unused, redundant, or optional. Because covert shells are different schemes than the covert channels we present in this dissertation, these techniques do not apply to our IP covert channels (i.e., our channels do not use packet headers or payload to transmit information).

2.2.5 Auditing and Detection

Auditing and detecting covert channels have received less attention compared to other covert channel handling techniques. Both methods are common practices in secure systems to monitor malicious activity [14] and applicable to covert channels for three reasons:

1. Both methods provide a mechanism to discourage the use of these channels and may work as a deterrent [5].

2. Most identification systems need input from system analysts (i.e., to specify the shared resources). Because of human error, a number of covert channels may remain undetected. Auditing and detection can help record the activity of these channels.

3. Covert channel elimination can be costly for high performance systems [17]. In this case, allowing these channels to exist but monitoring their activity is crucial.

Additionally, both auditing and detection can co-exist with channel elimination and limiting methods. In this context, auditing and detection can be used to identify the users participating in a covert channel.

Auditing is a required activity for secure systems. However, it has been shown that the audit practices described in the TCSEC requirements [4] are not sufficient for covert storage channel auditing [74]. The problems with the TCSEC auditing schemes are:

1. Determination of the shared resource in covert channel use.

2. Distinguishing covert channel senders and receivers.

3. Determination of whether a primitive alters or views the shared resource.

4. Circumvention of covert channel audit.

Sheih et al [74] discuss that these problems occur because 1) both view and alter operations are allowed for the primitives, 2) a primitive can access many shared resources used by many covert channels, and 3) many primitives can access the same shared resource used by one covert channel. They propose an audit scheme that identifies the shared resource for each covert channel and employs *audit vectors* to unambiguously record who alters/views which shared resource in what order. Once a trace of a covert channel is identified, its bandwidth can be determined. However, the actual information leaked through this channel may not be recovered (i.e., the sender can encrypt the message [74]).

Detection of covert channels is a closely related problem. The detection problem can be classified as *online* and *offline* detection. Online methods focus on the speed of the detection algorithm to implement real-time on-the-fly detection schemes. In contrast, offline detection mechanisms can be run as a batch process. Online detectors use little data for fast detection, whereas offline detectors perform more complex data analysis using as much data as needed. Both online and offline schemes rely on the quality of the data logs generated by the above audit schemes.

Unusual traffic patterns may lead to discovery of storage covert shells. For example, multiple ping requests within a small time interval may indicate a storage shell in the ICMP protocol such as that used by Loki [34]. In addition, storage shells can sometimes be detected by observing variations in unused packet header fields [73]. Sohn et al [75, 76] illustrate an offline detection scheme that uses support vector machines (SVM) to detect ICMP (payload) and TCP/IP header (IP ID and TCP ISN) covert shells. Murdoch et al [40] argue that TCP and IP specifications exhibit sufficient structure to define what is normal, and that learning methods such as SVM is an overkill. They present a suite of tests to detect whether the IP ID and TCP ISN generating processes are in compliance with the specifications.

3. SIMPLE NETWORK COVERT CHANNELS

In this chapter, we introduce simple network channels that exploit timing of IP network packets to leak information from Alice to Eve. We first describe our implementation of an IP covert channel, both as a storage and a timing channel, and discuss the subtle issues that arose in its design. We then report our results of channel efficacy in terms of accuracy and bandwidth in Section 3.2. In Section 3.3, we show that the regularity of a noiseless IP simple covert channel can be used to differentiate it from legitimate channels and present two efficient online detection measures that provide detection rates over 95%. In Section 3.4, we repeat similar analysis for noisy IP covert channels in which legitimate and covert traffic are mixed in an attempt to hide the covert channel. We show that our online detection measures fail to identify the covert channel for noise levels higher than 10%. For such channels, we investigate effective yet computationally expensive search mechanisms that are more suitable for offline detection to locate the hidden covert channels locally for which the global measures fail.

3.1 Covert Channel Creation

Within a distributed MLS system that properly follows policy, an observer with a LOW security level may still be able to observe the presence of HIGH network traffic without having access to the encrypted contents. Assuming that Alice is a HIGH sender, Bob is a HIGH receiver communicating with Alice, and Eve is a LOW observer, the only information shared between Alice and Eve is the timing of the packets traveling from Alice to Bob. The IP simple covert channels (SCC) we present in this chapter use this timing information to covertly transmit a secret message from Alice to Eve by adjusting the transmission intervals of the network packets Alice sends to Bob. In this section, we detail the covert channel operation and implementation for both storage and timing IP SCCs.

3.1.1 Channel Design

IP SCCs transmit a covert message from Alice to Eve bit-by-bit; hence the covert channel symbol set is composed of *zero*s and *one*s. The lone event in the system is `packet_arrival`, which is controlled by Alice and observed by Eve. To send consecutive bits to Eve, Alice generates a sequence of packets each separated by a timing interval (i.e., inter-arrival time). Depending on how this timing interval is chosen, a storage or a timing channel can be devised. If Alice uses a single timing interval τ to denote the inter-arrival times between the packets throughout the covert communication, then the resulting channel is a storage IP SCC. Alternatively, if Alice uses a set of timing intervals τ_1, τ_2, ..., τ_n for the same purpose, the resulting channel is a timing IP SCC. While the underlying operation is similar, we show that the latter channel protocol is easier to implement and is more robust in the presence of timing (i.e., synchronization) errors.

The storage IP SCC operates as follows: Alice and Eve agree a priori on a constant timing interval (τ) and the starting protocol (either a particular time or in response to a network event, such as the first packet sent). To send a *zero*, Alice maintains silence throughout the interval τ and does not send any packets to Bob. To send a *one*, she transmits a single packet to Bob in the middle of τ. On the receiving end (of the covert channel), Eve monitors the transmission line between Alice and Bob. She records a *zero* if no packets are observed throughout the interval τ, a *one* otherwise. By observing each τ consecutively, Eve records the entire binary string transmitted over the covert channel. Figure 3.1 illustrates a running example of a storage IP SCC.

The timing IP SCC operates as follows: Alice and Eve agree on a set of timing intervals $\tau_1, \tau_2, ..., \tau_n$, a set of associations for each timing interval (e.g., (τ_1, *zero*), (τ_2, *one*), etc.), and the starting protocol. For simplicity, we assume that they agree on

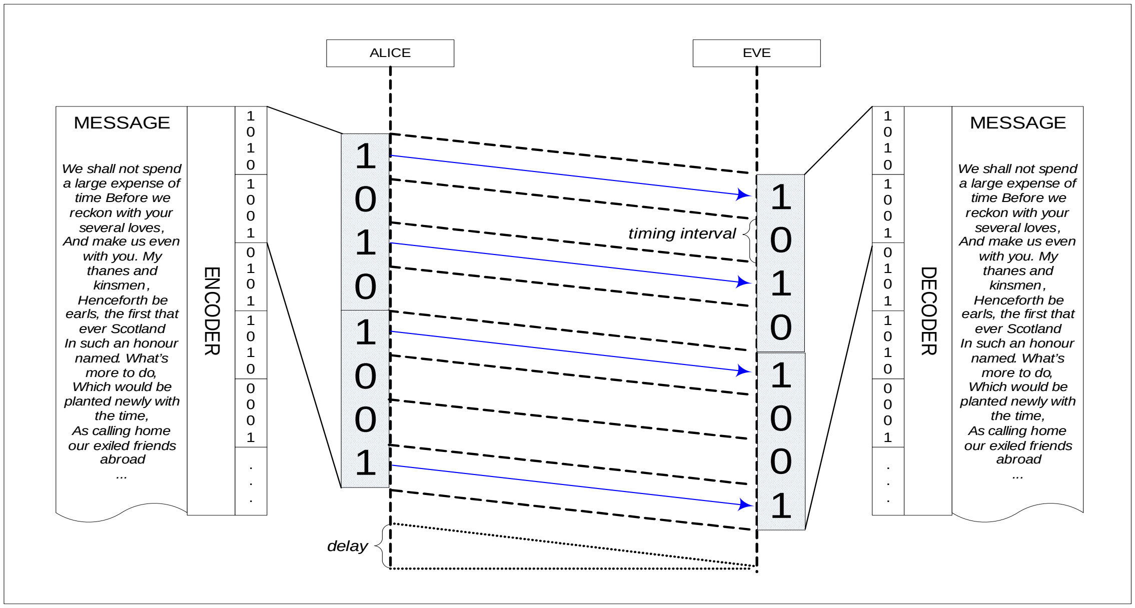
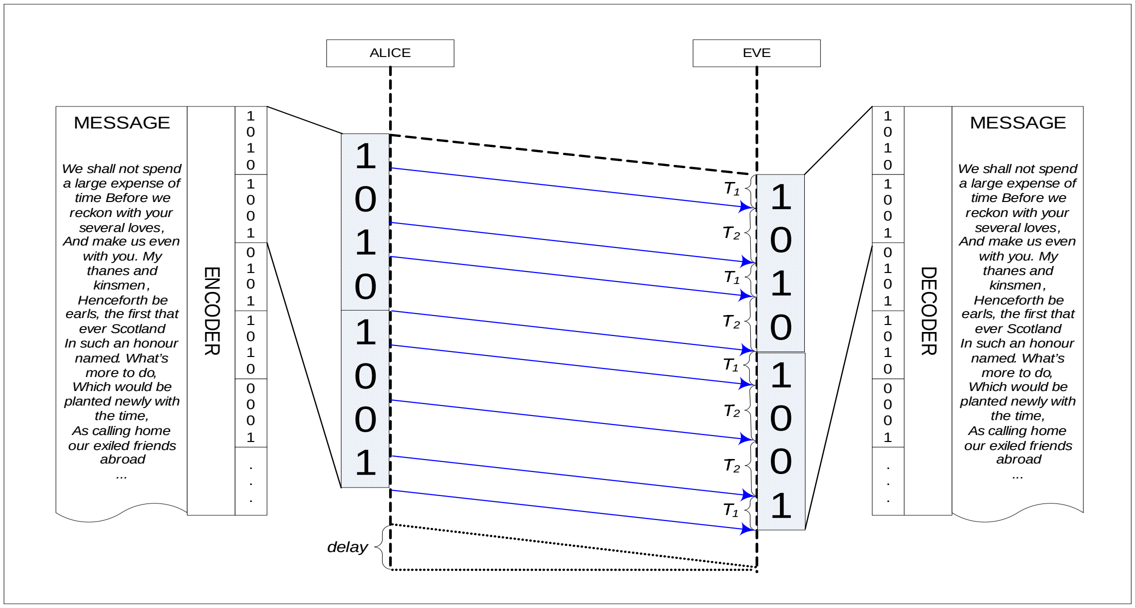

ALICE
EVE
MESSAGE
We shall not spend a large expense of time Before we reckon with your several loves, And make us even with you. My thanes and kinsmen, Henceforth be earls, the first that ever Scotland In such an honour named. What's more to do, Which would be planted newly with the time, As calling home our exiled friends abroad ...
ENCODER
1 0 1 0
1 0 0 1
0 1 0 1
1 0 1 0
0 0 0 1
. . .
1 0 1 0 1 0 0 1
timing interval
1 0 1 0 1 0 0 1
delay
DECODER
MESSAGE
We shall not spend a large expense of time Before we reckon with your several loves, And make us even with you. My thanes and kinsmen, Henceforth be earls, the first that ever Scotland In such an honour named. What's more to do, Which would be planted newly with the time, As calling home our exiled friends abroad ...

Fig. 3.1. IP storage SCC. The example text is first encoded with a coding scheme and then sent bit by bit to the receiving end. The message is rebuilt by decoding the bit stream.

two timing intervals τ_1 and τ_2 with associations (τ_1, *zero*) and (τ_2, *one*). To send a *zero*, Alice *sleeps* for τ_1 units of time and sends a packet at the end of this interval. To send a *one*, she sends a packet after sleeping for τ_2 units of time. On the receiving end (of the covert channel), Eve monitors the transmission line between Alice and Bob. Upon the observance of a network packet, Eve 1) records the inter-arrival time τ, 2) compares τ to both τ_1 and τ_2, and 3) records a *zero* or *one* depending on the previous comparison. Again by observing consecutive inter-arrival times, Eve records the entire binary string transmitted over the covert channel. Figure 3.2 illustrates a running example of a timing IP SCC. Our observations and assumptions in designing both types of IP SCCs are:

1. Alice and Eve use a single covert channel to communicate, thus our research does not address multiple channels that aggregate packet traffic to leak information.



We shall not spend
a large expense of
time Before we
reckon with your
several loves,
And make us even
with you. My
thanes and
kinsmen,
Henceforth be
earls, the first that
ever Scotland
In such an honour
named. What's
more to do,
Which would be
planted newly with
the time,
As calling home
our exiled friends
abroad
...

We shall not spend
a large expense of
time Before we
reckon with your
several loves,
And make us even
with you. My
thanes and
kinsmen,
Henceforth be
earls, the first that
ever Scotland
In such an honour
named. What's
more to do,
Which would be
planted newly with
the time,
As calling home
our exiled friends
abroad
...

Fig. 3.2. IP timing SCC. The example text is first encoded with a coding scheme and then sent bit by bit to the receiving end. The message is rebuilt by decoding the bit stream.

2. Alice has a ready supply of network packets that she can send to Bob at any given time and the channel protocol allows arbitrary sends from Alice (e.g., she sends HTTP `GET` requests with arbitrary content to Bob who is a web server with HIGH content).

3. Our research does not address the cases in which Bob might become suspicious as a result of the amount of network traffic generated by Alice.

4. The original payloads of the network packets transmitted over the direct channel between Alice and Bob are legitimate, hence do not violate any MLS policies. Further, the covert message sent over the indirect channel is independent of the original packet payloads.

5. The raw data that flows across the covert channel is binary but the actual interpretation of the binary string is up to Alice and Eve.

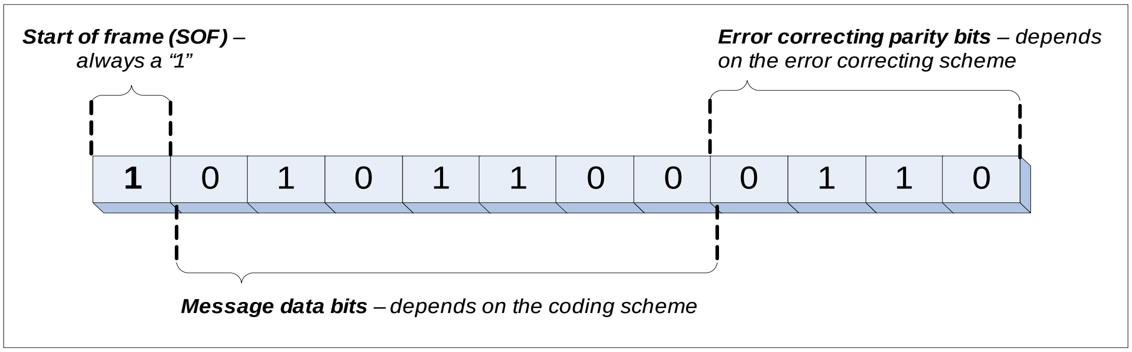

Start of frame (SOF) – always a "1"
Error correcting parity bits – depends on the error correcting scheme
1
0
1
0
1
1
0
0
0
1
1
0
Message data bits – depends on the coding scheme

Fig. 3.3. Components of an example IP SCC frame: Data, error correcting, and synchronization bits.

6. Additional bits may also be included in the transmission for three reasons: 1) Additional parity bits may be appended to the data to add redundancy for error correction caused by transmission errors (e.g., errors arising when a packet is lost/delayed), 2) additional bits may be added for purposes of maintaining synchronization between Alice and Eve's event clocks, and 3) the covert data may be encrypted to add a further layer of privacy and obfuscation. In our implementation, we employ error-correcting codes but not encryption.

7. The covert message for transmission is subdivided into smaller blocks of binary data, referred to as *frames* in this paper. An example frame consists of data bits, synchronization bits, and error-correcting bits. Figure 3.3 shows a representation of a frame. While all the frames are of equal length, the actual length, as well as the interval between frames, is influenced by parameters of the encoding scheme and the network. In our implementation, frames are composed of the eight-bit ASCII encoding of each character, a start of frame (SOF) bit (only for storage IP SCCs as we discuss in Section 3.1.2), and an optional six

bits for error correction using (7-4)-Hamming codes. We explain the use of the SOF bit and detail the use of Hamming codes in Section 3.1.2.

8. We assume a unidirectional communication model for the covert channel. Only the indirect covert channel is assumed to be unidirectional; The direct channel itself is still bidirectional and the TCP/IP packets are ACKed. Assuming a unidirectional channel means that Eve cannot communicate with Alice using the covert channel itself. Restricting the channel to be unidirectional increases the difficulty in implementing an error-free channel. In detail, Eve cannot 1) acknowledge the correct receipt of covert bits, 2) rate-limit Alice, or 3) indicate when to resynchronize.

3.1.2 Channel Implementation

We implemented our covert channels using the C Berkeley socket library for our communication protocol, and Python version 2.3 to encode/decode the data transmitted over the channel and as a wrapper that called the C library functions. The software was developed for and ran under Red Hat Linux 9.0 kernel version 2.4.22.

For simplicity, our IP SCC prototype transmits packets from Alice to Eve directly. This scenario apparently violates policy because such direct communication is not allowed in an MLS system. However, we argue that the traffic pattern generated by this channel is the same as the packet traffic that would be generated in any scenario described in Appendix A. This is because in this channel the traffic pattern is shaped by the covert message and the timing interval, not by the direction of the communication (i.e., Alice shapes the traffic and Eve has no effect on it). Further, we argue that in terms of channel bandwidth, our IP SCC prototype provides an upper limit. This is because in any other scenario in which Eve is eavesdropping, Eve can only lose information caused by an error in the observation. Further, because the traffic pattern remains the same, the detection schemes we introduce in Sections 3.3 and 3.4 that monitor the channel for anomalous traffic patterns will be unaffected.

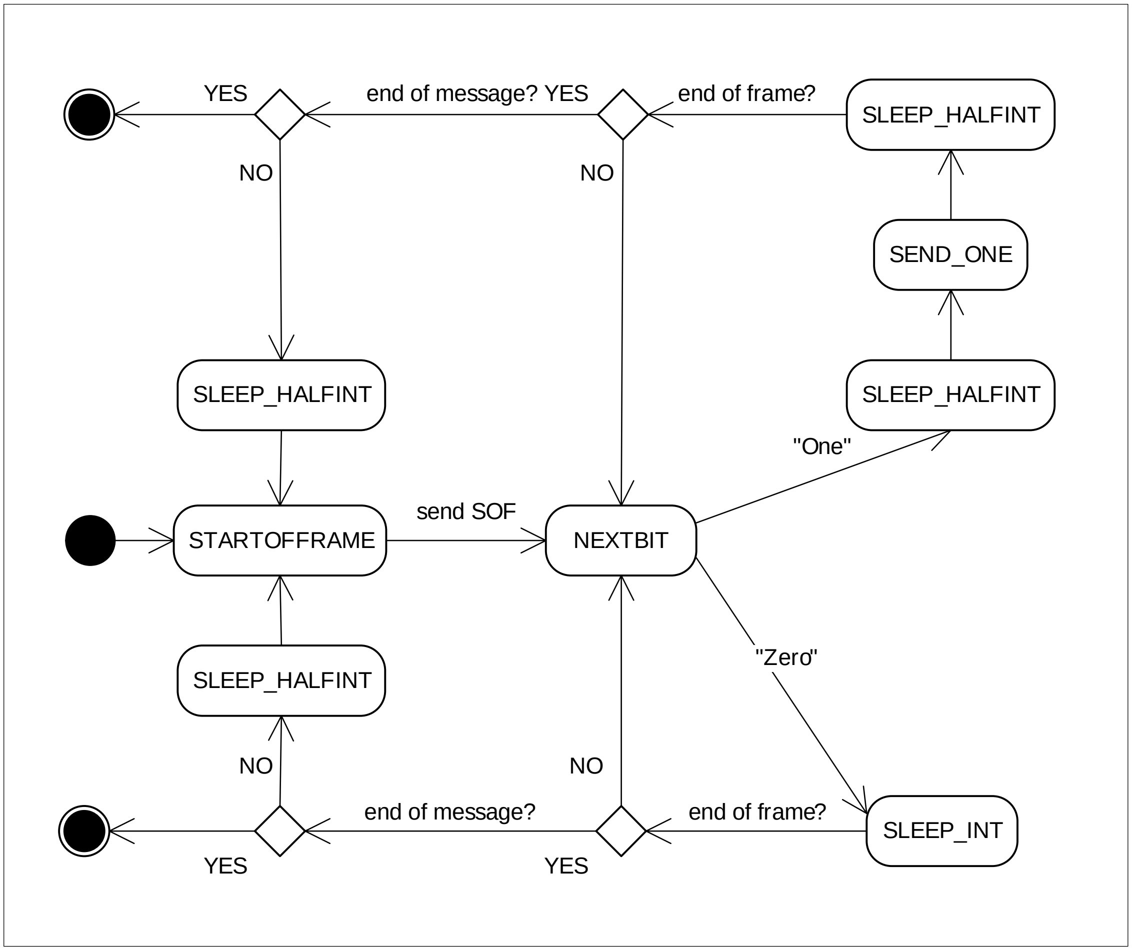


Fig. 3.4. Alice's execution flow diagram for a storage IP SCC.

The channel implementation differs slightly for the storage and timing IP SCCs. The storage IP SCC software uses both blocking and non-blocking Berkeley sockets and alternates between them to transmit a covert message to Eve. In detail, Eve uses a blocking socket to wait for the SOF packet to arrive at the beginning of each frame. Once this packet is observed, she switches to a non-blocking socket to record the remaining bits of the frame and reverts back to the blocking socket to wait for the SOF packet of the next frame. We illustrate the execution flow of Alice and Eve for a storage IP SCC in Figures 3.4 and 3.5. Given an encoded message, Alice sends a packet in the middle of the timing interval for each *one*, and stays

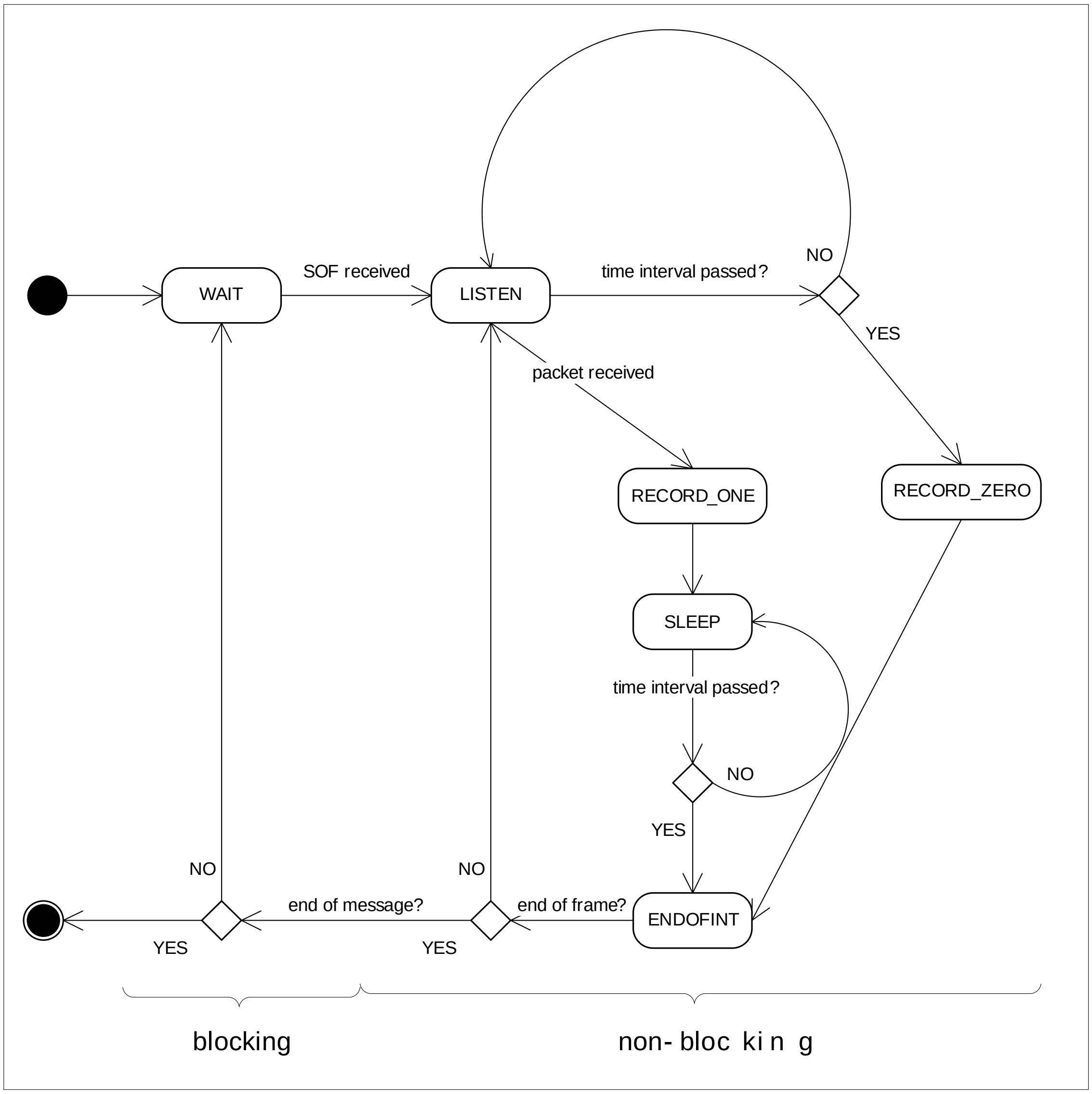


Fig. 3.5. Eve's execution flow diagram for a storage IP SCC.

silent for each *zero*. Before sending the data bits of a frame, Alice sends the SOF denoting the beginning of frame. After Eve receives the SOF bit, the blocking socket is resolved and Eve switches to non-blocking mode. During non-blocking mode, Eve's receiver program continues its execution and checks whether a packet arrives during the timing interval or not. If Eve receives at least one packet during the interval, a *one* is recorded. Otherwise a *zero* is recorded and Eve advances to the next interval

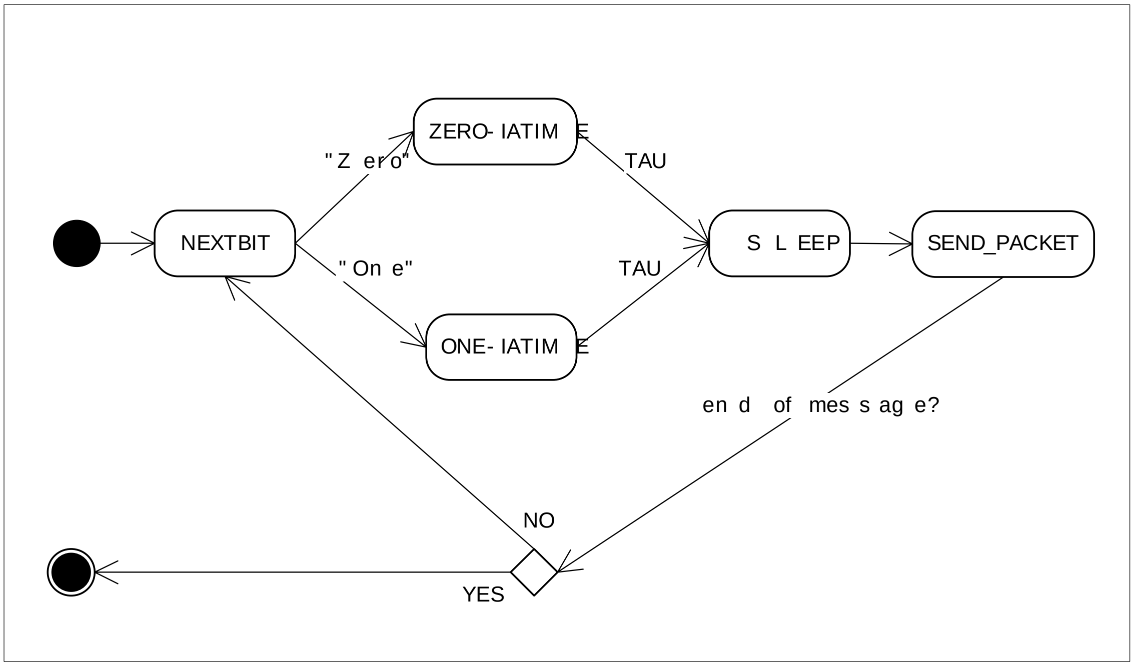


Fig. 3.6. Alice's execution flow diagram for a timing IP SCC.

until the necessary number of bits for a frame is received. Once a frame is completed, Eve switches back to the blocking mode and listens for the next SOF. Because the blocking socket pauses and waits for SOF in between two frames, this implementation provides an inherent synchronization. However, this scheme itself does not entirely solve the synchronization problem because of network congestion and jitter, and we discuss various other solutions for synchronization problems in Section 3.2.1.

In contrast, the timing IP SCC software is much simpler. The main difference from the storage channel is that in timing channels, the events (i.e., packet arrivals) are tied to the symbols (i.e., the bits), whereas in storage channels the events are tied to intervals. An advantage of this scheme is that we no longer need non-blocking sockets to wait for the silent intervals because each event is explicitly signaled by a packet arrival. Therefore, timing IP SCCs use a single blocking socket to record the packet arrival times and do not employ an SOF bit. The latter is because the implementation of timing IP SCCs is more straightforward and is less prone to timing

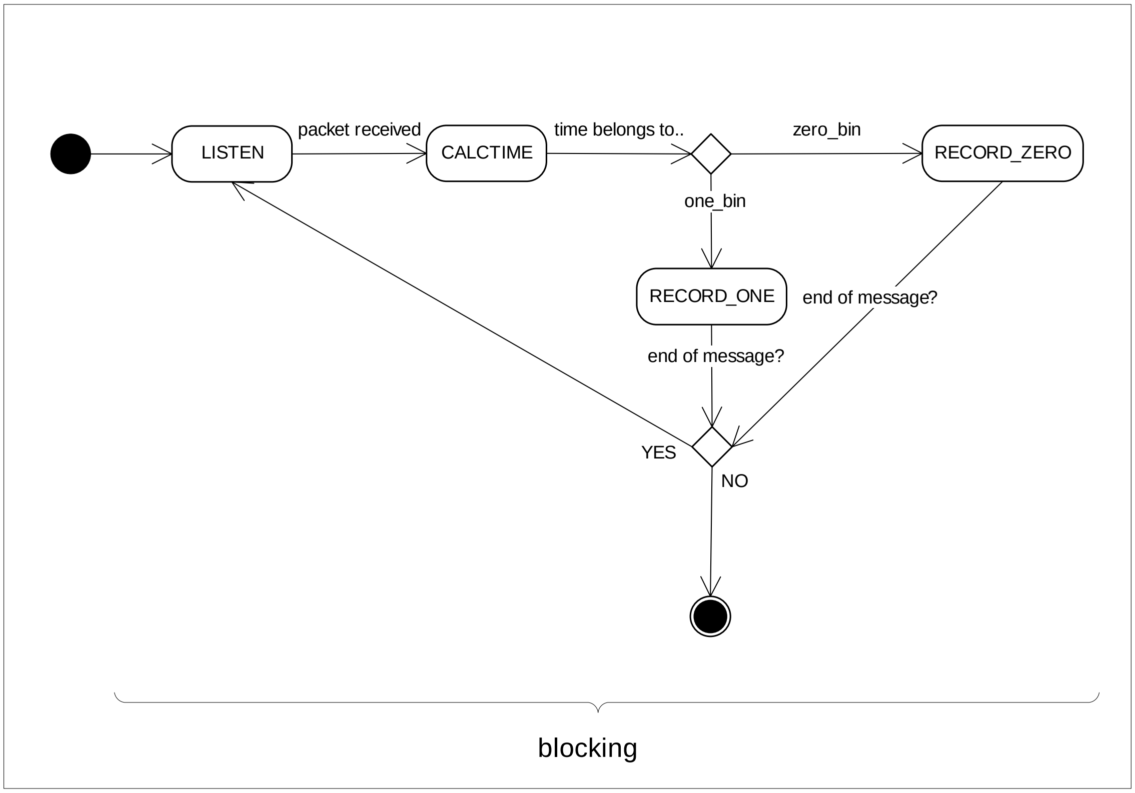


Fig. 3.7. Eve's execution flow diagram for a timing IP SCC.

(i.e., synchronization) errors as we discuss in Section 3.2.1. We illustrate the execution flow of Alice and Eve for a timing IP SCC in Figures 3.6 and 3.7.

Both IP SCCs can be configured to run on any application port. Because the traffic pattern is expected to vary based on the application, choice of the protocol in which to hide the channels can affect detection ability. For example, as we show experimentally in Section 3.3.2, UDP traffic generates patterns almost as regular as covert channel traffic caused by its best-effort nature. Therefore, running the covert channel software on a UDP port will hide the channel better than running it, for example, on the Telnet port.

3.2 Covert Channel Analysis

We assess the efficacy of IP SCCs in terms of bit/character accuracy (the number of bits/characters transmitted correctly divided by the total number of bits/characters) and channel bandwidth. Two factors affect channel accuracy and bandwidth:

1. *Contention noise* is the amount of non-covert traffic Eve observes in the covert channel and can potentially reduce channel accuracy [9].

2. *Clock skew* is the amount of jitter in the network and can potentially result in the loss of synchronization between Alice's and Eve's event clocks.

In this section, we investigate the effects of these factors on channel accuracy and bandwidth, and present the results of our empirical study with different choices of timing intervals.

3.2.1 Channel Accuracy

To evaluate the channel efficacy alone and provide an upper bound on character accuracy, we investigate noiseless IP SCCs. We assume that IP SCCs are free of contention noise, hence Eve is capable of isolating the network packets generated by Alice and destined to Bob from the ones originating from or destined to other users in the network. In this scenario, clock skew is the only factor that affects channel accuracy. IP SCCs leak information using the timings of packets generated by Alice and observed by Eve. In distributed systems, there may be a delay between the time a packet is generated and it is observed. If the delay is constant, Eve can simply adjust the packet clock to compensate for this delay. However if the delay is varying, that is the time that a packet *travels* from Alice to Eve jitters, the inter-arrival times observed by Eve are no longer exactly the same as the ones generated by Alice. This effect is called clock skew. Clock skew in a network is mostly caused by varying network conditions that result in slight increase or decreases in the latency between

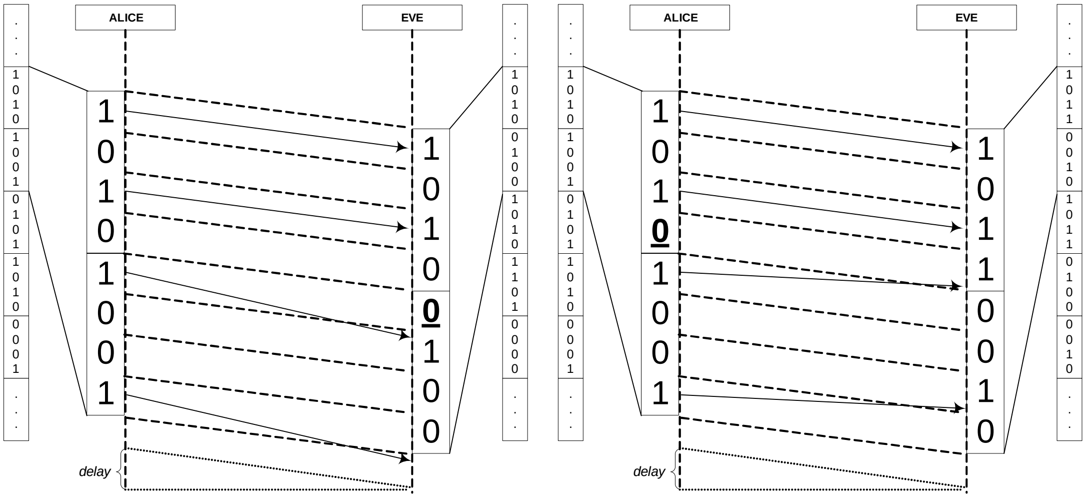


Fig. 3.8. A *zero*-insertion and a *zero*-deletion example for a storage IP SCC.

Alice and Eve (also known as network jitter). This can potentially affect the timing of network packets traveling from Alice to Bob (Eve) and thus reduce channel efficacy.

Channel Synchronization

An important problem for IP SCCs related to clock skew is the synchronization of Alice and Eve's packet clocks. For example clock skew can cause Eve to record packets as arriving in a time interval before or after the intended one. For a storage IP SCC, this skew can result in an individual bit flip, a *zero*-insertion, or a *zero*-deletion as we illustrate in Figure 3.8. The individual bit-flips can be corrected using error-correcting codes; however, insertion and deletion errors can potentially cause an entire series of transmission to be shifted. *One*-insertions and *one*-deletions are not possible for IP SCCs. Further, insertion and deletion errors only apply to storage IP SCCs. This is because in timing IP SCCs, each network packet has a one-to-one correspondence with a single bit of information, whereas in a storage IP SCC a packet delay can cause additional *zero*s to be inserted. Therefore, in timing IP SCCs bit errors are independent from each other and are not carried to the next bit as in the storage case, in which an insertion can potentially shift the entire bit-string. This advantage over the storage IP channels makes the timing channel more suitable for one-bit error-correction which we discuss next.

One way to remedy clock skew is to use error-correcting codes (ECC) to correct the individual bit errors. ECCs add additional bits to the message that can be used to detect and correct some errors in transmission [77, 78]. The code rate of an ECC is defined as the ratio between the length of an original message word and the length of its codeword. A drawback of ECCs is that they increase the number of bits to be transmitted. In our study, we employed *Hamming codes*, which are computationally simple to encode and decode and are capable of correcting one error per codeword [78]. To increase accuracy, one could also employ more sophisticated ECCs such as the *Reed-Solomon codes* or *turbo codes* [77]. Another scheme one could also use for storage channels would be to employ coding schemes that take advantage of the lack of *one*-insertions and *one*-deletions.

An additional scheme we employed to prevent synchronization errors for storage IP SCCs is to use an SOF bit at the beginning of each frame and to switch to a blocking socket every time a frame is completely transmitted. This scheme isolates transmission errors within individual frames and stop the propagation of errors to the following frames by aligning Alice and Eve's timing windows each time an SOF bit is transmitted.

We enhanced the SOF scheme by introducing *silent intervals* between frames. During a silent interval, no packet transfer occurs between Alice and Eve, thus the channel recovers from the errors that occurred before this period. We assume that the parties have previously determined the length of the silent interval. This interval can either be a default value or the covert channel itself can be initially used to send this value before the actual data transfer starts. Alice can enter the silent state at any time during the transmission. However, she has no way of knowing whether Eve received the covert bits correctly or not (i.e., the covert channel is unidirectional). Therefore, it is up to Alice to observe the changing network conditions and make the decision about when to pause the transmission. As an example, a sudden change in the round-trip time (RTT) between Alice and Eve might be a good signal that Alice should enter the silent state. On the receiving end, Eve simply waits for the arrival of the SOF packet (i.e., the first packet that arrives after a string of possibly erroneous *zero*s) and takes no action. A simpler option is to enter the silent state periodically to clear the channel. This method increases channel accuracy at the expense of transmission rate.

Rather than slow down the transmission by introducing silent periods in which no transfer occurs, the channel could adapt to the changes gradually as the network conditions change. In our *interval adjusting* scheme, Eve closely monitors the time each packet arrives and compares it to the projected ideal case (i.e., the expected arrival time of the next packet) based on the current timing interval. Comparing the two, a δ is computed, which is the deviation between the ideal and actual times and can be positive or negative depending on whether the packet has arrived early or late. Eve then simply adds this value to the timing interval and adjusts the clock for the next arriving packet. This scheme is most useful when there is an incremental change in the network conditions that persists for longer than the lifetime of a single packet. It can however lead to errors if the change in the network delay is greater than 50% of the timing interval (e.g., adjust to an incorrect timing interval). As a precaution, we restrict the maximum adjustment to be less than 10% between two consecutive intervals.

More advanced solutions for combating synchronization errors caused by clock skew are self-synchronizing codes and phase-locked loops (PLL). The former method employs encoding schemes that are specifically designed to detect the loss of synchronization and to recover from this state (i.e., resynchronization). The latter method is a closed-loop feedback circuit that is designed to track or synchronize an output signal with an input signal in frequency and phase [79]. Both methods were proved to be effective in communications and could be applied to our channels to achieve better accuracy.

3.2.2 Channel Bandwidth

Ideally, using the shortest timing intervals possible yields the highest data rate. However, this rate is limited by three factors:

1. Alice's packet generator is the bottleneck as she cannot generate packets at a higher rate than the generator's rate. Another limiting factor on Alice's side is the precision of the packet delayer. As we use shorter timing intervals, the delayer routine may fail to *sleep* precisely for the required amount of time. The same constraints also apply to Eve's observer routine.

2. In networks with jitter, the negative impact of clock skew on channel accuracy increases as we use shorter timing intervals. This is because with short timing intervals a small variation on the interval may alter its symbol association (i.e., place it in a different partition) whereas the same variation may be negligible for long intervals.

3. Timing intervals that yield high data rates may be anomalous in comparison to normal traffic in the network.

3.2.3 Empirical Evaluation

To assess the efficacy of the covert channel experimentally, we used our implementation of a storage IP SCC. Our covert channel ran between Purdue and Georgetown university networks and was subject to changing network conditions. During *normal* network conditions, the route between communicating parties was twelve hops with an average RTT of 31.5 msec. For one-bit error-correction, we used Hamming codes that are computationally simple to encode and decode [78]. To calculate bit and character accuracy we used an implementation of the Levenshtein distance (i.e., the edit distance [80]). IP SCCs are capable of sending both binary and text data and we used ASCII encoding for the latter. In both cases, the software divided the

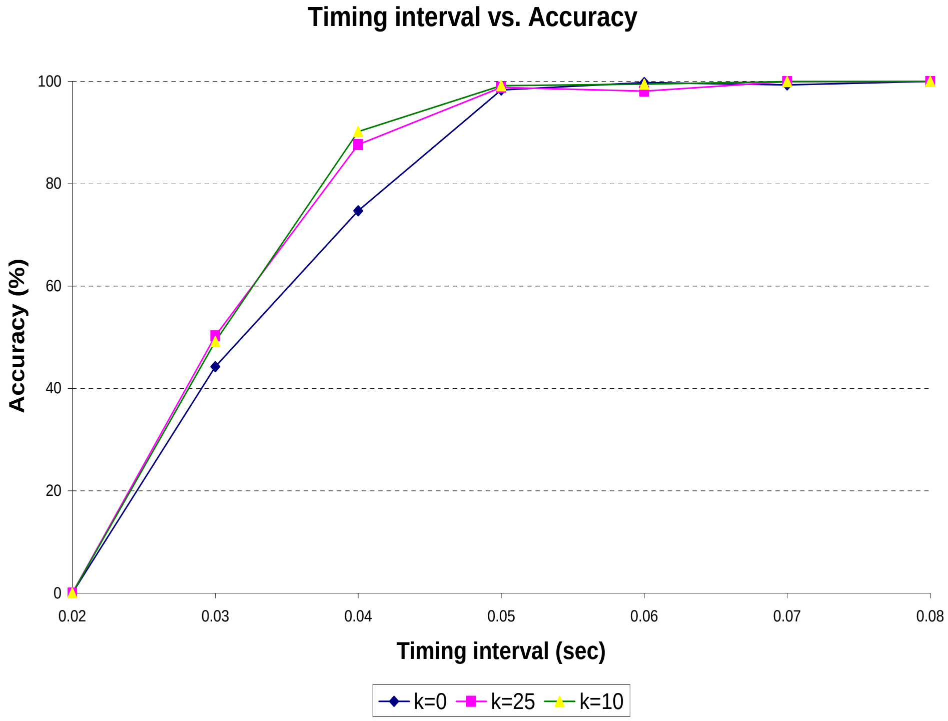


Fig. 3.9. Timing interval versus accuracy with different values of k for the silent interval synchronization scheme.

resulting bit-string into eight-bit codewords that were expanded to 14 bits when we used (7,4)-Hamming error-correction.

Effect of timing interval size

We first investigated the potential data rate of our channel by decreasing the timing interval until the accuracy drops. We mark this point as a threshold that can be thought as a boundary between lossless and lossy communication and calculate the corresponding channel bit and character rate. In this experiment, we employed a storage IP SCC channel with the periodic silent intervals synchronization scheme described in Section 3.2.1, with k denoting the frequency the synchronization scheme goes into a silent period (e.g., every twenty timing intervals). The character coding is eight bit ASCII with no error correction. We report the trade-off between the timing interval and the channel accuracy in Figure 3.9. Our channel provided nearly lossless communication for larger intervals at the cost of lower transmission bandwidth.

The results show that the threshold value for the timing interval is around 0.06 seconds and intervals greater than 0.06 guarantee nearly 98% character accuracy for all three values of k. The equivalent bit transfer rate is 16.666 bits per second (bps). With ASCII encoding and the SOF bit taken into account, we calculate the character transfer rate 2.083 characters per second. As expected, the channel accuracy remains high for larger timing intervals. It also remains slightly higher when the transmissions are periodically paused for resynchronization.

Effect of network conditions

We additionally investigated a case with network congestion and its effects on the performance of the covert channel. To do so, we ran our covert channel on a congested network with a highly varying RTT between Alice and Eve with mean RTT at 42.07 msec. The normal RTT values for this channel showed a steady behavior with mean RTT at 31.5 msec. Our evaluations showed that the congestion results in a lower accuracy rate. For example, with the timing interval set to 0.08, we observed 100% average character accuracy under normal conditions, but the accuracy dropped to 82.11% for the congested network. Clearly, the interval must be increased to retain accuracy during periods of high congestion.

3.2.4 Storage or Timing?

In designing IP SCCs, we investigated storage and timing channels separately and noted subtle differences between them. Despite these differences, storage and timing IP SCCs are related constructs and, for example, given a storage channel, one can provide a reduction to show that an equivalent timing channel can be constructed and vice versa. To see this, consider a timing IP SCC with a single state (response) packet_observed and timing intervals τ_1 and τ_2. Using this channel, we can construct a storage IP SCC with states packet_present and packet_absent (both derived from the single state packet_arrival), and $\tau = \tau_1 + \tau_2$. Similarly, given a storage IP SCC with timing interval τ, we can construct a timing IP SCC with $\tau_1 = \tau$ and $\tau_2 = 2\tau$.

Additionally, we claim that if we can detect a storage IP SCC that transmits a bit-string over the network, we can detect the timing equivalent of the same channel. As we present in the next two sections, our detection methods use measures that capture the regularity of packet inter-arrival times generated by these channels. Consider a storage IP SCC with timing interval τ that generates k different inter-arrival time values, where k is equal to the number of different number of *zero*s we can have between two *one*s in a codeword (e.g., for the set $\{11, 101, 1001\}$, $k = 3$). In contrast, the timing equivalent of this IP SCC uses two timing intervals τ_1 and τ_2 that generates only two clusters of inter-arrival times, thus generating a more regular inter-arrival time pattern. Therefore, we concentrate only on storage IP SCCs in covert channel detection and conjecture that detection measures that work for these channels would also work for timing IP SCCs.

3.3 Noiseless Channel Detection

For a storage IP SCC, there must be a pre-specified timing interval between consecutive network packets sent by Alice and observed by Eve. Similarly for a timing IP SCC, there must be pre-specified timing intervals. This behavior implies regularity in terms of packet inter-arrival times in comparison to packet inter-arrival times generated by a legitimate process and we argue that the packet traffic generated by IP SCCs is highly regular (thus anomalous) in comparison to the traffic generated by legitimate processes (e.g., as compared to WWW traffic). To show an example, we plot the inter-arrival times generated by both channels while sending a sample text[1] from Alice to Bob (Eve) in Figure 3.10. For the storage IP SCC, we observe that the

[1] "All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.[EOL]" [81]

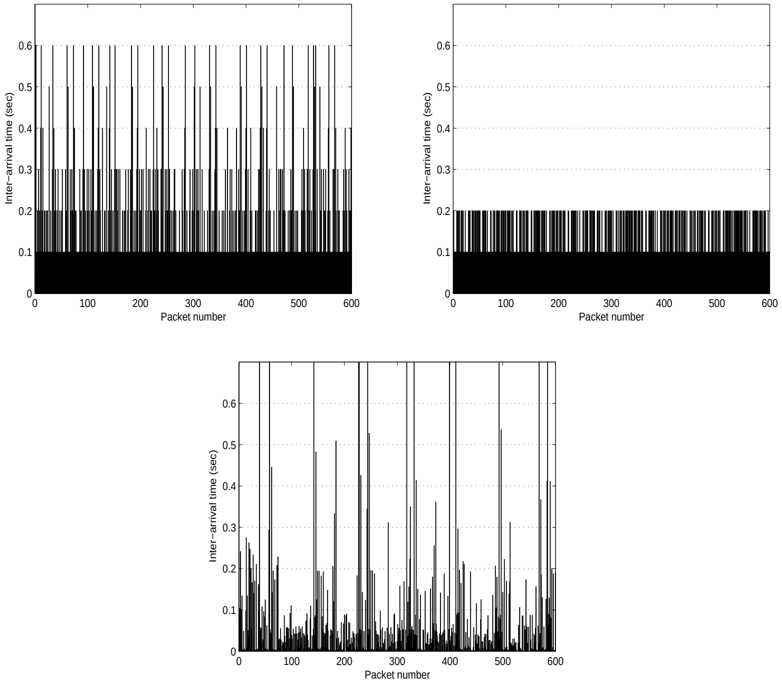


Fig. 3.10. Inter-arrival times for different traffic types. (a) A simulated IP storage SCC with $\tau = 0.2$ secs (no clock skew). (b) A simulated IP timing SCC with $\tau_1 = 0.1$ secs and $\tau_2 = 0.2$ sec (no clock skew). (c) An example NZIX-II WWW traffic.

packet inter-arrival times can be grouped into six clusters. Similarly, the inter-arrival times for the timing IP SCC can be grouped into two clusters. Figure 3.10(c) shows that no such clustering is obvious for a sample WWW traffic taken from the NZIX-II dataset [10].

To identify this type of a covert channel requires being able to capture such regularity. In this section, we introduce two measures to do so and empirically evaluate the efficacy of both in terms of Type I (false positive (FP)) and Type II (false negative (FN)) error rates.

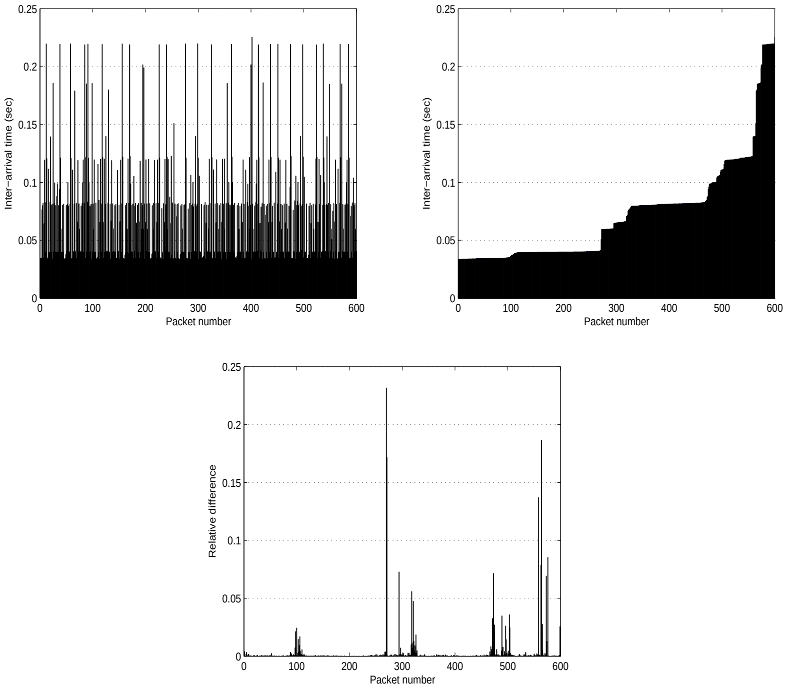


Fig. 3.11. Inter-arrival times from an actual run of a storage IP SCC with $\tau = 0.04$ secs. (a) Actual values. (b) Sorted values. (c) Relative differences.

3.3.1 Measures for Detection

To capture the type of regularity we described above, we introduce two detection measures: ϵ-similarity and compressibility.

ϵ-Similarity

To derive our first similarity measure, we sort the inter-arrival times we collected using IP SCCs. From this sorted list we compute the relative difference between each pair of consecutive points. For example, the relative difference between P_i and P_{i+1} is computed as $\frac{P_{i+1}-P_i}{P_i}$. We illustrate this process in Figure 3.11 for a compilation of inter-arrival times we collected using our implementation of a storage IP SCC we ran between Purdue and Georgetown universities. As a result of network jitter that created clock skew, the inter-arrival times no longer create perfect clusters as in Figure 3.10. From the sorted inter-arrival times we compute an efficient measure of similarity, which we call *ϵ-similarity*, by computing the percentage of relative differences that are less than a constant number ϵ. Our claim is that for noiseless IP SCCs, the majority of the pairwise differences in the sorted inter-arrival time list will be small. (It is large only for jumps in the step function.) Thus, the similarity scores for IP SCCs will be high. We further claim that the IP SCC ϵ-similarity scores will be comparably higher than that of legitimate channel scores.

Compressibility

The Kolmogorov complexity of a string S is defined as the shortest universal computer program that produces this string (see [82]). Denoted as $K(S)$, this notion provides a lower bound on the representation of the string and computes the maximum available compression on S, hence producing the shortest possible compressed string C. For an arbitrary string S, Kolmogorov complexity is not a computable [82]; however, off-the-shelf compression algorithms can be used as an approximation [83].

To derive our second similarity measure, let $S \in \Sigma^s$ be a string we want to compress with any off-the-shelf compressor, where Σ is the alphabet of symbols from which the string is drawn, and s is the length of S. Let the resulting compressed string be $C \in \Sigma^c$. We define the *compressibility* of S as the compression rate we obtain by dividing the length of S by the length of C. Formally,

$$\forall S \in \Sigma^s, \exists C \in \Sigma^c \text{ such that}$$

$$C = \Im(S), \text{ and } \kappa(S) = \tfrac{|S|}{|C|}$$

where $\Im$ is the compressor (e.g., `gzip`), $\kappa(S)$ is the compressibility of S, and $|.|$ is the length operator. Our claim is that the compressibility of the inter-arrival packet times generated by a noiseless IP SCC will be higher than the ones generated by legitimate channels. Inter-arrival times are numerical values whereas compression works on strings. In Section 3.3.2, we detail how we convert these numbers to strings as a pre-processing step to compression.

A Discussion of Other Approaches

We additionally investigated several approaches that were not fruitful, but were more obvious from a statistical point of view.

Indexes of dispersion of a point process have been used as a tool in network characterization [84, 85]. An *index of dispersion for intervals (IDI)* can be used to qualitatively compare the inter-arrival times of a point process with the Poisson process serving as the basis (for which the IDI is unity) [86]. IDI provides a finer measure for defining the *variability* of the process than does a second order moment analysis. Gusella defines the variability, or the burstiness, of the network traffic as "the changes in the variance of the sum of consecutive inter-arrivals" [84]. Although this measure appears promising, it requires a number of assumptions including *stationarity*, that need to be made for the correct interpretation of the results. In this study, we do not impose such assumptions on the distributions of covert or legitimate traffic.

Another avenue we examined was statistical non-parametric tests similar to those used in network traffic analysis [87–89]. Applications of these tests have mainly concentrated on network traffic characterization and modeling. The goal is often to determine whether two streams come from the same empirical distribution, for example, using a non-parametric goodness-of-fit test such as the Kolmogorov-Smirnov test. In our research, we are not seeking to model either the legitimate or the covert network traffic. Our goal is to define measures that differentiate covert from legitimate traffic, therefore, these methods are not directly applicable to the detection of IP SCCs.

3.3.2 Empirical Evaluation

In this section, we empirically show that both ϵ-similarity and compressibility measures are able to detect IP SCCs, and that compressibility performs better than ϵ-similarity. In our experiments, we applied these measures on both legitimate channels and IP SCCs. Our legitimate traces were extracted from the second version of NZIX datasets (NZIX-II) which is a collection of TCP (WWW, FTP-Data, and SSH) and UDP traces collected by the WAND research group [10]. We used twenty different traces for each legitimate traffic type and each trace contained a sufficient number of packets that we could run the covert channel over it without running out of packets. To collect the inter-arrival times for the covert channel, we ran three storage IP SCCs (with τ set to 0.04, 0.06, and 0.08 secs) that transmitted the sample text between Purdue and Georgetown universities.

For each trace, we first eliminated values larger than one second to eliminate large inter-arrival time values from the data that occur when there is no packet transfer. This allows us to deal only with short values and therefore concentrate on the network effects on the inter-arrival times rather than the user effects (e.g., time gaps between the HTTP requests). Our goal is not to model or identify a traffic distribution, but to determine whether we can accurately detect a covert channel in a short window (i.e., for on-line detection). Therefore, in our experiments we report the results for windows of size 2000. Although we ran the IP SCC on a real network between Purdue and Georgetown, for the legitimate traffic we used the recorded inter-arrival times in the datasets. A drawback is that we cannot have the same network conditions (e.g., number of hops, same jitter), but excluding the case of jitter, this does not impact our results. This is because none of our measures look at absolute inter-arrival time values, but rather compute measures of regularity in terms of the relative differences between these values.

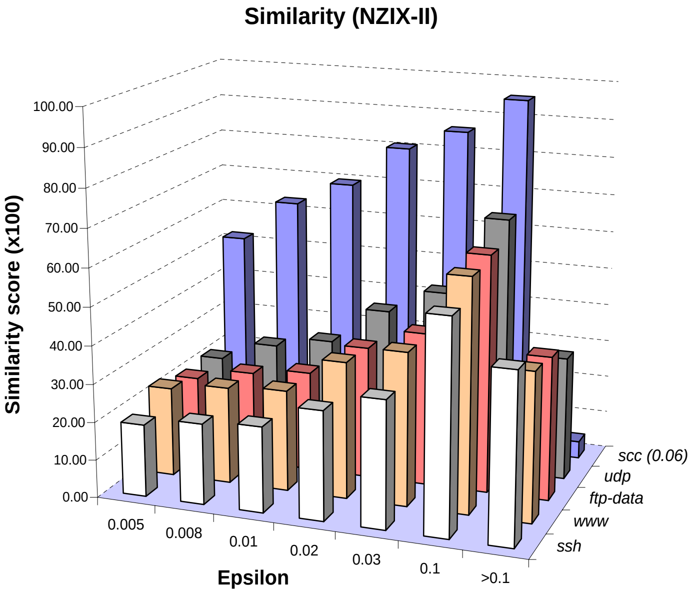


Fig. 3.12. ϵ-similarity scores for various types of covert and legitimate channels.

ϵ-Similarity Results

We report the results for our first measure in Figure 3.12 and Table 3.1. The x-axis shows seven different arbitrarily chosen ϵ values and the y-axis shows the mean ϵ-similarity score (averaged over twenty traces of each traffic type), that is the percentage of all pairs of sorted inter-arrival time values whose difference is less than ϵ. As anticipated, the results show a striking difference between the IP SCC and NZIX-II scores. For example, the 0.005-similarity score for the IP SCC with $\tau = 0.06$ is 52 while it is 21 for FTP-Data. Thus, on average 52% of the sorted IP SCC inter-arrival times relatively differ less than $\epsilon = 0.005$ while only 21% of FTP-Data differ less than the same ϵ. Another result we observe in Table 3.1 is that the higher τ we use for the storage channel (i.e., to get better accuracy), the higher the similarity scores. This implies that more accurate channels are easier to distinguish from normal traffic, hence they are easier to detect.

Table 3.1
ϵ-similarity scores for storage IP SCCs for various τ.

τ	0.005	0.008	0.01	0.02	0.03	0.1
0.04	39.97	52.88	58.63	72.84	79.84	91.90
0.06	52.08	63.28	69.68	80.74	86.24	95.60
0.08	64.88	75.19	78.54	87.89	91.55	97.50

Compressibility Results

Before applying the compressibility measure, we needed to convert our numerical dataset into a set of strings because compression works on strings. To do so, we first smoothed each dataset by taking two significant digits (and rounding the third digit), and added a letter at the beginning depending on the number of zeros after the decimal point. For example, we transformed 0.00247 to B25, and 0.0247 to A25, etc. This way, we could still identify the same values without using repeating zeros, which would create more compressible datasets[2].

After this added pre-processing phase, we performed our experiments using `gzip` as the compressor [90]. We report the mean compressibility score for each traffic type in Figure 3.13 averaged over twenty traces for the legitimate channels. Our results show that the compressibility scores for the SSH, WWW, and FTP-Data datasets are much lower than IP SCC compressibility scores. For example, the compressibility score for the storage IP SCC with $\tau = 0.06$ is 6.76 while it is 3.27 for WWW on average. In addition, we observe that as we increase the timing interval for IP SCC, it becomes more compressible. For example, the compressibility score for a 0.04 IP SCC is 5.76, whereas it is 7.01 for 0.08. We conjecture that the more accurate the covert channel, the more compressible its dataset.

[2] Even though B25 and A25 compress better than B25 and B31, this effect is evenly distributed among all datasets, whereas repeating zeros favor datasets with data points with small magnitudes.

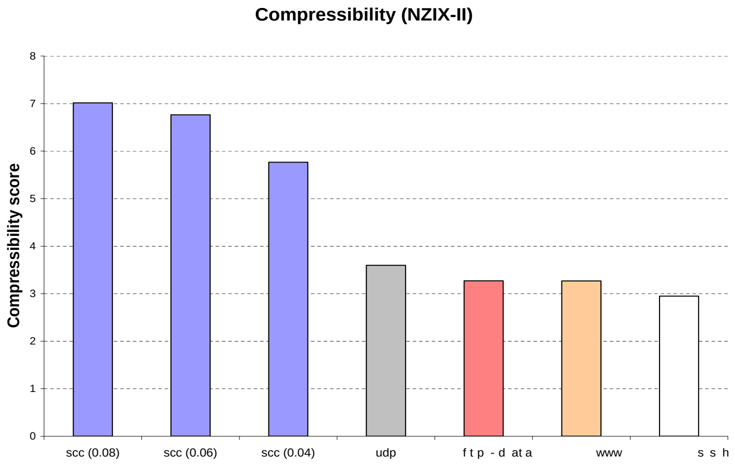


Fig. 3.13. Compressibility scores for various types of covert and legitimate channels.

Automatic Detection

To assess the efficacy of channel detection experimentally, we evaluated the efficacy of both measures in terms of FN and FP rates. Our procedure worked as follows: For each legitimate traffic type (WWW, FTP-Data, and UDP[3]), we randomly chose ten traces out of twenty for training. We reserved the remaining ten *independent* traces for testing. In the training phase, we calculated the ϵ-similarity or compressibility scores for each training trace and computed the mean μ and the standard deviation σ to determine our thresholds $\mu + 1.5\sigma$ and $\mu + 2\sigma$. The main difference between ϵ-similarity and compressibility was that in the former we calculated seven different learned thresholds for each of seven ϵ values (0.005, 0.008, 0.01, 0.02, 0.03, 0.1, and $<$ 0.1) that were chosen arbitrarily. For compressibility we calculated only one threshold.

To classify a test trace as legitimate or covert, we calculated its ϵ-similarity (for each ϵ value) or compressibility score that we compared to the learned threshold. A dataset was classified as covert if its combined score was above the learned threshold and legitimate otherwise. For ϵ-similarity, the combined score was determined by

[3] We omit the SSH results because NZIX-II did not contain enough SSH traces for training and testing.

Table 3.2
IP SCC detection efficacy for ϵ-similarity and compressibility with thresholds $\mu + 1.5\sigma$ and $\mu + 2\sigma$.

Threshold	Application	FP	FN(0.04)	FN(0.06)	FN(0.08)
ϵ-similarity					
$\mu + 1.5\sigma$	WWW	0.06	0.00	0.00	0.00
	FTP-DATA	0.04	0.00	0.00	0.00
	UDP	0.14	0.65	0.19	0.00
$\mu + 2\sigma$	WWW	0.04	0.08	0.00	0.00
	FTP-DATA	0.00	0.28	0.00	0.00
	UDP	0.08	0.90	0.66	0.38
Compressibility					
$\mu + 1.5\sigma$	WWW	0.14	0.00	0.00	0.00
	FTP-DATA	0.09	0.00	0.00	0.00
	UDP	0.12	0.19	0.00	0.00
$\mu + 2\sigma$	WWW	0.07	0.00	0.00	0.00
	FTP-DATA	0.03	0.00	0.00	0.00
	UDP	0.09	0.63	0.02	0.00

taking the majority vote over seven different ϵ values. For example, we classified the trace as covert if the scores for four out of seven ϵ values were over the threshold. For compressibility, the combined score was simply the compressibility score.

In our results, we report the averages of 100 runs of the above procedure. Each run used a different partition of the twenty legitimate traces into training and testing. We report the average FP and FN rates for the ϵ-similarity and compressibility measures in Table 3.2. We list our observations and conclusions as follows:

1. The results suggest that we can distinguish IP SCC traces from WWW and FTP-Data traces (i.e., FN = 0%) with low false alarm rates using both ϵ-

similarity and compressibility measures with thresholds $\mu + 1.5\sigma$ and $\mu + 2\sigma$ respectively.

2. UDP results with high FN rates suggest that we cannot distinguish IP SCC traces from UDP traces with a reasonable false alarm rate. UDP is a best-effort protocol in which the packets are sent as soon as they are ready (i.e, there is no flow control). Therefore, UDP inter-arrival times tend to be regular under stable network conditions, and thus generate higher ϵ-similarity and compressibility scores as compared to TCP inter-arrival times.

3. The compressibility measure fares better than the ϵ-similarity measure in terms of the FN rates with comparable FP rates. Further, compressibility uses only one measurement whereas ϵ-similarity requires a majority vote over seven measurements. Therefore, we conjecture that compressibility is a superior measure.

4. As claimed before, IP SCCs with higher τ yield higher detection rates. This is because using higher timing intervals yields more accurate channels which generate more regular inter-arrival times as a result.

5. We claim that the FP rates for both measures can potentially be lowered by increasing the number of datasets we use in the training phase. Using only ten datasets for training was an experimental short-coming as we were limited by the available legitimate data. In a real detection environment, hundreds of such traces can be used to train the classifier.

3.4 Noisy Channel Detection

A noisy IP SCC is a network covert channel in which we observe both legitimate and covert traffic. This channel may arise in cases where Eve cannot isolate the network packets originating from Alice as a result of the shared use of the transmission line that is being tapped. In addition, introducing noise into the channel artificially has been shown to be effective in jamming the covert channel and limiting its bandwidth [11]. In our study, we assume that Alice injects noise into the transmission line intentionally to increase the irregularity of the packet inter-arrival times and thwart the regularity-based detection schemes.

In this section, we first empirically show that if channel noise reaches a certain level, the regularity-based detection measures fail to identify the covert channel and Alice can potentially hide the covert channel by mixing it with legitimate traffic. Given this observation, we argue that to detect such hidden IP SCCs, we need more *localized* measures to focus on the suspicious segments of the dataset. To do so, we employ compression-based similarity measures accompanied with a sliding window algorithm to search for covert channels in mix datasets. We conclude the section with an empirical study to evaluate the efficacy and performance of our methods.

3.4.1 Empirical Evaluation with Noiseless Measures

In this section, we show that regularity-based measures fail to detect noisy IP SCCs with reasonable false alarm rates for noise levels higher than 10%. We apply the ϵ-similarity and compressibility measures to mix datasets that we formed by mixing packet inter-arrival times extracted from WWW, FTP-Data, and UDP datasets with the ones generated by IP SCCs. As an example, to create a 10% FTP-Data mix dataset with 2000 data points, we picked 200 consecutive data points from a FTP-Data dataset and inserted it as a block within 1800 data points we selected from the IP SCC dataset at a randomly selected location. This scheme resembles the cases in which we apply our measures on a larger window than the duration of the covert communication. Actually, how we mix these data points does not matter for our detection measures because neither measure pays attention to the relative ordering of the individual data points. This is because both measures rearrange the data points as a pre-processing step (e.g., the ϵ-similarity measure sorts the data points).

In our experiments, we used the same setup and datasets as in Section 3.3. We report the average scores for the ϵ-similarity measure for each mix dataset with noise

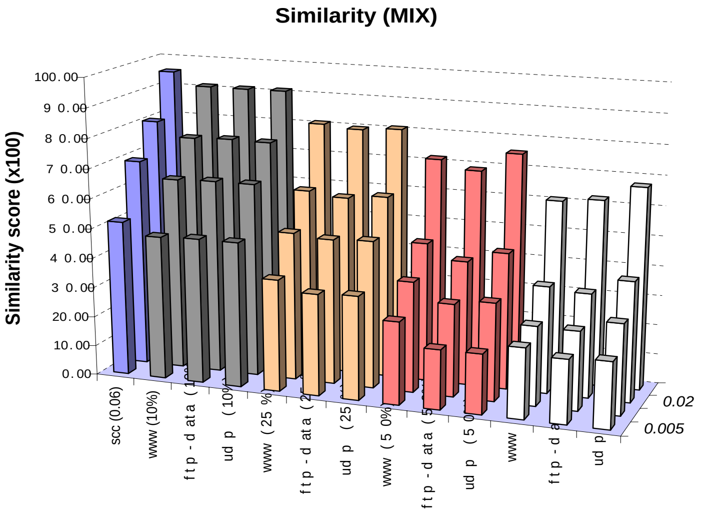


Fig. 3.14. ϵ-similarity scores for a noisy IP SCC with different traffic types and noise levels set at 10%, 25%, and 50%.

levels at 10%, 25%, and 50% in Figure 3.14. The ϵ-similarity scores for the mix datasets are comparably lower than the ones we calculated in Section 3.3. For example, the mean 0.005-similarity score for the pure IP SCC ($\tau = 0.06$) is 52 while a 25% FTP-Data mix dataset generates a score of 34.

The average scores for the compressibility measure show a similar trend which we illustrate in Figure 3.15. The compressibility scores for the mix datasets are substantially lower than the ones we calculated for the pure covert channels in Section 3.3. For example, the mean compressibility score for the pure IP SCC ($\tau = 0.06$ secs) is 6.76 while a 25% FTP-Data mix dataset generates a score of 4.98.

To determine how automatic detection with both measures would fare in the presence of noise, we repeated the automatic detection study we presented in Section 3.3 to compute the FN rates. We chose ten training traces out of twenty legitimate datasets to compute the mean μ and the standard deviation σ to determine our thresholds $\mu + 1.5\sigma$ (for ϵ-similarity) and $\mu + 2\sigma$ (for compressibility). We reserved the remaining ten *independent* traces for testing. We averaged the results over 100 runs each time

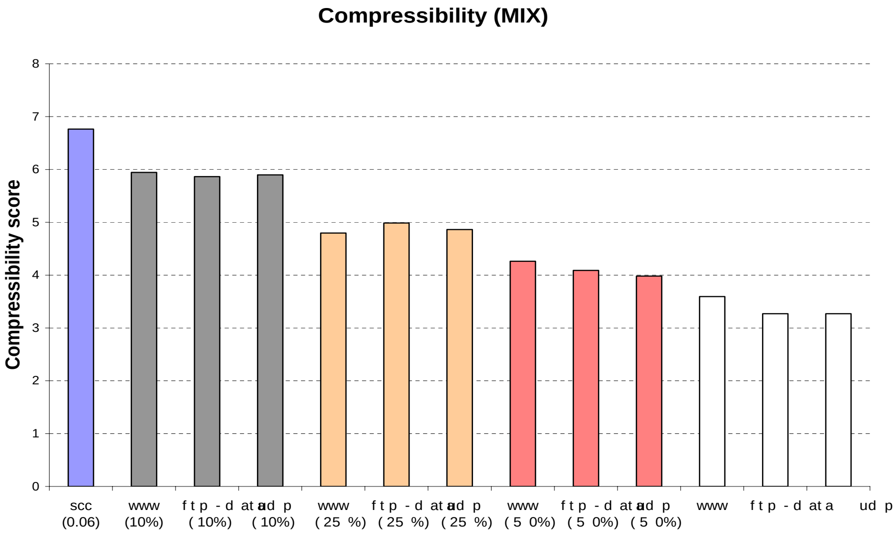


Fig. 3.15. Compressibility scores for a noisy IP SCC with different traffic types and noise levels set at 10%, 25%, and 50%.

using a different partition of the twenty legitimate traces into training and testing. We report the results in Table 3.3 and list our observations and conclusions as follows:

1. The results suggest that both measures can detect WWW and FTP-Data mix datasets up to 10% noise level. We conclude that our measures work for low noise levels.

2. Compressibility further detects WWW and FTP-Data mix datasets up to 25% noise level for which the ϵ-similarity measure completely fails. We conclude that compressibility is a more robust measure when detecting noisy IP SCCs.

3. Neither measure can detect 50% mix datasets with a reasonable false alarm rate regardless of the mix type and UDP mix datasets regardless of the noise level.

Table 3.3
Noisy IP SCC detection efficacy for ϵ-similarity and compressibility with thresholds $\mu + 1.5\sigma$ (ϵ-similarity) and $\mu + 2\sigma$ (compressibility).

Application	Noise	fn(0.04)	fn(0.06)	fn(0.08)
ϵ-similarity				
www	10%	0.00	0.00	0.00
	25%	0.76	0.23	0.00
	50%	1.00	0.98	0.80
ftp-data	10%	0.04	0.00	0.00
	25%	0.99	0.64	0.00
	50%	1.00	1.00	1.00
udp	10%	0.81	0.45	0.08
	25%	0.99	0.95	0.80
	50%	1.00	1.00	0.99
Compressibility				
www	10%	0.00	0.00	0.00
	25%	0.00	0.00	0.00
	50%	0.95	0.92	0.74
ftp-data	10%	0.00	0.00	0.00
	25%	0.52	0.00	0.00
	50%	1.00	0.96	0.95
udp	10%	0.71	0.48	0.22
	25%	0.94	0.90	0.84
	50%	0.99	0.99	0.99

3.4.2 Measures for Detection

The previous section suggests that our regularity-based detection measures alone fail to detect the presence of IP SCCs with reasonable detection rates given a mix dataset. To detect noisy IP SCCs, we take the approach of identifying and eliminating the noise from the mix dataset instead finding a measure that is robust in the presence of noise. To do this, we need a more *localized* methodology that focuses on the suspicious segments of the dataset individually and also in relation to each other. Therefore, we employ a sliding window methodology that traverses the mix dataset using windows of size w and strides of size s. The idea is to collect local information about each window individually, and also in relation to each other. To collect individual and relative information about each window, we introduce the *compressibility-walk* and the *CosR-walk* methods, respectively.

Once we identify the mixed dataset windows that are suspected to be generated by a covert channel, we can then eliminate the other windows and apply our measures only on the suspicious window. Because our regularity-based measures work effectively for low noise levels, this local approach can potentially generate higher detection rates as compared to the cases where we apply the measures globally.

Compressibility-Walk

Compressibility-walk traverses the mixed dataset in strides of size s and computes the compressibility score for each window of size w. The windows are allowed to overlap in cases where $s < w$ and we investigate cases in which w is much larger than the covert channel window (i.e., a noisy covert channel). Our claim is that because the compressibility scores of the datasets generated by IP SCCs are higher than the ones generated by legitimate channels, the windows that contain covert channels will reveal themselves as peaks when plotted on a graph. As an alternative, we could also devise an ϵ-similarity-walk methodology, but we omit this approach because as we showed in Section 3.3, compressibility is a superior measure as compared to ϵ-similarity, particularly in the presence of noise.

CosR-Walk

In the next section where we evaluate our sliding window approach empirically, we show that examining individual windows alone is not sufficient. Therefore we need to investigate the relation between two consecutive windows, and to do so we need a metric that measures the similarity between the two.

Compression can be used as a similarity metric, which is an approximation to conditional Kolmogorov complexity [91]. Given two strings S and T, the conditional Kolmogorov complexity $K(S|T)$ is defined as the shortest representation of S given T. Intuitively, the more information S and T share, the shorter the representation of S given T.

Compression-based similarity metrics have been used in various research areas [83, 92–96]. Let $\Im$ be a compressor that approximates Kolmogorov complexity. In [92], the conditional compression of S given T is approximated by the compression-based distance measure (CDM), defined as:

$$CDM(S,T) = \frac{\Im(S|T)}{\Im(S)+\Im(T)}$$

where | is the concatenation operator. CDM is a dissimilarity measure. In [91], the authors show that the $CosR$ metric performs better than CDM, which is defined as:

$$CosR(S,T) = 1 - \frac{\Im(S)+\Im(T)-\Im(S|T)}{\sqrt{\Im}(S)\Im(T)}$$

Combining this metric with our sliding window approach, CosR-walk traverses the mixed dataset in a similar fashion to compressibility-walk, only this time computing the CosR scores for consecutive windows. Again, we allow the windows to overlap. Our claim this time is that because CosR is a similarity measure, the score when one window has data points generated by a legitimate channel and the other generated by a covert channel will be lower than the score when both windows contain data points generated by a legitimate channel.

3.4.3 Empirical Evaluation

In this section, we empirically show that both compressibility-walk and CosR-walk were successful in identifying hidden covert channels in mix datasets, although CosR yielded superior performance (i.e., it was able to identity the covert channel in more cases). In our experiments, we used the same setup and datasets as in Section 3.3. To demonstrate the sliding window approach, we ran compressibility-walk on a WWW dataset with 2000 data points, with w and s set to 500 and 20. The dashed lines in top two graphs in Figure 3.16 illustrate the cases in which the dataset was purely generated by a WWW channel. The solid line on each graph illustrates the results for a 50% WWW mix dataset. Clearly, the peaks in each of the upper two graphs show the location of each hidden covert channel. We conclude that IP SCCs will reveal themselves as peaks as long as the compressibility of the legitimate and covert channels are distinguishable.

Similarly, we ran our conditional compressibility method CosR-Walk on the same datasets, with w and s set to 500 and 20. The dashed lines in the bottom two graphs in Figure 3.16 show the results when the procedure was run on pure WWW datasets. The solid line on each graph again illustrates the results for a 50% WWW mix dataset. For this metric, a V-shaped curve in each bottom graph suggests the existence of a covert channel. As expected, the locations of these channels are consistent with the compressibility graphs.

To show that CosR-walk is superior to compressibility-walk, both methods traversed an example 50% UDP mix dataset. In Figure 3.17 on the left, we illustrate one unsuccessful attempt by compressibility-walk to locate the covert channel. The curve on this graph appears to be flat suggesting that no covert channel exists, even though there is a covert channel hidden in the middle section. In contrast, the CosR-walk

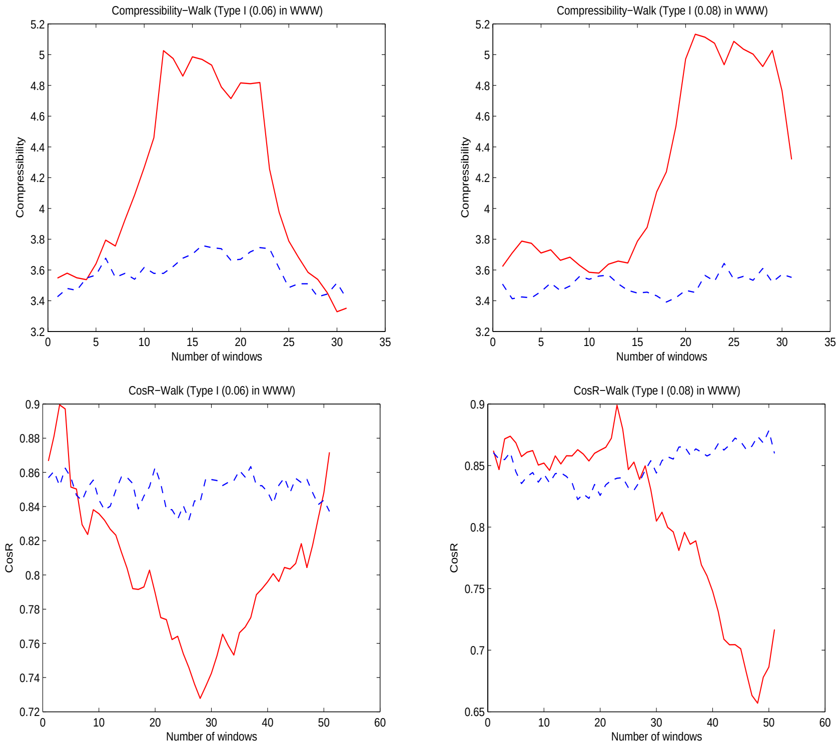


Fig. 3.16. (Top row) Compressibility-walk results for (dashed) a pure WWW channel, (solid) a 50% WWW mix channel. (Bottom row) CosR-walk results for (dashed) a pure WWW channel, (solid) a 50% WWW mix channel.

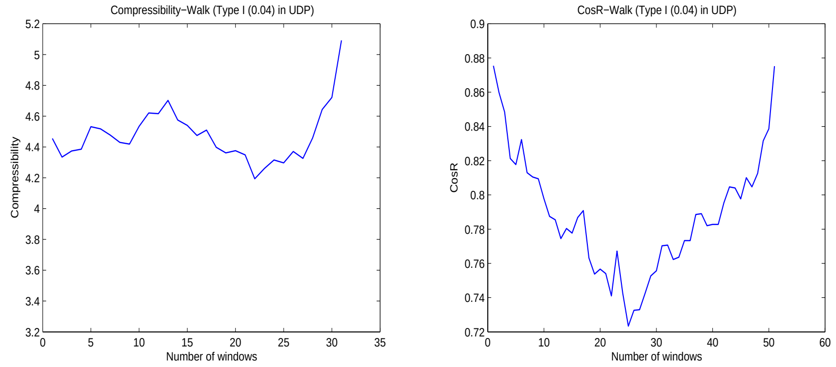


Fig. 3.17. (Left) Compressibility-walk fails to identify the hidden channel in a 50% UDP mix dataset. (Right) CosR-walk identifies the hidden channel.

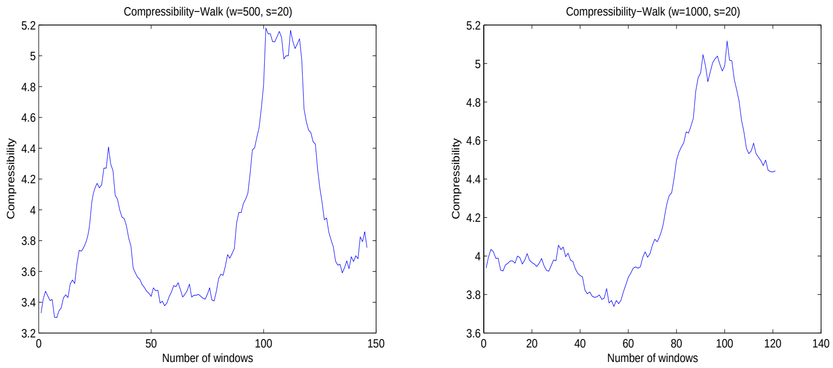


Fig. 3.18. (Left) Compressibility-walk identifies both channels when $w = 500$. (Right) Compressibility-walk identifies only one channel when $w = 1000$.

performs as it should do and the covert channel is again identified by the signature V-shape as we illustrate on the right graph. This phenomenon occurs because the compressibility scores for the UDP datasets are as high as the IP SCC scores, which causes compressibility-walk to fail. (Compressibility-walk works only when the compressibility score of the covert channel is significantly higher than the legitimate channels' scores.) In contrast, CosR-walk is a relative measure for which the individual compression scores do not matter.

3.4.4 Limitations

We conclude the section with a discussion on the limitations of these two measures in detecting noisy IP SCCs. In our experiments we used window and stride sizes that gave us the best results. However, in reality both of our search methods are sensitive to the choice of parameters w and s and choosing a window size larger than the covert channel size may not reveal the channel at all. We illustrate one such example for compressibility-walk in Figure 3.18. On the left graph, both covert channels are identified by setting $w = 500$, whereas on the right only one covert channel is visible by setting $w = 1000$. Because the size of the covert channel is not known at the time of detection, the methods need to be run with different sets of parameters to identify the hidden covert channels.

The time complexity of both methods is a function of data points n, window size w, slide size s, and the running time of the compression algorithm $C(.)$. Compressibility-walk performs one compression per window and CosR-walk performs three. Hence the running time for both algorithms is $O((n/s)C(w))$. Practically, compression is an expensive operation and our implementation of the search algorithms compress each window independently. However, the performance can be improved if some compression information is carried over to the current window from the previous window when windows are overlapping.

Lastly, one can always create a mix dataset that does not reveal the covert channel hidden inside. One example is to use a scattered approach and hide short bursts of covert channel communication throughout the dataset to create balanced windows (i.e., balanced in terms of compression scores and similarity). Although this type of a noisy IP SCC would be difficult to identify, the resulting covert channel is rate-limited. For example if Alice uses a 50% mix dataset to avoid detection, in reality this means that the covert channel bandwidth is halved. Additionally, by forcing Alice to use short bursts of covert channel traffic, we avoid a continuous covert channel between Alice and Eve (i.e., Alice needs to switch between covert and legitimate channels and both parties need to know which packets belong to the covert channel at all times).

3.5 Chapter Summary

A crucial step in covert channel design is to find and exploit a shared resource that is unlikely to be used as a communication medium by design. Less emphasis has been given to hiding the traffic generated by these channels – the secrecy of a covert channel mostly relies on the secrecy of knowing the shared resource. In this chapter, we argued that once this shared resource is identified, unhidden covert traffic can be distinguished from legitimate traffic in most cases because of the way in which event-based covert channels adjust the timing of the events to signal information. We first described our implementation of an IP covert channel, discussed the subtle issues that arose in its design, and presented a discussion on its efficacy. Further, we showed that the regularity of a noiseless IP covert channel can be used to differentiate it from legitimate channels and presented two efficient online detection measures that provided detection rates over 95%. We repeated similar analysis for “noisy” IP covert channels in which legitimate and covert traffic were mixed to obfuscate the covert traffic. We showed that our online detection measures failed to identify the covert channel with a reasonable false alarm rate for noise levels higher than 10%. For such channels, we investigated effective search mechanisms to find regions of the traffic that appear to be covert channels. Such “local” methods are computationally expensive and are thus best deployed offline.

4. TIME-REPLAY NETWORK COVERT CHANNELS

Time-replay channels hide the channel traffic in normal traffic patterns by replaying a previously recorded or a specifically engineered sequence of timing intervals. In certain cases, the event sequence generated by a time-replay covert channel can be equal in distribution to the sequences generated by legitimate channels in the system; Thereby creating channels that are undetectable through analyzing localized distribution of inter-arrival times. Formally, a *time-replay covert channel* is an event-based covert channel in which the timing intervals between *events* (i.e., inter-event times) are taken from a *sequence* of values each of which is associated with a *symbol* by a *rule* and used only once.

Time-replay covert channels represent a family of event-based covert channels rather than a particular channel or a channel type. Therefore, a time-replay covert channel can be categorized as a storage, timing, or a hybrid channel depending on the number of different events and timing intervals it uses. In this chapter, we first detail the time-replay covert channel definition, design, and operation in Sections 4.1 and 4.2. We provide a discussion on channel efficacy and list the factors that affect both channel accuracy and bandwidth in Section 4.3. In Section 4.4, we discuss detection schemes that are applicable to time-replay covert channels in general and investigate the conditions under which these schemes can potentially work. In Section 4.5, we investigate the efficacy of the existing channel prevention and elimination techniques as applied to these channels and provide a discussion how the existing elimination techniques can be improved.

4.1 Covert Channel Definition

Time-replay covert channels replay previously recorded event activity using a simple but an effective three-fold strategy: 1) obtain a pre-recorded event sequence as input, 2) divide the sequence into k partitions, where k is the size of the message alphabet, and associate each partition with a symbol using a set of thresholds (i.e., rules), and 3) send the symbol s by delaying for the amount of time indicated by the timing value in the corresponding partition before generating the event. For example, suppose that a malicious user Alice wishes to leak code C to an eavesdropper Eve using a time-replay covert channel. Suppose C is composed of two types of symbols: s_1 and s_2 (e.g., *zero* and *one* for binary code). To send s_1, Alice chooses a timing value τ_{s_1} from the s_1-partition of the input sequence and generates an event after idling for τ_{s_1} time. To send s_2, she uses the s_2-partition. Each timing value in each partition is used only once. On the receiving side, Eve monitors the events. Upon observing an event, Eve 1) calculates the inter-event time τ between the current and the most recent event, 2) determines to which partition τ belongs, and then 3) records s_1 or s_2 depending on the decision in step 2.

In a time-replay covert channel, Alice's sending routine takes code C and sequence S as the inputs and generates a sequence of events ES with the message encoded along with the inter-event times of this event sequence. To do so, Alice uses the rules R that divide S into k partitions. A rule always applies to an interval of timing values associated with a symbol, such as $< t_i, t_j, s >$ where t_i and t_j are timing values ($t_i \leq t_j$) and s is a symbol. For example, if C is composed of *zeros* and *ones*, the rule $< 0, t_{median}$,*zero*$>$ divides S into two partitions where partition $[0, t_{median})$ is associated with a *zero* and $[t_{median}, t_{max}]$ is associated with a *one.* One can also create a rule for a single timing interval by setting $t_i = t_j$. Applying R on C using S, the sending routine generates a sequence of events that are *transmitted* to Eve. On the receiving end (of the covert channel), Eve's observing routine takes the observed event sequence ES as the only input and computes C by reversing Alice's sending routine operations.

For this channel to work, Alice and Eve must be able to pre-negotiate on the rules, event types, and the encoding scheme. One alternative is to have the information hard-coded in the covert channel software so that no pre-negotiation is needed between the parties. A second alternative is to use another covert channel (e.g., an implementation of a storage SCC) to exchange the necessary information on the fly. In either case, Eve's receiving routine does not need to know the original time sequence S, but does need to know the rule set R.

4.2 Covert Channel Creation

Within a distributed MLS system that properly follows policy, an observer with a LOW security level may still be able to observe the presence of HIGH network traffic without having access to the encrypted contents. Assuming that Alice is a HIGH sender, Bob is a HIGH receiver communicating with Alice, and Eve is a LOW observer, the only information shared between Alice and Eve is the timing of the packets traveling from Alice to Bob. The IP time-replay covert channels (TRCC) we present in this section use this timing information to covertly transmit a secret message from Alice to Eve by adjusting the transmission intervals of the network packets Alice sends to Bob under the following assumptions:

1. Alice and Eve use a single covert channel to communicate, thus our research does not address multiple channels that aggregate packet traffic to leak information.

2. Alice has a ready supply of network packets that she can send to Bob at any given time and the channel protocol allows arbitrary sends from Alice (e.g., she sends HTTP `GET` requests with arbitrary content to Bob who is a web server with HIGH content).

3. Our research does not address the cases in which Bob might become suspicious as a result of the amount of network traffic generated by Alice.

4. The original payloads of the network packets transmitted over the direct channel between Alice and Bob are legitimate, hence do not violate any MLS policies.

5. The raw data that flows across the covert channel is binary but the actual interpretation of the binary string is up to Alice and Eve.

6. Additional bits may also be included in the transmission for three reasons: 1) Additional parity bits may be appended to the data to add redundancy for error correction caused by transmission errors (e.g., errors arising when a packet is lost/delayed), 2) additional bits may be added for purposes of maintaining synchronization between Alice and Eve's event clocks, and 3) the covert data may be encrypted to add a further layer of privacy and obfuscation. In our implementation, we employ error-correcting codes but not encryption.

7. The covert message for transmission is subdivided into smaller blocks of binary data, referred to as *frames* in this paper. An example frame consists of data bits, synchronization bits, and error-correcting bits. While all the frames are of equal length, the actual length, as well as the interval between frames, is influenced by parameters of the encoding scheme and the network. In our implementation, frames are composed of the eight-bit ASCII encoding of each character and an optional six bits for error correction using (7-4)-Hamming codes.

8. We assume a unidirectional communication model for the covert channel. Only the indirect covert channel is assumed to be unidirectional; The direct channel itself is still bidirectional and the TCP/IP packets are ACKed. Assuming a unidirectional channel means that Eve cannot communicate with Alice using the covert channel itself. Restricting the channel to be unidirectional increases the difficulty in implementing an error-free channel. In detail, Eve cannot 1)

acknowledge the correct receipt of covert bits, 2) rate limit Alice, or 3) indicate when to resynchronize.

4.2.1 Channel Design

IP TRCCs transmit a covert message from Alice to Eve bit-by-bit, hence the replay channel symbol set is composed of *zeros* and *ones.* The lone event in the system is the packet arrival, which is controlled by Alice and observed by Eve. To leak consecutive bits to Eve, Alice generates a sequence of packets and encodes the covert message by adjusting the timing between these packets (i.e., inter-arrival times). To impersonate legitimate behavior further and to avoid generating regular (thus anomalous) packet sequences, IP TRCCs adjust packet timings according to a pre-recorded normal sequence. Therefore, in IP TRCCs, the choice of which inter-arrival sequence directly impacts the secrecy of the channels and the better the input inter-arrival sequence impersonates normal behavior, the more covert the channel. We illustrate the covert channel operation in Figure 4.1.

We present three alternative IP TRCC designs; They differ in how the channel symbols are associated with the inter-arrival times:

Binary-Matching Channel (BMC)

Given an inter-arrival time sequence, suppose that Alice and Eve are capable of negotiating a single cutoff value τ_{cutoff}, for example the median of the sorted inter-arrival time sequence as calculated by Alice. In the *binary-matching channel* (BMC), the parties share one rule to partition the sequence into $[0, \tau_{cutoff})$ and $[\tau_{cutoff}, \tau_{max}]$. The inter-arrival times shorter than τ_{cutoff} are associated with a *zero* and inter-arrival times longer are associated with a *one.* Alternatively, one can use a buffered cutoff value $\tau_{cutoff} \pm \delta$ to increase channel accuracy. In either case, Alice maintains two

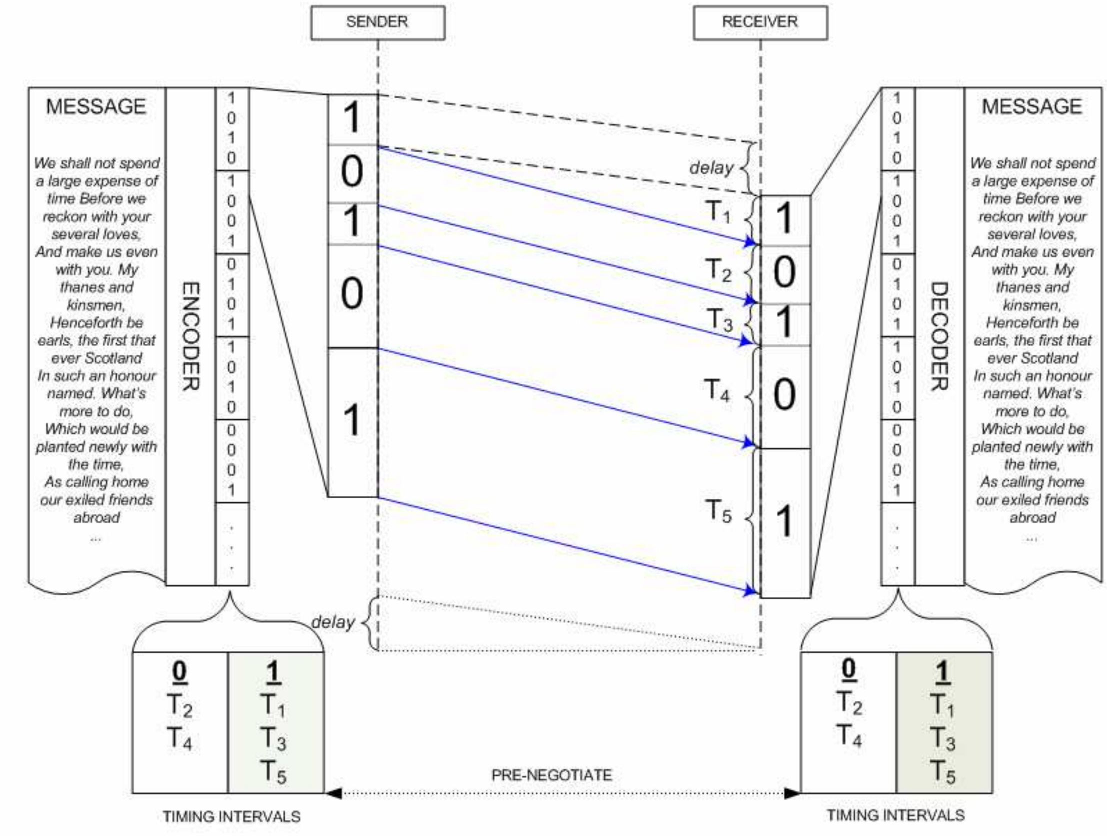


Fig. 4.1. IP time-replay covert channel. The example text is first encoded with a coding scheme and then sent bit by bit to the receiving end. The message is rebuilt by decoding the bit-string.

bins; one for each timing value partition. On the receiving end, Eve does not need to know the exact inter-arrival times, only the cutoff value τ_{cutoff} and δ.

BMC operates as follows: To send a *zero*, Alice randomly chooses a timing value τ_{zero} from the *zero*-bin and sends a packet to Bob (Eve) after idling for τ_{zero} time. To send a *one*, she uses the *one*-bin for selection. Each timing value is used only once. On the receiving side, Eve monitors the transmission line between Alice and Bob. Upon detecting the presence of a packet traveling from Alice to Bob, Eve 1) calculates the inter-arrival time $\tau_{observed}$ between the currently observed packet and the last one, 2) compares $\tau_{observed}$ to $\tau_{cutoff} \pm \delta$, and she 3) records a *zero* if $\tau_{observed} \leq \tau_{cutoff} - \delta$, a *one* if $\tau_{observed} \geq \tau_{cutoff} + \delta$, and nothing otherwise.

Rule-Matching Channel (RMC)

Arranging inter-arrival times as in BMC may lead to detection because similar timing intervals are associated with the same symbol (i.e., short values with a *zero* and long values with a *one*). The two other alternative channels that we designed use more complex rules to hide the channel structure better. In the *rule-matching channel* (RMC), we use k rules that divide n inter-arrival time values into $\lceil \frac{n}{k} \rceil$ partitions. As an example, one can use a sample rule set $(0, \tau_1,$*zero*$)$, $(\tau_1, \tau_2,$*one*$)$, and $(\tau_2, \tau_3,$*zero*$)$, that partitions the inter-arrival space into three mutually exclusive blocks. Each rule partition is associated with a symbol randomly, hence no relation exists between the partitions.

RMC operates as follows: To send a *zero*, Alice randomly chooses a timing value τ_{zero} from the *zero*-bins and sends a packet to Bob (Eve) after idling for τ_{zero} time. To send a *one*, she uses the *one*-bins for selection. On the receiving side, Eve monitors the transmission line between Alice and Bob. Upon detecting the presence of a packet traveling from Alice to Bob, Eve 1) calculates the inter-arrival time $\tau_{observed}$ between the currently observed packet and the last one, 2) compares $\tau_{observed}$ to the complete set of rules, and 3) records a *zero* if $\tau_{observed}$ belongs to a rule partition with a *zero* association, a *one* otherwise.

Exact-Matching Channel (EMC)

The last alternative IP TRCC that we designed uses an independent rule for each timing interval, which we call the *exact-matching channel* (EMC). The EMC rule set is composed of 3-tuples such as $(\tau_1, \tau_1,$*zero*$)$, $(\tau_2, \tau_2,$*zero*$)$, $(\tau_3, \tau_3,$*one*$)$, etc. In this scheme, there is a one-to-one correspondence between the rules and the timing values. As in RMC, each rule is associated with a symbol randomly, hence no relation exists between the timing intervals. The EMC operation is identical to RMC, only this time we have as many rules as timing intervals.

4.2.2 Comparison and Limitations

Both BMC and EMC are special cases of RMC where we have $k = 1$ or $k = n - 1$ rules to divide the sequence into two and n partitions, where n is the length of the input sequence. In the rest of this chapter, we will mainly concentrate on BMC-type IP TRCCs and refer to EMC for the sake of comparison. This is because BMC-type channels have many design advantages over the other two as we list below:

Rule Negotiation: To initiate the covert channel, Alice and Eve need to pre-negotiate on one rule for BMC, k rules for RMC, and $n - 1$ rules for EMC.

Rule Comparison: On observing a packet arrival, Eve needs O(1) rule comparisons for BMC, O(k) comparisons for RMC, and O(n) comparisons for EMC to decide on the observed symbol.

Correctness: For the correct operation of the channel, RMC and EMC need the same timing values to be associated with the same symbols, whereas BMC requires only timing values equal to the cutoff value to be associated with the same symbol. For example, in EMC, if τ occurs more than once, each value needs to be associated with the same symbol s. In BMC, this is guaranteed by design except the τ_{cutoff} value.

Accuracy: For the accurate operation of the channel, RMC and EMC need similar timing values to be associated with the same symbols, which is guaranteed in BMC by design.

Stability: Both EMC and RMC are *unstable* channels that are extremely vulnerable to perturbations on individual inter-arrival time values caused by channel noise or intentionally, whereas BMC is a *stable* channel that is not affected by small changes of each inter-arrival value.

Given these observations, we concentrate only on BMC-type channels in the next section where we discuss the efficacy of IP TRCCs. In Sections 4.4 and 4.5, we include EMC into our discussion to compare the two in terms of channel detection and elimination.

4.3 Covert Channel Analysis

We assess the efficacy of IP TRCCs in terms of bit/character accuracy (the number of bits/characters transmitted correctly divided by the total number of bits/characters) and channel bandwidth. Three factors affect channel efficacy:

1. *Contention noise* is the amount of non-covert traffic Eve observes in the covert channel and can potentially reduce channel accuracy [9].

2. *Clock skew* is the amount of jitter in the network and can potentially result in the loss of synchronization between Alice's and Eve's event clocks.

3. *Sequence selection* is the choice of which inter-arrival sequence will be used and can potentially affect channel accuracy.

In this section, we investigate the effects of these factors on channel accuracy and bandwidth, and present the results of our empirical study with different choices of inter-arrival time sequences.

4.3.1 Channel Accuracy

To evaluate the channel efficacy alone and provide an upper bound on character accuracy, we assume that IP TRCCs are free of contention noise. Thus, Eve is capable of isolating the network packets generated by Alice and destined to Bob from the ones originating from or destined to other users in the network. In this scenario, clock skew and sequence selection are the only factors that affect channel accuracy. Clock skew in a network is mostly caused by varying network conditions that result in a slight increase or decrease in the latency between Alice and Eve (i.e., network jitter). This factor can potential affect the timing of network packets traveling from Alice to Bob (Eve), thus reduce channel accuracy. For example, a temporary congestion in the network can delay a packet further and change its symbol association at the receiving side. Sequence selection also plays an important role in determining how much damage clock skew can cause and the choice of which sequence affects channel accuracy. Ideally, in a sequence, timing intervals that are associated with different symbols need to be as distinct from each other as possible. As an example, in BMC, Eve needs to be able to differentiate timing values that belong to the *zero* partition from the ones that belong to the *one* partition. This depends on both sequence selection as well as how individual timing intervals are associated with each partition. In this section, we illustrate the effects of sequence selection on channel accuracy empirically.

An effective way to remedy clock skew and sequence selection deficiencies is to use error-correcting codes (ECC). ECCs add additional bits to the codeword that can be used to detect and correct some errors in transmission [77, 78]. The code rate of an ECC is defined as the ratio between the length of an original codeword and the length of the codeword with added error-correcting bits. A drawback of ECCs is that they increase the amount of information to be transmitted over the channel. In this study, we use Hamming codes [78] for one-bit error correction.

4.3.2 Channel Bandwidth

Ideally, using sequences with the shortest inter-arrival times possible yields the highest data rate. However, this rate is limited by three factors:

1. Alice's packet generator is the bottleneck as she cannot generate packets at a higher rate than the generator's rate. Another limiting factor on Alice's side is the precision of the packet delayer. As we use shorter inter-arrival times, the

delayer routine may fail to *sleep* precisely for the required amount of time. The same constraints also apply to Eve's observer routine.

2. In networks with jitter, the negative impact of clock skew on channel accuracy increases as we use shorter timing intervals. This is because with short timing intervals, a small variation on the interval may alter its symbol association (i.e., place it in a different partition) whereas the same variation may be negligible for long intervals.

3. Sequences that yield high data rates may be anomalous in comparison to normal sequences in the network.

4.3.3 Empirical Evaluation

To assess the efficacy of the covert channel experimentally, we implemented a BMC prototype using the Berkeley socket library in Python. For one-bit error-correction, we used Hamming codes that are computationally simple to encode and decode [78]. To calculate bit and character accuracy, we used an implementation of the Levenshtein distance (i.e., the edit distance [80]). In our experiments, BMC replayed ten WWW, FTP-data, UDP, and five SSH inter-arrival time sequences we extracted from the NZIX-II dataset[1]. Each sequence contained 3000 data points and we eliminated inter-arrival times longer than two seconds to avoid long delays between the packets. BMC is capable of sending both binary and text data and we used ASCII encoding for the latter. In both cases, BMC divided the resulting bit-string into eight-bit codewords that were expanded to 14 bits when we used (7,4)-Hamming error-correction.

Accuracy Analysis

To get an upper-bound on channel accuracy, we ran BMC between two machines on the same LAN to send the ASCII-encoding of the sample text listed in Chap-

[1] A collection of real world IP traces captured by the WAND research group [10].

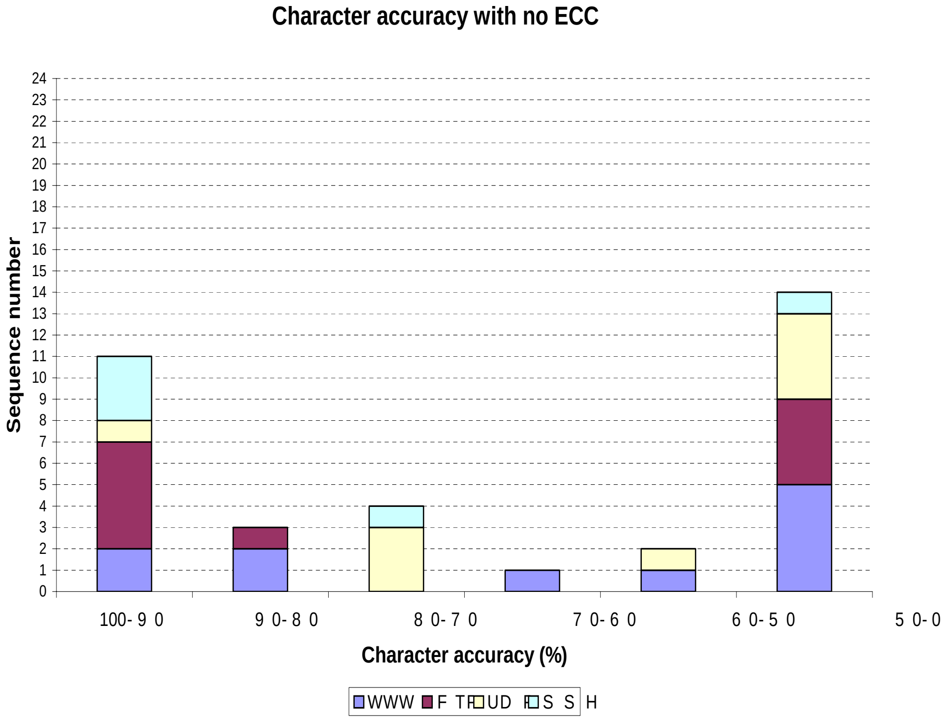


Fig. 4.2. Character accuracy results for BMC replaying NZIX-II sequences with no error correction.

ter 3. We report the character accuracy results with and without error-correction for all 35 NZIX-II inter-arrival time sequences in Figures 4.2 and 4.3 using histograms. We observe that with no error-correction, 11 out of 35 sequences yielded over 90% character accuracy and 14 sequences yielded less than 50% accuracy because of certain sequence properties as explained in the next section. With error-correction, 22 sequences yielded over 80% character accuracy rates, 17 of which yielded over 90% accuracy. On average, error-correction yielded a ten percent increase in character accuracy and we observed no specific advantage of using one type of traffic type over another as sequences from each traffic type gave a mixture of good/bad results. (As an exception, SSH sequences fared slightly better as four out of five sequences yielded over 98% accuracy with error-correction.)

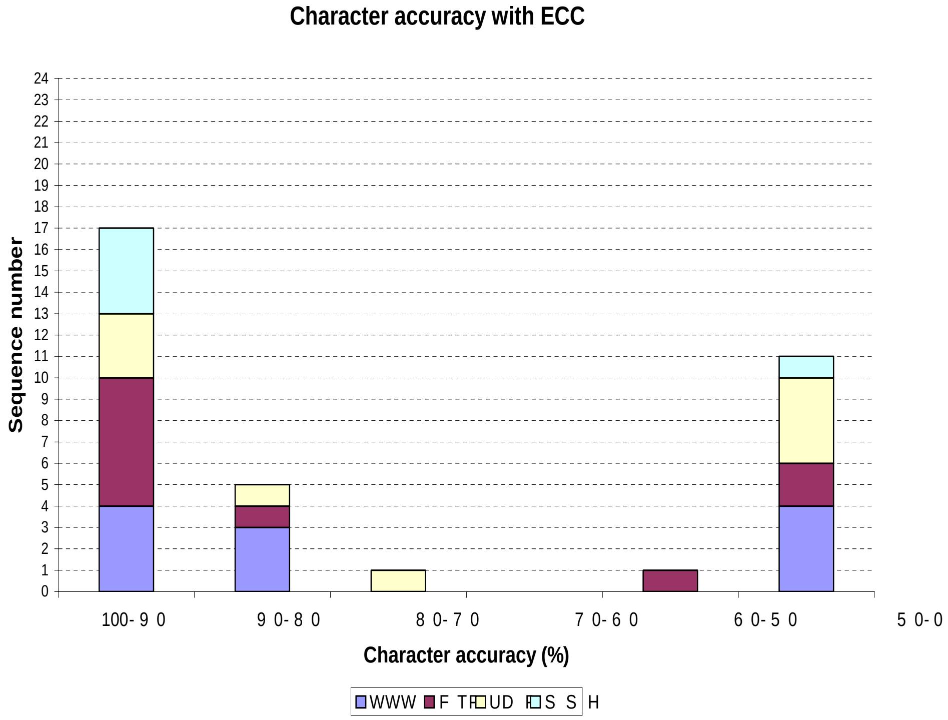


Fig. 4.3. Character accuracy results for BMC replaying NZIX-II sequences with (7,4)-Hamming error correction.

Bit accuracy results followed a similar trend as we illustrate in Figure 4.4. Our results show that, 22 out of 35 sequences yielded better than 90% accuracy rates and the rates for the remaining sequences were uniformly distributed over the interval 90-40%. We also observed that a sequence with bit accuracy over 90% yielded character accuracy over 80% with error-correction. Additionally, as expected, bit accuracy rates with and without error-correction were similar.

As a final note, whether a 100% accuracy rate is necessary or not depends on the application type. While an 80% accuracy rate may be sufficient for Eve to decode a reasonably understandable message, 100% accuracy is absolutely necessary if the covert channel is transmitting a binary and/or an encrypted file. Although, for the cases in which 100% accuracy is not required (e.g., streaming traffic), the covert

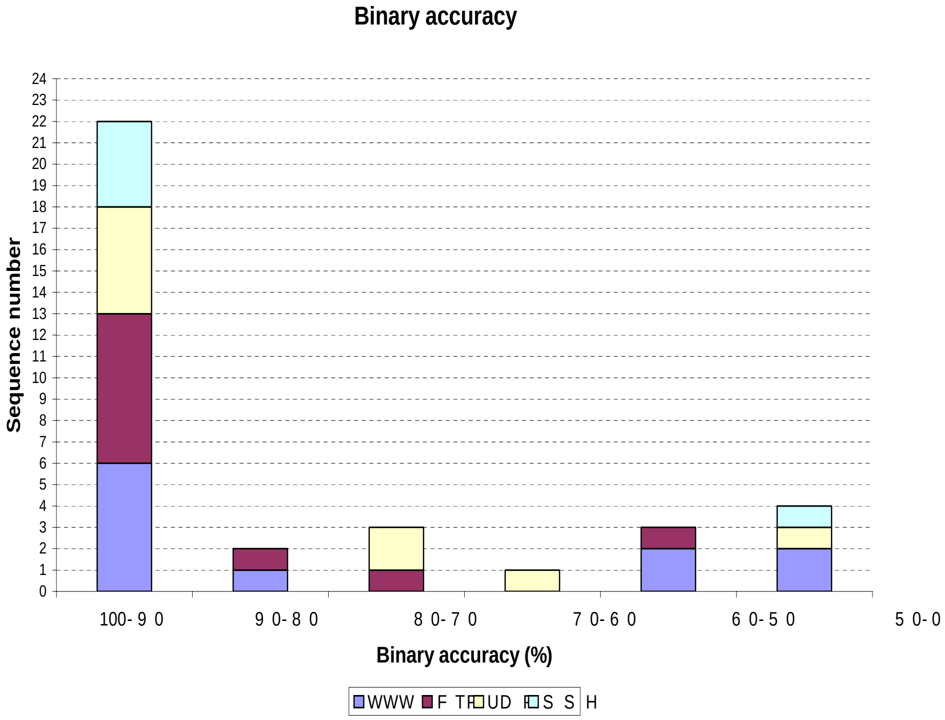


Fig. 4.4. Bit accuracy results for BMC replaying NZIX-II sequences.

channel can employ sequences that yield higher data rates at the expense of lowered accuracy.

Sequence Analysis

In Figures 4.5 and 4.6, we illustrate four example NZIX-II inter-arrival time sequences we used in our experiments. We sorted the inter-arrival times and used a different precision for each graph's y-axis to focus on the intersection point between the cutoff value and the sequence. In each graph, the solid line represents the cutoff value (τ_{cutoff}, which is the median), and the dashed line is the packet generator's *minimum* processing time (P_{gen}), which was 0.0044 secs for Python. (In Figure 4.5, these appear to lie on the x-axis because of resolution.) The example sequences in Figure 4.5 provided 98% and 96% character accuracy with no error-correction and

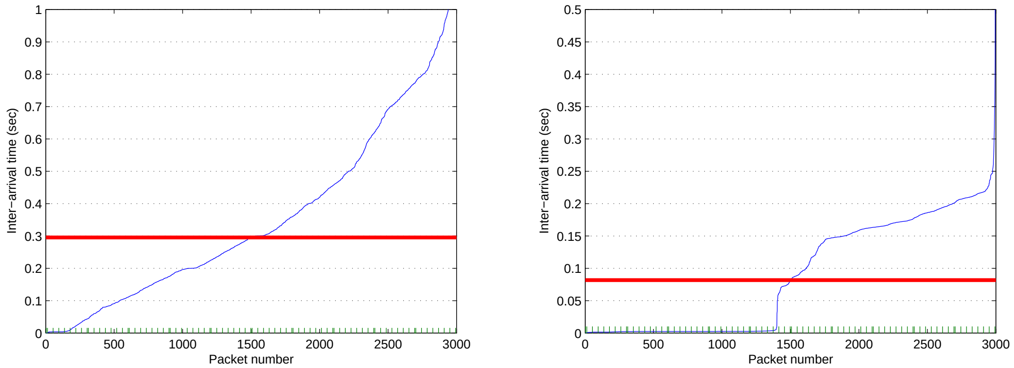


Fig. 4.5. Example NZIX-II WWW and FTP inter-arrival time sequences that yield high character accuracy with one-bit error correction. The solid line is τ_{cutoff} and the dashed line is P_{gen}.

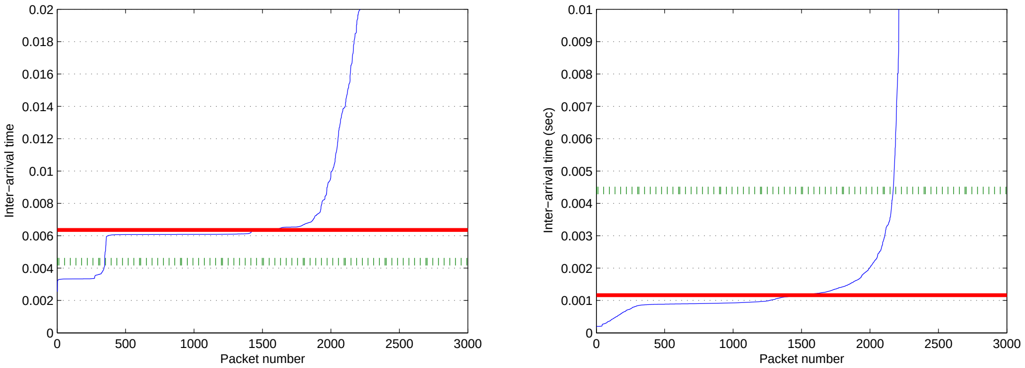


Fig. 4.6. Example NZIX-II FTP and UDP inter-arrival time sequences that yield low character accuracy with one-bit error correction. The solid line is τ_{cutoff} and the dashed line is P_{gen}.

100% with error-correction. The results in this graph are as a result of three factors that can be generalized to all BMC sequences: 1) Each sequence provides a good separation of *short* and *long* inter-arrival times, 2) the cutoff value intersects the sequence at one point, 3) $P_{gen} \ll \tau_{cutoff}$. In contrast, the example sequences in Figure 4.6 provided 0.23% and 0% character accuracy with no error-correction and 0.35% and 0% with error-correction. This is because both sequences in the latter graph violate the above conditions – particularly the third, in which the packet generator cannot keep up with the desired packet rate dictated by the input inter-arrival time sequence. However, this is an experimental deficiency as one can design a faster packet generator that will allow using sequences with short timing values. Further, Alice can devise a more accurate channel by specifically engineering a more suitable sequence using the original sequence. To do so, one alternative is to avoid the timing values that are within the δ-neighborhood of τ_{cutoff} (e.g., the flat lines in Figure 4.6). However, choosing a big δ value can create anomalous output sequences that can potentially lead to detection. A second alternative is to use the sequence as it is but ignore the timing values that are within the δ-neighborhood of τ_{cutoff} (i.e., do not associate a symbol with a timing value that falls within this interval). In this scheme, Alice still sends packets with such inter-arrival times, but these carry no information bits. Upon observing an inter-arrival time value that falls under this category, Eve updates the arrival time but does not record any bits. The resulting channel is rate-limited but is more difficult to detect as compared to the former alternative.

As a final note, one can specifically engineer an ideal sequence that follows all three conditions we listed above for a BMC channel. However, in doing so, the resulting sequence should not deviate from the normal sequences to avoid creating anomalous traffic patterns.

4.4 Covert Channel Detection

Detection is a common practice in secure systems to monitor malicious activity [14]. In this section, we discuss the detection of EMC and BMC-type IP TRCCs, which represent two extreme cases of the generalized RMC-type channels. We investigate the cases in which these channels are detectable/undetectable through analyzing localized distribution of inter-arrival times.

IP TRCCs leak a covert message from Alice to Eve by replaying a pre-recorded inter-arrival time sequence. Let S_{in} be one such sequence we want to use to transmit a message text M. We can generate S_{in} by either utilizing a pre-recorded sequence that is *recently* generated by a legitimate channel, or by producing a sample from the legitimate distribution if no previous sequence is accessible. In the former case, the first line of defense could be to devise methods to determine if certain inter-arrival times are the exact same ones that were observed recently. However, because the covert channel window size is not known at the time of detection, this scheme would need to employ costly search mechanisms similar to the ones we presented in Chapter 3. Further, Alice can perturb the timing values deterministically (e.g., by adding a constant to the timing values that is only known to Eve) or randomly (which would only work for BMC at the expense of lowered channel accuracy) to by-pass such schemes.

In this section, we focus on schemes that analyze localized distribution of inter-arrival times sequences to detect time-replay covert channels. Suppose we are given an inter-arrival time sequence S_{in}, a covert message M, and a rule set R. To generate an output sequence S_{out}, Alice first encodes M into a bit-string B (e.g., using ASCII encoding) and chooses the timing values from S_{in} according to B and R as we detailed in Section 4.2 and illustrate in Figure 4.7. Using this protocol, the channel generates S_{out}, which is a permutation of the timing values in S_{in} chosen in a particular way to embed B. Further, we assume that $|S_{out}| \ll |S_{in}|$.

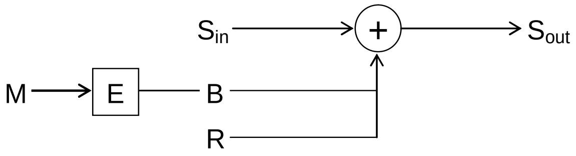


Fig. 4.7. Factors that affect the output sequence S_{out} for a time-replay channel: the input sequence S_{in}, the input message M, the encoder E, the input bit-string $B = E(M)$, and the rule set R.

Given this setup, S_{in} is equal in distribution to a normal sequence N by construction and we investigate whether S_{out} is also equal in distribution. If Alice can generate S_{out} in such a way that this is true, then the channel is virtually undetectable through traffic distribution analysis. If no such sequence can be generated, that is one can always distinguish the distribution of S_{out} from that of N, a detection scheme can be devised to detect the anomalous activity. In this section, we investigate the cases in which Alice can devise such indistinguishable S_{out}s and the cases in which this is not possible.

Whether Alice can generate such an output sequence S_{out} or not depends on various factors such as the input sequence S_{in}, the message encoder $E(M) = B$, and the rule set R. Further, because the input sequence is a replay of a previously recorded sequence, Alice does not have control of S_{in}. Hence, she can only devise a generation scheme by choosing different rule sets and message encoders. In terms of time-replay channels, Alice can employ three types of channels (BMC, RMC, and EMC) that use different rule sets. Additionally, we show that by adjusting the encoding scheme, Alice can ensure that S_{out} is generated by uniformly sampling S_{in}, which can potentially preserve its distribution for certain distribution types (e.g., uniform distribution).

4.4.1 Detecting EMC-type Channels

First, we examine the case in which Alice uses an EMC-type IP TRCC to create a covert channel that is virtually undetectable through distribution analysis under certain conditions. An EMC-type IP TRCC uses $n-1$ rules to divide the inter-arrival sequence into n partitions, where n is the size of the input sequence. Further, Alice arranges the channel rule set such that, given the symbol to be transmitted (i.e., a *zero* or a *one*), the probability of selecting a timing value is equally likely for all remaining timing values. In this setup, we argue that if Alice can also ensure the uniform sampling of the timing values from S_{in} *regardless of the transmitted message*, S_{out} can potentially be equal in distribution to S_{in} depending on the distribution type. A trivial example is the uniform distribution for which uniform sampling produces an output that is also uniformly-distributed. Another example is the Poisson distribution. It can be shown that randomly tagging a Poisson-distributed sequence produces two sub-sequences, each Poisson-distributed (see [97]). Therefore, if a parametric test does not reject the null hypothesis that S_{in} is D-distributed, it will not reject the same hypothesis for S_{out}.

Furthermore, in cases S_{in} does not follow a certain distribution and does not retain a rigid structure (e.g., a timing value is always followed by a certain value), uniform sampling can potentially preserve the structure of the sequence. This is because a series of numbers (with no assumed distribution) and its uniform sampling are equivalent under a non-parametric comparison. Thus, uniform sampling only reduces the sample size but it can potentially preserve the structure. Therefore, if S_{out} is generated by uniformly sampling S_{in}, a non-parametric test (e.g., the Kolmogorov-Smirnov test) comparing the two will not reject the null hypothesis that S_{in} and S_{out} are drawn from the same underlying population.

One way to ensure uniform sampling for EMC-type channels is to make sure that the bit-string that will be transmitted (i.e., $E(M) = B$) is uniformly distributed. This is because it is B that determines which values will be selected from S_{in}. Because Alice does not have control of the message M, she can only choose an appropriate encoding for the message, or use a transformation on B following the message encoding. However, finding an appropriate encoding that will ensure the uniform distribution of *one*s and *zero*s in B is a hard problem as Alice will also need to take character frequencies into account. A simpler solution is to find an appropriate transformation T that will produce a uniformly distributed output B_T when input any bit-string B. One such transformation is bit-by-bit encryption. Indeed, Alice can employ a probabilistic encryptor such that the transformed (i.e., encrypted) bit-string B_T is guaranteed to be indistinguishable from a random bit-string [98].

In cases uniform sampling preserves the distribution, Alice can devise an EMC prototype that is virtually undetectable through distribution analysis by using the following protocol:

1. Run the covert channel on an application port for which the inter-arrival times are characteristically divided into several clusters (e.g., Telnet (Yes), Streaming (No)).

2. Use an EMC-type channel with a recently recorded normal input sequence S_{in} and a rule set R for which the probability of selecting a timing value is uniform.

3. Either exchange a session encryption key K with Eve or use a publicly known bit-string as the key.

4. Encode M into B and encrypt B bit-by-bit with a bit-secure probabilistic encryptor into B_E using the session key K.

5. Send B_E bit-by-bit using the covert channel.

However, uniform sampling does not guarantee preserving the distribution of the input sequence for every distribution type. For example if the input sequence is bursty, uniform sampling might fail to spot the windows in which this characteristic property is observed. Further, for such distribution types how accurate the output sample represents the input sample depends on the sampling rate. We conjecture that the EMC-type covert channel is undetectable through distribution analysis only for the cases in which S_{out} is a sample for which the distribution of S_{in} is preserved.

4.4.2 Detecting BMC-type Channels

As we discussed in Section 4.2, implementing EMC-type channels and finding input sequences (S_{in}) that are characteristically divided into several clusters can be infeasible. In comparison, BMC-type channels are easy to implement channels that are more resilient to the factors that decrease channel accuracy. In this section, we examine the case in which Alice uses a BMC-type IP TRCC to generate output sequences (S_{out}) that are indistinguishable from normal sequences. BMC-type channels use only one rule to divide S_{in} into two partitions. Further, Alice arranges the timing values such that, given the symbol to be transmitted (i.e., a *zero* or a *one*), the probability of selecting a timing value is equally likely for all timing values associated with that symbol. Again, if Alice can also ensure the uniform sampling of the timing values from S_{in} *regardless of the transmitted message*, S_{out} can potentially be equal in distribution to S_{in} depending on the distribution type.

Creating a uniform sampling scheme in BMC-type channels, however, is not as trivial as in the EMC case. This is because, unlike EMC, the BMC rule set *deterministically* divides S_{in} into two partitions using a cutoff value τ_{cutoff}, whereas in EMC, each timing value is associated with a symbol randomly. This difference has a major effect on the structure of the transformed sequence (i.e., S_{out}). As an example to illustrate this effect, assume that S_{in} is a Poisson-distributed sequence[2]. Again, it is easy to show that deterministically tagging the sample points (as in the BMC case) creates two sub-samples neither of which are Poisson-distributed (see [97]). Therefore, a detection scheme could use a parametric test to determine that the output sequence S_{out} is not Poisson-distributed.

The cases for which the input sequences are not from a particular distribution also suffer from deterministic tagging. For example, consider the case in which S_{in} is composed of a group of *short* inter-arrival time values followed by a group of *long* values and so on. In this case, if Alice applies uniform sampling as in the EMC case,

[2] We are not assuming that normal sequences are Poisson-distributed but use this assumption as an example.

the output sequence S_{out} will not show the same behavior as the S_{in} as it will be a mixture of *short* and *long* values that can be detected as anomalous behavior by a detection scheme. To avoid this, Alice could choose an appropriate transformation over B or S_{out} to make the output sequence look more like S_{in}. For example, Alice can use redundant bits to increase consecutive *zero*s and *one*s; hence, consecutive *short* and *long* inter-arrival time values. Eve also needs to know the transformation to apply the inverse at the receiving end.

In either case, whether the output sequence S_{out} shows similar structure to S_{in} depends on whether Alice can find and implement an appropriate sampler or transformation on B or S_{out}. This depends on the distribution type (parametric case) and how rigid the structure of the input sequence is (non-parametric case). However, Alice has access to S_{in}, thus she can check whether the generated S_{out} resembles the input sequence. Further, she can apply various transformations to determine which yields the closest match to S_{in}.

4.4.3 Freshness of the Input Sequence

In the previous sections we assumed that Alice has access to an inter-arrival time sequence S_{in} that is equal in distribution to a normal sequence by construction (i.e., it is a recently recorded normal sequence). This assumption is valid for systems in which the normal behavior does not change over a long period of time. However, for distributed systems in which network conditions change over time, S_{in} may become outdated as conditions change and may no longer be indistinguishable from the sequences generated under the new conditions.

In such cases, S_{in} can no longer represent the normal behavior and the output sequence S_{out}, which is a permutation of the values in S_{in}, will also be anomalous. Hence, a distribution analysis scheme comparing the distribution of S_{out} to that of a normal sequence may detect the anomalous behavior, hence the covert channel. In this case, Alice needs to refresh S_{in} to reflect the current conditions. For example, she can record a new sequence under the current network conditions to use as the input sequence to the time-replay covert channel. In this case, Eve also needs to relearn the sequence information to be able to decode the message correctly. In a BMC-type channel, Alice only needs to exchange the new cutoff value, which can be transmitted to Eve using the channel. Upon receiving the new cutoff value, Eve simply adjusts her packet monitor that decides on the symbol. In an EMC-type channel, the entire sequence needs to be transmitted to Eve along with the symbol associations.

4.5 Covert Channel Prevention and Elimination

Eliminating high-capacity covert channels is also a crucial requirement for trusted systems [5]. In the previous section, we showed that time-replay channels can potentially be hidden from detection schemes that perform distribution analysis under certain conditions. Therefore, to counter these channels, one needs to ensure that certain prevention and elimination schemes are in place. In this section, we present systems in which time-replay channels can be prevented by design. We then investigate two covert channel elimination schemes, jamming [11] and the network pump [12], and provide methods that aid these schemes in stopping time-replay channels.

4.5.1 Preventing Time-Replay Channels

Time-replay covert channels are not applicable to every system. As an example, consider the case in which Alice runs an IP TRCC on an application port that is normally used to transmit streaming traffic. In this case, to hide the covert channel Alice uses a recently recorded sequence that was previously used in streaming communication. However, it is easy to see that covert channels that employ such sequences will be extremely error-prone. This is because the inter-arrival times generated by a streaming application show uniform behavior over time. Thus, the inter-arrival times cannot be grouped into well-separated partitions (e.g., *short* and *long* clusters) as required by a time-replay covert channel. Therefore, we say that a time-replay covert channel is inherently prevented for such setups in which the normal traffic is not suitable for covert communication.

In our empirical evaluation of the efficacy of IP TRCCs in Section 4.3.3, we showed that certain sequences are not suitable to be used in covert communication because they provided extremely low character accuracy rates. These sequences suffered from the fact that many values were grouped around the τ_{cutoff} value; thus, there was not a clear separation between *short* and *long* values that would ensure that Eve can determine which timing value belongs to which partition (see Figures 4.5 and 4.6). Hence, sequence selection plays an important role in covert channel accuracy. Further, if the normal sequences belong to the set of sequences for which time-replay covert channels fail the system is inherently protected against channels of this type.

We argue that one can utilize this characteristic as a strategy to design systems that explicitly prevent time-replay covert channels and event-based covert channels in general. Time-replay channels use the differences between the event timings as a medium to leak information. Contrapositively, if the time between the packets (i.e., packet inter-arrival times) is uniformly distributed, time-replay channels can be prevented completely. The main difficulty with this approach is that the packets are generated and transmitted following a protocol and are responsive to changes in the environment (e.g., network) conditions. One solution to this problem is to allow packets to be transmitted according to the protocol but employ certain constructs (e.g., a data pump) along the transmission line that could transform the inter-arrival times to make them more uniform. In the next two sections, we discuss two such existing channel elimination schemes, jammers [11] and the network pump [12], that adjust packet timings to disrupt the channel in a similar way.

4.5.2 Eliminating EMC-type Channels

As we discussed in Section 4.2, EMC-type time-replay channels are unstable channels that are extremely vulnerable to perturbations on individual inter-arrival time values. As a result, one can eliminate these channels easily by adding small perturbations to the packets traveling through the channel. Therefore, both jammers and data pumps work effectively on EMC-type channels without any modification because both schemes add a probabilistic delay to each packet disrupting the association between the symbols and the inter-arrival times.

4.5.3 Eliminating BMC-type Channels

Unlike EMC, BMC-type channels are less affected by perturbation on packet arrival times. This is because in BMC, each symbol is not tightly associated with a timing value as in EMC but it is associated with an interval (i.e., $[0,\tau_{cutoff})$ to *zero*, and $[\tau_{cutoff},\tau_{max})$ to *one*). Therefore, small perturbations on timing values do not necessarily change the timing value's symbol association. For example, added packet delays with a low variance (regardless of the mean) cannot eliminate BMC-type channels because Eve observes relative timing values (i.e., packet inter-arrival times) rather than the actual arrival time. This effect is especially visible for channels that use sequences with a good separation of inter-arrival times into short and long values. For these channels, both jammers and data pumps may fail to eliminate the covert activity completely because neither scheme has a policy that observes by how much the timing values should be perturbed. As a result, BMC-type channels may survive these elimination schemes with a slightly lowered data rate. (The actual amount by which the data rate is reduced depends on the amount of added average delay to the packets.)

To eliminate BMC-type channels as well as EMC, elimination schemes need to transform packet timings more intelligently to remove the associations between the timing values and symbols. For BMC-type channels it is important that the timing values are perturbed significantly so that their symbol association can be disrupted. One method to do so is to use a random transformation with high variance that results in sequences with different symbol associations compared to the original sequence.

However, this transformation can only partially eliminate the channel because some of the timing values (e.g., the ones that are most distant from τ_{cutoff}) can potentially retain their original association. A more effective way is to mimic systems for which these channels are prevented by design and transform the sequences in a way such that no separation into short and long values is possible for the resulting sequence (i.e., create sequences with more uniform inter-arrival times). Using such a transformation, time-replay channels can be completely eliminated as the transformation additionally disrupts the shape of the output sequence in a way a time-replay channel cannot be deployed. However, such schemes introduce additional control over packet timings usually at the expense of performance. Therefore, one needs to investigate the effects of these transformations on system performance to guarantee fair-scheduling of the packets as an example.

4.6 Chapter Summary

In this chapter, we introduced IP time-replay covert channels (TRCC). This is a new covert channel family that hides channel traffic by replaying a previously recorded or a specifically engineered sequence of timing intervals. We presented different types of IP TRCCs each of which utilized a different rule set to associate time values with the symbols from the encoded message (e.g., *zero* or *one* for a binary message). Our analysis showed that the binary-matching channel (BMC), which has only one rule to partition the timing value space into two, has many advantages over the other two alternatives we presented: the rule-matching channel (RMC) and the exact-matching channel (EMC). Accuracy analysis on our BMC prototype implementation that replayed example sequences extracted from the NZIX-II dataset [10] yielded high data-rates for some sequences and low rates for some others. We concluded that some traffic inter-arrival sequences are more suitable for time-replay covert channels simply because they provide a good separation of inter-arrival time values.

To counter time-replay channels, we investigated detection, prevention, and elimination techniques. Our analysis showed that for EMC-type channels, a motivated user can adjust the message encoder such that she can devise a channel that can potentially be undetectable through distribution analysis. We also argued that searching for such a transformation is more difficult for a BMC-type channel because of the way its rule set deterministically partitions the timing value set. We conjectured that whether a time-replay covert channel is detectable through distribution analysis depends on whether Alice can find and implement a suitable sampling scheme that preserves the distribution of the input sequence. Our analysis showed that certain systems can inherently avoid time-replay covert channels because of the uniformity of the inter-arrival times they generate (e.g., streaming traffic). We argued that one can mimic this behavior and avoid time-replay channels by creating systems that explicitly transform their event sequences into more uniform ones. Lastly, we investigated the efficacy of the existing channel eliminating schemes such as jammers [11] and the network pump [12] as applied on EMC and BMC-type channels. We argued that EMC-type channels can be easily eliminated with current schemes as they are very sensitive to changes in packet timing. For BMC-type channels, though, one needs to transform packet timings more intelligently to remove the associations between the timing values and symbols.

5. CONCLUSIONS AND FUTURE DIRECTIONS

A crucial step in covert channel design is to find and exploit a shared resource that is unlikely to be used as a communication medium by design. Little attention has been given to methods for hiding the traffic generated by these channels – the secrecy of a covert channel mostly relies on the secrecy of knowing the shared resource. In this dissertation, we showed that traffic analysis can counter traditional event-based covert channels, which do not employ any additional scheme to obfuscate the channel further, although rate-limited and more complex channels can be indistinguishable from normal channels through traffic distribution analysis. To prove this statement, we argued that once the shared resource is identified, covert traffic can often be distinguished from legitimate traffic because of the way in which event-based covert channels adjust the timing of the events to signal information. We first described our implementation of an IP covert channel prototype, discussed the subtle issues that arose in its design, and presented a discussion on its efficacy. Further, we showed that the regularity of a noiseless IP covert channel can be used to differentiate it from legitimate channels and presented two efficient online detection measures that provided detection rates over 95%. We repeated our analysis for "noisy" IP covert channels in which legitimate and covert traffic were mixed to obfuscate the covert traffic. We showed that our online detection measures failed to identify the covert channel with a reasonable false alarm rate for noise levels higher than 10%. For such channels, we investigated effective search mechanisms to locate regions of the traffic that appear to be covert channels. Such "local" methods are computationally expensive and are thus best deployed offline.

In the second part of the dissertation, we introduced IP time-replay covert channels (TRCC) that are a new covert channel family that hide channel traffic by replaying a previously recorded or a specifically engineered sequence of timing intervals. We presented different types of IP TRCCs, each of which utilized a different rule set to associate time values with the symbols from the encoded message (e.g., *zero* or *one* for a binary message). Our analysis showed that the binary-matching channel (BMC), which has only one rule to partition the timing value space into two, has many advantages over the other two alternatives we presented: the rule-matching channel (RMC) and the exact-matching channel (EMC). Accuracy analysis on our BMC prototype implementation that replayed example sequences extracted from the NZIX-II dataset [10] yielded high data-rates for some sequences and low rates for some others. We concluded that some traffic inter-arrival sequences are more suitable for time-replay covert channels simply because they provide a good separation of inter-arrival time values.

To counter time-replay channels, we investigated detection, prevention, and elimination techniques. Our analysis showed that for EMC-type channels, a motivated user can adjust the message encoder such that she can devise a channel that can potentially be undetectable through distribution analysis. We also argued that searching for such a transformation is more difficult for a BMC-type channel because the way its rule set deterministically partitions the timing value set. We conjectured that whether a time-replay covert channel is detectable through distribution analysis depends on whether Alice can find and implement a suitable sampling scheme that preserves the distribution of the input sequence. Our analysis additionally showed that certain systems can inherently avoid time-replay covert channels because of the uniformity of the inter-arrival times they generate (e.g., streaming traffic). We argued that one can mimic this behavior and avoid time-replay channels by creating systems that explicitly transform their event sequences into more uniform ones. Lastly, we investigated the efficacy of the existing channel eliminating schemes such as jammers [11] and the network pump [12] as applied on EMC and BMC-type channels. We argued that EMC-type channels can be easily eliminated with current schemes as they are very sensitive to changes in packet timing. For BMC-type channels, though, one needs to transform packet timings more intelligently to remove the associations between the timing values and symbols.

In summary, the contributions of this dissertation are:

1. We presented an implementation of an IP covert channel prototype that effectively transmits covert information over the network using IP packet timings.

2. We introduced novel detection measures that effectively detect both noiseless and noisy IP covert channels (and all event-based covert channels in general) using traffic analysis and force malicious users to either design more complex channels and/or rate-limit the channel to avoid detection.

3. We introduced time-replay covert channels (TRCC) that are a covert channel family that hides event-based channels, and argued that TRCCs are virtually undetectable through distribution analysis under certain assumptions.

4. We discussed prevention and elimination techniques for time-replay covert channels that aid current elimination schemes in stopping these channels.

Additionally, we provided simple bandwidth analysis for both unhidden and hidden IP covert channels and investigated the efficacy of detection and elimination techniques through an experimental study.

5.1 Future Directions

Future directions one might take to extend this work can be grouped into two categories: Designing more effective covert channels and increasing the efficacy of countermeasures against these channels. To design better simple and time-replay covert channels, one alternative is to employ better synchronization techniques that can potentially have a positive impact on channel accuracy. For example, one can employ more advanced solutions for combating synchronization errors caused by clock skew such as self-synchronizing codes and phase-locked loops (PLL). The former method employs encoding schemes that are specifically designed to detect the loss of synchronization and to recover from this state (i.e., resynchronization). The latter method is a closed-loop feedback circuit that is designed to track or synchronize an output signal with an input signal in frequency and phase [79]. Both methods were proved to be effective in communications and could be applied to our channels to achieve better accuracy. Further, one can implement faster packet generators that will allow the sender to use inter-arrival time sequences with shorter timing intervals. Additionally, one can extend this work by applying the same channel design principles to implement other examples of event-based covert channels that use different events to transmit information. As an example, one can use the inter-arrival times of CPU requests to design a simple or a time-replay covert channel and investigate the efficacy of the channel using techniques similar to the ones presented in this dissertation.

To improve the countermeasures against time replay covert channels, one can provide further analysis on detection, prevention, and elimination schemes. An interesting direction one could take is to investigate the cases in which Alice can devise a scheme to hide a binary-matching channel completely and the cases in which this is not possible. This depends on whether Alice can find a way to imitate the input sequence such that the output sequence shows similar behavior, regardless of the message transmitted. Another direction is to investigate schemes that can help prevent time-replay covert channels completely. To do so, one need to introduce techniques that transform the inter-event sequences in such a way that it is no longer possible to leak information using the differences in timing values (e.g., by making them uniform). Further, one needs to investigate the effects of these transformations on system performance to guarantee fair-scheduling of the events as an example.

LIST OF REFERENCES

LIST OF REFERENCES

[1] B. W. Lampson, "A note on the confinement problem," *Communications of the ACM*, vol. 16, pp. 613–615, October 1973.

[2] D. Bell and L. LaPadula, "Secure computer system: Unified exposition and multics interpretation," tech. rep., Belmont, MA, March 1976.

[3] G. J. Simmons, "The prisoner's problem and the subliminal channel," in *Proceedings of Crypto-83 Advances in Cryptography*, 1983.

[4] U. D. of Defense, "Trusted computer system evaluation criteria TCSEC," *DoD 5200.28-STD Washington: GPO:1985*, 1985.

[5] J. Patrick R. Gallagher, "A guide to understanding covert channel analysis of trusted systems," *National Computer Security Centre NCSC–TG–030*, no. Library No. S–240,572, 1993.

[6] M. Schaefer, B. B. Gold, R. Linde, and J. Scheid, "Program confinement in KVM/370," in *Proceedings of the 1977 ACM Annual Conference*, (Seattle, WA), pp. 404–410, October 1977.

[7] P. A. Karger and J. C. Wray, "Storage channels in disk arm optimization," in *Proceedings of the 1991 IEEE Computer Society Symposium of Research in Security and Privacy*, (Oakland, CA), pp. 52–61, May 1991.

[8] W.-M. Hu, "Reducing timing channels with fuzzy time.," *Journal of Computer Security*, vol. 1, no. 3-4, pp. 233–254, 1992.

[9] I. S. Moskowitz, "Variable noise effects upon a simple timing channel.," in *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pp. 362–372, 1991.

[10] W. R. group, "NZIX-II trace archive, data available at http://pma.nlanr.net/traces/long/nzix2.html."

[11] J. Giles and B. Hajek, "An information-theoretic and game-theoretic study of timing channels," *IEEE Transaction on Information Theory*, vol. 48, pp. 2455–2477, September 2003.

[12] M. H. Kang, I. S. Moskowitz, and D. C. Lee, "A network pump," *IEEE Transactions on Software Engineering*, vol. 22, no. 5, pp. 329–338, 1996.

[13] J. McHugh, "Covert channel analysis," tech. rep., December 1995.

[14] CC, *Common Criteria*. National Institute of Standards and Technology, 1998.

[15] M. Bishop, *Computer Security: Art and Science.* Addison Wesley Professional, 2002.

[16] L. Helouet, C. Jard, and M. Zeitoun, "Covert channels detection in protocols using scenarios," in *Proceedings of SPV'2003, Workshop on Security Protocols Verification*, 2003.

[17] I. S. Moskowitz and M. H. Kang, "Covert channels - Here to stay?," in *Proceedings of the Ninth Annual Conference on Computer Assurance*, (Gaithersburg, MD), pp. 235–244, National Institute of Standards and Technology, 1994.

[18] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the art of virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.

[19] S. B. Lipner, "A comment on the confinement problem," in *Proceedings of the Fifth ACM symposium on Operating systems principles*, (Austin, TX), pp. 192–196, ACM Press, 1975.

[20] H. Dietel, *An Introduction to Operating Systems.* Reading, MA: Addison-Wesley, revised 1st ed., 1984.

[21] B. D. Gold, R. R. Linde, and P. F. Cudney, "KVM/370 in retrospect," in *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, p. 13, 1984.

[22] S. N. Foley, "A model for secure information flow.," in *IEEE Symposium on Security and Privacy*, pp. 248–258, 1989.

[23] R. A. Kemmerer, "Shared resource matric methodology: An approach to identifying storage and timing channels," *ACM Transactions on Computer Systems*, vol. 1, pp. 256–277, August 1983.

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

[25] J. Millen, "20 years of covert channel modeling and analysis," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pp. 113–114, 1999.

[26] K. Ahsan, "Covert channel analysis and data hiding in TCP/IP," Master's thesis, University of Toronto, 2000.

[27] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proceedings of the 2002 Workshop on Multimedia Security at ACM Multimedia*, December 2002.

[28] J. W. G. III, "Countermeasures and tradeoffs for a class of covert timing channel," tech. rep., 1994.

[29] K. Eggers, "Characterizing network covert storage channels," in *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pp. 275–279, December 1988.

[30] P. A. Henry, "Covert channels provided hackers the opportunity and the means for the current distributed denial of service attacks," *CyberGuard Corporation*, 2000.

[31] C. Abad, "IP checksum covert channels and selected hash collision," tech. rep., 2001.

[32] M. Bauer, "New covert channels in HTTP: Adding unwitting web browsers to anonymity sets," in *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, (Washington, DC), pp. 72–78, ACM Press, 2003.

[33] S. Li and A. Ephremides, "A network layer covert channel in ad-hoc wireless networks," in *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, pp. 88–96, October 2004.

[34] Daemon9, "Project Loki," *Phrack Magazine*, vol. 49, August 1996.

[35] Daemon9, "Loki2 (the implementation)," *Phrack Magazine*, vol. 51, September 1997.

[36] S. Castro, "Covert channel tunneling tool," 2003.

[37] J. C. Smith, "Covert shells," *SANS Institute Information Security Reading Room*, November 2000.

[38] C. G. Girling, "Covert channels in LANs," *IEEE Transactions on Software Engineering*, vol. SE-13, February 1987.

[39] V. Hauser, "Placing backdoors through firewalls," *WindowsSecuriy.com*, May 1999.

[40] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proceedings of the 2005 Information Hiding Workshop*, May.

[41] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *Workshop on Privacy Enhancing Technologies*, vol. 2482, pp. 194–208, April 2002.

[42] T. Handel and M. Sandford, "Hiding data in the OSI network model," in *Proceedings of the First International Workshop on Information Hiding*, May 1996.

[43] J. Rutkowska, "The implementation of passive covert channels in the linux kernel," *Chaos Communication Congress*, December 2004.

[44] C. Rowland, "Covert channels in the TCP/IP protocol suite," *First Monday: Peer-reviewed Journal on the Internet*, vol. 2, no. 5, 1997.

[45] T. M. Dogu and A. Ephremides, "Covert information transmission through the use of standard collision resolution algorithms.," in *Proceedings of the Third International Workshop, IH'99*, pp. 419–433, 1999.

[46] G. J. Simmons, "Subliminal channels: Past and present," *European Transaction on Telecommunications*, vol. 5, no. 4, pp. 459–473, 1994.

[47] G. J. Simmons, "The history of subliminal channels," in *Proceedings of the First International Workshop on Information Hiding*, pp. 237–256, Springer-Verlag, 1996.

[48] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding – A survey," *Proceedings of the 1999 IEEE*, vol. 87, pp. 1062–1078, July 1999.

[49] D. E. Denning, "A lattice model of secure information flow," *Commun. ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[50] P. A. Porras and R. A. Kemmerer, "Covert flow trees: A technique for identifying and analyzing covert storage channels," May 1991.

[51] C.-R. Tsai, V. Gligor, and C. Chandersekaran, "A formal method for the identification of covert storage channels in secure XENIX," in *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April 1987.

[52] C. Wang and S. Ju, "Searching covert channels by identifying malicious subjects in the time domain," in *Proceedings of the Fifth Annual IEEE SMC Information Assurance Workshop*, May 1991.

[53] C.-R. Tsai and V. Gligor, "A bandwidth computation model for covert storage channels and its applications," in *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pp. 108–121, 1988.

[54] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication.* Urbana, IL: The University of Illinois Press, 1964.

[55] J. Millen, "Finite-state noiseless covert channels," in *Proceedings of the 1989 Computer Security Foundations Workshop*, pp. 81–85, 1989.

[56] I. S. Moskowitz and A. R. Miller, "Simple timing channels," in *Proceedings of the 1994 IEEE Symposium in Security and Privacy*, pp. 56–64, May 1994.

[57] O. L. Costich and I. S. Moskowitz, "Analysis of a storage channel in the two-phase commit protocol," in *Proceedings of the Fourth IEEE Computer Security Foundations Workshop (CSFW'91)*, pp. 201–208, June 1991.

[58] I. S. Moskowitz and A. R. Miller, "The channel capacity of a certain noisy timing channel.," *IEEE Transactions on Information Theory*, vol. 38, no. 4, pp. 1339–, 1992.

[59] S. H. Son, R. Mukkamala, and R. David, "Integrating security and real-time requirements using covert channel capacity.," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 6, pp. 865–879, 2000.

[60] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems.* Reading, MA: Addison-Wesley, 1987.

[61] S. W. Golomb, "The limiting behavior of the Z-channel," *IEEE Transactions on Information Theory*, p. 372, May 1980.

[62] I. S. Moskowitz, S. J. Greenwald, and M. H. Kang, "An analysis of the timed Z-channel," in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, (Washington, DC), p. 2, IEEE Computer Society, 1996.

[63] S.-P. Shieh, "Estimating and measuring covert channel bandwidth in multilevel secure operating systems.," *J. Inf. Sci. Eng.*, vol. 15, no. 1, pp. 91–106, 1999.

[64] J. T. Trostle, “Modelling a fuzzy time system.,” *Journal of Computer Security*, vol. 2, no. 4, pp. 291–310, 1993.

[65] J. W. G. III, “On introducing noise into the bus contention channel,” in *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pp. 90–98, IEEE Press, 1993.

[66] J. W. G. III, “On analyzing the bus-contention channel under fuzzy time.,” in *Proceedings of the 1993 Computer Security Foundations Workshop (CSFW)*, pp. 3–9, 1993.

[67] M. H. Kang and I. S. Moskowitz, “A pump for rapid, reliable, secure communication,” in *Proceedings of the First ACM Conference on Computer and Communications Security*, (Fairfax, VA), pp. 119–129, ACM Press, 1993.

[68] M. H. Kang and I. S. Moskowitz, “A data pump for communication,” tech. rep., Washington, DC, 1995.

[69] M. H. Kang, A. P. Moore, and I. S. Moskowitz, “Design and assurance strategy for the NRL pump.,” *IEEE Computer*, vol. 31, no. 4, pp. 56–64, 1998.

[70] B. Montrose and M. H. Kang, “An implementation of the pump: The event driven pump,” tech. rep., May 1996.

[71] M. H. Kang, I. S. Moskowitz, and D. C. Lee, “A network version of the pump,” in *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, (Washington, DC), p. 144, IEEE Computer Society, 1995.

[72] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, “Eliminating steganography in Internet traffic with active wardens,” in *Proceedings of the Fifth International Workshop on Information Hiding*, vol. 2578, pp. 18–35, October 2002.

[73] M. Handley and V. Paxson, “Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics,” in *Proceedings of the 10th USENIX Security Symposium*, August 2001.

[74] S.-P. Shieh and V. D. Gligor, “Auditing the use of covert storage channels in secure systems.,” in *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pp. 285–295, 1990.

[75] T. Sohn, J. Moon, S. Lee, D. H. Lee, and J. Lim, “Covert channel detection in the ICMP payload using support vector machine.,” in *ISCIS*, pp. 828–835, 2003.

[76] T. Sohn, J.-T. Seo, and J. Moon, “A study on the covert channel detection of TCP/IP header using support vector machine.,” in *ICICS*, pp. 313–324, 2003.

[77] C. Berrou, A. Glavieux, and P.Thitimajshima, “Near shannon limit error-correcting coding and decoding: Turbo codes,” in *Proc. IEEE Int. Conf. Comm.*, pp. 1064–1070, 1993.

[78] R. Hamming, “Error detecting and correcting codes,” *The Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, 1950.

[79] R. E. Best, *Phase-locked loops: Design, simulation and applications.* McGraw-Hill Professional, 5th ed., 2003.

[80] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21, pp. 168–173, January 1974.

[81] UN, "Universal declaration of human rights," vol. 217A, no. 3, 1948.

[82] M. Li and W. Lampson, *An introduction to kolmogorov complexity and its application*. Springer, 2nd ed., 1997.

[83] M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi, "The similarity metric," in *Proceedings of the 14th annual ACM-SIAM Symposium on Discrete algorithms*, pp. 863–872, Society for Industrial and Applied Mathematics, 2003.

[84] R. Gusella, "Characterizing the variability of arrival processes with indexes of dispersion," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 203–211, February 1991.

[85] C. Rosenberg, F. Guillemin, and R. Mazumdar, "New approach for traffic characterisation in ATM networks," in *IEE Proceedings - Communications*, vol. 142, pp. 87–90, April 1995.

[86] D. R. Cox and P. A. W. Lewis, *The statistical analysis of series of events*. Chapman and Hall, 1966.

[87] V. Paxson and S. Floyd, "Wide area traffic: The failure of Poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, 1995.

[88] H. Balakrishnan, M. Stemm, S. Seshan, and R. H. Katz, "Analyzing stability in wide-area network performance," in *Measurement and Modeling of Computer Systems*, pp. 2–12, 1997.

[89] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Application of sampling methodologies to network traffic characterization," in *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, (New York, NY, USA), pp. 194–203, ACM Press, 1993.

[90] GNU, "GNU zip, available online at http://www.gzip.org."

[91] D. Sculley and C. E. Brodley, "Compression and machine learning: A new perspective on feature space vectors," in *To Appear in Data Compression Conference (DCC)*, 2006.

[92] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards parameter-free data mining," in *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (Seattle, WA), pp. 206–215, 2004.

[93] R. Cilibrasi, P. M. B. Vitanyi, and R. D. Wolf, "Algorithmic clustering of music based on string compression," *Comp. Music Journal*, vol. 28, no. 4, pp. 49–67, 2004.

[94] R. Cilibrasi and P. M. B. Vitanyi, "Clustering by compression," *IEEE Trans. Information Theory*, vol. 51, no. 4, 2005.

[95] S. Wehner, "Analyzing network traffic and worms using compression," 2004.

[96] D. Loewenstem, H. Hirsh, P. Yianilos, and M. Noordewier., "DNA sequence classification using compression-based induction," tech. rep., April 1995.

[97] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes.* McGraw-Hill, 1991.

[98] S. Goldwasser and S. Micali, “Probabilistic encryption,” *Journal of Computer and Systems Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

APPENDICES

A. IP SCC SCENARIOS

Scenario I: To observe packet presence, Eve can monitor the shared communication line at MAC level to see if there is any packet collision. If there is such collision, she can deduce that Alice has issued a packet.

Scenario II: Another option is to monitor the shared communication line between Alice and Bob at the physical level and observe the signal. If there is a high signal, Eve can deduce that Alice has issued a packet.

Scenario III: A relatively different scenario is to use the TCP/IP acknowledgment (ACK) packets in MLS systems that allow Alice to notify the successful receipt of a packet sent by Eve to Alice (write-up). In this case, the timing of these ACK packets can be used to create a covert channel.

VITA

VITA

Serdar Cabuk was born in Ordu, Turkey on June 16, 1979. He earned his bachelor's degree in computer engineering from Bogazici (Bosphorus) University, Istanbul, Turkey in 2001. He enrolled in the Electrical and Engineering Department at Purdue University in 2002 and started his research under the guidance of Dr. Carla E. Brodley and Dr. Eugene H. Spafford. He was awarded a Fulbright Scholarship for the 2002 and 2003 academic years and earned his Ph.D. degree in electrical and computer engineering from Purdue University in 2006.