The Hong Kong Polytechnic University

Department of Computing

Two Classes of Novel TCP Exploits and the Countermeasures

by

LUO XIAPU

A thesis submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

June 2007

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signature)

LUO XIAPU (Name of Student)

To my beloved parents.

ABSTRACT

This thesis presents two classes of novel TCP exploits and the countermeasures. In the first exploit, we have proposed a new breed of low-rate Denial-of-Service (DoS) attacks, referred to as *pulsing DoS* (PDoS) attacks, which abuse TCP congestion control mechanisms to throttle a victim's throughput. Comparing with traditional flooding-based DoS attacks, PDoS attacks use much less attack traffic to cause similar damage to TCP flows. Besides TCP, the dominant transport protocol today, the emerging SCTP and DCCP will also be vulnerable to PDoS attacks. On the defense side, we have proposed two new effective schemes to detect PDoS attacks. In the second exploit, we have proposed two novel network timing channels, *TCPScript* and *Cloak*, which facilitate stealthy communications in the Internet. By exploiting TCP's flow concept, sliding window, and acknowledgement mechanisms, TCPScript and Cloak provide much higher channel capacity, camouflage flexibility, and reliability than existing covert channels. Since the protocol features exploited by TCPScript and Cloak are widely adopted by modern transport protocols, similar covert channels could be imbedded in other protocols. We have also proposed new detection schemes to uncover TCPScript and Cloak channels.

In the PDoS attacks, we have fully exploited TCP congestion control mechanisms to effectively deny TCP flows from using the available bandwidth. Unlike traditional flooding-based attacks, the PDoS attack sends out a train of attack pulses, each of which will cause packet drops at the affected routers. Due to TCP's additive-increase/multiplicative-decrease and timeout mechanisms, the periodic packet losses will cause the TCP victim's throughput to stay at a very low value. We have evaluated the effectiveness of the PDoS attack on popular TCP variants, TCP-friendly protocols, and active queue management schemes based on analytical modeling and

test-bed experimentations. Since PDoS attacks could be configured in their intensity and periodicity of the attack pulses, we have also studied the tradeoff between attack damage and attack cost. Subsequently, we have optimized the PDoS attack to achieve the best tradeoff. Finally, we have generalized the PDoS attacks, other low-rate DoS attacks, and flooding-based attacks under a single framework: polymorphic DoS attacks.

On the countermeasures, we have designed a two-stage detection mechanism for PDoS attacks and Vanguard for PMDoS attacks. The two-stage detection mechanism is designed to detect PDoS attacks at the network under protection. It employs a wavelet analysis to monitor the variability in the incoming TCP data traffic and outgoing TCP acknowledgment traffic. The second stage is to detect the attack based on change-point detection of the monitored statistics in the first stage. Vanguard, on the other hand, is designed to detect different forms of DoS attacks, i.e., the PMDoS attacks. As a result, Vanguard uses three anomalies for an accurate detection and for reducing false positives and false negatives. We have conducted extensive experiments on a test bed to evaluate their performance in terms of detection accuracy and computational requirement, and have compared them with several detection systems proposed by others.

In the second class of TCP exploits, we have fully utilized TCP's protocol features to design more effective network timing channels. The first idea is to exploit TCP's bursty traffic for imbedding covert messages. In particular, we have designed TCPScript to imbed messages in the TCP data burst size. Moreover, we have built into TCPScript additional mechanisms based on TCP acknowledgements to increase its channel reliability. The second idea is to imbed covert messages in the packet-flow combinations which are used in the design of Cloak. Cloak possesses many outstanding properties which cannot be achieved by existing network timing channels, including high channel throughput, full reliability, and very high flexibility. For the proof of concept, we have prototyped TCPScript and Cloak, and have evaluated them in a test bed and PlanetLab. Experiment results have showed that TCPScript and

Cloak enjoy better performance as compared with two other network timing channels. On the countermeasures, we have proposed new detection schemes for detecting TCP-Script and Cloak channels which are based on identifying anomalies in the TCP data and acknowledgment traffic. We have evaluated the detection rates of the schemes based on public traces.

ACKNOWLEDGMENTS

This thesis would not have been possible without the help of many people. First and foremost, I would like to express sincere gratitude to my supervisor, Prof. Rocky K. C. Chang, for his guidance, support and encouragement throughout my graduate studies. I really appreciate his insightful comments, constructive criticism, and the freedom he gave to develop my own interests. He always patiently listens to my diverse thoughts and helps me extract the essentials. He is always there when I feel frustrated and inspires me to keep moving forward. I have learned a lot from his expertise, rigorous research attitude, enthusiasm and dedication to high-quality research.

I am also grateful to my thesis committee members: Prof. Wenke Lee of Georgia Tech, Prof. Wing-Cheong Lau of the Chinese University of Hong Kong, and Prof. Henry C. B. Chan of the Hong Kong Polytechnic University for their insightful comments and suggestions to this dissertation and my research.

I am deeply indebted to my parents and my elder brother for their endless love and unconditional support. They have made countless efforts to let me focus on research and pursue my dream without worrying about anything else. They deserve all the credits for what I have achieved and what I may in the future.

I would like to express my appreciation to the following professors who have taught me courses and have given me useful suggestions to doing research: Prof. Daniel So Yeung, Prof. Eric C. C. Tsang, Prof. Will W. Y. Ng, Prof. Jiannong Cao, Prof. John C. S. Lui, and Prof. Qian Zhang.

I would like to thank all the members in the Internet Infrastructure and Security Research Group: Edmond W. W. Chan, Grace Xie Yi, Sam Lam Sum, Kathy Tang Yajuan, Samantha S. M. Lo, Steve W. K. Poon, Hu Guangmin, Le Yu, and Zhou

Wenchao. I appreciate very much for their collaborations, feedbacks, discussions, and support. I treasure their friendship and the memorable time spent together.

I want to acknowledge my fellow colleagues and friends who shared with me the pleasure as well as the pain of postgraduate study: Tang Jianhui, Wu Weigang, Yang Jin, Ding Xiaoping, Zheng Yuan, Cheng Hui, Kirk H. M. Wong, Wing W. Y. Ng, and Patrick P. K. Chan.

The research conducted in this thesis was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. PolyU 5080/02E), a grant from the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99), and a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ACK	Pure acknowledgement in TCP
AIMD	Additive-increase/multiplicative-decrease
AQM	Active queue management
AVQ	Adaptive Virtual Queue
BER	Bit error rate
BGP	Border Gateway Protocol
CBR	Constant bit rate
CUSUM	Cumulative sum
cwnd	Congestion window
DCCP	Datagram Congestion Control Protocol
DDoS	Distributed denial-of-service
DNS	Domain Name System
DoS	Denial-of-service
DWTM	Discrete wavelet transform
DTWP	Dynamic time warping
FDDoS	Flooding-based DoS
FR	Fast retransmit and fast recovery
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISN	Initial sequence number
LBNL	Lawrence Berkeley National Laboratory
minRTO	Minimal retransmission timeout (i.e., 1 second)
MAC	Media access control

MSS	Maximum segment size
NAT	Network address translation
PA-PDoS	Periodic AIMD-PDoS
PDoS	Pulsing denial-of-service
PI	Proportional integral
PLR	Packet loss rate
PMDoS	Polymorphic DoS
PSD	Power spectrum density
PT-PDoS	Periodic timeout-PDoS
RED	Random Early Detection
RED-PD	RED-preferential dropping
REM	Random Exponential Marking
RoQ	Reduction-of-quality
RTO	Retransmission timeout
RTT	Round-trip time
rwnd	Receive window
SCTP	Stream Control Transmission Protocol
ssthresh	Slow start threshold
TCP	Transmission control protocol
TO	Timeout
TTL	Time-to-live
UDP	User Datagram Protocol
WLAN	Wireless local area network

1. INTRODUCTION

Old and new security problems continue to plague the Internet and its communities. More security measures are constantly introduced to network devices, systems, and software; however, attacks are also becoming more sophisticated at the same time. Addressing network security problems is therefore the main driving force behind the recent initiative for a "clean-slate" design of the Internet [1]. This initiative implicitly admits that it is not possible to address the security problems confronting the current Internet which was not designed for security. However, it is too early to tell whether a new Internet design, if any, could completely eliminate these problems. For instance, many security problems in network protocols are inherent in nature; according to our experience, network protocols could *always* be exploited for purposes other than the originally intended.

In particular, this thesis argues that transport-layer protocols, if cleverly exploited, could be used for devising powerful network attacks. We substantiate this argument by demonstrating that TCP's rich protocol features and behavior can be misused for two kinds of Internet-wide security problems: denial-of-service (DoS) attacks and network covert communications. Although the actual number of reported DoS or distributed DoS (DDoS) has dropped in 2006, this threat remains as one of the most serious threats to the Internet, according to the CSI/FBI Computer Crime and Security Survey in 2004 and 2006 report [2] and the Symantec Internet security threat report [3]. We propose in this thesis a new and more effective DoS attack, coined as *pulsing denial-of-service* (PDoS) attack, because its ON/OFF attack pattern resembles pulses in the field of signal processing [4]. Unlike the classic flooding-based attacks, the PDoS attacks TCP congestion control algorithm by introducing intermittent false feedback signals. Therefore, it could achieve similar damage achieved by the classic DoS attacks but with much less attack traffic. We have also designed two

online algorithms to detect an ensuing PDoS attack and evaluated its performance on a test bed.

Although receiving relatively fewer reports as compared with DoS attacks, network covert communications could pose a serious threat to the Internet security. Covert channels have been used to steal information, for example, password [5] and data [6]. Covert channels have also been used to communicate attack commands in the previous large-scale DDoS attacks [7,8], to transfer infection information in a new class of worms [9], and to control botnets [10, 11]. A recent report also points out that the use of network covert channels for concealing malicious activities is on the rise [12]. On the other hand, covert channels could be instrumental in permitting different communities to bypass censorship or any privacy intrusion devices [13, 14]. In this thesis, we present two new channels—TCPScript and Cloak—which exploit TCP's traffic burstiness, acknowledgments, and packet-flow combinations for embedding covert messages. Both possess many attractive properties unmatched by others, such as a high channel capacity, resilience to varying network conditions, and a high detection cost. On the countermeasures, we have designed new algorithms to detect the two network covert channels. We have thoroughly evaluated the covert channels and detection algorithms on a test bed and in the PlanetLab.

1.1 TCP insecurity

Transmission Control Protocol (TCP), being the dominant transport protocol for numerous application protocols, such as HTTP, FTP, Telnet, SSH, and SSL, is perhaps one of the most complex Internet protocols. TCP provides a wide range of services to a TCP connection, such as full-duplex communication, reliability, message orderliness, and flow control. Moreover, the TCP congestion control prevents congestion collapse in the Internet, provides fairness for best-effort traffic, and optimizes performance on throughput, delay, and loss [15]. TCP provides all of the above

through a sliding window algorithm, packet sequencing, positive acknowledgment, retransmission algorithms, and congestion control algorithms.

Like other TCP/IP protocols, TCP was designed at the time when network security was not a concern and trust was assumed. Indeed, the original TCP design [16] suffers from a number of inherent limitations and vulnerabilities that have been exploited by various attacks [17–21]. Some attacks aim at TCP connections. For example, RST and FIN attacks attempt to tear down normal TCP connections [17]), and the *Land* attack employs malformed packets to crash some earlier TCP/IP stacks [20]. Moreover, other TCP attacks abuse TCP features for the purpose of bypassing firewalls and censorship, and fingerprinting remote systems. We will present a detailed survey on different kinds of TCP exploits in Chapter 2.1.3.

1.2 Denial-of-service attacks

Unlike system-specific attacks, DoS attacks are more generic in nature. Different network protocol packets can be used in an attack, and the attacks are generally system independent. Therefore, their impact can be very significant in scope and damage, as evidenced by the large-scale attacks in 2000 that caught Yahoo, eBay, and many other major web sites off guard [22]. The conventional (D)DoS attacks are flooding-based [22]. That is, an attacker sends out an unusually large number of packets to overwhelm a victim from a single or multiple infected hosts. These attack packets exhaust the victim's bandwidth or system resources, preventing them from serving legitimate requests. Detecting them, however, is generally not difficult (except for the SYN flooding attack) on the victim's side based on an anomalous rise in the traffic rate and slow network responses.

We classify the existing TCP vulnerabilities that can be exploited by DoS attackers into three categories. The first category covers the protocol design weaknesses. For example, a SYN flooding attack could deplete a system's memory for accepting new connections by not completing the TCP three-way handshake [23, 24]. The second

category concerns TCP/IP stack implementations [25]. For example, the *Bubonic* attack could induce a high CPU/memory utilization and then a DoS attack against some TCP/IP stacks by crafting a sequence of TCP packets with random settings [20]. The third category concerns injections of false congestion signals into a TCP flow. For example, an attacker could send forged ICMP Source Quench messages to a TCP sender to reduce its transmission rate [26]. The PDoS attack proposed in this thesis belongs to the third category. Note that this vulnerability is also closely related to the design of TCP congestion control algorithms [27].

1.3 Low-rate TCP-targeted DoS attacks

The main principle for the PDoS attack is to inject false congestion signals each of which will cause the sender's congestion window to drop. If the congestion signals are spaced appropriately, the congestion window will be suppressed to a low value, thus reducing the TCP throughput. To induce those false congestion signals, a PDoS attacker dispatchs a sequence of packet bursts, or pulses, to temporarily congest the bottleneck that will cause packet drops in legitimate TCP flows. As a result, the main advantage of the PDoS attack is that its average traffic rate is usually much lower than the bottleneck capacity, which also makes it hard to detect the attack. Moreover, this type of *low-rate DoS attacks* has been studied under two other names: shrew attack [28] and reduction-of-quality (RoQ) attack [29]. As we shall explain later, the PDoS attack includes the shrew attack as a special case. Moreover, we have analyzed general PDoS attacks which admit nonconstant inter-pulse period, whereas only constant periods are considered in [29].

The seminal work on low-rate DoS attacks began with the shrew attack that attempts to confine a TCP sender to the timeout state by dispatching attack pulses whenever the sender retransmits lost packets after a timeout period [28]. By exploiting the pre-defined minimal retransmission timeout value (RTO) (i.e., 1 second) suggested in [30], an attacker could predict a TCP sender's behavior and launch a shrew attack

with a fixed period. The RoQ attack, on the other hand, sends periodic attack pulses to force a router's active queue management (AQM) mechanism to enter the transient state, thus increasing its packet loss probability. Consequently, the throughput of TCP flows passing through the router will be degraded due to the attack-inflated loss probability [29]. The analysis in [29] addresses mainly RED-like AQMs and the effects of transient state on the TCP throughput. Note that our work on the PDoS attacks was done in parallel with the work on RoQ attacks; we were not aware of the RoQ attacks [29] when publishing our first paper on the PDoS attacks [4].

1.4 Network covert channels

Encryption prevents unauthorized access to protected communications, whereas a covert channel conceals the existence of communication [31]. Since its debut as a confinement problem in operating systems in 1973 [32], covert channels have attracted considerable interest from both academic and hackers. The trusted computer system evaluation criteria (TCSEC) defines a covert channel as "any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy" [33, 34].

Traditionally, covert channels are classified into two groups: storage channel and timing channel. The former 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." The latter allows "one process signals information to another process by modulating its own use of system resources in such a way that this manipulation affects the real response time observed by the second process" [33]. However, there is no fundamental distinction between the two in theory [34]. A mixed or hybrid channel is a combination of the two [35]. Venkatraman and Newman-Wolfe gave a specific definition for network covert channels: "they implement a valid interpretation of a consistent security policy and are based on the observation of the extrinsic characteristics of the communication without necessarily having access to the information

contained within messages (due to encryption) or the necessity to modulate internal states or variables" [36].

Network covert channels operate in a similar manner as the classic covert channels in trusted computer systems. However, there are marked differences between the two. First, the "high" or "low" level processes used in the the classic covert channels are not well defined in network covert channels. For example, when a compromised host tries to leak information to a remote attacker, the host may be considered as a "high" process. However, if a compromised host is receiving malicious commands from a remote attacker, then the host may be regarded as a "low" process. Second, there are potentially uncountable entities that could engage in network covert communications, but the number is quite limited in the classic covert channels. Third, implementing the Bell LaPadula model for network covert channels is practically very difficult. Most enterprise networks do not implement the Bell LaPadula and permit two-way communications to support various TCP-based applications. For example, most network administrators may allow users to browse Web sites but they will inspect the outgoing and incoming traffic.

1.5 TCP-based covert timing channels

Most network storage channels hide information in the header fields of different protocols (e.g., MAC, IP, TCP, ICMP, DNS, and HTTP [37, 38]). However, these channels could be thwarted by active warden [39], protocol scrubber [40] and middlebox (e.g., NAT) which may modify the values in those fields. On the other hand, some recently proposed channels convey information in the packets' timing information. For example, Cabuk et al. [41] have proposed an IP timing channel, in which an IP packet arrival during a timing interval is decoded as `1` and its absence decoded as `0`. Berk et al. [42] have considered using inter-packet delay of ICMP packets to encode one or multiple bits: bit `1` is encoded by a longer inter-packet delay and bit `0` encoded by a smaller inter-packet delay. Shah et al. [43] have recently proposed

JitterBug, another timing channel modulating binary bits into the inter-packet delay. Although these timing channels are more robust to the network intermediaries, they suffer from three serious problems: low bit rate, deviation of the modulated packets' behavior from the normal profile, and being vulnerable to adverse network conditions, such as packet loss and large jitter.

In this thesis we have explored other parts of the design space for designing network covert channels. As a result of the exploration, we have designed two new network covert channels: *TCPScript* and *Cloak*. TCPScript imbeds covert messages in the burstiness of TCP data traffic. In particular, TCPScript uses a packet-counting approach to encode multibit messages and takes the advantages of TCP's sequence number and acknowledgement mechanism to increase its robustness to adverse network conditions. Moreover, we have proposed an information-theoretic model to estimate TCPScript's capacity and have compared it with IP timing channel's and Jitterbug's. We have conducted extensive experiments in a test bed and the PlanetLab platform [44]. The experiment results confirm that TCPScript offers not only much higher throughput but also more reliable service than the other two timing channels. On the countermeasures, we have proposed a new online detection approach to discover ongoing TCPScript instances which has also been evaluated experimentally.

Cloak, another new timing channel, takes an entirely different approach to convey covert messages. Each message in Cloak is encoded into a unique combination of TCP flows and packets. In other words, a partition of packets into TCP flows represents a hidden message. Cloak has many attractive properties that cannot be found in other network covert channels. First of all, Cloak provides reliable service in the same manner as TCP: each covert message transmission is guaranteed reliable. Second, Cloak offers *ten* different encoding and decoding methods each of which has a unique tradeoff among several important considerations, such as the channel capacity and the need for packet marking. Third, the packet transmissions modulated by Cloak could be carefully crafted to mimic the normal TCP flows in a typical TCP-based application session. Extensive experiments conducted in a test bed and PlanetLab

platform show that Cloak outperforms other timing channels in terms of throughput under various network conditions. Moreover, we have designed and evaluated a two-step detection algorithm for Cloak.

1.6 Contributions of this work

1.6.1 Pulsing DoS attacks and the countermeasures

1. We have proposed and analyzed a new class of DoS attacks: pulsing DoS (PDoS) attacks. Unlike traditional attacks, a PDoS attack effectively throttles TCP throughout by sending false feedback signals. We have also designed a practical two-stage mechanism to detect them [4, 45].

   The PDoS attacks exploit TCP congestion control mechanisms, including time-out and additive-increase/multiplicative-decrease (AIMD) algorithm. The widely analyzed shrew attack is a special case of the PDoS attacks. We have derived analytical models for the damage caused by different PDoS attack variants. Moreover, PDoS attacks will affect other TCP-friendly protocols, such as SCTP and DCCP which adopt similar congestion control mechanisms.

   The two-stage detection mechanism is based on two traffic anomalies caused by a PDoS attack: periodic fluctuations in the incoming TCP data traffic and a decline in the trend of the outgoing ACK traffic. The computation complexity of this two-stage detection mechanism is lower than other existing detection schemes. The test-bed experiment results also show that it can promptly detect PDoS attacks.

2. We have generalized the PDoS, other low-rate attacks, and the flooding DoS attacks under a single framework: polymorphic DoS (PMDoS) attacks. We have proposed Vanguard, a new detection method to detect PMDoS attacks [46].

   The PMDoS attack is a general framework for the low-rate attacks and flooding-based attacks. In particular, we have modeled the AIMD-based attack as an

alternating renewal process and analyzed its impact under different parameter settings. This unified framework is very useful for comparing different low-rate attacks and for understanding various tradeoff involved. It is also possible to identify new design points which could yield more powerful attacks.

We have proposed Vanguard to detect various forms of low-rate attacks and flooding-based attacks (i.e., PMDoS attacks). Vanguard employs three anomalies to discover PMDoS attacks: decline in the outgoing TCP ACK traffic, increase in the ratio of the incoming TCP traffic to the outgoing TCP ACK traffic, and change in the distribution of the incoming TCP traffic. Vanguard enjoys a low computational complexity as the two-stage detection scheme. The experiment results support that Vanguard can promptly detect many PMDoS attacks.

3. We have investigated the problem of optimizing PDoS attacks and have obtained insightful understanding on the tradeoff between attack power and attack cost [47]. As a result, we could distinguish between three types of PDoS attacks based on this tradeoff.

   An optimal PDoS attack balances the attack power (measured by TCP throughput degradation) and the attack cost quantified (measured by attack traffic rate). By varying the attack parameters, we could study the three types of attacks launched by risk-loving, risk-neutral, and risk-averse attacks. Moreover, we have obtained the optimized attack parameters for the three classes of attacks. We have validated the analytical results using both ns-2 simulation and test-bed experiments.

Although we have primarily considered attacking standard TCP protocols, our models and analysis could be extended to new TCP flavors and even nonTCP protocols, such as SCTP [48] and DCCP [49]. In fact, several works have already investigated the impact of PDoS attacks on FAST TCP flows [50, 51] and UDP-based VoIP flows [52]. Chertov et al. have evaluated the impact of PDoS attacks experimentally

on the DETER and Emulab test beds [53]. Moreover, recent studies have assessed the effects of PDoS attacks on TCP-based applications. For example, Zhang et al. report that PDoS attacks could cause session resets and delayed routing convergence in Border Gateway Protocol (BGP) [54]. Mirkovic et al. show that the PDoS attacks could induce a high failure ratio, a metric for measuring DoS attack's effect [55]. We will further discuss these related works in Chapter 2.

1.6.2 Network covert channels and the countermeasures

1. We have proposed TCPScript, a new approach to designing timing channels that exploit traffic burstiness in reliable, end-to-end protocols. We have also identified two anomalies for detecting TCPScript channels.

   A TCPScript channel camouflages itself in bursty TCP traffic and embeds information into the number of packets in each data burst. TCPScript increases its reliability by further exploiting TCP's acknowledgement mechanism. Moreover, we have proposed an information-theoretic model for estimating TCPScript's capacity and comparing it with an IP timing channel and Jitterbug. Extensive experiments conducted in both a test bed and PlanetLab support that TCPScript attains a higher throughput and is more robust to adverse network conditions, as compared with the IP timing channel and JitterBug. Moreover, we have proposed a new detection scheme for TCPScript based on two features: burst size and inter-ACK-data delay.

2. We have proposed Cloak, a class of novel timing channels which embed covert messages into a combination of TCP flows and packets [56]. We have also proposed an anomaly detection algorithm to uncover Cloak channels.

   Cloak is fundamentally different from other timing channels in several aspects. First, Cloak encodes a message by a unique distribution of N packets over X TCP flows. The combinatorial nature of the encoding methods increases the channel capacity exponentially with (N, X). Second, Cloak offers ten different

encoding and decoding methods, each of which has a unique tradeoff among several important considerations, such as channel capacity and the need for packet marking. Third, the packet transmissions modulated by Cloak could be carefully crafted to mimic the normal TCP flows in a typical TCP-based application session. Although Cloak's basic idea is simple, we will show how we have tackled a number of challenging issues systematically. Our experiment results collected from PlanetLab nodes and a test bed suggest that Cloak is feasible under various network conditions and different round-trip. We have proposed and implemented an anomaly detection algorithm for uncovering Cloak channels.

Since sequence number, acknowledgement, packet-flow are general concepts in communication protocol design [57] and have been adopted by many new protocols (e.g., DCCP and SCTP), we believe that the basic ideas of TCPScript and Cloak could be applied to other protocols. Moreover, Cloak's combination-based framework can be used to design new storage channels.

1.7 Organization

The rest of this thesis consists of four main sections: Chapter 2 on background and related works, Chapters 3-4 on PDoS and the countermeasures, Chapters 5-6 on two new network covert channels and countermeasures, and finally Chapter 7 on conclusions and future works.

In Chapter 2, we first provide relevant background on TCP congestion control mechanism, flow control, and existing TCP exploits. After that, we survey previous works on low-rate DoS attacks and the countermeasures. We will explain why the PDoS attacks proposed in this thesis are more powerful and difficult to detect. We will then present the covert communication problem and the model assumed in this thesis. Besides pointing out some classic works done for this problem, we propose a new taxonomy for network storage channels which could also help identify new

ones. Lastly, we introduce the state-of-the-arts on network timing channels and the countermeasures.

Chapters 3-4 are devoted to the PDoS attacks and the two new detection systems. In Chapter 3, we first present and analyze a suite of PDoS attacks that exploit TCP's timeout and AIMD mechanisms in a synchronous or asynchronous manner. To go further, we generalize the PDoS attacks, other low-rate attacks, and the traditional flooding-based by a general framework referred to as polymorphic DoS (PMDoS) attack—a DoS attack existing in different forms. Finally, we have obtained optimal attack parameters for three different attack profiles. In Chapter 4, we present a two-stage mechanism for detecting PDoS attack and Vanguard for detecting PMDoS attacks.

Chapters 5-6 are devoted to TCPScript and Cloak and the anomaly detection algorithms. In Chapter 5, we will first discuss the basic design of TCPScript which embeds covert messages into the burst size. We will then further improve TCPScript's decoding accuracy using TCP's acknowledgment channel and a clustering algorithm. We finally end the chapter with experiment results and the anomaly detection algorithm. In Chapter 6, we detail the design of Cloak and its ten different encoding and decoding methods. After that, we address a practical head-of-line blocking problem by introducing a D-limited codeword scheme. We then present extensive test-bed and PlanetLab experiment results and conclude the chapter with the anomaly detection algorithms.

2. BACKGROUND AND RELATED WORKS

In this chapter, we will first present a short primer on TCP protocol that is relevant to this work, traditional TCP exploits, and a few recently proposed TCP-based attacks. After that, we will explain the covert communication problem in the Internet and present a brief survey on the common approaches to designing two main types of network covert channels: storage channels and timing channels.

2.1 A TCP primer

In this section, we briefly introduce some of TCP's salient features (i.e., reliability, flow control, and congestion control), which will be exploited for designing PDoS attacks and new network covert channels. Interested readers may refer to [16, 27, 58] for detailed information about TCP specifications in the standard documents. Moreover, we will summarize in section 2.1.3 existing TCP exploits, including those discovered in this thesis.

2.1.1 Reliability and flow control

TCP uses a quadruple—source and destination addresses, and source and destination port numbers—to uniquely distinguish a connection. Each connection is established after a successful three-way handshaking and is closed by a successful four-way handshaking or an RST signal.

In order to provide reliable transmission service, a TCP sender assigns a sequence number to each byte sent and requires acknowledgement (ACK) from the TCP receiver. If a TCP sender cannot receive the ACK within a timeout interval or receive three duplicate ACKs, it will retransmit the data [16, 27]. A TCP receiver uses the

sequence number to remove duplicates and to order data before passing them to the upper-layer applications.

TCP adopts a window-based flow control mechanism. The receiving TCP announces a receiver window, denoted as *rwnd*, to the sending TCP [16]. The sending TCP maintains a congestion window, denoted as *cwnd*, which is governed by its congestion control algorithms [27]. The window-based flow control mechanism allows a TCP sender to transmit at most $\min\{rwnd, cwnd\}$ data without receiving ACKs for the data sent [27].

2.1.2 Congestion control

Four congestion control algorithms are employed in the standard TCP: slow start, congestion avoidance, fast retransmit, and fast recovery. These algorithms control the value of *cwnd*, whereas the minimum of *cwnd* and *rwnd* controls the data transmission rate. Here, we briefly introduce these algorithms based on RFC 2581 [27]. The new recommendations and changes are described in detail in section 7 of a new IETF draft [59].

TCP uses a slow start threshold (*ssthresh*) to determine whether the slow start or congestion avoidance algorithm should be used to control the value of *cwnd*. The slow start algorithm is used at the beginning of a transfer and after the retransmission timer expires. The initial value of *cwnd* is no larger than two segments. The initial value of *ssthresh* is usually a very large value (e.g., the value of *rwnd*) and will decrease in response to congestion. A TCP sender uses the slow start algorithm when $cwnd < ssthresh$ and invokes the congestion avoidance algorithm when $cwnd > ssthresh$. Either algorithm may be used when $cwnd == ssthresh$. The slow start algorithm will increase *cwnd* by at most one receiver Maximum Segment Size (MSS) bytes for each received ACK that acknowledges new data. The congestion avoidance algorithm will increase *cwnd* by roughly one full-sized segment per round-trip time (RTT). When a TCP sender detects loss using the retransmission timer, *cwnd* is set

to one full-sized segment. At the same time, *ssthresh* is set to no more than the value given in Eq. (2.1):

$$ssthresh = max(FlightSize/2, 2 \times SMSS), \tag{2.1}$$

where SMSS denotes the sender MSS in bytes and *FlightSize* is the amount of unacknowledged data [27].

Without receiving (or a sufficient number of) ACKs, the sender eventually timeouts. The retransmission timeout value (RTO) is computed according to [30]:

$$RTO = max\{minRTO, SRTT + max(G, 4 \times VRTT)\}, \tag{2.2}$$

where minRTO, the lower bound on RTO, is recommended to be one second for the purpose of avoiding spurious retransmissions [60], and G is the clock granularity. SRTT is the smoothed RTT and VRTT is the RTT variation, which are updated according to Eq. (2.3) and Eq. (2.4) upon receiving a new RTT measurement rtt, respectively:

$$SRTT = 7/8 \times SRTT + 1/8 \times rtt. \tag{2.3}$$

$$VRTT = 3/4 \times VRTT + 1/4 \times |SRTT - rtt|. \tag{2.4}$$

Since a TCP receiver will immediately send a duplicate ACK when receiving an out-of-ordered segment, the TCP sender uses the fast retransmit algorithm to detect and recover loss. The fast retransmit algorithm regards the recipient of three duplicate ACKs as a signal of packet loss and therefore performs a retransmission of what appears to be the lost segment. At the same time, *ssthresh* is set to no more than the value given in Eq. (2.1), and *cwnd* is set to *ssthresh*+3 $\times$ SMSS. After that, the fast recovery algorithm controls the transmission of new data until a nonduplicate ACK arrives. For each additional duplicate ACK received, the *cwnd* value is incremented by one SMSS. The TCP sender will transmit a segment if it is allowed by both *cwnd* and *rwnd*. Upon the arrival of a nonduplicate ACK, *cwnd* is set to *ssthresh*.

2.1.3 Existing TCP exploits

The original TCP design [16] has inherent limitations and vulnerabilities that have been exploited by many attacks [17–21]. Some attacks try to tear down TCP connections or to smash TCP/IP stacks. Other attacks try to abuse TCP for bypassing firewalls/censorship or fingerprinting remote systems.

We further classify those attacks that aim at TCP connections into three categories. The attacks in the first category exploit the protocol's inherent limitations. Without completing the TCP three-way handshake, the SYN flooding attack depletes a victim system's memory for new connections and prevents it from serving legitimate connections [23, 24]. A malicious client can also launch DoS attacks by intentionally prolonging the duration of its connection with a server through delaying the transmission of acknowledgement packets [61]. Moreover, if an attacker knows a TCP connection's 4-tuple (the source and destination IP addresses and ports) and the 32-bit sequence numbers, he could mount injection attacks [62, 63] (e.g., injecting RST or SYN packets to tear down the connection [17]). Exploiting the fact that a TCP receiver will reject an incoming packet if its sequence number is unacceptable, the desynchronization attack intentionally makes an offset between a TCP receiver's ACK number and the TCP sender's sequence number [64]. Furthermore, an attacker could employ ICMP protocol to attack TCP connections; for example, forged Destination Unreachable and Time to Live Exceeded messages could reset existing connections [18, 19, 26]. On the other hand, various detection and defense approaches have been proposed. For example, the TLS protocol has been proposed to provide privacy and data integrity [65]. SYN cookies and cache are devised to mitigate the effect of SYN-flooding attacks [24].

The attacks in the second category exploit the protocol's implementation problems in many TCP/IP stacks [25]. For example, attackers construct malformed packets to launch DoS attacks. One of such notorious threats is the *Land* attack which sends TCP packets having the same source and destination address and identical source

and destination port numbers [20]. Some earlier TCP/IP stacks could crash when processing this type of packet. Another example is setting both SYN and FIN flags in the TCP header [21]. Since the TCP specification [16] does not clarify how to handle such malformed packet, some TCP/IP stacks process the SYN flag first by generating an ACK packet and performing a state-transition to the state *SYN-RCVD*. And then the stack handles the FIN flag by transiting to the state *CLOSE-WAIT* and sending an ACK. Since the attacker will not send any other packets to the victim, the victim's TCP connection gets stuck in the *CLOSE-WAIT* state until the expiry of the keepalive timer [21]. *Hping* [66], *Scapy* [67] and *Nemesis* [68] are popular packet crafting tools that could initiate such attacks. To carry out TCP spoofing and session hijacking attacks, an attacker has to predict TCP's sequence number if he could not capture packets exchanged by the victims. Although the sequence number should be generated randomly, some TCP/IP stacks leave many hints for predicting their values. For example, some TCP/IP stacks increase the Initial Sequence Number (ISN) by a finite amount each time period or adopt constant increments. More implementation problems can be found in [69].

The TCP exploits discussed above have also been widely used for system fingerprinting [20, 21, 70]. When conducting an active fingerprinting, an attacker sends TCP probes to the target system and then waits for the response. Based on a couple of probing packets and responses, an attacker could distinguish among different TCP/IP stack implementations. Popular techniques include FIN probes, ACK value sampling, bogus flag probes, TCP option handling, and ISN sampling [20]. An excellent introduction can be found in [71]. Moreover, *Nmap* [72], *xprobe* [73], and *p0f* [74] are popular TCP/IP fingerprinting tools.

The attacks in the third category exploit an increasingly invalid assumption that all TCP senders and receivers behave correctly, and congestion signal generated by the network is trustful. While most of existing TCP-targeted attacks belong to the first and the second categories, the attacks in this category receive much less atten-

tion. Savage et al. have examined misbehaving TCP receivers that can increase its download throughput through several approaches:

- ACK division: The receiver replies many ACKs for portions of each received segment, and each ACK increases the congestion window by one MSS;
- DupACK spoofing: The receiver replies a lot of duplicate ACKs no matter whether a segment is lost or not. Each duplicate ACK increases the congestion window; and
- Optimistic ACKing: The receiver acknowledges segments to be received. It will quickly increase the sender's congestion window at a cost of possible segment lost [75].

Moreover, these techniques have been abused to launch DDoS attacks [76].

2.2 Low-rate DoS attacks against TCP

The seminal research on PDoS attacks began with the shrew attack that attempts to constrain a TCP sender to the timeout state by dispatching attack pulses whenever the sender retransmits lost packets after a timeout period [28]. We have proposed a general framework for the PDoS attacks in [4], which consists of four kinds of attacks: synchronous timeout-based attacks, asynchronous timeout-based attacks, synchronous AIMD-based attacks, and asynchronous AIMD-based attacks.

Both timeout-based attacks drive a victim TCP sender to continuously enter the timeout state. Consequently, the victim TCP sender has very low throughput. The difference between these two timeout-based attacks lies in whether an attacker uses fixed inter-pulse period or adapts the period according to the behavior of TCP's slow start algorithm. We have further investigated the impact of asynchronous timeout-based attacks on a victim TCP's throughput by considering three difference cases in [45]. These three cases differ in how a PDoS attack constrains the value of *ssthresh*: (1) no constraint, (2) constraining it to the lowest value of two, and (3) constraining

it to a constant value larger than two. We will show that the shrew attack can be considered as one of three asynchronous timeout-based attacks. Kanhere et al. [77] have considered the third case and proposed a variant that will constrain the *ssthresh* within a range. This variant allows an attacker to vary the attack period and the burst for the purpose of evading detection at the cost of a lesser damage [77]. An interesting work from Taghizadeh et al. has demonstrated how to employ short-lived TCP flows to launch a shrew attack on long-lived TCP flows [78].

Both AIMD-based attacks cause a victim TCP sender to repeatedly enter the fast retransmit and fast recovery state. The result is the same as before: the victim TCP connection suffers from a very low throughput. The difference between these two AIMD-based attacks again lies in whether an attacker uses fixed inter-pulse period or adapts the period according to the behavior of TCP's congestion avoidance algorithm. Furthermore, we have generalized the AIMD attacks into the polymorphic DoS (PMDoS) attacks that randomize the attack period in order to evade detection and, at the same time, inflicts significant damage to TCP flows.

Another closely related work is the reduction-of-quality attack (RoQ), which also targets at TCP flows going through a router [29]. The RoQ attack forces the active queue management (AQM) scheme deployed in a router to go into the transient state, and then to increase packet loss rate by sending periodic attack pulses. The analysis in [29] mainly considers the RED-like AQM and the effect of the transient state on the TCP throughput. Guiruis et al. recently proposed two variants of shrew attacks and two variants of RoQ attacks (i.e., shrew attack at saturation, shrew attack at full buffer, RoQ attack at saturation, and RoQ attack at full buffer). The basic idea is that an attack can have high potency, defined as the ratio of the damage caused by the attack to the cost for launching the attack, if the attacks are launched right after the victim TCP flows already saturate the bottleneck or even use up the buffer [79]. However, the authors have also acknowledged that their assumptions may not hold in practice [79].

Chertov et al. have carefully examined the effect of PDoS attacks through ns-2 simulation and emulation experiments in the DETER and Emulab test beds [53]. They compare the result from a simple analytical model of TCP performance degradation, which is a special case of our analytical model proposed in [4], with the simulation result and emulation result. They found that the analytical and simulation results match well for a set of attack pulse lengths and router buffer sizes. The test-bed results match the results from analytical model and simulations when routing nodes are overloaded. Their extensive experiments confirm that our AIMD-based PDoS attacks are still effective when the period is not the same as the minimal RTO value used in the attack [53]. Moreover, Dong et al. reported that the PDoS attacks can significantly degrade the throughput of FAST TCP [51]. The FAST TCP is a new variant of TCP designed for high-speed networks that uses queueing delay in addition to loss probability as a congestion signal [50]. But FAST TCP maintains higher throughput than TCP Reno does under the PDoS attacks, because the former re-increases *cwnd* faster and therefore results in a larger *cwnd* than the latter does.

Since many existing Internet applications rely on TCP, the impact of PDoS attacks will be amplified through these applications. For example, Zhang et al. demonstrated that the PDoS attacks have severe impact on the Border Gateway Protocol (BGP) by causing session resets and delaying routing convergence [54]. Moreover, they investigated how to carry out coordinated PDoS attacks through careful time synchronization and selection of attack hosts and destinations. A new study from Mirkovic et al. also illustrates the effectiveness of the PDoS attacks through the changes of failure ratio, a new user centric metric for measuring DoS attacks [55]. The failure ratio indicates the percentage of transactions that are still alive just before the attack, but fail during the attack. They found that the failure ratio oscillates with the PDoS attack and does not vanish even in the absence of PDoS attacks. The major reason is that applications suffer significant periodic loss when attack pulses arrive and cannot recover to their QoS criteria quickly enough when there are no attack pulses [55]. Based on extensive ns-2 simulations under realistic scenarios, Shevtekar

et al. showed that the PDoS attacks could also seriously degrade VoIP quality by inducing considerable packet losses [52].

2.2.1 Countermeasures

Comparing with traditional flooding-based attacks, the PDoS attacks are harder to detect because they usually have a relatively low average traffic rate. Existing detection schemes could be grouped into two classes. The methods in the first class employ flow-level detection schemes. These methods try to identify malicious flows that will periodically occupy a large portion of the bandwidth and cause packet loss in well-behaved flows. For example, the system in [80] will consider a flow as a potential shrew attack if it exhibits a periodic pattern and its burst length is greater than or equal to RTTs of other connections with the same server and its time period is equal to the fixed minimum RTO. The HAWK system will pick up suspicious flows if they are too bursty over an extended period of time (e.g., 5 seconds) with very high-rate bursts appearing in short time-spans (e.g., 100ms) [81]. Based on the observation that the PDoS attacks may evade the RED-PD's detection because of its slow response [82,83], Xu et al. proposed a double horizon system that uses cut-tail dropping with instant response to recognize aggressive flows which send more packets in a short period than the threshold [84].

An interesting analysis from Sarat et al. [85] shows that while using smaller buffer will not affect link utilization, it makes the PDoS attacks more effective and harder to detect [86]. Based on a simple mathematical model, they argued that a relatively small increase in the buffer size over the value suggested in [85] is sufficient to detect malicious flows that cause the PDoS attacks [86]. Moreover, these systems will always punish identified malicious flows by dropping their packets to prevent normal flows from the shrew attacks [80, 81, 84, 86]. The deficit round robin (DRR) scheduler has also been used to guarantee fair bandwidth sharing among malicious flows and legitimate flows [87]. However, these methods may miss PDoS attacks that consist of

many attack flows each of which sends only a few attack packets but the aggregated volume is high enough to cause transient congestion.

The approaches in the second class adopt aggregate level detection schemes. Most of them are tailored for the *shew* attack. Therefore, other kinds of the PDoS attacks could bypass such schemes by changing parameters (e.g., period and rate). Sun et al. used dynamic time wrapping to isolate and discover shrew attack's rectangular attack pulses [87]. However, this method could be rendered ineffective when the duration of attack pulse is shorter than the sampling period. Chen et al. identified the anomalies in traffic's power spectrum density (PSD) induced by the shrew attack's periodic attack pulses [88, 89]. Nevertheless, an attacker may change the period or just employ aperiodic PDoS attacks (e.g., PMDoS attacks) to evade the detection.

We have proposed a two-stage detection scheme for the PDoS attacks, which is based on two kinds of traffic anomalies near the TCP receiver: periodic fluctuations in the incoming TCP data traffic and a decline in the trend of the outgoing ACK traffic [4]. This method, however, could not detect all kinds of PMDoS attacks. Hence, we have proposed Vanguard that synthesizes three anomalies (i.e., decline in the outgoing TCP ACK traffic, increase in the ratio of the incoming TCP traffic to the outgoing TCP ACK traffic, and change in the distribution of the incoming TCP traffic) to discover ongoing PMDoS attacks [46].

The aforementioned detection and accompanying defense systems are usually located close to the bottleneck router or the TCP receiver side. On another front, it is proposed to randomize the timeout value in [90, 91] for defending against the timeout-based PDoS attack. However, this method cannot defend the AIMD-based attack, because the attack's timing does not rely on the TCP timeout values.

2.2.2 Application-level DoS attacks

The low-row nature of shrew attack has motivated several new application-level DoS attacks. Macia-Fernandez et al. proposed a low-rate DoS attack that tries to

keep iterative servers busy and their service queue full all the time by sending a new request as soon as the server finishes processing a request. Consequently, other users may infer that the server is unavailable [92]. Chan et al. demonstrated that an attacker can cause temporary overload on a victim server in a relatively small interval by either herding normal users' requests or generating requests. Consequently, the increased delay and loss rate could discourage new users from accessing the server [93].

Guirguis et al. proposed an RoQ attack that tries to prevent Internet end-systems' proportional-integral (PI) controller from converging to the steady-state [94]. Using a series of bursts of requests to overload the system, such RoQ attack can force the admission controller to decrease the admission rate whenever a burst of requests has been admitted and consequently to drop requests from legitimate users [94]. Recently, Guirguis et al. further extended the RoQ attack to the dynamic load balancers and illustrated that a series of attack bursts will bring significant damage in terms of additional delay to legitimate requests [95].

Moreover, similar attacks have been applied to ad-hoc network and discussed their potential impacts [96,97]. On the other hand, shrew attacks can be used to watermark TCP connections going through low-latency anonymous networks [98–100]. In other words, an attacker can easily carry out traffic analysis by matching TCP flows that are affected by shrew attacks.

2.3 Network covert channels

In this section, we will first introduce the security model adopted by most network covert channels. After that, we will review existing storage and timing channels. In particular, we have proposed a new taxonomy for storage channels according to their hiding methods which is different from the traditional taxonomy based on network layers [37, 38]. An important advantage of this new taxonomy is that we can easily discover potential storage channels in any protocol.

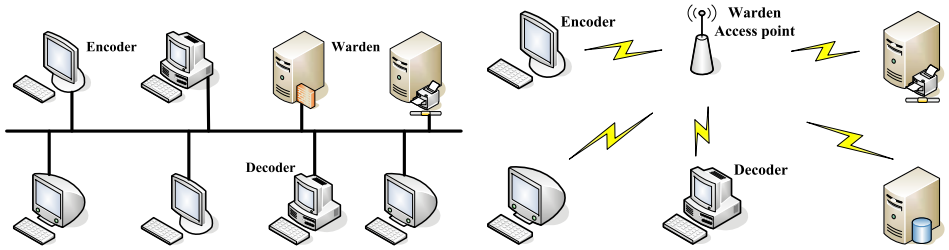
2.3.1 Model

In the general context of network covert channels, there are three distinguishable roles: encoder, decoder, and warden. An encoder tries to transmit covert messages stealthily to a decoder. A warden inspects all traffic coming from and going to its administrative zone, which consists of legitimate hosts, an encoder or a decoder, or both, depending on the network model.

We define two kinds of network models: broadcast model in Figure 2.1 and unicast model in Figure 2.2. In the broadcast model, a decoder could hear all traffic generated by an encoder regardless of its destination and whether the frame is correct or not. The traditional Ethernet and widely deployed wireless LAN (WLAN) belong to this model. Both encoder and decoder are located in compromised hosts, whereas the warden is placed at the gateway for Ethernet or the AP for WLAN. Many works have discussed how to embed covert channels into Ethernet frames [38,101,102] and 802.11 frames [103, 104], and to build timing channels by exploiting collision and contention resolution algorithms [105–108] in MAC protocols.

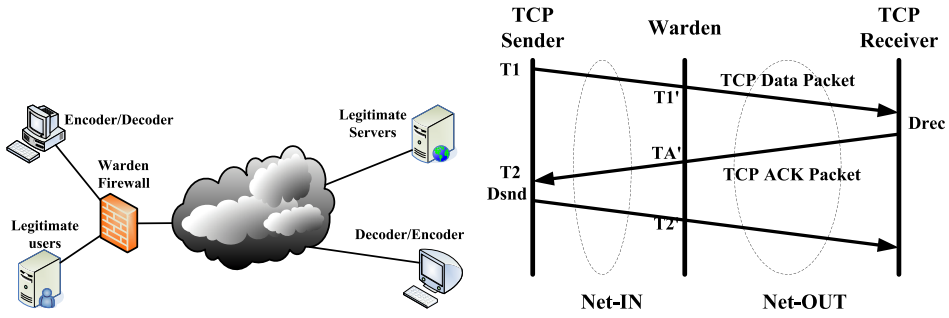
In the unicast model, a decoder may only receive the packets sent to itself or to a set of receivers, such that the decoder could sniff the traffic on the path between them. However, packets sent by the encoder cannot be malformed in this case, because the warden can easily detect them, and network equipment (e.g., switch and router) and middle-boxes (e.g., NAT) will drop them. There are a lot of works done in this category, ranging from IPv4/v6 and TCP protocols [37, 109, 110] to various application-layer protocols (e.g., HTTP and DNS) [13,14,111–113]. Clearly, all methods used in the unicast model are also suitable for the broadcast model.

There are two types of wardens: passive warden and active warden [31]. The passive warden could detect covert channels using either signature-based or anomaly-based algorithms. The active warden, on the other hand, could proactively manipulate the traffic to throttle the covert channels without detection. Existing approaches



(a) Ethernet (b) Wireless LAN

Fig. 2.1. A broadcast model for covert channels.



(a) Internet (b) TCP-based application

Fig. 2.2. A unicast model for covert channels.

include "scrubbing" the fields in various protocols [39, 40, 114] or inserting fuzzy delays [115, 116].

Based on information theory, a covert channel's capacity is defined as the maximal information rate in bits per second. In the seminal work, Millen analyzed the capacity of a class of finite-state noiseless channels with nonuniform transition times [117,118]. Moskowitz and Miller have used a discrete, memoryless and noiseless model to analyze a simple timing channel [119] and employed a Z-channel to compute the capacity of timing channels with noise [35,120]. Venkatraman and Newman-Wolfe used a noiseless model to analyze the capacity of network covert channels that exploit the spatial and temporal variation in transmission characteristics [36]. Berk et al. used the

well-known Arimoto-Blahut algorithm [121–123] to estimate the capacity of network timing channels [42]. However, since the one-way delay or RTT is a time-varying quantity [124], it is hard to obtain a constant value for the capacity. Hence, in this thesis we use the original definition of channel capacity in information theory (i.e., bits per symbol) to evaluate the capacity of covert channels. For a storage channel, the symbol denotes a packet or a frame; for a timing channel, the symbol indicates a timing event.

Both storage channels and timing channels have their own advantages and disadvantages. On one hand, by embedding information into the header of packets, storage channels usually enjoy a higher capacity and are more robust to adverse network conditions, such as packet loss, delay jitter, and packet reordering, than timing channels. However, most storage channels are vulnerable to specification-based active wardens, which will eliminate the messages in the packet headers. Moreover, it is not difficult to discover these storage channels by using either signature-based detection algorithms or anomaly-based detection algorithms. For example, the former can discover storage covert channels exploiting unused fields and ambiguities in protocols' definitions. The latter can identify storage covert channels abusing the flexibilities provided by protocols, for example, improper parameter settings.

On the other hand, although timing channels have a lower capacity and are vulnerable to adverse network conditions, it is difficult to detect them or entirely stop them for two reasons. First, the timing information exists in network protocols and there are no rigorous specifications for it. Second, an encoder may use a very slow covert channel, for example, sending one packet within 24 hours or not to denote `1` or `0`. To sum up, the selection of covert channels depends on its usage and the network model.

2.3.2 Storage channels

Many studies on storage channels use specification-based approaches to scrutinize possible covers [39, 110]. Fisk et al. classify the information carried in network protocols into structured carriers and unstructured carriers [39]. The former has well-defined, objective semantics for its content (e.g., TCP header), whereas the latter lacks objectively defined semantics and is usually interpreted by humans rather than computers. Here, we also employ the specification-based approach to trace possible storage channels in structured carriers. But we further propose a methodology taxonomy as shown in Figure 2.3, where we use $Lx.y$ to denote different method (x is the level number, and y is the letter next to each rectangle in Figure 2.3). Based on this taxonomy, we could not only categorize existing storage channels, but also identify new ones. Existing literature and tools on storage channels have already covered many popular protocols, for example, 802.3 MAC [38, 101, 102], 802.11 MAC [103, 104], IPv4 [102, 109, 125–127], ICMPv4 [6, 8, 128, 129], IPv6 [110, 130], ICMPv6 [130], TCP [37, 109, 126, 127, 131, 132],SSH [133], HTTP [13, 134–138] and DNS [111–113].

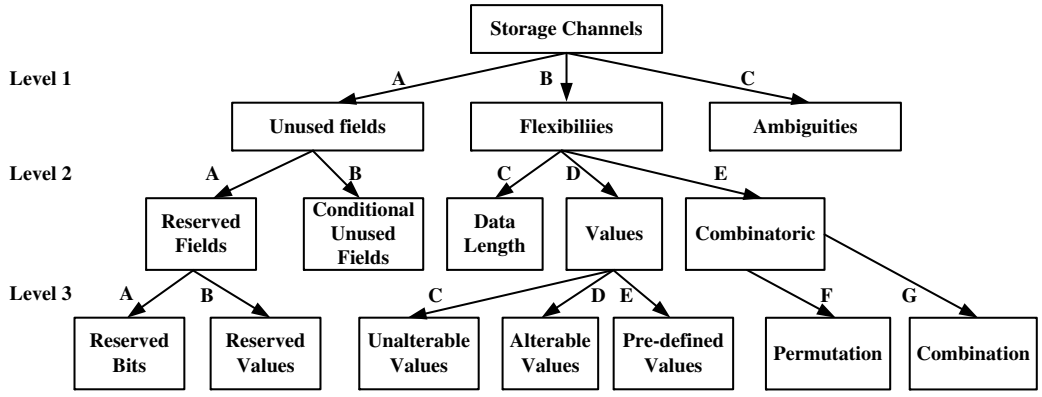


Fig. 2.3. A taxonomy for storage channels.

Storage covert channels can exploit unused fields, flexibilities, and ambiguities in the protocol definitions to embed hidden messages. The unused fields include reserved

values ($L3.A$) and conditional unused bits ($L3.B$). There are two types of reserved values. One could indicate one or several continuous bits that are not used in a protocol's current version. For example, there are three reserved bits between the data offset field and the ECN field in the TCP header. The other could denote the values that are not utilized in a used field. For example, in the OPCODE field of the DNS protocol, value 3 is not used to specify any kind of query. The difference between these two types of reserved values lies in the fact that the former has at least two continuous unused values because one bit could be set to `0` or `1`, whereas the latter could represent individual values. The conditional unused bits ($L2.B$) refer to fields that are not used if some conditions are satisfied; for example, the fragment offset field in the IP header if the DF bit is set.

Some covert channels abuse the flexibilities provided by the protocol to embed messages. Generally, there are three ways to exploit the flexibilities. First, an encoder could exploit the packet length or padding data length to transmit information ($L2.C$). It could dispatch packets with different lengths to represent different meanings. If both encoder and decoder could manipulate the packets, then the encoder could insert new data into normal packets passing through and modify the corresponding fields in the protocol header, for example, the total length and checksum fields in the IP header. The decoder then removes the piggyback data from those packets and then restores the original value to corresponding fields in the protocol header.

Second, the encoder could use valid values in different fields to convey messages. There are at least three common approaches to achieve it:

1. Some fields could accept a range of arbitrary values ($L3.C$), and the value will not be changed by routers, for example, the IP's identification value (IPID), TCP's initial sequence number, and so on. Hence, the covert message could be directly encoded into these fields. Other examples include the address fields [38, 101], the FCS field, the duration/ID field in 802.11 frames [103, 104], and IPID field [37, 127].

2. Some fields will be changed ($L3.D$) by routers, for example, the TTL field in IPv4 and hop limit field in IPv6. A simple way to overcoming this problem is to set a threshold for the final TTL value. If an encoder wants to transmit `1`, it could send a packet whose TTL is much larger than the summation of the threshold and estimated maximal number of hops. Otherwise, intending to transmit `0`, the encoder could set the TTL to be less than the threshold but larger than the estimated maximal number of hops. Consequently, the decoder could get the information by comparing the final TTL with the threshold.

3. Some fields have a set of pre-defined valid values ($L3.E$). Then the messages could be encoded by selecting different valid values. For example, there are different types of management frames in 802.11 MAC, and the encoder could manipulate the subtype values to convey covert messages [38, 101, 102, 104].

Third, encoder could embed information into the combinations ($L3.G$) or permutations ($L3.F$) of fields or values. For example, if there are N_{TCP} available TCP options that could be attached to a packet, then such a packet could encode at least $\log_2 \sum_{k=0}^{N_{TCP}} \begin{pmatrix} N_{TCP} \\ k \end{pmatrix} = N_{TCP}$ bits of information by utilizing the combination technique. Another example is to make use of the permutation technique. If a packet could only carry M_{TCP} TCP options, then it could convey $\log_2(M_{TCP}!)$ bits of information by permutating the locations of these options. Note that all selected options or bits in various fields are legal according to the protocols specifications. The method ($L3.G$) stems from our original combination-based framework, Cloak, which will be presented in section 6.

Some covert channels exploit obscurity in a protocol's specification. For example, since RFC793 does not mention whether a SYN packet with FIN flag is correct or not when TCP receiver is in the LISTEN state [16], some implementations accept it [139]. Note that we do not consider those cases that are explicitly forbidden in protocols' specifications. For example, according to RFC793, SYN+RST should be ignored by a TCP receiver when it is in the LISTEN state [16]. Therefore, there are

many different combinations that could be used "legitimately" to convey messages. Another kind of ambiguities is the collision in the checksum function [125].

Although various fields in different headers could be used to embedded information, they could be detected if their statistical features are not consistent with those of normal packets. Neural network and support vector machines have been adopted to detect storage covert channels based on the ISN values of TCP flows [37]. Moreover, statistical approaches have been proposed to detect covert channels over HTTP [14,140]. Besides detection, another approach is to neutralize covert channels by performing active operations on the traffic. In addition to protocol scrubbers [40], traffic normalizers [114], and active wardens [39], NAT can also adversely affect the quality of some covert channels [141].

The storage media used by the covert channels discussed above are permanent in the sense that their values will not be changed in the absence of adversary, for example, active warden. Some newly proposed channels utilize the shared states in the network to transmit messages, which are ephemeral in the sense that their values will only be kept for a while and then will be changed. Its advantages include:

1. There are no strict constraints on an decoder's location as long as it could access the share states, whereas traditional covert channels require the decoder to sniff the traffic generated by the encoder.

2. The ephemeral nature of the shared state could help users camouflage their actions.

On the other hand, such kind of storage covert channels have to handle some special issues, for example synchronization, noise handling, and so on. Jones suggested employing board posting as the shared state as long as the messages are not deleted [134]. Danezis proposed using the IPID number in a server's TCP/IP stack to convey messages based on the assumption that the IPID value will monotonically increase for each outgoing packets [142]. Hence, an encoder could transfer a value, say 3, by inducing the server to transmit three packets and consequently to increase

its IPID value by 3. As another example, we have designed *WebShare* that conveys hidden messages by increasing the values of ubiquitous Web counters [143].

2.3.3 Timing channels

Existing timing channels could be divided into two categories according to their timing references. The first type of timing channels uses the absolute time interval as the timing reference. For example, if the encoder and decoder are synchronized, they could use the number of packets observed in a predefined time interval to represent covert messages. The second type of covert timing channels employs a single or a group of packets as the timing reference. For example, an encoder may send out two packets each time. The decoder regards the first packet as the timing reference and decodes the message based on the time interval between these two packets.

At the MAC layer, Girling proposed a covert channel based on the delays between successive transmissions of frames [38]. A similar method is also mentioned in [101]. However, noise will be introduced if the encoder cannot transmit two frames according to the pre-defined interval. Handel et al. proposed a covert channel where an encoder first causes a packet collision and then retransmits the frame either leading or lagging the frame retransmitted by others according to the intended binary value [102].

There are also studies on timing channels in ideal MAC channels that may not be strictly consistent with practical specifications. Dogu and Ephremides proposed a covert channel for an ideal slotted ALOHA environment that uses splitting algorithms for contention resolution [105]. The covert message is transmitted through the number of collisions caused by the encoder during the contention resolution period. The decoder uses a maximum likelihood decoder to determine the number of collisions deliberately caused by the encoder. Bhadra et al. proposed an alternative collision based covert channel in the slotted ALOHA environment [106]. To transmit bit `1`, an encoder will intentionally cause a collision by jamming a transmission during that time slot. Consequently, the decoder will interpret each collided transmission

as bit 1 and successful transmission as bit 0. Li and Ephremides proposed another timing channel based on splitting algorithm [107], which encodes the message into an encoder's reactions to collision events (i.e., the encoder will retransmit the frames or not in the next slot). A recent work for multiple encoders requires each encoder to adopt an action according to a biased probability and the bit to be conveyed [108]. For example, to convey bit 1, an encoder may choose action A with a very high probability in next step. Otherwise, the encoder may choose action B. Therefore, the decoder may infer that the information is bit 1 (or 0) if most other users choose action A (or B).

At the IP layer, Cabuk et al. proposed a binary IP timing channel [41]. An encoder delivers bit 1 by sending one packet during a predefined time interval and bit 0 by keeping silent [41]. Being a simple solution, the IP timing channel is vulnerable to adverse network conditions, such as packet loss and jitter, and requires synchronization between encoder and decoder, because it employs absolute time interval as the timing reference.

Ahsan and Kundur utilize the order information between IPSec packets to deliver messages, because different order sequences could represent different values [127]. With an alias of ordered channel [144], such kind of methods need two order indices. One index is used to encode messages, and the other index records the original sequence. In their paper, the 32-bit sequence number field in the IPSec authentication header (AH) and encapsulating security payload (ESP) is used to record the original packet sequence, and the arrival sequence of packets is used to carry covert messages. Clearly, this encoding method's accuracy will be affected by packet reordering events in the Internet.

Berk et al. examined an ICMP-based timing channel that encodes the information into the inter-arrival time between two ICMP packets [42]. At the TCP layer, Chakinala et al. proposed another ordered channel based on TCP. They used TCP sequence number to maintain the original packet sequence [144]. Shah et al. proposed Jitterbug, a covert channel based on the inter-arrival time between two consecutive

packets [43]. Venkatraman et al. proposed two types of network covert channels: spatial covert channels and temporal covert channels. The former conveys covert messages by manipulating the volume of communication between nodes, whereas the latter embeds messages into the order, frequency or duration of transmissions [36].

It is rather difficult to detect timing channels, because there may not exist detectable signatures in the timing information. As a result, most studies on the defense side focus on how to decrease the capacity of timing channels by introducing noise. Hu proposed using fuzzy time to reduce the capacity of timing channel in an multi-level OS environment [115, 116]. Although it is a feasible solution, it has the cost of performance degradation. In a network environment, this cost could be more significant, because the TCP throughput is inversely proportional to latency. Using game theory, Giles et al. [145] further investigated the equilibriums between timing channel and various delay jammers (e.g., pure delay jammers with a maximum delay constraint, an average delay constraint, or a maximum buffer size constraint) that will introduce timing noise. Chakinala et al. [144] examined the games between order channels and several defense schemes, including the k-distance permuter, the k-buffer permuter, and the k-stack permuter. Based on packet inter-arrival time, two metrics—*regularity* and ϵ-similarity—were proposed to locate the anomalies caused by IP timing channels [41]. In [42], Berk et al. proposed an interesting detection algorithm based on the optimal statistical distribution of intervals for approximating the capacity.

2.4 Summary

In this chapter, we have reviewed the basic knowledge of TCP that is relevant to the two TCP exploits to be discussed in the rest of this thesis. The most important elements include congestion control algorithms, reliability, and flow control mechanisms. We have also reviewed existing TCP exploits based on protocol design, implementations, and assumptions.

For the DoS attacks, we have concentrated on low-rate DoS attacks and the countermeasures, because the PDoS attack can be classified as a low-rate attack. We have discussed the recent works on devising more effective low-rate attacks against TCP and AQMs and their impact on Internet protocols and stability. For the countermeasures, we have reviewed several recently proposed detection systems and pointed out their limitations.

For the covert channels, we have first presented the attack model assumed in this thesis and surveyed existing storage channels and timing channels. In particular, we have proposed a new taxonomy for storage channels which could help discover new storage channels. Moreover, we have pointed out several serious weaknesses of the available storage and timing channels, including unreliability, low data rate, and vulnerability to active wardens or similar active intermediaries in the Internet.

3. PULSING DENIAL-OF-SERVICE ATTACKS

In this chapter, we will introduce the PDoS attacks, including the attack mechanism, analytical modeling, experiment results and analysis, and attack optimization. We will also present a general framework, referred to as polymorphic DoS attacks, for analyzing different low-rate attacks and flooding-based attacks.

Note that PDoS attack is in fact a suite of attacks, as depicted in Figure 3.1. The specific attacks are labeled by a number (1 to 10). The PDoS attacks are first classified into timeout-based PDoS attacks and AIMD-based PDoS attacks, depending on whether the attack exploits TCP's timeout or AIMD mechanism. On the next level, the attacks can be periodic or aperiodic, depending on whether the attack period is fixed or not. Besides, there are additional levels of classifications which we will explain later in this chapter.

Since different arrangements of attack pulses will result in different kinds of aperiodic PDoS attacks, it is impossible to investigate all kinds of aperiodic PDoS attacks. We have addressed the attacks (1)-(4), (8)-(9), and (11) in this chapter; others will be reserved for future works. For (5) and (7), we have investigated their special cases, i.e. synchronous timeout-based PDoS attacks (i.e.(8)) and synchronous AIMD-based PDoS attacks (i.e. (11)). These attacks will synchronize their attack epochs with certain values in TCP stack, e.g. RTO and cwnd. For the stochastic AIMD-based PDoS attack, we have investigated a large class of such attacks (i.e. Renewal process PDoS (9)). It is relatively easy to obtain the analytical results for such attacks. Moreover, based on similar idea, we will examine other kinds of stochastic AIMD-based PDoS attacks (i.e. (10)). Similar to the stochastic AIMD-based PDoS attack, we will study the stochastic timeout-based PDoS attack (i.e. (6)) in the future work.

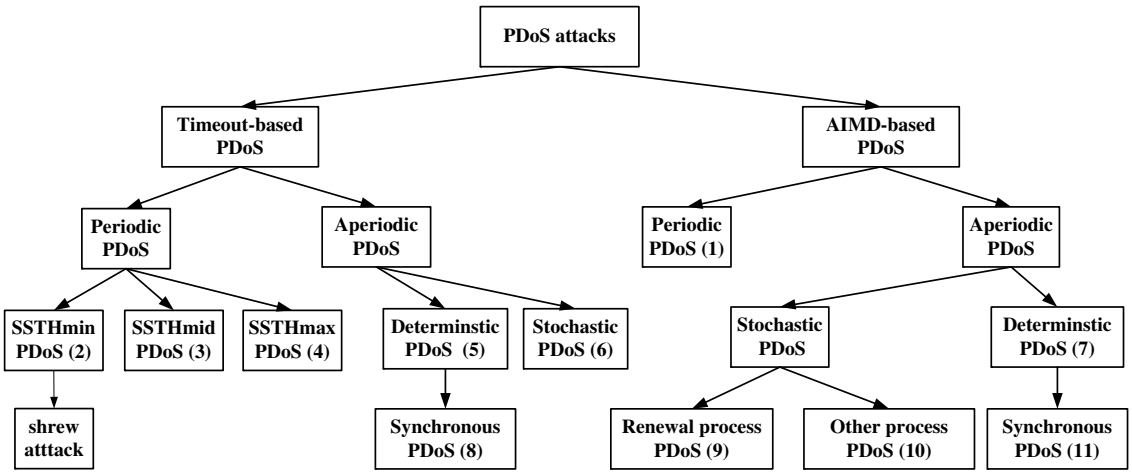


Fig. 3.1. A classification of the PDoS attack suite.

3.1 Modeling the PDoS attacks

One can view the traditional flooding-based attack as a brute-force attack, which exploits the finiteness of network and system resources. However, the PDoS attack is more sophisticated in the sense that it exploits TCP congestion control mechanism. There are two main mechanisms in a typical end-to-end congestion control algorithm. The first is the generation of a congestion signal that serves to notify the sender of possible congestion. The second is the sender's response to the receipt of such a congestion signal. Table 3.1 summarizes the mechanisms used in TCP.

In essence, a PDoS attacker induces a sequence of *congestion signals* to a TCP sender using attack pulses, so that the sender's *cwnd* is constrained to a low value. Therefore, the magnitudes of the attack pulses must be significant enough to cause packet drops in a router. We illustrate a sequence of attack pulses in Figure 3.2 and formally model them using the following notations: $\mathbb{A}(T_{extent}(n), R_{attack}(n), T_{space}(n), N)$, where

- N is the total number of pulses sent during an attack.
- $T_{extent}(n), n = 0, 1, \ldots, N$, is the duration of the nth attack pulse.

Table 3.1
TCP's network congestion signals and responses

	Congestion Signals	Sender's Responses
1	Retransmission timer expired	Reduce the congestion window (*cwnd*) to one and perform a slow start (the sender is said to enter the *timeout* state).
2	Three duplicate ACKs received	Halve the *cwnd* and increase the *cwnd* by one per RTT. (the sender is said to enter the fast retransmit/ fast recovery state).

- $R_{attack}(n), n = 0, 1, \ldots, N$, determines the sending rate of the nth attack pulse.
- $T_{space}(n), n = 0, 1, \ldots, N-1$, measures the time between the end of the nth attack pulse and the beginning of the $(n+1)$th attack pulse. If $T_{space}(n) = 0, \forall n$, the PDoS attack is the same as a flooding-based attack.

To simplify the analysis, we assume that an attacker uses identical pulses. That is, $T_{extent} = T_{extent}(n)$ and $R_{attack} = R_{attack}(n)$.

Back to Table 3.1, a TCP sender's response to the recipient of three duplicate ACKs is generally known as an *additive-increase, multiplicative-decrease* (AIMD) algorithm. Although TCP is the prime target for such PDoS attacks in the Internet today, it is useful to examine more general AIMD algorithms in a similar manner as in [146]. That is, we denote an AIMD algorithm by AIMD(a, b), $a > 0,\ 1 > b > 0$. In this general AIMD algorithm, a sender will decrease its *cwnd* from W to bW when it enters the fast retransmit/fast recovery (FR) state, and then it will increase its *cwnd* by a per RTT until receiving another congestion signal. Many TCP variants, such as Tahoe, Reno, and New Reno, employ AIMD$(1, 0.5)$. Moreover, many TCP implementations do not send an ACK for every received packet. Instead, they send a delayed ACK after receiving d consecutive full-sized packets, where d is typically equal to 2 [27]. In this case, the sender's *cwnd* is only increased by $\frac{a}{d}$ per RTT.

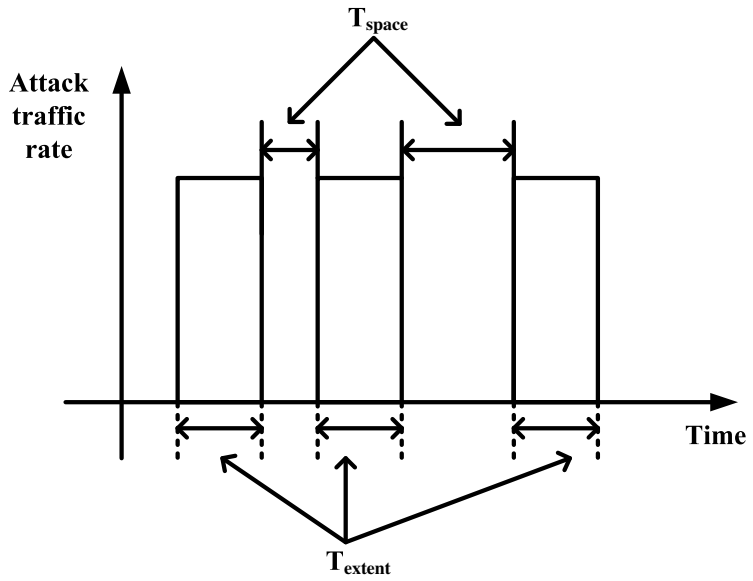


Fig. 3.2. Attack pulses of a PDoS attack.

A PDoS attack can force a victim TCP sender to frequently enter the timeout state (TO) state or to frequently enter the FR state. The result is a persistently low value of *cwnd*, which translates into a very low throughput for the victim TCP connection. Accordingly we classify the PDoS attacks into *timeout-based* attacks and *AIMD-based* attacks.

Consider that there are N_f legitimate TCP flows traversing through a router, and a DoS attack causes the router to drop packets. When using the subscript i, we denote the ith TCP sender or its corresponding parameters (e.g., RTT_i and RTO_i). Otherwise, we denote parameters belonging to all victim TCP senders. We use R_{bottle} to represent the router's bandwidth and define *attack power* and *attack cost* for a DoS attack in Definitions 3.1.1-3.1.2, respectively. Note that the DoS attacks include the PDoS attacks. While the attack power measures the impact of a DoS attack on legitimate TCP flows, the attack cost measures the intensity of a DoS attack, in terms of the traffic rate normalized by the bottleneck bandwidth. In the rest of this

thesis, whenever we compare the PDoS attacks with the flooding-based DoS (FDDoS) attacks based on their power, we assume that they have the same attack cost.

Definition 3.1.1 *The power of a DoS attack, denoted by* Γ*, is defined as*

$$\Gamma = 1 - \frac{\sum_{i=1}^{N_f} \Psi_{i,attack}}{\sum_{i=1}^{N_f} \Psi_{i,normal}}, \tag{3.1}$$

where $\Psi_{i,attack}$ *and* $\Psi_{i,normal}$ *denote the amount of data (bytes) sent by the* i*th TCP flow in the presence of and in the absence of a DoS attack within the same period, respectively.* $\Psi_{i,attack}$ *can be* $\Psi_{i,attack}^{TO}$ *(or* $\Psi_{i,attack}^{AIMD}$*) for the case under the timeout-based PDoS attacks (or AIMD-based PDoS attacks).*

Definition 3.1.2 *The cost of a DoS attack, denoted by* γ*, is defined as*

$$\gamma = \frac{R_{DoS}}{R_{bottle}},$$

where R_{DoS} *is the average traffic rate of a DoS attack. For a FDDoS attack, we have* $\gamma = \frac{R_{FDDoS}}{R_{bottle}}$*, whereas the cost of a PDoS attack is given by*

$$\gamma = \frac{R_{attack} T_{extent}}{R_{bottle} T_{attack}}, \tag{3.2}$$

where $T_{attack} = T_{extent} + T_{space}$ *is the period of the PDoS attack.*

3.2 Timeout-based PDoS attacks

In a timeout-based PDoS attack, the attack pulses are severe enough to cause a victim TCP sender to frequently enter the timeout state. An attacker may launch aperiodic attacks or periodic attacks. We will discuss them respectively in the following sections.

3.2.1 Aperiodic attacks

Since aperiodic attacks may have arbitrary patterns, we analyze only one special case, which is synchronous with the RTO value. That is, if an attacker knows the RTO_i value of the ith victim TCP sender and is able to cause every retransmitted

packet to drop, then the victim TCP sender's *cwnd* stays always at one. We refer it to as synchronous timeout-based attack. This type of attack is shown in Figure 3.3, which shows that the attack epoches coincide with the retransmission epoches. These epoches can be obtained through Proposition 3.2.1.

Proposition 3.2.1 (Attack epoches for synchronous timeout-based attacks) *Let $t_n, n = 0, \ldots,$ be the nth attack epoch. The first attack starts at t_0, and for $n > 0$, the nth attack epoch in a synchronous timeout attack is given by*

$$t_n = \begin{cases} t_0 + (2^n - 1)RTO_i & \text{if} \quad 1 \le n \le N_{i,max} \\ t_0 + (2^{N_{i,max}} - 1)RTO_i + (n - N_{i,max})RTO_{i,max} & \text{if} \quad N_{i,max} < n \le A_{i,max}, \end{cases} \tag{3.3}$$

where $RTO_{i,max}$ is the maximum value of RTO_i, and $A_{i,max}$ is the maximum number of retransmission attempts before the TCP sender gives up. Moreover, $N_{i,max} = \left\lfloor \log_2 \frac{RTO_{i,max}}{RTO_i} + 1 \right\rfloor$.

Proof According to RFC 2988, a TCP sender will double its current RTO_i value when the retransmission timer expires [30]. If the new RTO_i is not more than $RTO_{i,max}$, then the sender will use the new RTO_i. Hence, $t_n = t_0 + \sum_{j=0}^{n-1} 2^j RTO_i = t_0 + (2^n - 1)RTO_i$ for $1 \le n \le N_{i,max}$, where $N_{i,max} = \max\{n : 2^{n-1}RTO_i \le RTO_{i,max}\}$. Otherwise, the RTO_i value is set to $RTO_{i,max}$, and the sender continues to retransmit the lost packet until the total number of attempts reaches $A_{i,max}$. ■

3.2.2 Periodic attacks

The synchronous timeout-based attack is a perfectly timed attack that is obviously not quite feasible in practice. It is easier for an attacker to send pulses with fixed periods. Kuzmanovic and Knightly demonstrated that such periodic timeout-based attack is indeed possible [28]. Their attack scheme, commonly referred to as shrew attack, works in the following way. After a victim TCP sender enters the timeout state after the launch of the first attack pulse, the attacker lets the sender send data within a short period T_{Shift}. This period should be small but long enough for the sender

to transmit some packets successfully. Therefore, T_{Shift} may be set to $2 \sim 3$ RTTs. Thus, the SRTT, VRTT, and RTO values are updated according to Eqs. (2.2-2.4) during T_{Shift}. Assume that the RTO value at the attack epoch is given by minRTO (i.e., the value of second argument of the max function in Eq. (2.2) is smaller than minRTO). If the new RTT samples do not deviate too much from SRTT, it is very likely that the RTO value is still given by minRTO at the end of T_{Shift}. Thus, subsequent attack pulses can be launched with a fixed period of $minRTO + T_{Shift}$, as depicted in Figure 3.4.

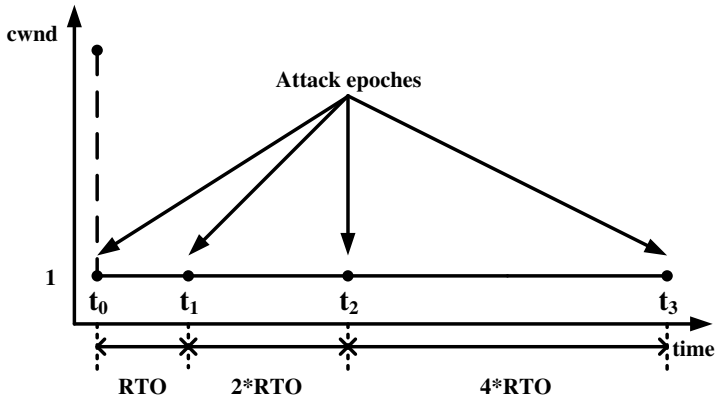


Fig. 3.3. An example of a synchronous timeout-based PDoS attack.

The shrew attack requires that a victim TCP sender uses minRTO as its RTO value and dispatches attack pulses with a fixed period around minRTO. In the following, we analyze a general case where a sending TCP is forced to enter the TO state by each attack pulse and its RTO value remains constant during a PDoS attack. Although the RTO may change in practice, the analysis helps us to understand the capability of such periodic timeout-based PDoS attacks. Moreover, we adopt a widely used assumption that the victim TCP sender always has data to send and the *rwnd* is not a constraint. Therefore, we can simplify Eq. (2.1) as:

$$ssthresh = max(cwnd/2, 2 \times SMSS). \tag{3.4}$$

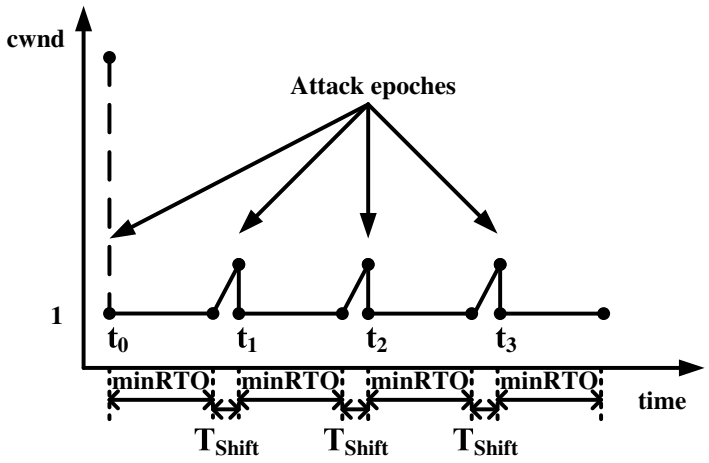


Fig. 3.4. An example of a timeout-based PDoS attack with fixed periods.

We let W_i^U be the maximal value of *cwnd* achieved by the ith victim TCP sender before a PDoS attack and use $ssthresh_{min}$ to denote the minimal $ssthresh_i$ in unit of SMSS (i.e., 2). According to the value of $ssthresh_i$, we classify the periodic timeout-based PDoS attacks into three groups: SSTHmin-PDoS, SSTHmid-PDoS, and SSTHmax-PDoS. The attacks in the first group constrain $ssthresh_i$ to $ssthresh_{min}$. The attacks in the third group permit $ssthresh_i$ to reach $\frac{W_i^U}{2}$. The attacks in the second group force $ssthresh_i$ to converge to a value within $(ssthresh_{min}, \frac{W_i^U}{2})$.

As depicted in Figure 3.5, we can obtain two values: $T_{i,mincvg}$ and $T_{i,maxcvg}$. The former indicates the required duration after which the ith victim TCP sender can increase its *cwnd* from 1 to $2ssthresh_{min}$ following a timeout (its $ssthresh_i$ is equal to $ssthresh_{min}$). The latter denotes the necessary period after which the victim TCP sender can increase its *cwnd* from 1 to W_i^U following a timeout (its $ssthresh_i$ is equal to $\frac{W_i^U}{2}$). Let $T_{i,period} = \lceil \frac{RTO_i}{T_{extent}+T_{space}} \rceil (T_{extent} + T_{space})$. Therefore, we can estimate $T_{i,mincvg}$ and $T_{i,maxcvg}$:

$$(\tau_d)^x = \frac{1}{2} W_{i,TO}, \quad \frac{a}{d} y = \frac{1}{2} W_{i,TO}, \quad x > 0,\ y > 0, \tag{3.5}$$

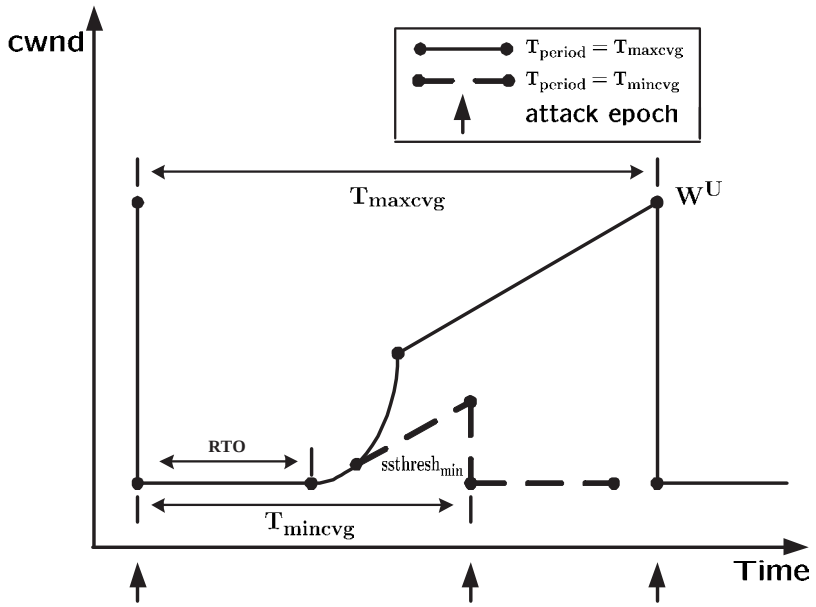


Fig. 3.5. Relationship among W^U, T_{maxcvg}, $ssthresh_{min}$, and T_{mincvg} in a PDoS attack.

where x is the number of RTTs when the TCP sender is in the slow start state and y is the number of RTTs when the TCP sender is in the congestion avoidance state. $W_{i,TO}$ is the value of *cwnd* at the end of $T_{i,mincvg}$ or $T_{i,maxcvg}$. τ_d is a constant value depending on d. When $d = 1$, the number of packets sent in the nth RTT is 2^n during the slow start phase [27]. When $d = 2$, we employ a simple model proposed in [147] to approximate the number of packets sent in the nth RTT to 1.5^n. Hence, we have

$$\tau_d = \begin{cases} 2 & \text{if} \quad d = 1, \\ 1.5 & \text{if} \quad d = 2. \end{cases} \tag{3.6}$$

The above equations mean that the ith victim TCP sender needs x RTTs to increase its *cwnd* from 1 to $\frac{1}{2}W_{i,TO}$ during the slow start state and requires y RTTs to increment its *cwnd* by $\frac{1}{2}W_{i,TO}$ in the congestion avoidance phase. Finally, we obtain

$T_{i,mincvg}$ and $T_{i,maxcvg}$ by substituting $W_{i,TO} = 4$ and $W_{i,TO} = W_i^U$ into Eq. (3.5) respectively.

$$\begin{aligned} T_{i,mincvg} &= \left(1 + \frac{2d}{a}\right) RTT_i + RTO_i, \\ T_{i,maxcvg} &= \left(\frac{\ln(W_i^U/2)}{\ln(\tau_d)} + \frac{d}{2a} W_i^U\right) RTT_i + RTO_i \end{aligned}$$

Based on the relationship between T_{period} and (T_{mincvg}, T_{maxcvg}), we can define SSTHmin-PDoS, SSTHmid-PDoS and SSTHmax-PDoS as illustrated in Figure 3.6.

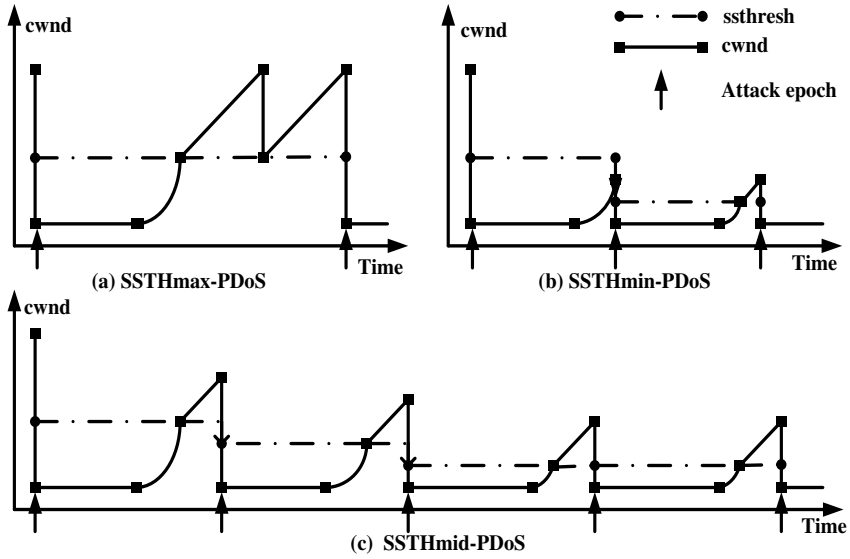


Fig. 3.6. The trajectory of *cwnd* and *ssthresh* under periodic timeout-based PDoS attacks.

Lemma 3.2.1 *The maximal congestion window of the ith legitimate TCP flow in the steady state, denoted by* W_i^U*, can be estimated by*

$$W_i^U = \frac{2R_{bottle}}{(1+b)S_p} \left(\sum_{j=1}^{N_f} \frac{1}{RTT_j}\right)^{-1}. \tag{3.7}$$

Proof According to [148], the N_f TCP flows share the bandwidth R_{bottle} in an inverse proportion to their RTTs. That is, $\sum_{i=1}^{N_f} BW_i = R_{bottle} \quad and \quad \frac{BW_i}{BW_j} = \frac{RTT_j}{RTT_i}$. Therefore,

$$BW_i = \frac{R_{bottle}}{RTT_i} \left(\sum_{j=1}^{N_f} \frac{1}{RTT_j} \right)^{-1}, \tag{3.8}$$

where BW_i is the bandwidth obtained by the ith flow in an attack-free scenario. Let S_p be the packet size which is assumed to be the same for all legitimate flows. By assuming that all TCP flows stay in the congestion avoidance state, we have $\frac{(W_i^U + W_i^L)}{2} \frac{T}{RTT_i} S_p = BW_i T$ and $W_i^L = bW_i^U$ for a period of T. By solving the equation for W_i^U, we can obtain Eq. (3.7). ■

Proposition 3.2.2 *The amount of data sent by the ith legitimate TCP flow under a periodic timeout-based PDoS attack, denoted by $\Psi_{i,attack}^{TO}$, is given by*

1. *If $T_{i,maxcvg} < T_{i,period}$ (i.e., SSTHmax PDoS),*

$$\Psi_{i,attack}^{TO} = (T_{i,period} - RTO_i) BW_i \left\lfloor \frac{(N-1)T_{attack}}{T_{i,period}} \right\rfloor.$$

2. *If $T_{i,mincvg} \leq T_{i,period} \leq T_{i,maxcvg}$ (i.e., SSTHmid-PDoS),*

$$\Psi_{i,attack}^{TO} = \left(\frac{3d}{8a} W_{i,TO}^2 + \frac{\frac{1}{2}\tau_d W_{i,TO} - 1}{\tau_d - 1} \right) S_p \left\lfloor \frac{(N-1)T_{attack}}{T_{i,period}} \right\rfloor.$$

3. *If $RTO_i < T_{i,period} < T_{i,mincvg}$ (i.e., SSTHmin-PDoS),*

$$\Psi_{i,attack}^{TO} = [1 + \frac{a}{2d} \left(\frac{T_{i,period} - RTO_i}{RTT_i} - 1 \right)^2 + 2 \left(\frac{T_{i,period} - RTO_i}{RTT_i} - 1 \right)] S_p \left\lfloor \frac{(N-1)T_{attack}}{T_{i,period}} \right\rfloor.$$

where,

$$W_{i,TO} = 2e^{\frac{-LambertW(\mathfrak{L}) RTT_i + \ln(\tau_d)(Ti,period - RTO_i)}{RTT_i}}, \ \mathfrak{L} = \frac{a}{d} \ln(\tau_d) e^{\frac{\ln(\tau_d)(T_{i,period} - RTO_i)}{RTT_i}}, \ and$$

LambertW() denotes the Lambert's $\mathbb{W}$ function [149].

Proof Since each attack pulse forces all TCP flows to enter the TO state, $\Psi_{i,attack}^{TO}$ is equal to the amount of data sent during the period starting from the end of a

timeout to the beginning of the next attack pulse (i.e., $T_{i,period} - RTO_i$), which we call a *run*. Consequently, the amount of data sent by the legitimate TCP flows under a PDoS attack with N pulses is equal to that sent during $\left\lfloor \frac{(N-1)T_{i,attack}}{T_{i,period}} \right\rfloor$ runs. Similar to the previous analysis of TCP [150, 151], we assume that each TCP flow's RTT is a constant value. Moreover, we assume that the RTO is a constant value, because the TCP sender recomputes the RTO value only after retransmitting the lost packets.

The scenario for $SSTHmax$ PDoS attacks is depicted in Figure 3.6(a). As shown, this scenario corresponds to the case where $T_{i,period}$ is long enough, so that the TCP sender recovers the lost packets as well as increases its *cwnd* to the maximal value of W_i^U. As a result, the *ssthresh* maintains its maximal value of $\frac{W_i^U}{2}$. Therefore, we may use the ratio $\frac{T_{i,period}-RTO_i}{T_{i,period}}$ to estimate the throughput degradation. A similar method has been applied to analyze the shrew attack in [28].

The scenario for SSTHmin-PDoS attacks is depicted in Figure 3.6(b). In this scenario, $T_{i,period}$ is short enough that the *cwnd* cannot reach 4. Thus, the *ssthresh* will be constrained to the minimal value of 2 [27]. Accordingly, during the attack-free period, the TCP sender enters the congestion avoidance phase after sending a packet, because the *cwnd* will reach the *ssthresh* after receiving an ACK. As a result, the amount of data sent in a run is the summation of data segments sent in the slow start phase (i.e., 1), and those sent in the congestion avoidance phase (i.e., $\frac{1}{2}[2 + 2 + \frac{a}{d}(\frac{T_{i,period}-RTO_i}{RTT_i} - 1)](\frac{T_{i,period}-RTO_i}{RTT_i} - 1)S_p = [\frac{a}{2d}(\frac{T_{i,period}-RTO_i}{RTT_i} - 1)^2 + 2(\frac{T_{i,period}-RTO_i}{RTT_i} - 1)]S_p$).

The scenario for SSTHmid-PDoS attacks is depicted in Figure 3.6(c) in which the PDoS attack will drive the *ssthresh* to a constant value. Consequently, the *cwnd* reaches $W_{i,TO}$ before the next attack pulse' arrival, and the *ssthresh* converges to $\frac{1}{2}W_{i,TO}$. By adding an additional equation to Eq. (3.5), we can calculate $W_{i,TO}$ as:

$$(\tau_d)^x = \frac{1}{2}W_{i,TO}, \quad \frac{a}{d}y = \frac{1}{2}W_{i,TO}, (x+y)RTT + RTO = T_{i,period}, \quad x > 0, \ y > 0,$$
$$\Rightarrow W_{i,TO} = 2e^{\frac{-LambertW(\mathfrak{L})RTT_i + \ln(\tau_d)(Ti,period-RTO_i)}{RTT_i}}, \ \mathfrak{L} = \frac{a}{d}\ln(\tau_d)e^{\frac{\ln(\tau_d)(T_{i,period}-RTO_i)}{RTT_i}} \quad (3.9)$$

Therefore, the total transmitted TCP data in each run consists of those sent during the slow start phase (i.e., $S_p \sum_{i=0}^{x} (\tau_d)^i = S_p \frac{\frac{1}{2}\tau_d W_{i,TO} - 1}{\tau_d - 1}$) and those sent during the congestion avoidance phase (i.e., $S_p(\frac{W_{i,TO}}{2} + W_{i,TO})\frac{y}{2} = S_p \frac{3d}{8a} W_{i,TO}^2$). ■

3.3 AIMD-based PDoS attacks

In an AIMD-based attack, the attack pulses cause a victim TCP sender to frequently enter the fast retransmit/fast recovery state (FR). Recall that we consider a general AIMD algorithm denoted by AIMD(a, b), $a > 0,\ 1 > b > 0$. If an attack pulse is able to cause some packet losses in a TCP connection and yet a sufficient number of duplicate ACKs can still be received by the sender, the *cwnd* will drop by $(1 - b)\%$. After that, the *cwnd* will increase by one MSS every RTT.

Since it will take at least $\frac{(1-b)d}{a} W$ number of RTTs to restore the *cwnd* to W after a decrease from W to bW, the *cwnd* value could drop continuously if the attack pulses are launched frequently enough. Moreover, when the *cwnd* is dropped to a certain level, there may not be enough duplicate ACKs to trigger the fast recovery process. Thus, the AIMD-based attack can also achieve a similar effect as the timeout-based attack without causing timeouts at the beginning of the attack. On the other hand, the AIMD-based attack could also launch a degradation-of-service attack by lowering the attack frequency. Similar to the timeout-based attacks, there are two types of AIMD-based attacks: aperiodic and periodic.

3.3.1 Aperiodic attacks

Since aperiodic attacks may have arbitrary patterns, we analyze only one special case, which is referred to as synchronous in the sense that the attack epoches always coincide with a fixed set of *cwnd* values. We therefore refer it to as synchronous AIMD-based attack. For example, consider a general AIMD algorithm AIMD(a, b), where $a > 0,\ 1 > b > 0$, and the sender's *cwnd* is increased by $\frac{a}{d}$ per RTT. Let $W_{i,n}, n = 0, 1, 2, \ldots$, be the *cwnd* value of the ith victim TCP connection *just before*

the nth attack epoch. Therefore, $W_{i,0}$ is the *cwnd* value just before the attack. Suppose that an attack epoch always occurs at the instant when the *cwnd* rises from $W_{i,n-1}$ to $fW_{i,n-1}, n \geq 1, 1 \geq f > b$ after a multiplicative decrease. Figure 3.7 shows an example of a synchronous AIMD-based attack. The dashed line depicts the trajectory of *cwnd* controlled by AIMD$(1, 0.5)$. The solid line, on the other hand, depicts the trajectory of *cwnd* when the TCP sender is experiencing a synchronous AIMD-based attack. The attack epoches for this type of synchronous AIMD-based attack are given in Proposition 3.3.1.

Proposition 3.3.1 (Attack epoches for a synchronous AIMD-based attack) *For the synchronous AIMD-based attack just described, the number of attack pulses required to reduce the ith sending TCP's cwnd to 2 (the minimum value) is given by* $\frac{log(2/W_{i,0})}{log f}$.

Assume that the RTT value is fixed. Let t_0 *be the first attack epoch. For* $n > 0$*, the* n*th attack epoch is then given by*

$$t_n = t_0 + \frac{1 - f^n}{1 - f} \frac{(f - b)dW_{i,0}}{a} \times RTT_i, \quad n \geq 1. \tag{3.10}$$

Proof Since the *cwnd* is decreased from W_i to fW_i at each attack epoch, $cwnd = f^n W_{i,0}$ after the nth attack. Hence, the attack will bring down the *cwnd* to 2 by launching a sequence of $\frac{log(2/W_{i,0})}{log f}$ attack pulses.

According to the attack strategy, before the arrival of the nth ($n \geq 1$) attack pulse the *cwnd* can only be increased to $(f - b)W_{i,n-1}$, which takes $\frac{(f-b)dW_{i,n-1}}{a} RTT_i$ amount of time, according to the AIMD algorithm. Therefore, the nth attack epoch should take place at

$$t_n = t_{n-1} + \frac{(f - b)dW_{i,n-1}}{a} RTT_i, \quad n \geq 1. \tag{3.11}$$

Furthermore, by substituting $W_{i,n} = fW_{i,n-1}$ into Eq. (3.11),

$$t_n = t_{n-1} + \frac{(f - b)df^{(n-1)}W_{i,0}}{a} RTT_i, \quad n \geq 1. \tag{3.12}$$

We can therefore obtain Eq. (3.10) by a repeated substitution of t_{n-1}. ■

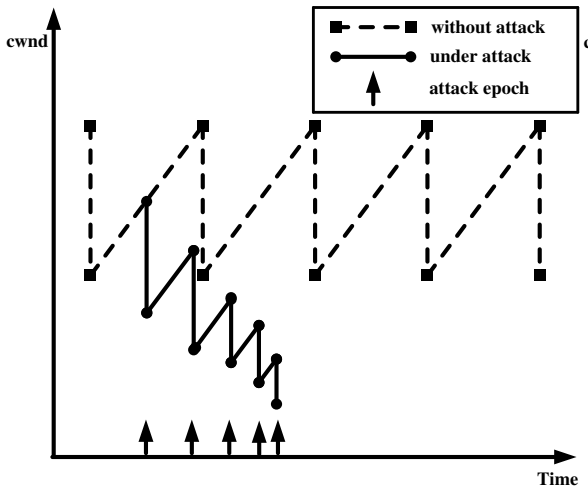


Fig. 3.7. An example of a synchronous AIMD-based attack against AIMD$(1, 0.5)$.

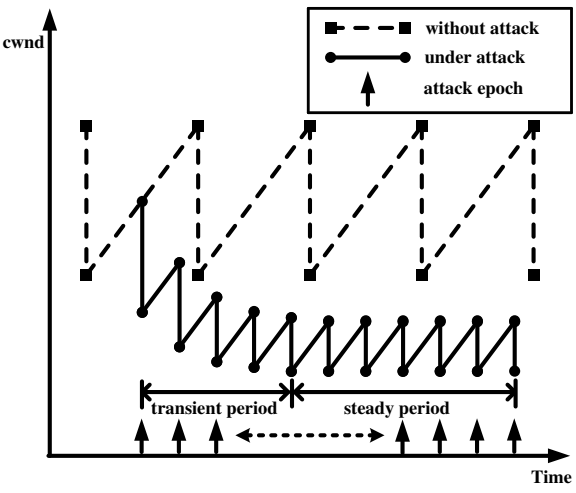


Fig. 3.8. An example of an AIMD-based attack with fixed periods.

3.3.2 Periodic attacks

Similar to the case of synchronous timeout-based attacks, it is difficult to launch a synchronous AIMD-based attack because of the difficulty in estimating the attack epoches. Therefore, we remove the synchronization requirement and consider an AIMD-based attack with a fixed period of $T_{attack} = T_{space} + T_{extent}$, as shown in Figure 3.8. Proposition 3.3.2 presents the steady-state value of *cwnd* in the midst of such attack. After that, Proposition 3.3.3 gives the minimum number of attack pulses for reducing the *cwnd* to the steady-state value.

Proposition 3.3.2 (Convergence of the *cwnd*) *Consider an AIMD-based attack with a fixed period of T_{attack} against the ith TCP connection using AIMD(a, b). If the cwnd of the victim connection will converge during the attack, then the converged value is given by*

$$W_{i,C} = \frac{a}{(1-b)d} \frac{T_{attack}}{RTT_i}. \tag{3.13}$$

Proof Just before the arrival of the $(n+1)$th attack pulse, the *cwnd* value is given by

$$W_{i,n+1} = bW_{i,n} + \frac{a}{d}\frac{T_{attack}}{RTT_i} = b^{n+1}W_{i,0} + \frac{a}{d}\frac{1-b^{n+1}}{1-b}\frac{T_{attack}}{RTT_i}. \tag{3.14}$$

If the *cwnd* converges, $W_{i,n+1} = W_{i,n}$ for some n. Therefore, by substituting $W_{i,n+1} = W_{i,n}$ into Eq. (3.14), we obtain the result. ■

Proposition 3.3.3 (Minimum number of attack pulses) *Consider an AIMD-based attack with a fixed period of T_{attack} against AIMD(a, b) adopted by the ith TCP sender. Let $W_{i,0} = W_{i,C} + \delta$, where $\delta > 0$. Moreover, if $W_{i,n} - W_{i,C} < \epsilon$, where ϵ is a small value, $W_{i,n}$ is considered the same as $W_{i,C}$. Then, the minimum number of attack pulses required to reduce the cwnd from $W_{i,0}$ to $W_{i,C}$ is given by*

$$N_{attack} < \frac{log\epsilon - log\delta}{logb}. \tag{3.15}$$

Proof From Eq. (3.13) and Eq. (3.14), we have $W_{i,n} = b^n \times W_{i,0} + (1-b^n)W_{i,C}$. By substituting $W_{i,0} = W_{i,C} + \delta$ into the equation and solving for n, we obtain $n = \frac{\log(W_{i,n}-W_{i,C})-\log\delta}{\log b}$. Since $W_{i,n}$ is considered to be the same as $W_{i,C}$ if $W_{i,n} - W_{i,C} < \epsilon$, we obtain Eq. (3.15). ■

In Figure 3.9 we plot Eq. (3.15) for different values of b. The figure shows that the flow throughput of a typical TCP ($b = 1/2$) can be brought to the converged value using fewer than 10 attack pulses. With a higher value of b, more attack pulses will be required to achieve the same effect, because the *cwnd* drops with a slower rate in these cases. With a higher value of δ, it will also take a longer time for *cwnd* to converge.

To drive the point further, in Figure 3.10 we use Eq. (3.13) to show the relationship between W_C and $\frac{T_{attack}}{RTT}$ for a TCP flow and a TCP-friendly flow (AIMD$(0.31, 0.875)$) with $d = 2$. According to [146], a flow with AIMD(a, b) is considered to be *TCP-friendly* if its parameters satisfy $a = \frac{4\times(1-b^2)}{3}$. Therefore, the converged *cwnd* value for a TCP-friendly flow is given by $W_C = \frac{4\times(1+b)}{3\times d} \times \frac{T_{attack}}{RTT}$. The figure also shows a lower bound and an upper bound on W_C.

The figure shows that if $\frac{T_{attack}}{RTT}$ is small, the flow's *cwnd* will be constrained to a very low value that will severely limit the flow's throughput. For example, consider those flows with RTT between 200ms and 500ms. In [28], the simulation results have shown that these flows will survive a periodic timeout-based attack. However, Figure 3.10 shows that a periodic AIMD-based attack with a period of one second is sufficient to degrade their throughput to the extent that the *cwnd* will be confined within $(4/3, 20/3)$. Note that the TCP fast recovery algorithm usually requires three duplicate ACKs. Therefore, even if the *cwnd* value is given by the upper bound, it is very likely that the fast recovery procedure cannot be started and that a timeout will therefore occur.

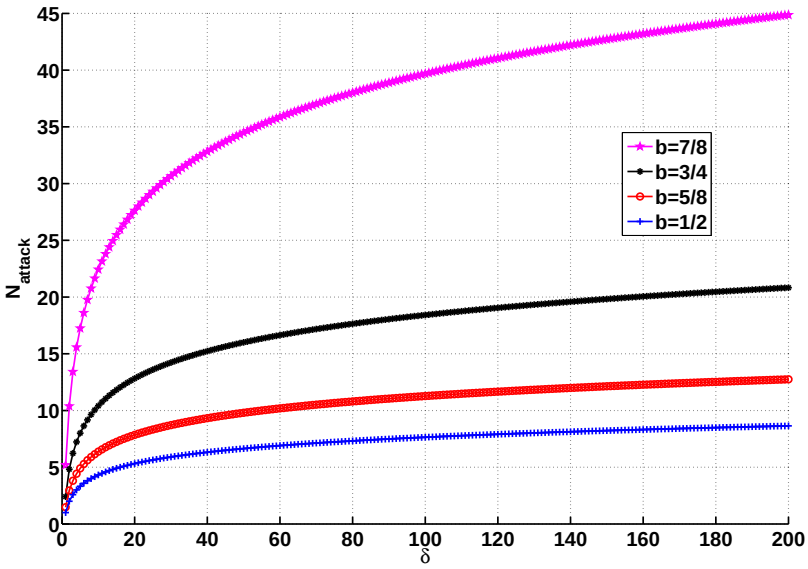


Fig. 3.9. Relationship between N_{attack} and δ.

Proposition 3.3.1 gives the throughput of the ith victim TCP connection under a periodic AIMD-based PDoS attack.

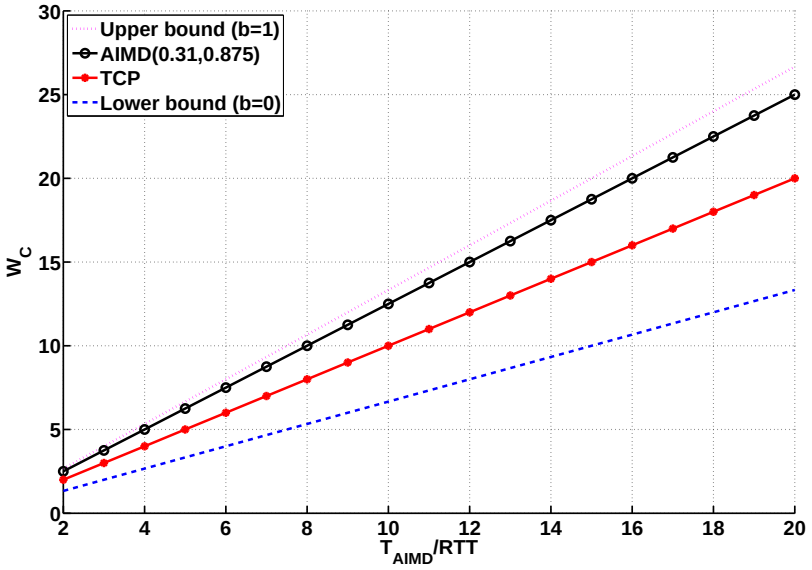


Fig. 3.10. Relationship between W_C and $\frac{T_{attack}}{RTT}$.

Theorem 3.3.1 *The throughput of the ith victim TCP connection under an AIMD-based PDoS attack with a period of* T_{attack} *is given by*

$$\Psi_{i,attack}^{AIMD} = \left\{ \sum_{n=1}^{N_{attack}-1} \left(bW_{i,n} + \frac{a}{2d} \frac{T_{attack}}{RTT_i} \right) \frac{T_{attack}}{RTT_i} + \frac{a(1+b)}{2d(1-b)} \left(\frac{T_{attack}}{RTT_i} \right)^2 (N - N_{attack}) \right\} S_p,$$

Proof We divide the whole TCP process into two distinct phases, as shown in Figure 3.8. The first phase is the transient period, which starts from the beginning of the attack and ends when the *cwnd* converges to $W_{i,C}$. During this period, the attacker will send N_{attack} attack pulses; therefore, there are $N_{attack} - 1$ free-of-attack intervals for TCP sender to transmit packets. During the interval between jth and $(j+1)$th attack epoch $(1 \leq j < N_{attack})$, the TCP sender can send $(bW_{i,j} + \frac{a}{2d} \frac{T_{attack}}{RTT_i}) \frac{T_{attack}}{RTT_i}$ packets. Therefore, the first item within the curly brackets in Eq. (3.16) gives the number of packets sent during the transient state.

The second phase, which is referred to as the steady period, follows immediately after the transient phase. In this phase, the *cwnd* exhibits a periodic sawtooth pattern. There are a total of $N - N_{attack}$ such periods, each of which begins after the jth attack epoch and ends before the $(j+1)$th attack epoch $(N_{attack} \leq j < N)$.

The number of packets transmitted during each period is $(bW_{i,C} + W_{i,C})\frac{T_{attack}}{2RTT_i} = \frac{a(1+b)}{2d(1-b)}(\frac{T_{attack}}{RTT_i})^2$. Therefore, the second item within the curly brackets in Eq. (3.16) gives the number of packets sent during the steady period. ■

3.4 Polymorphic DoS attacks

In this section, we will first model the PMDoS attack which can be turned into existing PDoS attacks under certain configurations. And then we will analyze the impact of PMDoS attack on TCP flows.

3.4.1 Modeling the PMDoS attack

We model a PMDoS attack as an *alternating renewal process*, which controls the sending rate of attack packets that can be in one of two states: R_{attack} for a period of T_{on}, and 0 *bps* (bits per second) for a period of T_{off}. Therefore, based on the results from *Renewal Theory* [152], we obtain the average attack rate $R_{average}$ of a PMDoS attack in Eq. (3.16).

Lemma 3.4.1 *The average attack rate of a PMDoS attack is given by*

$$R_{average} = \frac{E[T_{on}]R_{attack}}{E[T_{on}] + E[T_{off}]}, \tag{3.16}$$

where T_{on} and T_{off} are i.i.d. random variables. Besides, according to the definition of low-rate attack, the PMDoS attack should satisfy the following constraint:

$$R_{average} \leq R_{bottle}, \tag{3.17}$$

where R_{bottle} is the bandwidth of the bottleneck.

Based on Eq. (3.16), we know that the PMDoS attack is equivalent to a PDoS attack when T_{on} and T_{off} are degenerate random variables (constant values). Moreover, if T_{off} is close to one second and T_{on} is approximately equal to the RTT of victim TCP flows, then the PMDoS attack is equivalent to a shrew attack. Furthermore,

when T_{off} goes to 0, the PMDoS attack becomes a flooding-based DoS attack with a constant sending rate, for which Eq. (3.17) becomes

$$R_{attack} \leq R_{bottle}. \tag{3.18}$$

3.4.2 Analyzing the PMDoS attack

Based on the relationship between R_{attack} and R_{bottle}, we divide the PMDoS attack into two categories:

1. When $R_{attack} \leq R_{bottle}$, the PMDoS attack will behave like a constant-bit-rate (CBR) source. We denote this class of PMDoS attacks by $\mathbb{A}^{-}$,

2. When $R_{attack} > R_{bottle}$, the PMDoS attack will dispatch intermittent attack pulses because $T_{off} > 0$. We denote this class of PMDoS attacks by $\mathbb{A}^{+}$,

Proposition 3.4.1 *If the average value of* cwnd *in the absence of PMDoS attack is* $\mathbf{E}[W]$*, then under* $\mathbb{A}^{-}$ *PMDoS attack it becomes*

$$\mathbf{E}[W^{-}] = \left(1 - \frac{R_{attack}}{R_{bottle}}\right) \mathbf{E}[W]. \tag{3.19}$$

Proof Since the victim TCP senders will increase their packet transmission rates whenever there is available bandwidth, their flows will make full use of the available bandwidth ($R_{bottle} - R_{attack}$) after the $\mathbb{A}^{-}$ PMDoS attack occupies R_{attack} portion of R_{bottle} [45] (i.e., TCP's packet transmission rate $\mathbf{E}[R_{tcp}] = R_{bottle} - R_{attack}$). Since R_{tcp} can be modeled as $\mathbf{E}[R_{tcp}] = \frac{3\mathbf{E}[W]}{4RTT} MSS$ [150], where MSS is the maximum segment size and RTT is assumed to be a constant value as [150, 153], $\mathbf{E}[W]$ will decrease the same percent ($1 - \frac{R_{attack}}{R_{bottle}}$) as R_{tcp} does. ■

For the $\mathbb{A}^{+}$ PMDoS attack, since $R_{attack} > R_{bottle}$, the bottleneck will be blocked for a period of $\frac{R_{attack} T_{on}}{R_{bottle}}$. We assume that during such period all the victim TCP flows will suffer from packet loss and then enter the fast retransmit/fast recovery (FR) state, during which each TCP flow will employ an AIMD algorithm to control

its *cwnd*. Therefore, we can model the ith sending TCP's *cwnd* in the presence of $\mathbb{A}^+$ PMDoS attacks as:

$$W_i^+(n+1) = bW_i^+(n) + \frac{a}{d}\frac{T_{on}(n) + T_{off}(n)}{RTT_i}. \tag{3.20}$$

Eq. (3.20) belongs to the general class of classical stochastic difference equation [154, 155]:

$$Y(n+1) = A(n)Y(n) + B(n), \quad n \geq 0. \tag{3.21}$$

Since the attacker can send an arbitrary sequence of attack pulses, we consider a general attack pattern for which $T_{period}(n) = T_{on}(n) + T_{off}(n)$ is a stationary and ergodic stochastic process. Based on Theorem 1 in [154], we can obtain the solution to Eq. (3.20) in Proposition 3.4.2.

Proposition 3.4.2 *If $T_{period}(n)$ is a stationary and ergodic stochastic process, then there is a unique stationary solution for Eq. (3.20) as follows:*

$$W_i^{+*}(n) = \frac{a}{dRTT}\sum_{j=0}^{\infty} b^j (T_{period}(n-j-1)). \tag{3.22}$$

Proof Let $A(n) = b$ and $B(n) = \frac{a}{d}\frac{T_{period}(n)}{RTT}$. It can easily be proved that $A(n)$ and $B(n)$ fulfill the following requirements in [154, 155]:

$$-\infty \leq \mathbf{E}[log(|A(0)|)] < 0. \tag{3.23}$$

$$\mathbf{E}[log(|B(0)|)]^+ < \infty, \; x^+ = max(0, x), \forall x \in \mathbb{R}. \tag{3.24}$$

■

We use the similar technique in [156] to characterize the *cwnd* under the $\mathbb{A}^+$ PMDoS attack. Since $W_i^+(n)$ will converge to $W_i^{+*}(n)$ absolutely almost surely according to Theorem 1 in [154], we can obtain the converged value by calculating the expectation of $W_i^{+*}(n)$.

Corollary 3.4.1 *The expectation of $W_i^{+*}(n)$ is given by*

$$\mathbf{E}[W_i^{+*}(n)] = \frac{a\mathbf{E}[T_{period}(n)]}{RTT_i(1-b)d}. \tag{3.25}$$

If both T_{on} and T_{off} are constant values and let $T_{attack} = T_{on} + T_{off}$, then we get the same converged value (i.e., $W_{i,C} = \frac{aT_{attack}}{dRTT_i(1-b)}$) in [4].

Proof

$$\begin{aligned}
\because \mathbf{E}[W_i^{+*}(n)] &= \frac{a}{dRTT_i}\sum_{j=0}^{\infty} b^j \mathbf{E}[T_{period}(n-j-1)]. \\
\mathbf{E}[T_{period}(n)] &= \mathbf{E}[T_{on}(n) + T_{off}(n)]. \\
\therefore \mathbf{E}[W_i^{+*}(n)] &= \frac{a(\mathbf{E}[T_{on}(n)] + \mathbf{E}[T_{off}(n)])}{dRTT_i(1-b)}.
\end{aligned}$$

■

3.5 Evaluating the impact of PDoS attacks

3.5.1 Simulation experiments

We have conducted extensive ns-2 simulation experiments to validate our analytical results and to evaluate the impact of DoS attacks on different AQMs. The network topology used in the simulations is depicted in Figure 3.11. The network consists of M pairs of TCP senders and receivers. All the links, except for the bottleneck between routers S and R, are $R_{access} = 50$ Mbps. The two routers are connected through a link of $R_{bottle} = 10$ Mbps. There are ten legitimate TCP flows traversing through the bottleneck link, all of which are based on TCP New Reno, and their RTTs range from 20ms to 460ms as suggested in [28]. The minRTO of each flow is equal to one second according to the recommendation in [30]. Based on the scripts provided by [28], all the simulation experiments were performed in the ns-2 2.28 environment. The queue size (QS) is 100 packets and the AQMs' parameters are listed in Table 3.2.

We derive $\sum_{i=1}^{N_f} \Psi_{i,normal}$ in Proposition 3.5.1 and $\sum_{i=1}^{N_f} \Psi_{i,attack}$ for FDDoS attacks in Proposition 3.5.2.

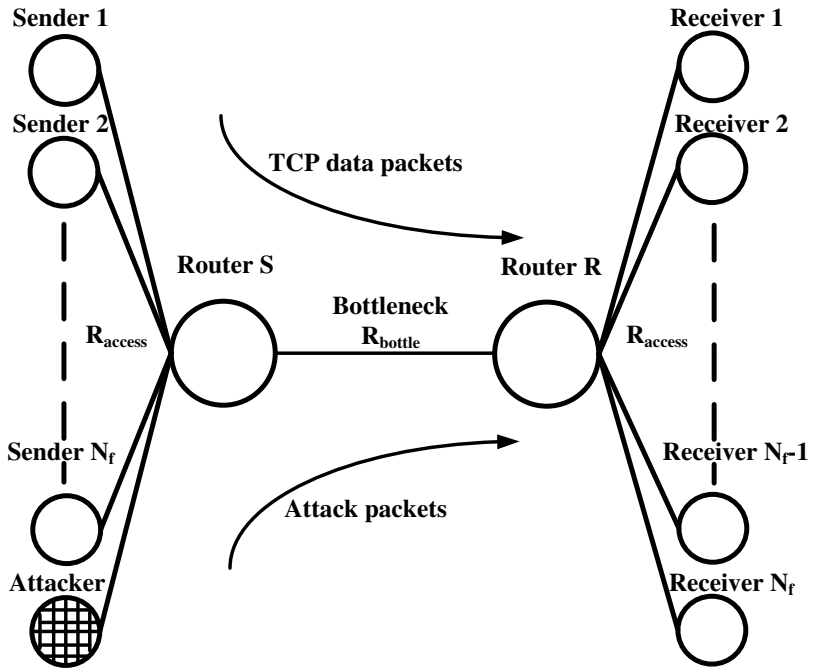


Fig. 3.11. Network topology for the simulation studies.

Table 3.2
Parameters for the four AQMs used in the simulation studies.

AQMs	Customized Parameters
RED	$max_{th} = 0.8QS$, $min_{th} = 0.2QS$, $max_p = 0.1$, $w_q = 0.002$, gentle=ture
REM	$b^* = 0.6QS$, $\gamma = 0.001$, $\phi = 1.001$
PI	$q_{ref} = 0.6QS$, $a = 0.00001822$, $b = 0.00001816$
AVQ	$\alpha = 0.15,\ \gamma = 0.98$

Proposition 3.5.1 *Since TCP flows will make a full use of the bottleneck bandwidth in the absence of attacks [148], we have*

$$\sum_{i}^{N_f} \Psi_{i,normal} = R_{bottle} T_{total}/8, \tag{3.26}$$

where T_{total} denotes the period that the TCP flows are under a DoS attack. For the case of PDoS attacks with N pulses, $T_{total} = (N-1)T_{attack}$.

Proposition 3.5.2 *In the presence of a FDDoS attack with $R_{attack} = \beta R_{bottle}$, $0 < \beta \leq 1$, $\sum_{i=1}^{N_f} \Psi_{i,attack} = (1-\beta) \sum_{i=1}^{N_f} \Psi_{i,normal}$.*

Proof Since we model a FDDoS attack as a traffic source with constant bit rate, its impact on the normal traffic is approximately the same as reducing the available bandwidth by the attack rate. According to the TCP congestion control mechanism, the TCP throughput will increase when there is additional bandwidth to transfer packets. Thus, the TCP flows will make a full use of the remaining bandwidth. ■

To simplify the notations, we label the periodic timeout-based PDoS attacks as PT-PDoS and the periodic AIMD-based PDoS attacks as PA-PDoS. In a PT-PDoS attack, each legitimate TCP flow is forced to enter the TO state by a PDoS attack pulse. Therefore, this scenario corresponds to the most severe impact inflicted by the attack. In a PA-PDoS attack, each legitimate TCP flow is forced to enter the FR state by an attack pulse. Obviously, there are many other possibilities, depending on the PDoS attack power.

The experiments

Figures 3.12-3.13 plot the attack power Γ verses the attack cost γ for the FDDoS and PDoS attacks for two different values of R_{attack}. Each figure has 4 sub-figures showing different values of T_{extent} for the PDoS attack scenarios, which obviously do not affect the FDDoS attack results. For the FDDoS attack, we only present the analytical results (the solid straight lines), because they match very well with the simulation results. As for the PDoS attacks, the 2 solid lines are obtained from the analytical results for the PT-PDoS and PA-PDoS attacks which are derived without considering specific queue management schemes. On the other hand, the 5 dashed lines are obtained from the simulation results for the 5 queue management schemes under the PDoS attack.

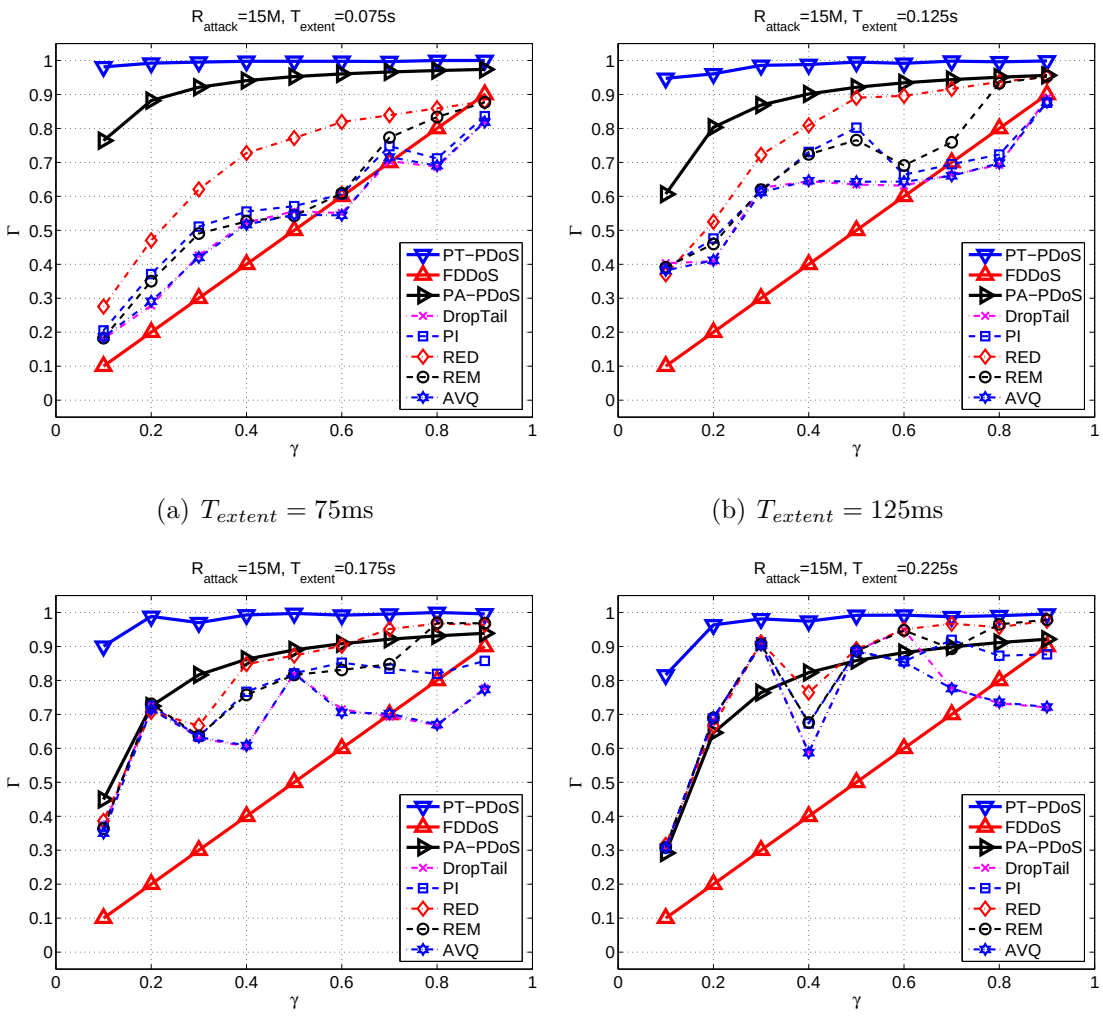
Figures 3.14-3.15 present the simulation results for the packet dropping rates, denoted as ζ, for the PDoS and FDDoS attacks, respectively. To clearly explain the results, we have also included the corresponding graphs for the attack power. In Figure 3.14, the PDoS attacks were launched with $T_{extent} = 125$ms and $R_{attack} = \{20, 30\}$ Mbps. We have computed ζ separately for the legitimate TCP packets (denoted by TP) and for the attack packets (denoted by AP). For example, *RED-TP* refers to the ζ for the legitimate TCP packets and RED is in use. This is similarly done for the FDDoS attacks in Figure 3.15.

Figure 3.16 gives the packet dropping probabilities used in the 3 RED-like AQM algorithms measured during the PDoS attacks with $T_{extent} = 125$ms, $R_{attack} = \{10, 20, 30\}$ Mbps, and $\gamma = 0.3$. As we shall see, this set of results is useful in explaining why RED drops more legitimate TCP packets than REM and PI do.

The PDoS attack power

According to Figures 3.12-3.13, the results for the PT-PDoS attack can be regarded as the upper bound for the PDoS attack power. Moreover, the figures show abrupt changes in the attack power for some parameter settings, for example, $\gamma = 0.3$ in Figure 3.12(d) and $\gamma = 0.6$ in Figure 3.13(c). In these cases, the attack periods ($T_{attack} = 1125ms, 1021$ms) are very close to that of the shrew attack [28]. Therefore, the PDoS attacks will drive the TCP flows into the TO state as soon as the TCP senders' retransmission timers expire, thus causing a very severe throughput degradation. These special attack parameters are referred to as *Shrew points* in [47].

For a given R_{attack}, the simulation results approach to those given by the PA-PDoS and PT-PDoS attacks as T_{extent} increases. That is, the PDoS attack power increases with T_{extent}, because more attack packets are sent in each attack pulse, which will quickly ramp up the packet dropping probability for the queue management schemes. As a result, more legitimate TCP packets will be dropped.



(a) $T_{extent} = 75$ms

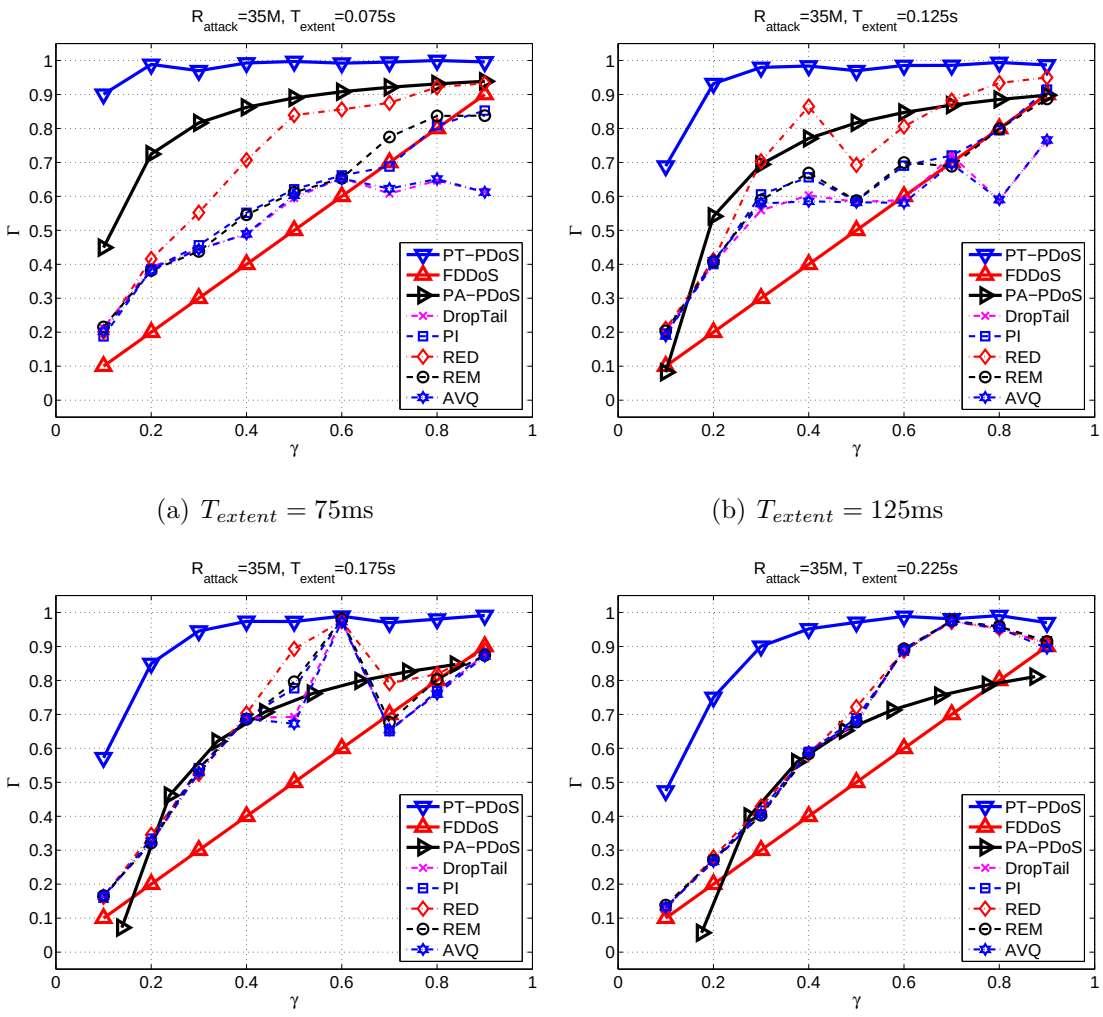
(b) $T_{extent} = 125$ms

(c) $T_{extent} = 175$ms

(d) $T_{extent} = 225$ms

Fig. 3.12. DoS attack power with $R_{attack} = 15$ Mbps.

Another interesting result is that the trend of the simulation results obtained for RED coincides very well with that of PA-PDoS attack in some cases, such as Figure 3.12(c) and Figure 3.13(b). Recall that a PA-PDoS attack forces each TCP flow to enter the FR state. On the other hand, *RED* uses an uniform dropping mechanism to avoid consecutive packet dropping [60], which therefore affects more TCP flows during a PDoS attack. Hence, the simulation results for RED are in good match with the analytical results obtained for the PA-PDoS attacks.



(a) $T_{extent} = 75$ms

(b) $T_{extent} = 125$ms

(c) $T_{extent} = 175$ms

(d) $T_{extent} = 225$ms

Fig. 3.13. DoS attack power with $R_{attack} = 35$ Mbps.

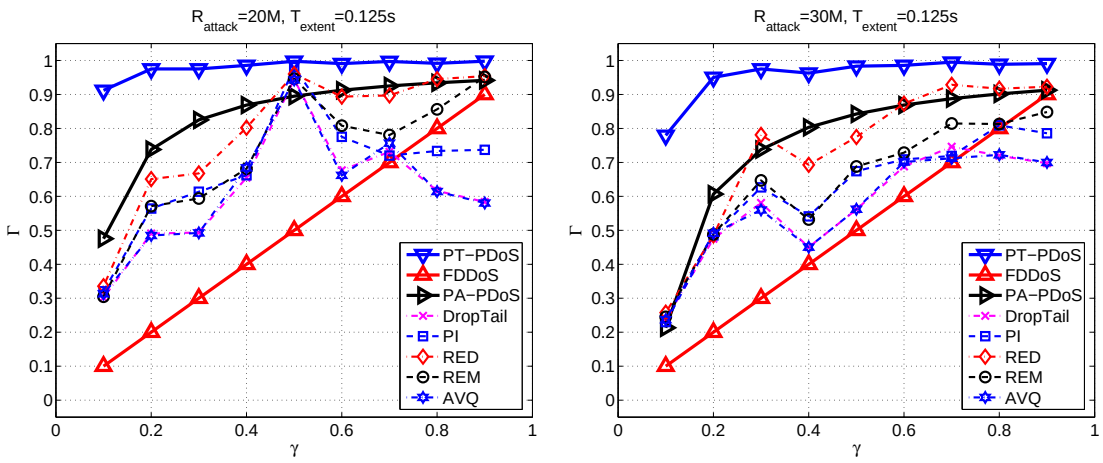
The resilience level of DropTail and AQMs under PDoS attacks

Based on the throughput degradation results in Figures 3.12-3.13, we can compare the resilience levels of the queue management schemes to the PDoS attacks. The figures have concluded the following order of resilience level for the five schemes: $\{AVQ, DropTail\} \geq \{PI, REM\} \geq RED$. The ones within $\{\}$ are considered to have very similar resilience levels.

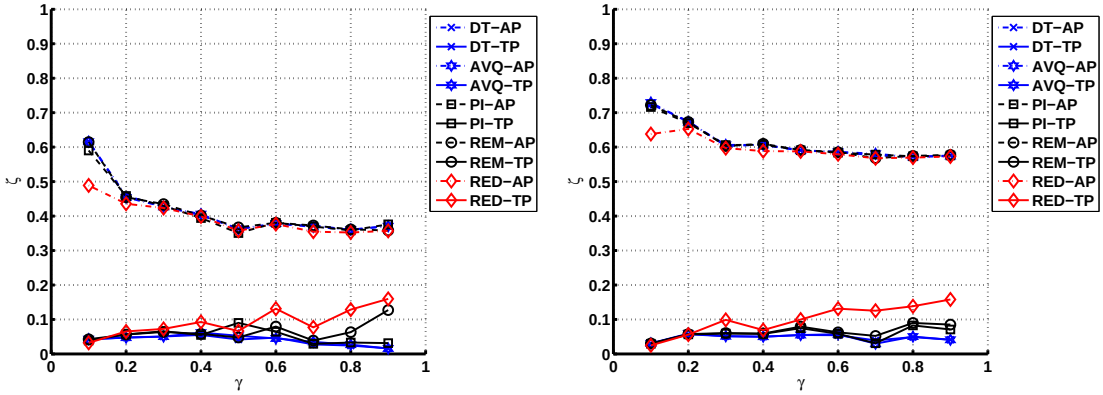
In Figure 3.14(c-d), the curves for the attack packets (AP) are all clustered together in the range of $\zeta = 0.35 - 0.6$. The curves for the legitimate packets (TP), on the other hand, lie below the curves for the attack packets. That is, the packet dropping rates for the attack packets are always higher than that for the legitimate packets. Besides, DropTail and AVQ drop relatively more attack packets but less TCP packets, while the RED-like AQMs drop relatively less attack packets but more TCP packets. In particular, RED drops the least number of attack packets but the largest number of TCP packets on average. This result is due to its random drop mechanism which lets the attack packets pass through the router even when the queue is full. These attack packets also push up the packet dropping probability for the legitimate TCP packets. On the contrary, DropTail and AVQ will drop all the subsequent attack packets whenever the queue is full, thus effectively dampening the power of the attack pulse.

Figure 3.16 reveals 2 factors responsible for the inferior performance of RED as compared with REM and PI. First, the abrupt arrivals of the attack packets increase RED's average queue length drastically, thus resulting in a very high packet dropping probability for both attack packets and legitimate TCP packets. However, RED's uniform dropping cannot drop the attack packets quickly enough, which instead increases the dropping of legitimate TCP packets. Second, RED's packet dropping probability decreases more slowly than REM and PI, because both REM and PI use the instantaneous queue length to compute the packet dropping probability.

Furthermore, as R_{attack} or T_{extent} increases, the results for AVQ and DropTail are almost the same, because they essentially have the same packet dropping strategy, except for the use of a virtual queue in AVQ. Similarly, REM and PI have very similar results, because both are designed based on the idea of proportional-integral controller.



(a) Γ for PDoS attack with $R_{attack} = 20$ Mbps (b) Γ for PDoS attack with $R_{attack} = 30$ Mbps



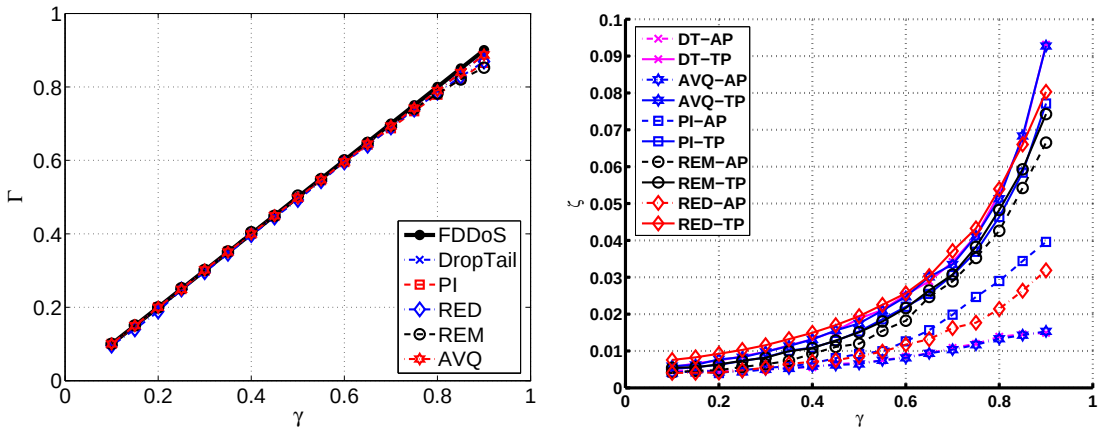
(c) ζ for PDoS attack with $R_{attack} = 20$ Mbps (d) ζ for PDoS attack with $R_{attack} = 30$ Mbps

Fig. 3.14. Attack power and packet dropping rates under PDoS attacks with $T_{extent} = 125$ms.

The resilience level of DropTail and AQMs under FDDoS attacks

Figure 3.15(a) shows that the simulation results for the FDDoS attack are very close to the analytical results. Figure 3.15(b) shows that the packet dropping rates for the attack packets and the TCP packets under DropTail and the 4 AQMs are very similar when γ is small, but they diverge as γ increases. Moreover, the difference is smaller for the RED-like AQMs when compared with DropTail and AVQ. This shows that the RED-like AQMs can achieve a higher resilience level than DropTail

and AVQ, which is opposite to the results obtained under the PDoS attacks. This can be explained by the fact that TCP flows always try to make a full use of the available bandwidth. Therefore, the random drop mechanism employed by the RED-like AQMs offers a better chance for the TCP flows to use the extra bandwidth by dropping the attack packets, while the DropTail and AVQ do not have such mechanism.



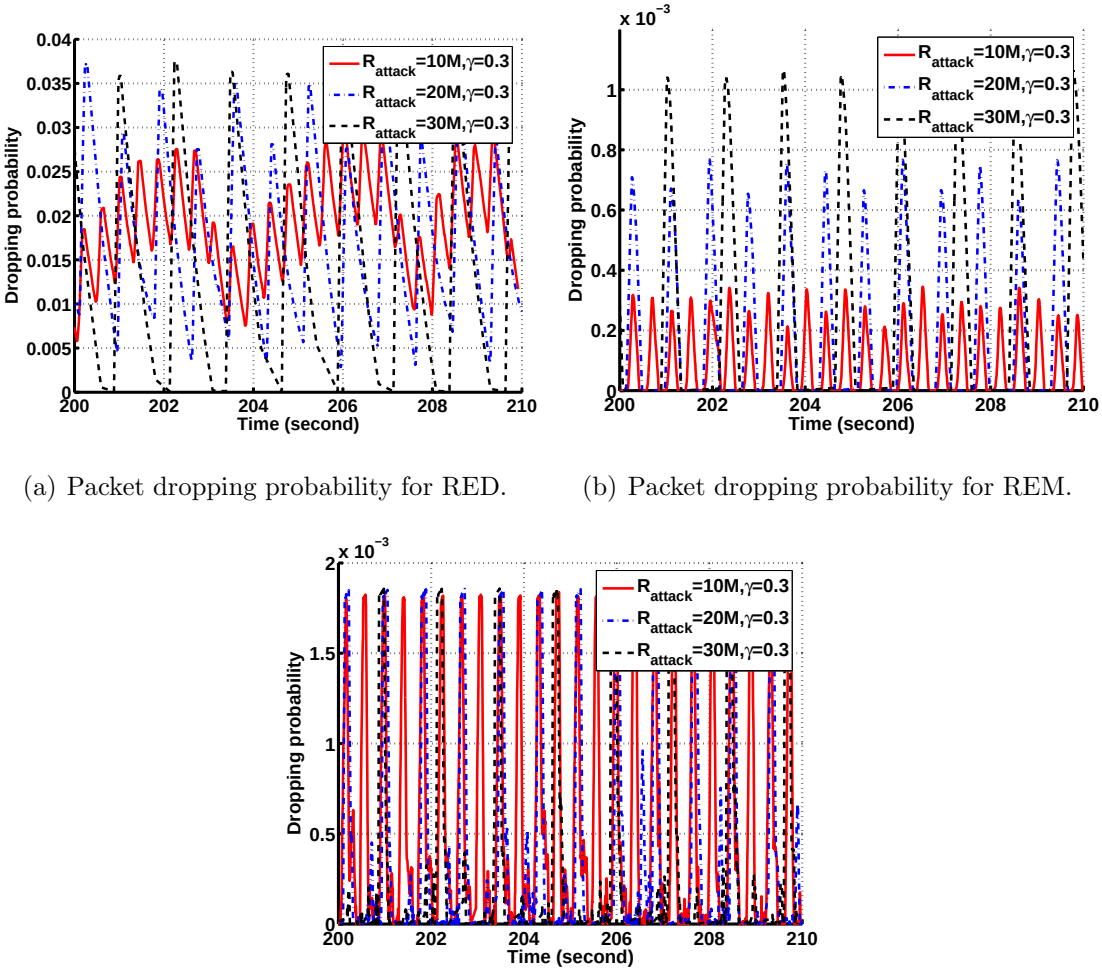
(a) Γ under FDDoS

(b) ζ under FDDoS

Fig. 3.15. Attack power and packet dropping rates under FDDoS attacks.

3.5.2 Test-bed experiments

We have carried out our evaluation using the topology in Figure 3.17, which consists of n routers shared by a number of legitimate TCP users and an attacker. All the links, except the bottleneck link, between the routers R_n (the bottleneck router) and R_{n+1} have a one-way propagation delay of $t_l$ms and a capacity of r_l Mbps. The link is shared by n_t long-lived legitimate TCP flows, an attack flow and cross traffic generated by n_c long-lived TCP flows. The bottleneck link has a one-way propagation delay of $t_b$ms and a capacity of r_b Mbps, and carries the traffic excluding the cross traffic.



(a) Packet dropping probability for RED.

(b) Packet dropping probability for REM.

(c) Packet dropping probability for PI.

Fig. 3.16. Packet dropping probabilities for RED, REM, and PI under PDoS attacks.

To demonstrate the impact of different variants of PMDoS attacks (periodic $\mathbb{A}^+$, stochastic $\mathbb{A}^+$, and $\mathbb{A}^-$ PMDoS attacks) on the n_t long-lived legitimate TCP flow aggregates, we have conducted extensive test-bed experiments.

We use the following parameter settings: $n = 2$, $n_t = 15$ (New Reno), $n_c = 10$ (New Reno), $t_l = 15$ms, $t_b = 30$ms, $r_l = 100$ Mbps, and $r_b = 10$ Mbps. Each of the legitimate TCP flows experiences a fixed RTT of 150ms and employs a minimum RTO of 1000ms. The queue at the bottleneck router R_1 is a droptail queue of size

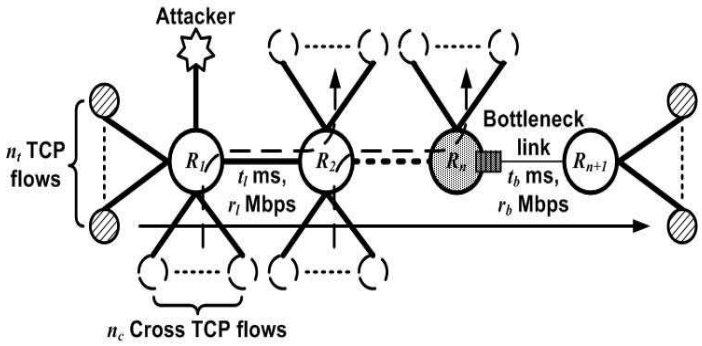
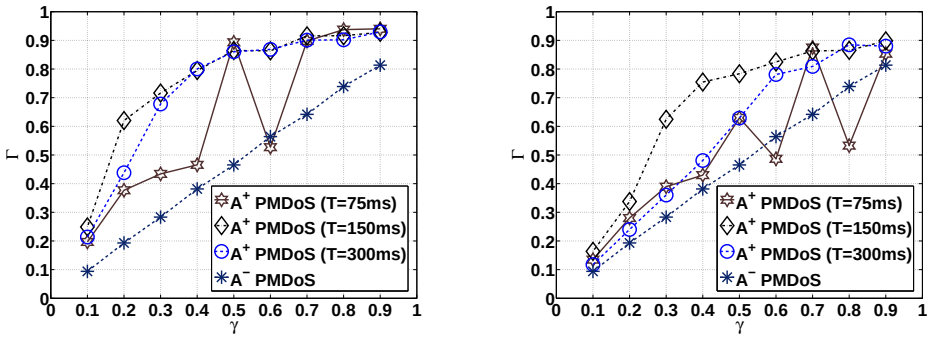


Fig. 3.17. Network topology for evaluating the impacts of PDoS attacks.

$Q = RTT \times r_b$. The period of each experiment is 730s. After 250s, the attacker launches a PMDoS attack until the end of the experiment period. Both the legitimate flows and the cross traffic are generated using `iperf` [157]. To simplify the notations, we use T and R to represent T_{on} and R_{attack} in the following figures, respectively.

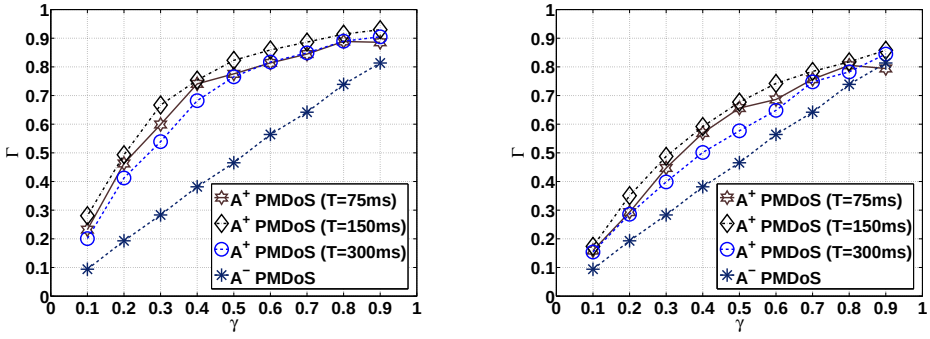
Figure 3.18 depicts the experiment results of Γ resulted from the periodic $\mathbb{A}^+$ PMDoS attacks with $R_{attack} = \{25, 50\}$ Mbps and $T_{on} = \{75, 150, 300\}$ms versus the attack cost γ. Figure 3.19 depicts the experiment results for the stochastic $\mathbb{A}^+$ PMDoS attacks. For the purpose of comparison, each sub-figure also includes the experiment results for the $\mathbb{A}^-$ PMDoS attacks.

These figures give insight into the performance difference among the three variants of PMDoS attacks. First, note that the throughput of the TCP flow aggregates is significantly reduced by both the $\mathbb{A}^+$ and $\mathbb{A}^-$ PMDoS attacks. However, the impact of the periodic and stochastic $\mathbb{A}^+$ PMDoS attacks are generally far more significant than the $\mathbb{A}^-$ PMDoS attack: As shown in Figures 3.18(a) and 3.19(a), while both the periodic and stochastic $\mathbb{A}^+$ PMDoS attacks with $\gamma = 0.5$ have already induced throughput degradation of flow aggregates by more than 80%, the $\mathbb{A}^-$ PMDoS attack with the same cost can only reduce the aggregates' throughput by not more than 50%. In order for the $\mathbb{A}^-$ PMDoS attack to achieve the similar level of degradation,



(a) $R_{attack} = 25$ Mbps (b) $R_{attack} = 50$ Mbps

Fig. 3.18. Normalized throughput degradation under periodic $\mathbb{A}^+$ and $\mathbb{A}^-$ PMDoS attacks versus attack cost.



(a) $R_{attack} = 25$ Mbps (b) $R_{attack} = 50$ Mbps

Fig. 3.19. Normalized throughput degradation under stochastic $\mathbb{A}^+$ and $\mathbb{A}^-$ PMDoS attacks versus attack cost.

the attacker has to increase its γ to 0.9, which also increases the chance of exposure to the detection mechanisms.

Besides, we notice that the choice of the attack pulse width T_{on} could affect the performance of the periodic $\mathbb{A}^+$ PMDoS attacks more significantly than the stochastic ones. Figures 3.18(a) and 3.18(b) show that when the attack pulse width of the periodic $\mathbb{A}^+$ PMDoS attack ($T_{on} = 75$ms) is smaller than the RTT of the flow aggregates, i.e., 150ms, the normalized aggregate TCP throughput degradation may experience

an unexpected decline even when the attack cost increases, for example, $\gamma = 0.6$ in Figure 3.18(a) and $\gamma = \{0.6, 0.8\}$ in Figure 3.18(b). For those attack scenarios, since T_{on} does not cover the RTT of the flow aggregates, a portion of the flow aggregates could possibly be affected by the attack traffic periodically. Those survived TCP flows can therefore utilize more available bandwidth, resulting in a smaller aggregate TCP throughput degradation. On the contrary, the randomized attack period of the stochastic $\mathbb{A}^+$ PMDoS attack enables its attack pulses to throttle different legitimate TCP flows. As a result, we observe from Figures 3.19(a) and 3.19(b) that when the attack operates with $E[T_{on}] = 75$ms, the throughput degradation generally increases with the attack cost.

3.6 Optimizing the PDoS attacks

Based on the above analysis, we can learn that there are three main differences between PDoS attacks and the traditional flooding-based DoS. First, by adjusting the attack parameters, PDoS attack can cause different levels of damage, ranging from degradation-of-service to absolute denial-of-service. Second, since the average traffic attack rate of a PDoS attack is much smaller than the flooding-based attacks, it can evade the detection methods designed for flooding-based attacks [158]. Third, the number of attack packets required by a PDoS attack is so small that the attacker can customize them with correct values in order to elude the feature-based detection methods [159–161].

Since an attacker can tune the parameters to achieve different objectives, he can choose to inflict a certain level of damage to the victim TCP connections and yet to evade attack detection mechanisms in place. The smartness of the PDoS attacks therefore lies on the intelligent choices of the attack parameters which is the primary focus of this section. Here, we consider only the periodic AIMD-based attack, because it offers more flexibility to control the attack intensity. We will investigate the optimization problems for other kinds of PDoS attacks in the future work.

First of all, since the primary objective of a PDoS attack is to cause throughput degradation, we use $\Gamma \in (0,1)$, shown in Eq. (3.27), to measure the throughput degradation in the midst of an attack, normalized by the throughput without the attack.

$$\Gamma = 1 - \frac{\Psi_{attack}}{\Psi_{normal}}, \tag{3.27}$$

where $0 < \Psi_{attack} < \Psi_{normal}$. When the attack is severe enough, Γ approaches to 1 as Ψ_{attack} approaches to 0.

Moreover, the PDoS attacker may want to evade those detection schemes that will raise an alert whenever the measured average traffic rate exceeds a pre-defined threshold. For the purpose of modeling the attacker's preference in this aspect, we use an average attack rate normalized by R_{bottle}:

$$\gamma = \frac{R_{attack} T_{extent}}{R_{bottle}(T_{extent} + T_{space})}. \tag{3.28}$$

In the analysis, we consider $\gamma \in (0,1)$, because for $\gamma \geq 1$ the PDoS attack becomes the traditional flooding-based attack which does not attempt to evade attack detection.

Since DoS attacks can be detected based on the drastic increase in the traffic rate, we use $(1-\gamma)^{\kappa},\ \kappa > 0$, to measure an attacker's risk preference. When $\kappa > 1$, the attacker can be considered *risk-averse*. That is, the attacker becomes less willing to take the risk of being exposed as the attack rate increases. When $0 < \kappa < 1$, the attacker is considered *risk-loving*, which means that the attacker is more eager to cause more damage than to the concealment of the attack as the attack rate increases. To be complete, the attacker is considered *risk-neutral* when $\kappa = 1$.

Figure 3.20 depicts $(1-\gamma)^{\kappa}$ as a function of γ for the three cases. The rate of the increase in the slopes of the curves differentiates the three types of an attacker's behavior in terms of his risk preference. Furthermore, there are two interesting limiting cases. First, $\lim_{\kappa \to 0}(1-\gamma)^{\kappa} = 1$. In this limiting case, the attacker is entirely unconcerned about the risk of being detected; the traditional flooding-based attack is a good example in this category. Second, in the case of $\lim_{\kappa \to \infty}(1-\gamma)^{\kappa} = 0$, the

attacker's decision is totally dominated by the risk of being detected to the extent that he could not launch an attack.

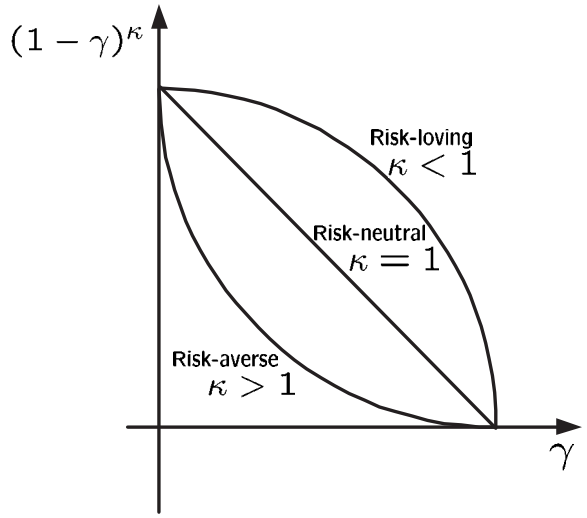


Fig. 3.20. Three different behaviors of a PDoS attacker modeled by $(1-\gamma)^{\kappa}$.

Now we combine the metric for characterizing the damage of a PDoS attack and that for the attacker's risk preference into a single metric in Eq. (3.29). Therefore, for any given finite value of κ, an attacker can optimize the attack by maximizing G_{attack}.

$$G_{attack} = \Gamma(1-\gamma)^{\kappa} = \left(1 - \frac{\Psi_{attack}}{\Psi_{normal}}\right)\left(1 - \frac{R_{attack}T_{extent}}{R_{bottle}(T_{extent}+T_{space})}\right)^{\kappa}. \quad (3.29)$$

3.6.1 A PDoS attack optimization problem

In the following we formulate the PDoS attack problem as a nonlinear optimization problem in which the only constraint is to ensure that $0 < \gamma < 1$ for the reasons discussed earlier.

$$\begin{cases} maximize & G_{attack} \\ subject\ to & 0 < \gamma < 1. \end{cases} \tag{3.30}$$

Let $T_{space} = \mu T_{extent}$, $\mu > 0$, which is the reciprocal of PDoS's duty cycle and $C_{attack} = \frac{R_{attack}}{R_{bottle}}$, which denotes the ratio of each pulse's sending rate to the bottleneck bandwidth. Then we can rewrite γ as

$$\gamma = \frac{R_{attack} T_{extent}}{R_{bottle}(T_{extent} + T_{space})} = \frac{C_{attack}}{1 + \mu}. \tag{3.31}$$

In the next two lemmas we present the analytical expressions for Ψ_{attack} and Ψ_{normal}, from which we can obtain a computable expression for Γ. Moreover, we can modify the optimization problem into a new form.

Lemma 3.6.1 *Assume that the TCP flows will make full use of the bottleneck bandwidth when there is no PDoS attack. Then, Ψ_{normal} is given by*

$$\Psi_{normal} = \eta R_{bottle}(N-1)T_{attack}/8, \quad \eta \in (0,1), R_{bottle} > 0 \tag{3.32}$$

Proof According to the TCP congestion control mechanism, a TCP sender will increase its *cwnd* every RTT until it experiences a packet loss. In other words, the throughput of the TCP flows will increase whenever there is additional bandwidth to transfer packets. Finally, their aggregated throughput will be approximately equal to the capacity of network ($\eta \to 1$) if the router's buffer is appropriately configured [85], because the incoming packets will keep the router busy all the time. Moreover, the attacker can estimate η by measuring the available bandwidth [162]. ■

Lemma 3.6.2 *The aggregated throughput of N_{flow} TCP connections under a PDoS attack $\mathbb{A}(T_{extent}, R_{attack}, T_{space}, N)$ can be approximated by*

$$\Psi_{attack} = \frac{a(1+b)T_{attack}^2 S_{packet}}{2d(1-b)}(N-1)\sum_{i=1}^{N_{flow}} \frac{1}{RTT_i^2} \tag{3.33}$$

Proof We have shown in the last chapter that the *cwnd* of a typical TCP flow (AIMD(1, 0.5)) can be brought to the converged value by using fewer than ten attack

pulses. Therefore, the period of the transient state will be very short. To simplify the following analysis, we use W_C in Eq. (3.13) to approximate the *cwnd* in the transient state. By summing up the throughput of all victim TCP flows during the $(N-1)$ free-of-attack intervals, we obtain Eq. (3.33). ■

Proposition 3.6.1 *Under a PDoS attack, the normalized throughput degradation* Γ *is given by*

$$\Gamma = 1 - \frac{\Psi_{attack}}{\Psi_{normal}} = 1 - \frac{C_\Psi}{\gamma}, \tag{3.34}$$

where

$$C_\Psi = \frac{4a(1+b)T_{extent}S_{packet}C_{attack}}{\eta(1-b)dR_{bottle}} \sum_{i=1}^{N_{flow}} \frac{1}{RTT_i^2}. \tag{3.35}$$

Proof By substituting Eq. (3.32) and Eq. (3.33) into Eq. (3.27) and some algebraic simplification. ■

Note that since $\Gamma \in (0,1)$, we have $0 < \frac{C_\Psi}{\gamma} < 1$; therefore, $C_\Psi < \gamma$. Moreover, since $\gamma \in (0,1)$, we have $0 < C_\Psi < 1$. Thus, the optimization problem in Eq. (3.30) becomes

$$\begin{cases} maximize & \left(1 - \frac{C_\Psi}{\gamma}\right)(1-\gamma)^\kappa \\ subject\ to & C_\Psi < \gamma < 1 \\ & 0 < C_\Psi < 1 \end{cases} \tag{3.36}$$

3.6.2 Optimized PDoS attack parameters

In this section we first solve the optimization problem in Eq. (3.36). From there we will immediately obtain three corollaries regarding the optimal values of γ for the three types of PDoS attackers. After that, we present the optimal value of μ, which enables the attacker to decide the length of the attack period.

Proposition 3.6.2 *The solution to the optimization problem stated in Eq. (3.36) is given by*

$$\gamma_-^* = \frac{C_\Psi(1-\kappa) - \sqrt{C_\Psi^2(1-\kappa)^2 + 4\kappa C_\Psi}}{-2\kappa}. \tag{3.37}$$

Proof We first obtain two roots to $\frac{\partial G_{attack}}{\partial \gamma} = 0$:

$$\gamma_{\pm}^{*} = \frac{C_{\Psi}(1-\kappa) \pm \sqrt{C_{\Psi}^{2}(1-\kappa)^{2} + 4\kappa C_{\Psi}}}{-2\kappa}. \tag{3.38}$$

It is easy to see that γ_{+}^{*} is not a feasible solution, because its value is less than zero. On the other hand, γ_{-}^{*} is a feasible solution by proving the following three results.

- $\gamma_{-}^{*} > 0$: This can be proved by observing that $\sqrt{C_{\Psi}^{2}(1-\kappa)^{2} + 4\kappa C_{\Psi}} > C_{\Psi}(1-\kappa)$ and putting this into γ_{-}^{*}.

- $\gamma_{-}^{*} < 1$: We prove this by contradiction. Assume that $\gamma_{-}^{*} \geq 1$. Then we have $\sqrt{C_{\Psi}^{2}(1-\kappa)^{2} + 4\kappa C_{\Psi}} - C_{\Psi}(1-\kappa) \geq 2\kappa$. After some simplification, the inequality is reduced to $C_{\Psi} \geq \kappa + C_{\Psi}(1-\kappa) \Rightarrow C_{\Psi} \geq 1$, which contradicts the first constraint in Eq. (3.36).

- $\gamma_{-}^{*} > C_{\Psi}$: We prove this also by contradiction by assuming that $\gamma_{-}^{*} \leq C_{\Psi}$. Thus, we get
  $\sqrt{C_{\Psi}^{2}(1-\kappa)^{2} + 4\kappa C_{\Psi}} - C_{\Psi}(1-\kappa) \leq 2\kappa C_{\Psi} \Rightarrow 1 \leq C_{\Psi}$, which also contradicts the first constraint in Eq. (3.36).

Now we can prove that γ_{-}^{*} is the only solution to Eq. (3.36) by observing that $\frac{\partial G_{attack}}{\partial \gamma}$ is a continuous function and

$$Sign\left(\frac{dG_{attack}}{d\gamma}\right) = \begin{cases} > 0 & \text{if} \quad \gamma \in (C_{\Psi}, \gamma_{-}^{*}) \\ = 0 & \text{if} \quad \gamma = \gamma_{-}^{*} \\ < 0 & \text{if} \quad \gamma \in (\gamma_{-}^{*}, 1). \end{cases} \tag{3.39}$$

■

Corollary 3.6.1 *For a risk-averse attacker ($\kappa > 1$), the optimal attack parameter is given by $\gamma_{-}^{*} = C_{\Psi}$ as κ goes to infinity, i.e., $\lim_{\kappa \to \infty} \gamma_{-}^{*} = C_{\Psi}$.*

Proof According to the L'Hospital's rule, $\lim_{\kappa \to \infty} \gamma_{-}^{*} = \lim_{\kappa \to \infty} \frac{\partial (C_{\Psi}(1-\kappa) - \sqrt{C_{\Psi}^{2}(1-\kappa)^{2} + 4\kappa C_{\Psi}})/\partial \kappa}{\partial(-2\kappa)/\partial \kappa} = C_{\Psi}$. ■

Corollary 3.6.2 *For a risk-loving attacker ($\kappa < 1$), the optimal attack parameter is given by $\gamma_-^* = 1$ as κ goes to 0, i.e., $\lim_{\kappa \to 0} \gamma_-^* = 1$.*

Proof According to the L'Hospital's rule, $\lim_{\kappa \to 0} \gamma_-^* = \lim_{\kappa \to 0} \frac{\partial(C_\Psi(1-\kappa) - \sqrt{C_\Psi^2(1-\kappa)^2 + 4\kappa C_\Psi})/\partial\kappa}{\partial(-2\kappa)/\partial\kappa} = 1$. ■

Corollary 3.6.3 *For a risk-neutral attacker ($\kappa = 1$), the optimal attack parameter is given by $\gamma_-^* = \sqrt{C_\Psi}$.*

Proof Substituting $\kappa = 1$ into Eq. (3.37). ■

According to modeling of the PDoS attack's impact, if each attack pulse could cause packets losses in different TCP flows, then the remaining TCP throughput is mainly determined by the attack period $T_{attack} = (1+\mu)T_{extent}$, as shown in Eq. (3.16). In other words, when R_{attack} and T_{extent} are given, we can determine the optimal value of μ that achieves the tradeoff between the damage and the risk preference.

Proposition 3.6.3 *The optimal $\mu_{optimal}$ is given by*

$$\mu_{optimal} = \frac{-2\kappa C_{attack}}{C_\Psi(1-\kappa) - \sqrt{C_\Psi^2(1-\kappa)^2 + 4\kappa C_\Psi}}. \tag{3.40}$$

Proof By substituting Eq. (3.37) into Eq. (3.31). ■

Corollary 3.6.4 *For a risk-neutral attacker,*

$$\mu_{optimal} = \sqrt{\frac{C_{attack}}{T_{extent} C_{victim}}} - 1, \tag{3.41}$$

where

$$C_{victim} = \sqrt{\frac{4a(1+b)S_{packet}}{(1-b)dR_{bottle}} \sum_{i=1}^{N_{flow}} \frac{1}{RTT_i^2}}. \tag{3.42}$$

3.6.3 Performance evaluation

Simulation experiments and results

The network topology, shown in Figure 3.11, consists of M pairs of TCP senders and receivers. All links, except the bottleneck between router S and R, are 50 Mbps. The two routers are connected through a link of 15 Mbps with RED. The TCP connections are based on TCP New Reno [163] and their RTTs range from 20ms to 460ms. We use the simulation scripts provided by [28].

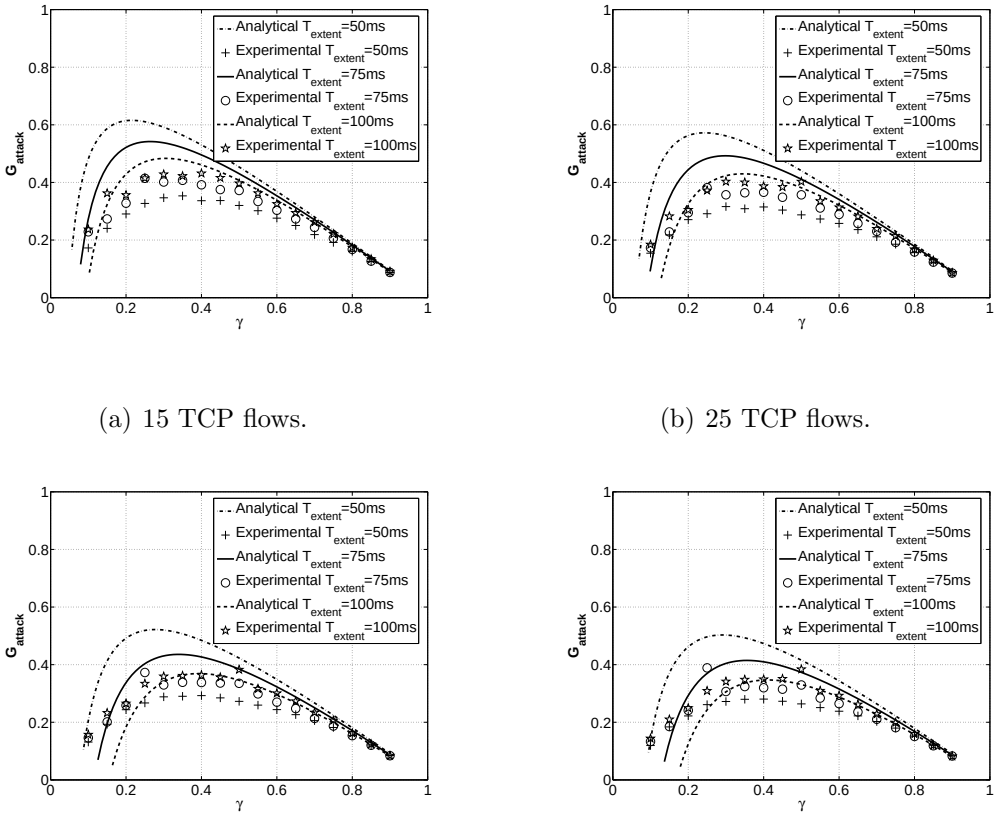
Figures 3.21, 3.22, 3.23 and 3.24 show the results for different number of TCP flows (15, 25, 35, and 45) under PDoS attacks with $R_{attack} = 25, 30, 35$ and 40 Mbps. The analytical results are presented by lines in the figures, while the simulation results are represented by symbols.

Normal-gain, under-gain, and over-gain attacks

Figures 3.21-3.24 show that all analytical results correctly predict the trends of the attack gains. However, the values may not be in exact match, because of the complex interplay between the attack pulses and the queue management mechanisms. For example, under certain attacks, the TCP sender may suffer from more throughput degradation when it enters the TO state instead of the FR state. Therefore, we classify the attacks into three categories according to the discrepancies between the experimental and analytical results.

The *normal-gain* attacks refer to those cases in which the simulation and analytical results are in close agreement, such as the case of $R_{attack} = 25$ Mbps, $T_{extent} = 100$ms in Figure 3.21 and the case of $R_{attack} = 35$ Mbps, $T_{extent} = 75$ms in Figure 3.23. The PDoS attack with such parameter settings can effectively constrain the TCP throughput to a low value by causing them to frequently enter the FR state.

The *under-gain* attacks refer to those cases in which the analytical results overestimate the actual attack gains, such as the cases when $T_{extent} = 50$ms in Figures 3.21-3.23. The cause for the discrepancy is due to the fact the attack rate is not high enough to affect all TCP flows. Moreover, we can observe that the longer the dura-



(a) 15 TCP flows.

(b) 25 TCP flows.

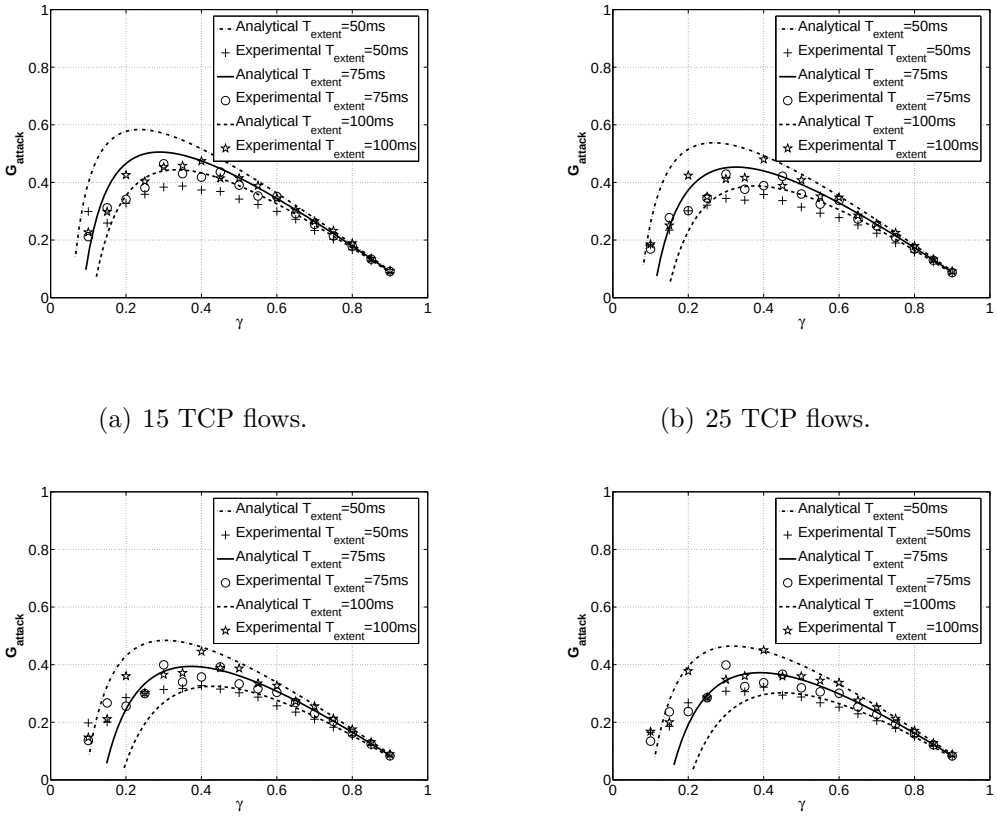
(c) 35 TCP flows.

(d) 45 TCP flows.

Fig. 3.21. Analytical and experimental results under PDoS attacks with $R_{attack} = 25$ Mbps.

tion of each attack pulse is, the more severe the PDoS attack inflicts on the normal TCP flows. This is because more legitimate packets will be dropped under such attacks when the attack packets occupy more buffer and/or use more computational resources from the router.

The *over-gain* attacks refer to those cases in which the analytical results underestimate the actual attack gains, such as the case of $R_{attack} = 40$ Mbps, $T_{extent} = 100$ms in Figure 3.24. This is because such attacks can force more TCP flows to enter the



(a) 15 TCP flows.

(b) 25 TCP flows.

(c) 35 TCP flows.

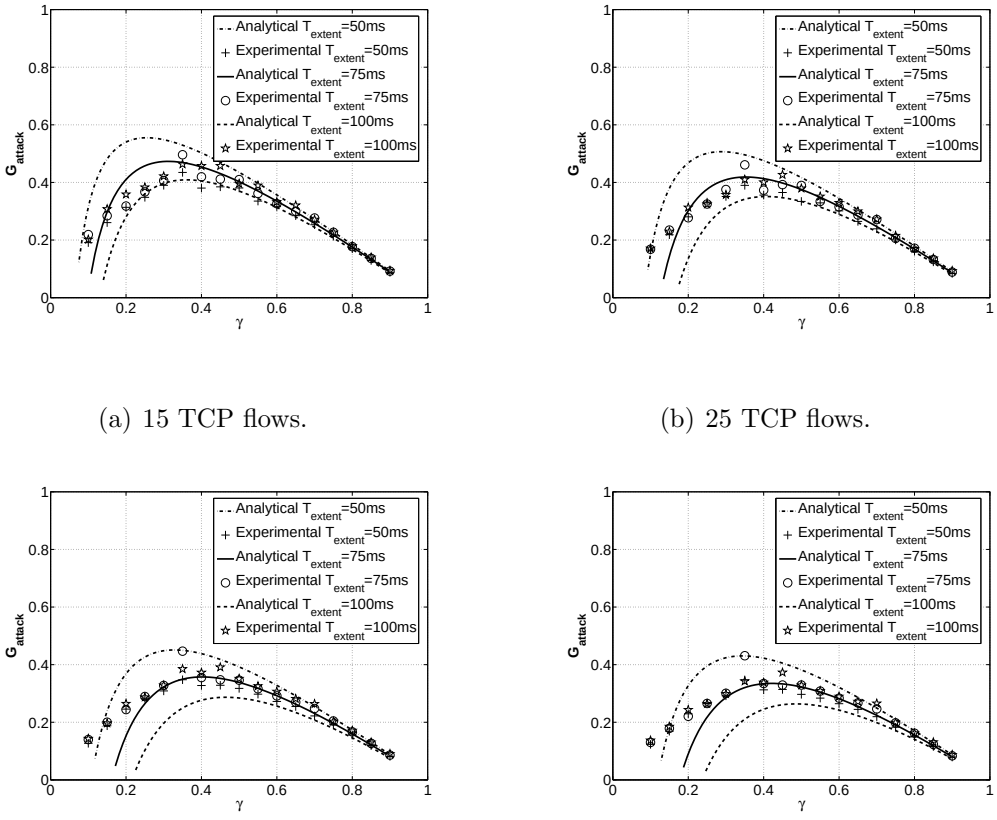
(d) 45 TCP flows.

Fig. 3.22. Analytical and experimental results under PDoS attacks with $R_{attack} = 30$ Mbps.

TO state instead of the FR state due to its high attack rate. Therefore, the analytical results consistently underestimate the extent of the throughput reduction.

Maximization points

Figures 3.21-3.24 show that for the normal-gain and over-gain attacks, most of the experimental results generally match very well with the analytical results in the maximization points. The exceptions are due to the shrew attacks that will be discussed in the following subsection. However, for the under-gain attacks, they do not match



(a) 15 TCP flows.

(b) 25 TCP flows.

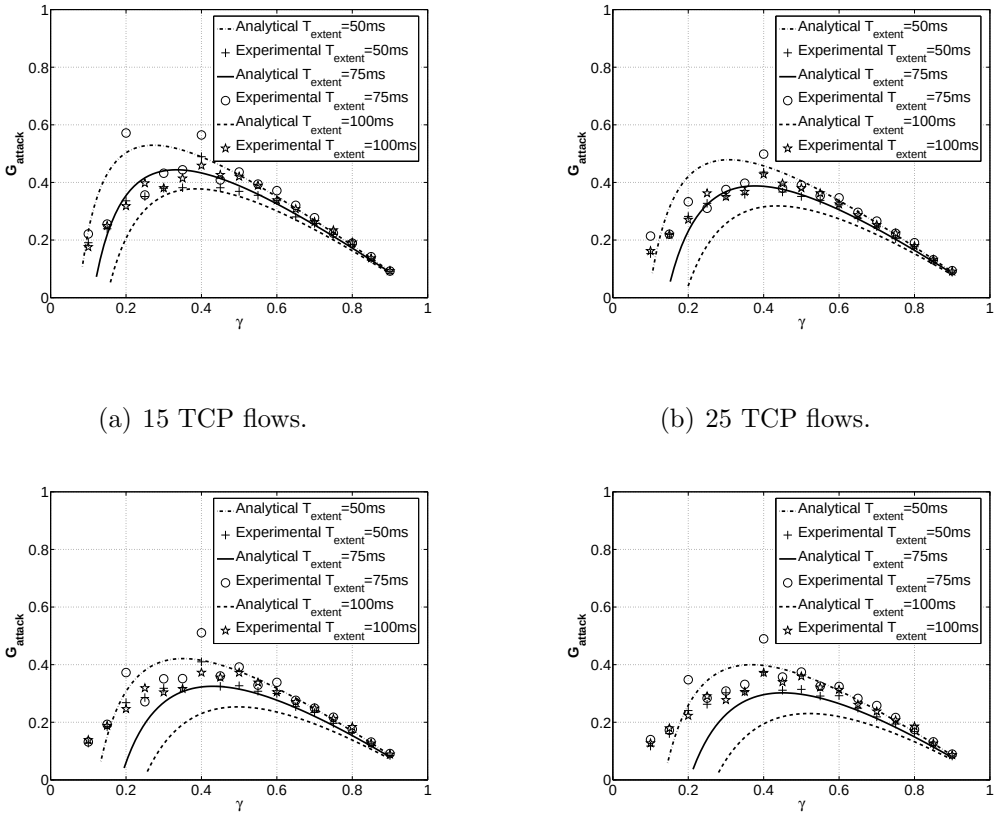
(c) 35 TCP flows.

(d) 45 TCP flows.

Fig. 3.23. Analytical and experimental results under PDoS attacks with $R_{attack} = 35$ Mbps.

as well, because the number of attack packets is too small to block the bottleneck and therefore not all the legitimate TCP flows are affected by the attack.

The figures also show that the experimental results located on the right-hand side of the maximization points are closer to the analytical results than those on the left-hand side of the maximization points. This is because when γ increases, more attack packets will be sent in each pulse, which will take up more resources in the bottleneck. Therefore, more legitimate TCP flows will be affected by the attack and consequently their throughput will be decreased as predicted from the analysis.



(a) 15 TCP flows.

(b) 25 TCP flows.

(c) 35 TCP flows.

(d) 45 TCP flows.

Fig. 3.24. Analytical and experimental results under PDoS attacks with $R_{attack} = 40$ Mbps.

Shrew attacks

According to the analysis in the last chapter, the AIMD-based and timeout-based attacks share similar attack scheme, but they exploit different aspects of the TCP congestion control mechanism. There are in fact some attack cases that correspond to the shrew attacks. That is, if a PDoS attack's T_{attack} is approximately equal to $\frac{minRTO}{n}$, $n \in [1, minRTO]$, then the attack may constrain the TCP sender to the TO state, instead of the FR state assumed by the analytical model. As a result, the actual throughput degradation will be grossly underestimated by the analysis. Even

if some TCP flows may survive these timeout-based attacks because of their large RTTs [28], they will still suffer from the AIMD-based attack.

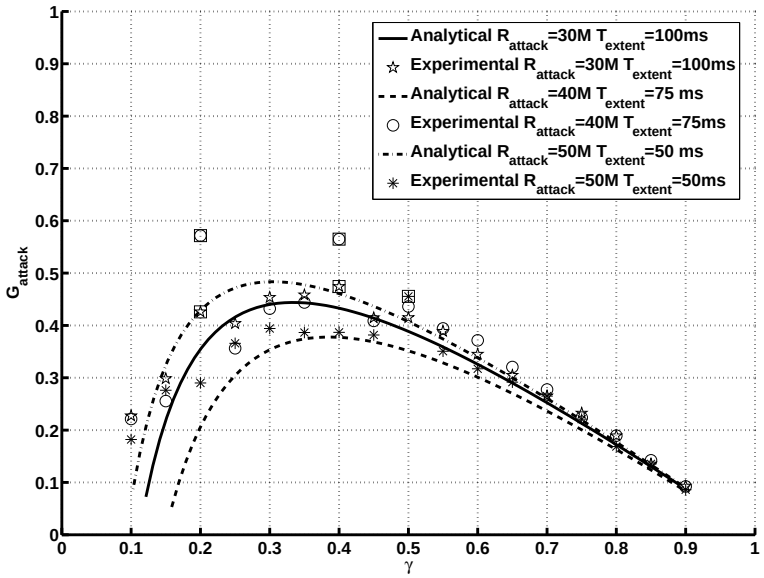


Fig. 3.25. Relationship between PDoS attacks and the shrew attacks.

We show some of the cases in Figure 3.25 and we use □ to mark them. For the normal-gain attack with $R_{attack} = 30$ Mbps and $T_{extent} = 100$ms, the shrew-attack points are $T_{attack} = 500ms, 1000$ms in which the attack gains are much higher than what are anticipated by the analysis. For the over-gain attack with $R_{attack} = 40$ Mbps and $T_{extent} = 75$ms, all points except the two shrew-attack points match well the trend given by the analysis. For the under-gain attack with $R_{attack} = 50$ Mbps and $T_{extent} = 50$ms, once again the shrew-attack point of $T_{attack} = 1000/3$ms gives a higher attack gain than the analytical result.

Test-bed experiments and results

We use Dummynet [164] to simulate the network by setting the bottleneck to 10 Mbps and introducing 150ms delay. We use `iperf` [157] to generate legitimate TCP flows. In the absence of attacks, the victim TCP flows will occupy all the bandwidth. The link between Dummynet and the victim is 10 Mbps, whereas the links connecting the legitimate users and the attacker to the Dummynet are 100 Mbps. In this setting, the legitimate user is running Linux Fedora with kernel 2 v2.6.5-1.358, whose minRTO is 200. We have conducted experiments under RED. The buffer size is set according to the rule-of-thumb $B = RTT \times R_{bottle}$ [85], and the RED parameters are configured as: $min_{th} = 0.2B$, $max_{th} = 0.8B$, $w_q = 0.002$, $max_p = 0.1$, and $gentle\_ = true$.

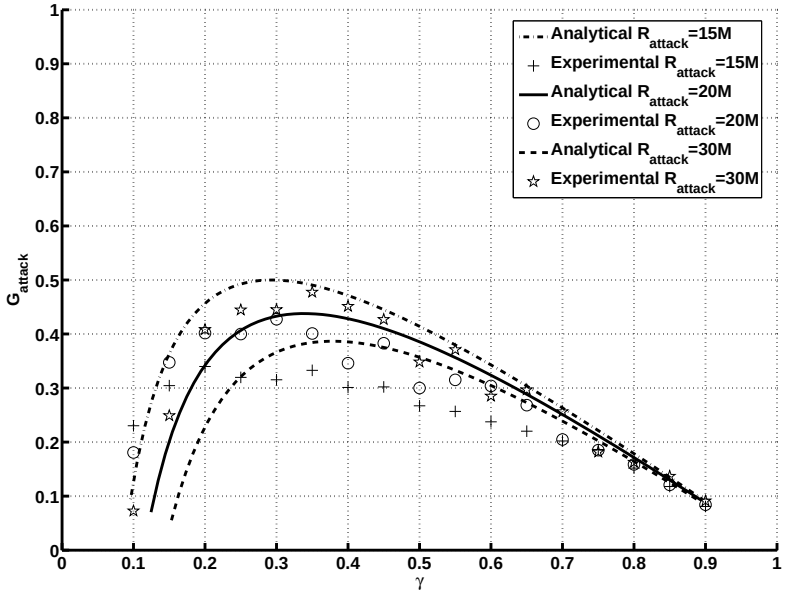


Fig. 3.26. Experiment results obtained in the test bed.

There are a total of ten victim TCP flows under three kinds of PDoS attacks, which have the same T_{extent} = 150ms but different R_{attack} values. The results are shown in Figure 3.26, where all of them match the trends of the analytical results.

Moreover, the normal-gain attack is observed when R_{attack} is equal to 20 Mbps. The analytical results usually underestimate the attack gains when R_{attack} is increased to 30 Mbps. On the other hand, the analytical results usually overestimate the attack gains when R_{attack} is decreased to 15 Mbps.

3.7 Summary

In this chapter, we have proposed a new class of PDoS attacks. Unlike the traditional DoS attack, the PDoS attack can effectively achieve the same damage with a much lower attack rate. We have presented two specific attack methods: timeout-based and AIMD-based, and their variants. We have further proposed polymorphic DoS (PMDoS) attacks which may result in various traffic patterns according to different parameter settings and consequently different impact on victim TCP flows. Our analysis has confirmed that the PDoS attacks can effectively force the affected TCP senders to continuously re-enter the FR state or the TO state.

Moreover, we have investigated the impact of the FDDoS and PDoS attacks on TCP throughput under different queue management schemes, including DropTail and the four AQM schemes. There are several important results obtained from the analytical and simulation results. First, under a PDoS attack, the RED-like AQMs suffer from a higher throughput degradation than the DropTail and AVQ do, because the latter discards the incoming packets only when the (virtual) queue is full. Second, the packet dropping rates under the queue management schemes behave quite differently for the FDDoS and PDoS attacks. During a PDoS attack, the packet dropping rates for the attack packets are almost the same, whereas they are different for legitimate TCP packets. In particular, both DropTail and AVQ tend to drop fewer legitimate TCP packets but more attack packets as compared with the RED-like AQMs. However, the results are opposite for a FDDoS attack. Third, the PDoS attack is indeed more effective than the traditional FDDoS attack, because the former has a much higher attack power and a smaller attack cost.

When evaluating the power of PDoS attacks in both ns-2 and test bed, we observed some differences between these two sets of experiment results. Although we observe significant throughput degradation under PDoS attacks in both ns-2 and test bed, the damage measured in ns-2 is severer than that observed in the test bed. The possible reasons include:

1. The TCP implementation in ns-2 is more vulnerable to the PDoS attacks than those in real Linux kernel because the former was originally based on source code of old BSD kernel while the latter has added many new features during the past decade [165]. A recent paper has mentioned the deviations between the TCP modules in NS-2 and those from the mainstream operating systems such as FreeBSD and Linux [165]. Moreover, they have done an excellent work by immigrating the TCP implementations in Linux to ns-2. Hence, we will re-conduct those ns-2 experiments in our future work to investigate the difference.

2. It is much easier to launch PDoS attacks in ns-2 than in real network because we can easily control the sending time of each attack packet in ns-2. In real network, we have tried several approaches to generate high-intensity attack pulses, for example, WinPcap [166] and Linux's raw socket [167]. We found that when using WinPcap and a large packet size an attacker can produce high-intensity attack pulses.

Furthermore, we have investigated how to optimize PDoS attacks. In particular, we have formulated the attack objective based on maximizing TCP throughput degradation and minimizing the risk of being detected. As far as we know, this is the first time to study such a tradeoff using an analytical framework. By adjusting the parameters in the attack objective, we can analyze the resulted attacks for different types of attackers who may be risk-averse, risk-loving, or risk-neutral. As a result, we have obtained the optimized attack parameters for a given attacker's behavior. Moreover, we have validated the analytical results using both ns-2 simulation and a test bed. The study of optimized pulsing attacks is the first step to adopting game

theory for analyzing a PDoS attacker's behavior. In practice, the variation of RTTs and packet dropping in the routers before the bottleneck router will affect the effect of optimized pulsing attacks. Therefore, one research issue in the future work is to explore how to launch more effective PDoS attacks on certain TCP flows, e.g. BGP connections [54], when considering not only attack power and attack cost but also the location of bottleneck and the effect of routers on the path.

4. DETECTING PULSING DENIAL-OF-SERVICE ATTACKS

In this chapter, we discuss how to detect various PDoS attacks. We propose a two-stage detection scheme that is based on two kinds of traffic anomalies caused by the PDoS attacks. To enhance its capability for detecting PMDoS attacks, we propose Vanguard, a new detection system. The extensive experiment results obtained in both ns-2 simulation environment and our test bed demonstrate that these two detection mechanisms can effectively and efficiently discover PDoS attacks and PMDoS attacks respectively.

4.1 A two-stage detection algorithm

In this section, we propose a novel two-stage scheme for detecting PDoS attacks. Although attackers can use UDP packets, arbitrary IP packets or ICMP packets to launch attacks, they prefer TCP packets as attack packets. The reason is that routers dispatch different types of packets into separate service queues, so that different types of flows could not affect one another. Hence, we only consider attack packets of TCP type. Though an attack or a selfish user may use non-compliant TCP flows to occupy more bandwidth and consequently affect other TCP flows' throughput, it is difficult to launch an effective PDoS attacks because its attack period is fixed to its RTT value. Moreover, such non-compliant feature can be easily detected by existing approaches [168]. If an attacker or a selfish user uses non-compliant TCP flows that send many packets in each burst similar to a PDoS attacker does, our detection schemes can also identify it. However, in this case, we cannot know whether the flow is controlled by an attack or a selfish user. In order to evade such meaningless

discussion, we do not consider the case that the attacker use non-compliant TCP flows to launch PDoS attacks.

Since a successful PDoS attack does not require a sustained high attack packet rate, the feasible location for detecting such an attack is at the victim network. The patterns of both incoming traffic and outgoing traffic are then under surveillance. Moreover, since the PDoS attack can be effectively launched even by a single source, our detection system is based on the detection of traffic pattern anomalies. We have discovered two anomalies incurred by a PDoS attack. The first is that the incoming data traffic will fluctuate in a more extreme manner during an attack, as depicted in Figure 4.1. In the absence of any attack, a *global synchronization* may occur when multiple TCP flows share the same bottleneck link and experience packet loss almost simultaneously [85, 169]. Similar synchronization phenomenon can also be caused by a PDoS attack which induces packet losses in the victim TCP connections simultaneously. Although this *quasi-global synchronization* phenomenon resembles that under the nonattack situation, their periods are different: the former one is dictated by the attack parameters, whereas the latter one by the bottleneck capacity.

The peaks of the incoming traffic, which consist of attack pulses and legitimate TCP packets, are usually of high-rate and short-duration [4, 28, 87], whose length is determined by T_{extent}. The valleys of the traffic rate are due to the congestion control algorithm of the affected TCP flows. Whenever an attack pulse arrives at the router, its instantaneous high volume traffic will fill the queue and induce packet drops. Depending on the volume of attack packets and the duration of congestion period, some TCP flows may timeout while others may enter the FR state. Of course, some TCP flows may survive the attack without experiencing any packet loss. Therefore, there are still some TCP traffic between two consecutive attack pulses. These fluctuations have a severe impact on the TCP performance (e.g., decrease in throughput and increase in jitter).

To visualize this quasi-global synchronization phenomenon, we have conducted both ns-2 simulation [170] and test-bed experiments, and the results are shown in

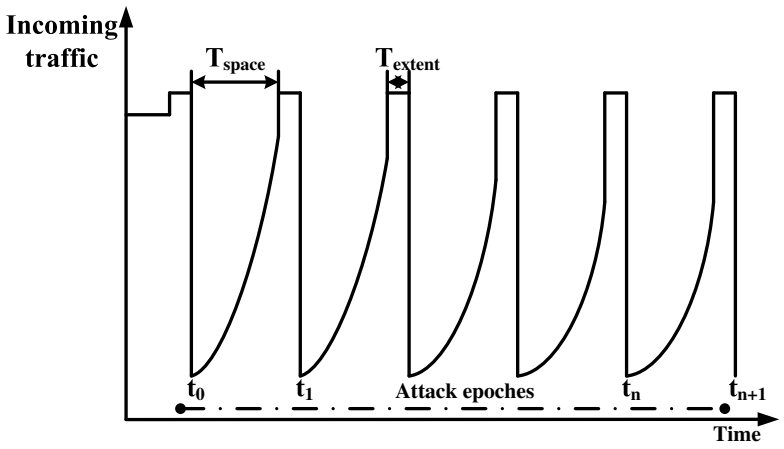
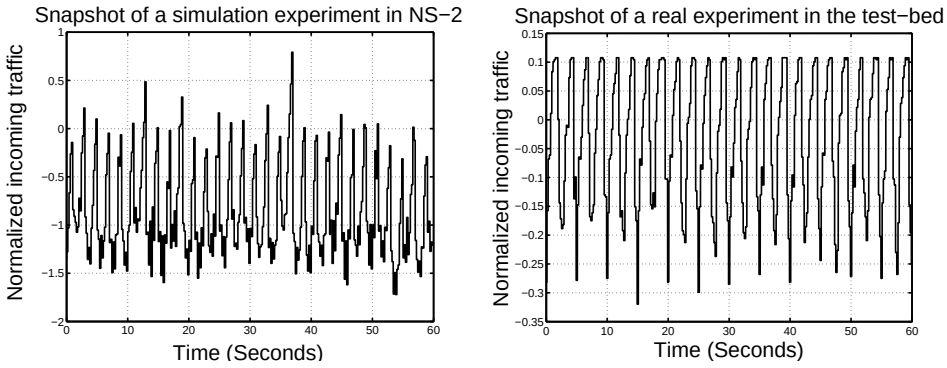


Fig. 4.1. Periodic pattern of the incoming data traffic during a PDoS attack.

Figure 4.2(a) and Figure 4.2(b), respectively. In order to display the results clearly, the incoming traffic has been first normalized so that the mean value is zero and then transformed into a piecewise aggregate approximation [171].

Figure 4.2(a) captures a one-minute snapshot of the incoming traffic in the ns-2 simulation, including the packets from 24 victims TCP flows and those belonging to a PDoS attack with T_{extent} = 50ms, T_{space} = 1950ms, R_{attack} = 100Mbps. We could observe not only the anticipated fluctuations but also its period. There are 30 pinnacles evenly distributed within a duration of 60 seconds, implying a periodic signal with a period of $60/30 = 2$ seconds which is in fact equal to the period of the PDoS attack.

Figure 4.2(b) displays a one-minute snapshot of the incoming traffic in the test-bed experiment, consisting of packets from 15 victims TCP flows and those belonging to a PDoS attack with T_{extent} = 100ms, T_{space} = 2400ms, and R_{attack} = 50 Mbps. Figure 4.2(b) exhibits the quasi-global synchronization phenomena in which 24 pinnacles are evenly spaced. The period of the incoming traffic is $60/24 = 2.5$ seconds which is also equal to the period of the attack pulses.



(a) The ns-2 simulation results (b) The test-bed experiment results

Fig. 4.2. Quasi-global synchronization phenomenon induced by a PDoS attack.

The second anomaly has to do with the outgoing TCP ACK traffic. As just mentioned, the incoming legitimate TCP traffic volume will decline because of the attack. However, the overall incoming TCP traffic volume may or may not decline during the attack, because the attack packets can also be TCP based. Hence, our detection system is also required to observe a possible decline in the outgoing TCP ACK traffic. It is important to note that both anomalies have to be used in order to avoid a high false-positive rate.

4.1.1 The first stage

Based on the discussion above, the first stage in the detection process is to monitor the variability in the incoming traffic and the outgoing TCP ACK traffic. Here we employ a discrete wavelet transform (DWT) for this purpose. The complexity of the pyramid algorithms that are used to compute the discrete wavelet transform is O(N), where N is the number of samples [172]. Any other simple scheme that is based on N samples also needs such computational complexity.

The DWT represents a signal $f(t) \in L^2(R)$ using scaling functions $\varphi_{j,k}(t)$, and a translated and dilated version of wavelet functions $\psi_{j,k}(t)$ [173]. Since the wavelet

functions operate like high-pass filters that use narrow time windows to compute differences in signals, they can capture the variability of the incoming traffic volumes. Similar approaches have been widely employed in the field of signal process to detect pulse-like signals, e.g. [174]. On the other hand, the scaling functions perform like low-pass filters; therefore, they can be used to extract the trend of the outgoing TCP ACK traffic. For the detection we do not argue that DWT is the best approach for computing the trend. However, since DWT can detect trends under different resolutions, besides detection a user may adjust the resolution level to obtain a full picture of the trend of the outgoing TCP ACK traffic for other monitoring purposes. In other words, DWT can provide more information than other simple approaches.

To realize an on-line detection, we use a moving window to group G contiguous samples to compute the DWT. Moreover, we define a statistic based on the signal energy to quantify the variability in the incoming traffic for the nth window of samples:

$$E_H(n) = \frac{1}{G} \sum_k |d_{1,k}^{In}|^2, \tag{4.1}$$

where $d_{1,k}^{In}$ is the wavelet coefficient at the finest scale ($j = 1$). Similarly, we define a statistic based on the signal energy to characterize the trend in the outgoing TCP ACK traffic for the nth window of samples:

$$E_L(n) = \frac{1}{G} \sum_k |c_{L,k}^{Out}|^2, \tag{4.2}$$

where $c_{L,k}^{Out}$ is the scaling coefficient at the highest decomposed scale ($j = L$). Further details about the DWT can be found in Appendix A.

4.1.2 The second stage

The second stage is to detect abrupt changes in $E_H(n)$ for the incoming traffic and in $E_L(n)$ for the ACK traffic. We employ a nonparametric CUSUM algorithm for this purpose. The details about the CUSUM algorithm can be found in Appendix B. The CUSUM method assumes that the mean value of the variable under surveillance

will change from negative to positive when a change occurs. Since both $E_H(n)$ and $E_L(n)$ are larger than zero, we first transform them into two random sequences:

$$Z_H(n) = E_H(n) - \beta_H \quad and \quad Z_L(n) = \beta_L - E_L(n),$$

where β_H and β_L are constants for determining the mean values of $Z_H(n)$ and $Z_L(n)$. Typically, we set β_H to the upper bound of $E_H(n)$ and β_L to $\overline{E_L(n)} - P_{tolerance} \times [\triangle(E_L(n))]$, where $\overline{E_L(n)}$ is the mean of $E_L(n)$ and $\triangle(E_L(n))$ is the standard deviation of $E_L(n)$. $P_{tolerance}$ controls the limit of the allowable decrease in $E_L(n)$. Therefore, $Z_H(n)$ and $Z_L(n)$ will have negative mean values under normal conditions.

Let T^{In} be the detection time for $Z_H(n)$ when $y_{Z_H}(n) > C^{In}_{cusum}$, where $y_{Z_H}(n)$ is the CUSUM value of $Z_H(n)$ and C^{In}_{cusum} is the corresponding threshold. Similarly, let T^{Out} be the detection time for $Z_L(n)$ when $y_{Z_L}(n) > C^{Out}_{cusum}$, where $y_{Z_L}(n)$ is the CUSUM value of $Z_L(n)$ and C^{Out}_{cusum} is the corresponding threshold. The detection system confirms the onset of a PDoS attack when both $y_{Z_H}(n) > C^{In}_{cusum}$ and $y_{Z_L}(n) > C^{Out}_{cusum}$ hold. Hence, the final detection time is determined by $T^{Final} = max\{T^{In}, T^{Out}\}$ and the detection delay is $T^{Final} - T_{Attack}$, where T_{Attack} is the start time of the PDoS attack.

4.2 Vanguard: detecting polymorphic DoS attacks

Although the two-stage detection scheme could efficiently detect the PDoS attacks, it may miss some PMDoS attacks that will induce only a few fluctuations in the traffic. To overcome this shortcoming, we propose Vanguard, which synthesizes three kinds of traffic anomalies induced by a PMDoS attack.

4.2.1 Anomalies induced by PMDoS attacks

Since the PMDoS attacks target at TCP flows, they will inevitably induce anomalies in the TCP traffic. In the following, we discuss three types of traffic anomalies that have been confirmed through analysis and extensive experiments.

Anomaly I - Decline in the outgoing TCP ACK traffic

Same as the two-stage detection system, the first anomaly used here is the drop in the number of TCP ACKs. However, this anomaly alone is not sufficient for the detection. On one hand, it may cause false alarms if the decline in the ACKs is a result of the decrease in the TCP data traffic. On the other hand, it may not be able to detect advanced attacks that manipulate normal TCP flows to trigger more ACK packets, for example, those using reordered TCP data packets to induce more ACK packets.

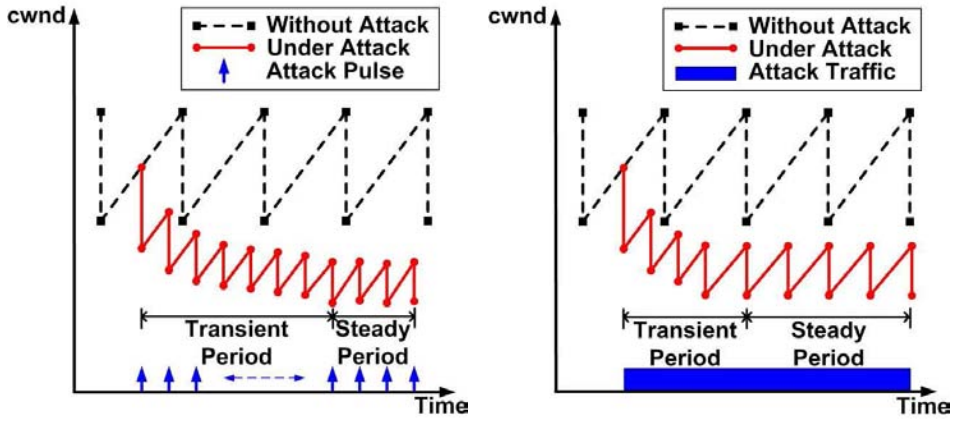
Anomaly II - Increase in the ratio of the incoming TCP traffic to the outgoing TCP ACK traffic

In order to eliminate the potential false alarms caused by anomaly I, we can monitor the ratio of the incoming TCP data traffic to the outgoing TCP ACK traffic, denoted by I_{Ratio}. When there is no packet loss, I_{Ratio} is typically equal to 2 [27], i.e., the ACK-every-other-segment mechanism. However, when an out-of-ordered packet (due to packet losses or packet reordering) arrives, I_{Ratio} is equal to 1, because a duplicate ACK is sent immediately. A similar technique has been used in D-WARD [175] to detect DDoS attack. However, D-WARD is installed in attackers' source networks, whereas Vanguard is located at the victim's network. Moreover, D-WARD cannot handle the attacks that make use of reordered TCP packets to trigger more ACK packets.

Anomaly III - Change in the incoming TCP data traffic distribution

Anomalies I and II can be utilized to discover most PMDoS attacks, except for more sophisticated ones that cleverly trigger more outgoing TCP ACK traffic to conceal the attack packets. To detect this advanced attack, we can exploit the third traffic anomaly: the change in incoming TCP traffic distribution. The basis of this

anomaly is that a PMDoS attack will perturb the distribution of the victim TCP data traffic. For example, as shown in Figure 4.3(a), the sender's *cwnd* will converge to a low value due to the periodic packet loss. Moreover, the *cwnd* could be constrained by a $\mathbb{A}^-$ PMDoS attack which behaves like a CBR flow. However in this case, the fluctuation of *cwnd* is modulated by the limited bandwidth instead of the attack.



(a) The decline of *cwnd* under an A^+ attack.

(b) The decline of *cwnd* under an A^- attack.

Fig. 4.3. Evolution of *cwnd* during PMDoS attacks.

4.2.2 Vanguard: a new detection scheme

In this subsection, we introduce Vanguard, a new scheme to detect PMDoS attacks by monitoring the three aforementioned anomalies. Vanguard first locates change points in statistics collected for detecting the three anomalies and then detect attacks based on a simple rule to be presented shortly.

In order to detect anomaly in the outgoing ACK traffic indicated in its trend, we apply the *moving average* algorithm in time series [176]. We use the following notations:

1. $D_{data}(i), i = 1, 2, \ldots$, is the ith sample of the incoming TCP data rate;

2. $D_{ACK}(i), i = 1, 2, \ldots$, is the ith sample of the outgoing ACK rate;

3. T_{sample} is the length of each sampling interval in seconds;

4. W_D is the length of detection window in seconds;

5. $N_{sample} = W_D/T_{sample}$ is the number of samples in a detection window.

We define the ratio of the incoming TCP data rate to the outgoing ACK rate as

$$I_{Ratio}(n) = \frac{I_{Data}(n)}{I_{ACK}(n)}, \tag{4.3}$$

where $I_{Data}(n)$ and $I_{ACK}(n)$ are given by

$$I_{Data}(n) = \frac{1}{W_D} \sum_{i=(n-1)N_{sample}+1}^{nN_{sample}} D_{Data}(i). \tag{4.4}$$

$$I_{ACK}(n) = \frac{1}{W_D} \sum_{i=(n-1)N_{sample}+1}^{nN_{sample}} D_{ACK}(i). \tag{4.5}$$

According to the analysis in section 4.2.1, I_{ACK} will decrease in the presence of unsophisticated PMDoS attack due to the decline of the victim incoming TCP traffic, whereas I_{Data} may or may not reduce depending on whether or not the attack packets can compensate for the dropped TCP data packets. I_{Ratio} will be larger than d if the attack packets cannot trigger more outgoing ACKs.

We employ the *color histogram indexing* method [177] to capture the change in the distribution of the incoming TCP data traffic. In the field of image retrieval, this method is a robust approach to computing similarity of two images [178]. Our basic idea is to measure the similarity index (SI) of the distribution of the incoming TCP traffic and that of the normal TCP traffic. Since the PMDoS attack will change the incoming TCP traffic distribution, there will be an abrupt change in the series of SI. The algorithm consists of three steps.

In the first step, we compute a traffic histogram for each detection window of samples. Given minimum (D_{data}^{min}) and maximum (D_{data}^{max}) values of the incoming TCP traffic samples, we divide the range $[D_{data}^{min}, D_{data}^{max}]$ into B disjoint subregions of equal

size, named as *histogram bins*. The traffic histogram $h(n)$ of the nth detection window is then obtained by counting the number of samples $h_{n,i}$ that fall in the histogram bin i, $1 \leq i \leq B$ (i.e., $h(n) = [h_{n,1}, \ldots, h_{n,B}]$). Second, a cumulative histogram $H(n) = [H_{n,1}, \ldots, H_{n,B}]$ is obtained by computing $H_{n,i} = \sum_{j \leq i} h_{n,j}$. Third, with the cumulative histogram of the normal TCP traffic $\widehat{H} = [\widehat{H}_1, \ldots, \widehat{H}_B]$ and $H(n) = [H_{n,1}, \ldots, H_{n,B}]$, define the SI of nth detection window as

$$I_{Dis}(n) = \sqrt{\sum_{j=1}^{B} (H_{n,j} - \widehat{H}_j)^2}. \tag{4.6}$$

We obtain I_{ACK}, I_{Ratio}, and I_{Dis} at the end of each detection window, and the system raises an alarm if the following statement is true:

$$I_{Ratio} \uparrow \vee \{I_{ACK} \downarrow \wedge I_{Dis} \uparrow\}, \tag{4.7}$$

where $\uparrow$ and $\downarrow$ represent abrupt increase and decrease, respectively. Accordingly, Vanguard first locates any abrupt change in the statistics collected for detecting the anomalies, and raises the alarm if I_{Ratio}, or both I_{ACK} and I_{Dis} are found to be anomalous.

The CUSUM, a nonparametric change-point detection algorithm [179], is used to capture abrupt changes in the sequences $\{I_{ACK}(n)\}$, $\{I_{Ratio}(n)\}$, and $\{I_{Dis}(n)\}$. This algorithm assumes that the mean of the variables being monitored will change from negative to positive. Since $I_{ACK}(n)$, $I_{Ratio}(n)$ and $I_{Dis}(n)$ are always nonnegative, we first transform them into three random sequences, $P_{ACK}(n)$, $P_{Ratio}(n)$ and $P_{Dis}(n)$, which have negative means under the normal period, as follows:

$$P_{ACK}(n) = \beta_{ACK} - I_{ACK}(n), \tag{4.8}$$

$$P_{Ratio}(n) = I_{Ratio}(n) - \beta_{Ratio}, \tag{4.9}$$

$$P_{Dis}(n) = I_{Dis}(n) - \beta_{Dis}, \tag{4.10}$$

where β_{ACK}, β_{Ratio}, and β_{Dis} are constants for determining the mean values of $\{I_{ACK}(n)\}$, $\{I_{Ratio}(n)\}$ and $\{I_{Dis}(n)\}$, respectively. Normally, we can set β_{ACK} to

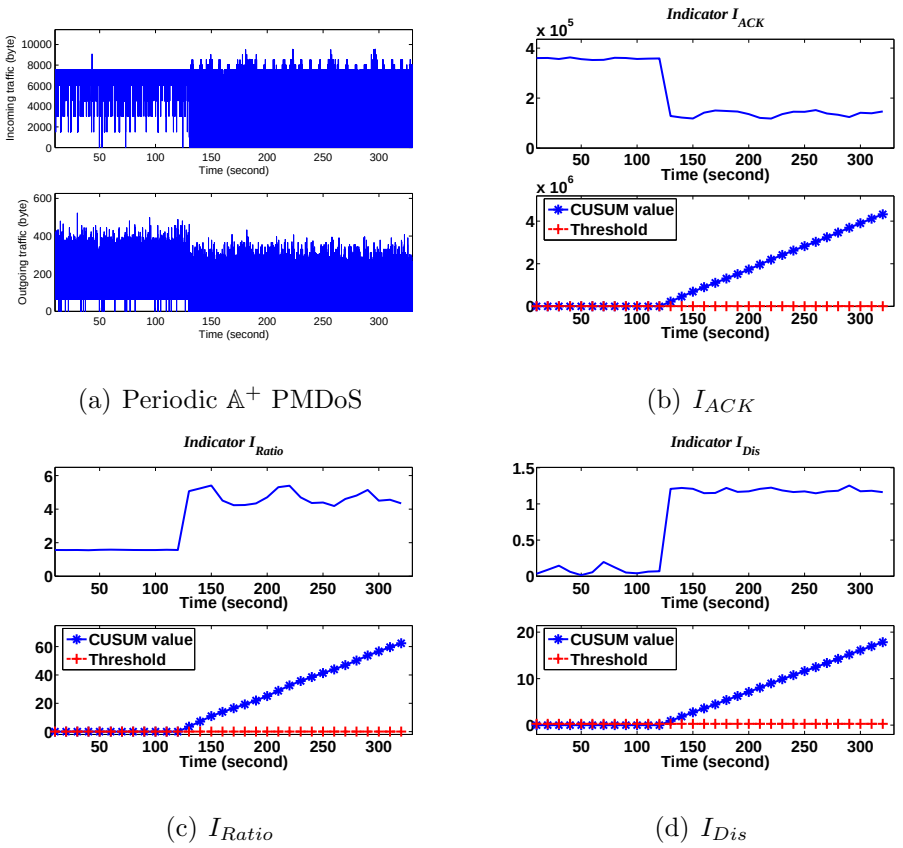
$\overline{I_{ACK}(n)} - P_{tolerance}[\triangle(I_{ACK}(n))]$, where $\triangle(I_{ACK}(n))$ is the standard deviation of $I_{ACK}(n)$ and $P_{tolerance}$ defines the sensitivity to the decline in the outgoing ACK traffic by controlling the allowable decrease during the transformation of $\{I_{ACK}(n)\}$. We set β_{Ratio} and β_{Dis} to the upper bound of $\{I_{Ratio}(n)\}$ and $\{I_{Dis}(n)\}$, respectively.

With $y_{P_{ACK}}(n-1)$, the CUSUM values of $P_{ACK}(n-1)$, and $P_{ACK}(n)$, Vanguard computes the CUSUM value $y_{P_{ACK}}(n)$ as:

$$y_{P_{ACK}}(n) = \max\{0, y_{P_{ACK}}(n-1) + P_{ACK}(n)\}. \tag{4.11}$$

Thus, the presence of abnormal decline in the outgoing ACK traffic is confirmed if $y_{P_{ACK}}(n) > C_{ACK}^{CUSUM}$, where C_{ACK}^{CUSUM} is the corresponding CUSUM threshold. Similarly, by computing the CUSUM values $y_{P_{Ratio}}(n)$ and $y_{P_{Dis}}(n)$ and by comparing with the corresponding CUSUM thresholds C_{Ratio}^{CUSUM} and C_{Dis}^{CUSUM}, we can confirm the presence of the increase in $I_{Ratio}(n)$ and the change in the distribution of incoming TCP traffic.

Figure 4.4, Figure 4.5 and Figure 4.6 demonstrate Vanguard's detection process for a periodic $\mathbb{A}^+$ PMDoS attack, a stochastic $\mathbb{A}^+$ PMDoS attack, and a $\mathbb{A}^-$ PMDoS attack respectively. The data is obtained from test-bed experiments with the presence of cross traffic and a bottleneck capacity of 10 Mbps. Both $\mathbb{A}^+$ PMDoS attacks operate with $R_{attack} = 25$ Mbps and $R_{average} = 6$ Mbps, whereas the $\mathbb{A}^-$ PMDoS attack operates with $R_{attack} = R_{average} = 6$ Mbps. All attacks have the same attack cost of $\gamma = 0.6$. The attack is started at the 131th second from the beginning of the experiment. For each row of subfigures, the first subfigure shows the raw incoming TCP traffic in the upper panel and raw outgoing ACK traffic in the lower panel. The second, third and forth subfigures plot the values of $I_{ACK}(n)$, $I_{Ratio}(n)$ and $I_{Dis}(n)$, respectively. For each set of those subfigures, the upper panel shows the raw data of the statistics, and the lower panel shows the CUSUM detection results of these statistics. We observe that the PMDoS attack induces different kinds of incoming TCP traffic and outgoing ACK traffic patterns. However, the abnormal change in the traffic can be instantly revealed from the three statistics, and thus effectively captured by Vanguard.



(a) Periodic $\mathbb{A}^+$ PMDoS

(b) I_{ACK}

(c) I_{Ratio}

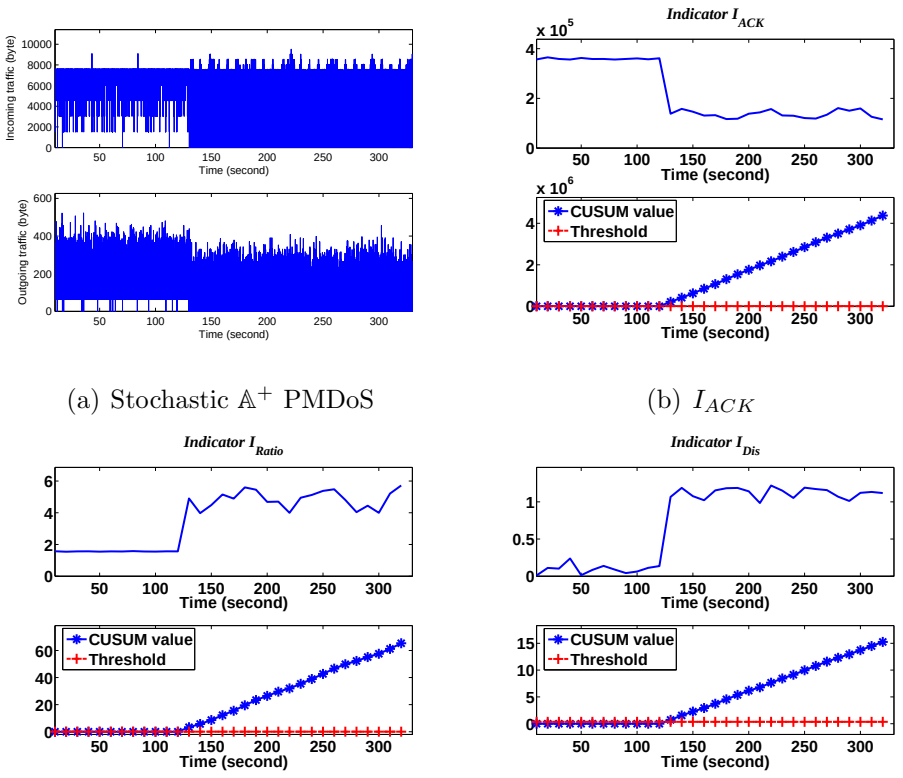
(d) I_{Dis}

Fig. 4.4. Demonstration of Vanguard's detection process for periodic $\mathbb{A}^+$ PMDoS attacks.

4.2.3 Other detection schemes

Since the defense system may need to process a huge volume of incoming packets, having a low computational complexity is a very important consideration in designing a practical defense system. In this subsection, we compare the computational complexity of Vanguard with other proposed detection schemes. We consider the two-stage detection scheme (named as DWTM-based detection scheme) proposed in the section 4.1, spectrum-based scheme [89, 180], and DTWP-based algorithm [87].

Since all the detection schemes under comparison make decision after collecting and manipulating N data samples in a detection window, their lowest complexity is $\Theta(N)$. For Vanguard, the values of statistics for anomaly I and II can be updated



(a) Stochastic $\mathbb{A}^+$ PMDoS

(b) I_{ACK}

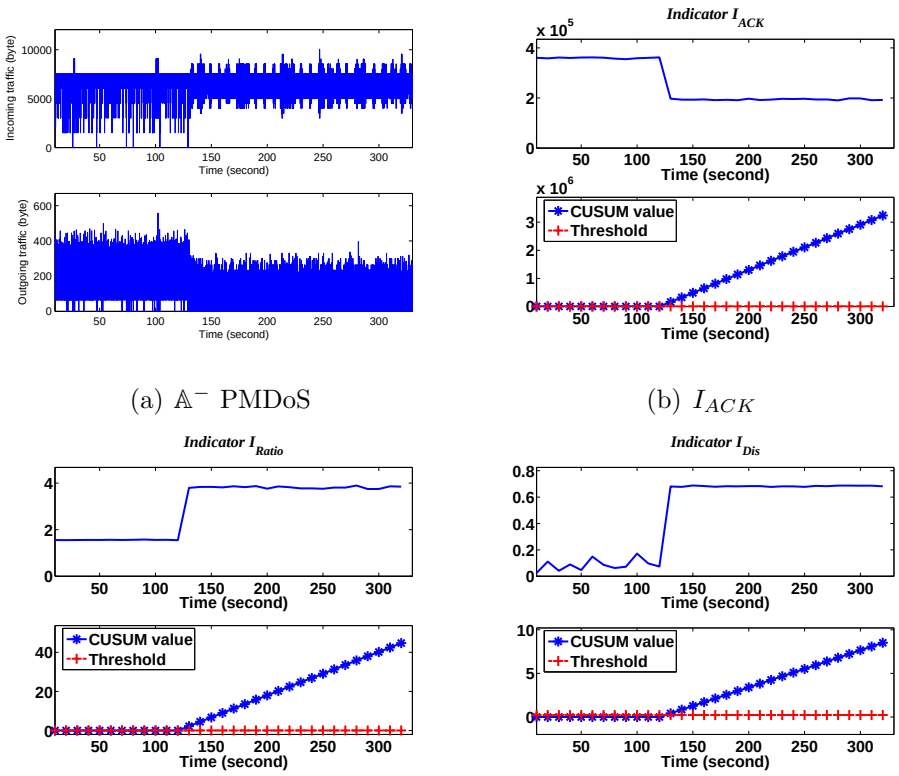
(c) I_{Ratio}

(d) I_{Dis}

Fig. 4.5. Demonstration of Vanguard's detection process for stochastic $\mathbb{A}^+$ PMDoS attacks.

upon receiving each data sample. By using bins with the same size, the data sample can quickly locate the bin it belongs to. After that, the burden of computing statistic for anomaly III is determined by B, which is the number of bins and is usually less than the number of samples (N) in each detection window. The CUSUM algorithm's complexity is $\Theta(1)$ [179]. Therefore, the complexity of Vanguard is $\Theta(N)$.

The spectrum-based detection has been used to differentiate between single-source and multi-source DoS attacks [180], which is employed to detect shrew attack by observing the change in the power spectral density (PSD) of the incoming TCP traffic [89]. Hence, its computational complexity is mainly determined by that of computing the PSD, which is $\Theta(NlogN)$ [181]. However, the spectral analysis cannot handle the



(a) $\mathbb{A}^-$ PMDoS

(b) I_{ACK}

(c) I_{Ratio}

(d) I_{Dis}

Fig. 4.6. Demonstration of Vanguard's detection process for $\mathbb{A}^-$ PMDoS attacks.

PMDoS attacks which could exhibit various frequencies under different settings of attacks. Just like the hop-frequency techniques, the frequency of the PMDoS attack (except for the shrew attack) can be changed easily.

A dynamic time warping (DTWP)-based algorithm is proposed in [87] to identify the shrew attack by matching the pattern of the incoming TCP data traffic with that of shrew attack traffic. It first employs auto-correlation to extract the signatures of the incoming traffic periodically and then compares the extracted signatures of the incoming traffic with the signatures of shrew attack traffic through a slightly modified DTWP algorithm. Since the computational complexity of the auto-correlation processing is $\Theta(N^2)$ and that of DTWP is $\Theta(NM)$, (M is the length of selected

signatures of shrew attack), the DTWP-based algorithm's computational complexity is $\Theta(N^2)$. However, the DTWP method may fail if the attack pulses are not separated by a constant interval. Moreover, the DTWP method will not be able to detect the $\mathbb{A}^-$ attacks as there will not be significant square-wave patterns in the incoming traffic.

Table 4.1 summarizes the computational complexities of the detection schemes. Together with the DWTM-based scheme, Vanguard achieves the lowest computational complexity. Moreover, as will be shown in section 4.3, Vanguard can achieve the highest detection rate for the class of PMDoS attacks.

Table 4.1
Computational complexity of four detection schemes.

Scheme	Computational complexity
Vanguard	$\Theta(N)$
DWTM-based scheme	$\Theta(N)$
Spectrum-based scheme	$\Theta(N \log N)$
DTWP-based scheme	$\Theta(N^2)$

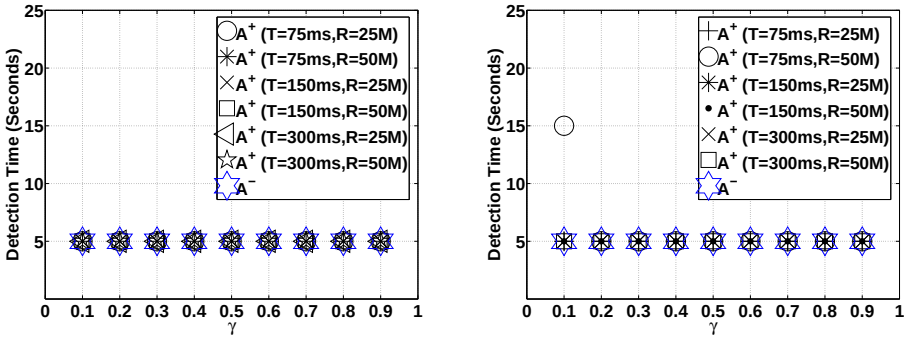
4.3 Evaluation and comparison with other detection schemes

In this section, we present the experiment results of the two-stage detection scheme and Vanguard through a number of test-bed experiments. Our evaluation is based on detection delays required to identify different PMDoS attack variants versus attack costs. We use the same procedures and settings for the test-bed experiments discussed in the previous section, except that we have installed a Snort IDS with the Vanguard preprocessor at R_{n+1} to sniff incoming TCP data traffic and outgoing ACK traffic. For the preprocessor configuration, we use $T_{Sample} = 0.005$s and $W_D = 5$s to achieve

a small detection delay, and $N_{W_D} = 40$ to obtain a training period of 200s. Moreover, we set $B = 25$ and $P_{tolerance} = 2$.

Figure 4.7 plots the detection time of PMDoS attacks against attack costs. Figure 4.7(a) plots the results for the periodic $\mathbb{A}^+$ PMDoS attacks with $T_{on} = \{75, 150, 300\}$ms and $R_{attack} = \{25, 50\}$ Mbps, and Figure 4.7(b) plots the results for the stochastic $\mathbb{A}^+$ PMDoS attacks. Each subfigure also includes the detection times for the $\mathbb{A}^-$ PMDoS attacks. Note that Vanguard can identify all PMDoS attacks with various attack costs within three detection windows (i.e., 15s). Specifically, it identifies all periodic $\mathbb{A}^+$ and the $\mathbb{A}^-$ PMDoS attacks immediately after a detection window. However, it requires slightly more time to identify the stochastic $\mathbb{A}^+$ PMDoS attacks with $\gamma = 0.1$, $T_{on} = 75$ms, and $R_{attack} = 50$ Mbps. Based on the packet traces, we found two reasons for it. Firstly, comparing with other attacks with $T_{on} > 75$ms, such attack's pulse is less significant. Secondly, comparing with attacks with $T_{on} = 75$ms, its high R_{attack} results in a larger $E[T_{off}]$ under the constraint of $\gamma = 0.1$. We observed long T_{off}s (i.e. $> E[T_{off}]$) between consecutive attack pulses at the beginning of the attacks because of the stochastic nature of attack period. As a result, Vanguard also needs a longer period to identify the attacks. Moreover, when γ varying from 0.01 to 0.09, Vanguard can detect 93.5% PDoS attacks and 88.9% flooding attacks. Most of them are detected within 10 detection windows (i.e. 50s).

Furthermore, we use two sets of real data traces to examine Vanguard's false alarm rate. One includes the traces obtained from WIDE backbone (samplepoint-B) during September 2005 and March 2006. The other is the LBNLs internal enterprise traffic [182]. We assume that these data sets do not contain PDoS or PMDoS attack traffic. By feeding these two sets of real data to the Vanguard system, we observed low false alarm rate: 8.2% for the first data set and 9.7% for the second data set.



(a) Periodic $\mathbb{A}^+$ PMDoS attack.

(b) Stochastic $\mathbb{A}^+$ PMDoS attack.

Fig. 4.7. Detection time for $\mathbb{A}^+$ PMDoS and $\mathbb{A}^-$ PMDoS attacks using Vanguard.

4.3.1 Comparison with other detection schemes

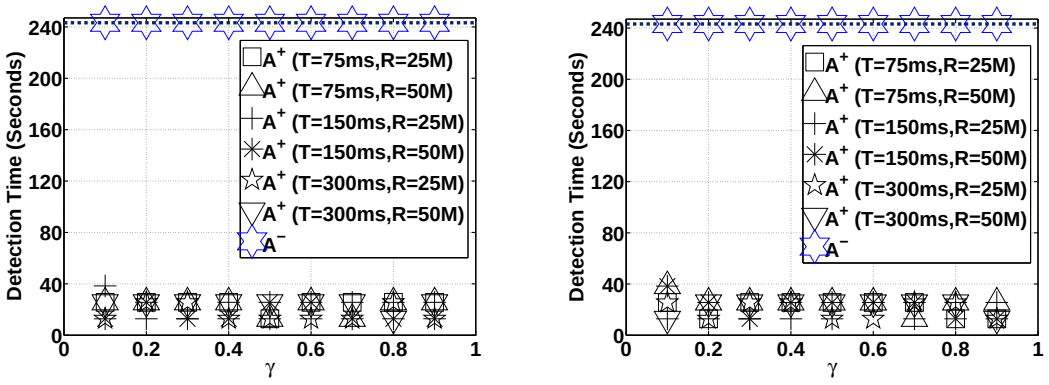
In this section, we further evaluate and compare the performance of Vanguard, the DWTM-based, the DTWP-based, and the spectrum-based detection schemes using test-bed experiments.

DWTM-based detection scheme

Figure 4.8 shows the detection times required for the DWTM-based detection scheme to discover the $\mathbb{A}^+$ and $\mathbb{A}^-$ PMDoS attacks with various attack costs. We have performed the experiments using the same network parameter settings. Each experiment lasts for 370 seconds and a PMDoS attack begins at 130 second. For the configuration of the DWTM-based detection system, each window of continuous samples lasts for 12.8 seconds to achieve a small detection delay, $N_{W_D} = 6$ to obtain a training period of 76.8 seconds, and $P_{tolerance} = 1$. The detection scheme employs the Haar wavelet [183] to capture the fluctuation in the incoming TCP data traffic, and the Daubechies wavelet with four vanishing moments ($DB(4)$) to extract the trend in the outgoing TCP ACK traffic. In each subfigure, any marker coinciding with the

dashed line represents that the corresponding PMDoS attack is undetected by the detection scheme.

From the figures, it is clear that the DWTM-based detection scheme shows a poorer performance than Vanguard, with an average detection rate of 89.18%. Specifically, while this mechanism can discover all the ongoing periodic and stochastic $\mathbb{A}^+$ PMDoS attacks within three detection windows (38.4s), it is unable to discover any $\mathbb{A}^-$ PMDoS attacks. Since the $\mathbb{A}^-$ PMDoS attack traffic constantly occupies a fixed portion of the bottleneck link capacity, the incoming TCP data traffic adapts to the remaining bandwidth without significant fluctuations. As a result, the attack traffic can elude the detection for the incoming traffic. We also evaluate its false alarm rate, which is 2.1% for WIDE traces and 5% for LBNL traces.



(a) Periodic $\mathbb{A}^+$ PMDoS

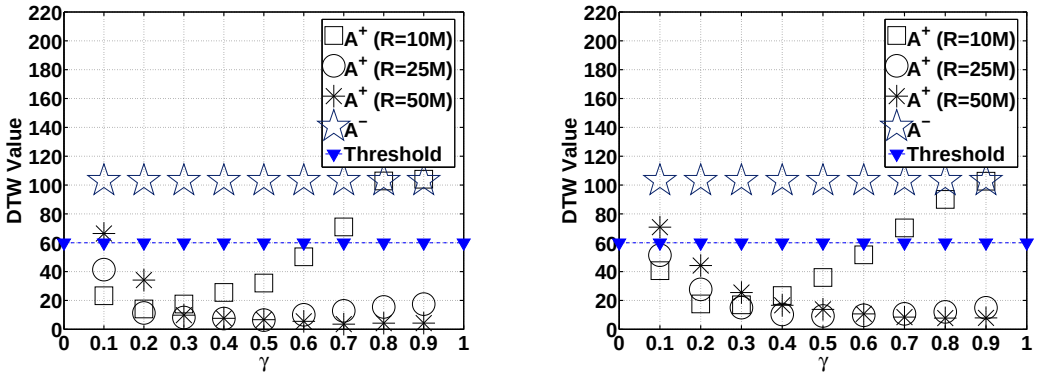
(b) Stochastic $\mathbb{A}^+$ PMDoS

Fig. 4.8. Detection time for $\mathbb{A}^+$ PMDoS and $\mathbb{A}^-$ PMDoS attacks using the DWTM-based scheme.

DTWP-based detection scheme

In Figure 4.9, we report the experiment results of the DTWP-based detection scheme under the $\mathbb{A}^+$ and $\mathbb{A}^-$ PMDoS attacks with different attack costs. Each subfigure reports results for $R_{attack} = \{10, 25, 50\}$ Mbps. The dashed line with downward-

pointing triangles (-▼-) is the DTWP threshold (DTWP=60) recommended in [87]. If the DTWP value is less than the threshold, the algorithm will confirm the presence of a PMDoS attack. We present the experiment results only for $T_{on} = 150$ms, because they are similar to those with $T_{on} = \{75, 300\}$ms. From the figures, we can observe that the DTWP-based scheme can identify many periodic and stochastic $\mathbb{A}^+$ PMDoS attacks, but it cannot detect any $\mathbb{A}^-$ PMDoS attacks. It is because that this detection algorithm is designed specifically for the shrew attack by matching the pattern of the incoming TCP data traffic with that of shrew attack traffic. Thus, this scheme is not able to detect the $\mathbb{A}^-$ PMDoS attacks operated with CBR. Moreover, its average detection rate is only 76.9%, which is relatively less than that obtained by Vanguard and DWTM-based detection schemes.



(a) Periodic $\mathbb{A}^+$ PMDoS attack with $T_{on} = 150$ms.

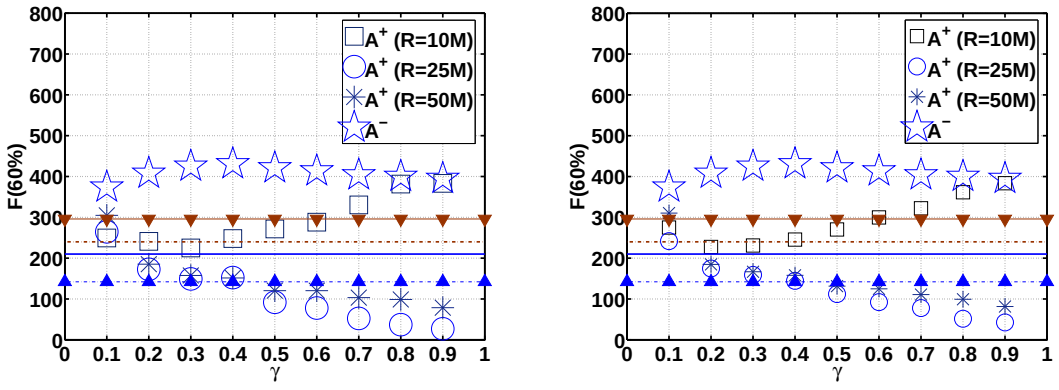
(b) Stochastic $\mathbb{A}^+$ PMDoS attack with $T_{on} = 150$ms.

Fig. 4.9. Detection time for $\mathbb{A}^+$ PMDoS and $\mathbb{A}^-$ PMDoS attacks using the DTWP-based scheme.

Spectrum-based detection scheme

Figure 4.10 shows the experiment results of the spectrum-based detection scheme under the $\mathbb{A}^+$ and $\mathbb{A}^-$ PMDoS attacks. In each subfigure, the area between the solid

line with downward-pointing triangles (-▼-) and the dashed line contains the range of frequencies for single-source DoS attacks. On the other hand, the area between the solid line and the dashed line with upward-pointing triangles (-▲-) contains the range of frequencies for multi-source DoS attacks. The experiment results show that the values ($F(60\%)$) of the $\mathbb{A}^+$ PMDoS attacks do not concentrate on a small range. Instead, they spread from low frequencies to high frequencies. Please note that all the experiments are actually conducted in a single-source manner. Therefore, the spectrum-based detection scheme will regard some single-source $\mathbb{A}^+$ PMDoS attacks as the multi-source ones by mistake. Moreover, such kind of detection schemes will miss those PMDoS attacks whose frequencies are similar to the frequency of normal traffic.



(a) Periodic $\mathbb{A}^+$ PMDoS with $T_{on} = 150$ Mbps. (b) Stochastic $\mathbb{A}^+$ PMDoS with $T_{on} = 150$ Mbps.

Fig. 4.10. Detection time for $\mathbb{A}^+$ PMDoS and $\mathbb{A}^-$ PMDoS attacks under the spectrum-based scheme.

4.4 Summary

In this chapter, we have proposed a two-stage detection scheme for the PDoS attacks. The detection is based on an unusually high variability in the incoming data

traffic and a drastic decline in the outgoing ACK traffic observed in the midst of a PDoS attack. As a result, we have employed wavelet transform to observe the incoming data traffic and outgoing ACK traffic, and a nonparametric CUSUM algorithm to detect change points. Moreover, our scheme is feasible for on-line detection because of the low time complexity for both the computation of the discrete wavelet transform and the CUSUM method.

To enhance the two-stage detection scheme with the capability of detecting some PMDoS attacks, we have proposed Vanguard, a new detection mechanism. Vanguard is based on three anomalies—a drastic decline in the outgoing ACK traffic, an unusually high variability in the ratio of the incoming TCP traffic and the outgoing TCP ACK traffic, and a significant change in the incoming TCP traffic distribution. We have implemented it as a Snort plug-in and experimented with it on a test bed. The experiment results show that Vanguard is very effective at detecting the PMDoS attack. Moreover, we have compared Vanguard with other proposed detection systems. Vanguard incurs the lowest computational complexity, while achieving the highest detection rate.

Although the experiment results obtained in our test bed demonstrates Vanguard's good performance, we cannot assure its performance in a real network environment, because our experiments are limited by the test bed's scale and the lack of real background traffic. Therefore, we conjecture that the third indicator (i.e. change in the incoming TCP data traffic distribution) might be sensitive to traffic variation in real network environment. We will investigate this issue in a real operational network and study how to select suitable parameters. Moreover, currently we only employ a heuristic decision rule. It is still possible for a smart attacker to evade it or cause false alarm. We will carry out formal analysis on the decision rule in the future work.

5. TCPSCRIPT: COVERT COMMUNICATIONS IN TCP BURSTINESS

In this chapter, we propose hiding covert messages into TCP's traffic bursts and introduce our design, TCPScript. By exploiting TCP's rich protocol features, TCPScript provides higher throughput and more robust service than other recently proposed timing channels. We evaluate its performance by carrying out experiments in both our test bed and the PlanetLab platform. Moreover, we have designed new detection metrics and a detection scheme to uncover TCPScript channels.

5.1 The basic design

5.1.1 Network model

There are four parties in the model: an encoder, a decoder, a TCP server, and a passive warden. The encoder and the warden are in a close proximity in terms of the communication delay. We assume that the passive warden is equipped with unlimited resources to detect network covert channels established between its network and the outside. Therefore, he can monitor and analyze all the incoming and outgoing network traffic, including those from the encoder's machine. Being a passive warden, however, he will not delay packets or control the transmission rate of hosts in his network.

The goal of the encoder is to leak out information to the decoder outside without being detected by the warden. In so doing, she establishes a "normal" TCP-based application session with the TCP server outside her network and embeds messages in the TCP data segments. Here we do not further differentiate between the encoder and the TCP sender; thus, we use encoder and TCP sender interchangeably. The

decoder is located such that he can eavesdrop packets sent from the encoder to the server. However, the decoder does not have to eavesdrop packets sent from the server to the encoder.

5.1.2 Encoding and decoding

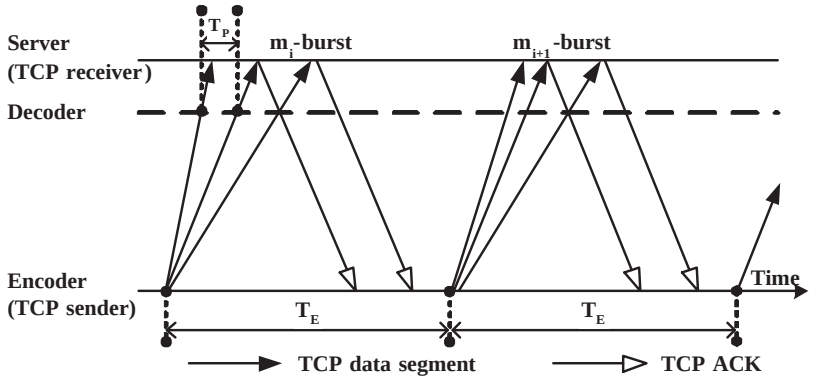
TCPScript encoding In the following we assume that each covert message m_i is represented by a positive integer: $m_i \in [1, M]$, where M is a constant. Figure 5.1(a) depicts the encoding process for messages m_i and m_{i+1}. To transmit m_i, the encoder sends m_i back-to-back TCP data segments to the server. To simplify the explanation, we set the size of all data segments to the TCP sender's *maximum segment size* (MSS) in bytes. Moreover, we denote the TCP sequence number (SN) of the jth data segment by $SN_j, j = 1, \ldots, m_i$. From the beginning of the transmission, the encoder awaits for the ACKs for a period of T_E, an *encoding period* for a covert message. Let the maximum value of the arrived ACKs at the end of T_E be ACK_{max}. In the new packet burst for m_{i+1}, the first packet's SN is set to ACK_{max}.

There are a couple of points to note for this simple encoding scheme:

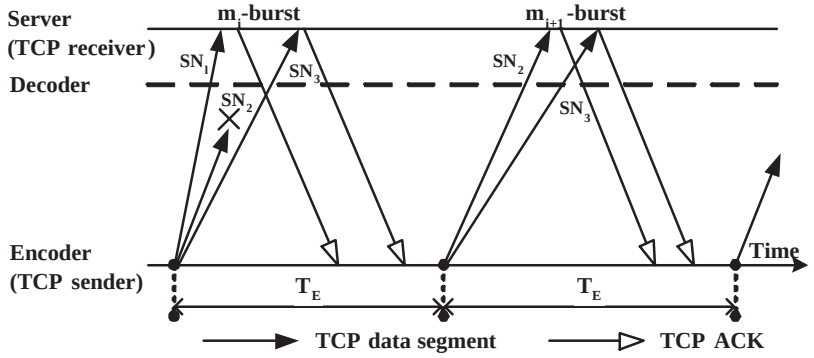
1. If the ACKs for all m_i segments arrive before the end of T_E, the encoder considers the transmission of m_i successful (it does not imply a correct decoding though). In the next burst, the first packet's SN is therefore set to $SN_{m_i} + MSS$ (because $ACK_{max} = SN_{m_i} + MSS$). Due to the TCP cumulative ACK, it is sufficient to receive an ACK with a value of $SN_{m_i} + MSS$.

2. If not all the ACKs arrive on time (due to packet losses, delay jitter, or packet reordering), then $ACK_{max} < SN_{m_i} + MSS$. Therefore, the encoder retransmits the unacknowledged data segments with $SN = ACK_{max}, \ldots, SN_{m_i} + MSS$. As we shall see next, the decoder does not care whether a TCP data segment is a retransmission or not.

TCPScript decoding Prior to the communication, the encoder and decoder will agree on two parameters: the number of bursts (denoted by N) and the maximal



(a) There are no packet losses for the two packet bursts, and all ACKs arrive within T_E.



(b) The segment with SN_2 is lost which, however, does not affect the decoding, and all ACKs arrive within T_E.

Fig. 5.1. Two scenarios for encoding messages m_i and m_{i+1} in TCPscript.

number of packets in a burst (denoted by M). The start of the communication can be signaled by the arrival of the first TCP data packet. There are several ways to decoding the messages. Here we present an off-line decoding algorithm based on packet clustering. There are two stages to this off-line algorithm:

1. The decoder first uses a clustering algorithm to partition all the captured TCP data segments into N clusters, each of which corresponds to a transmission burst. We use the hierarchical clustering algorithm, whose hierarchical cluster tree is constructed based on the centroid linkage [184]. Based on our extensive

decoding results, this method achieves better performance than other popular algorithms.

2. Consider the m_i-burst identified in the first phase. The decoder first obtains the maximum and minimum SNs from the packets received during T_E, denoted by $SN_{i,max}$ and $SN_{i,min}$, respectively. Then the message is decoded by counting the number of data segments based on $SN_{i,max}$, $SN_{i,min}$, and MSS:

$$\tilde{m}_i \longleftarrow \frac{SN_{i,max} - SN_{i,min}}{MSS} + 1. \tag{5.1}$$

Although the decoding algorithm in Eq. (5.1) is very simple, it has the following important properties:

(a) Even when some data segments are missing in the m_i-burst (due to packet losses or reordering), the decoding is still correct as long as the first and last segments are classified into the m_i-burst. Figure 5.1(b) illustrates such a case where the segment with SN_2 is lost; however, the arrival of the segment with SN_3 helps decode correctly.

(b) To recover missing packets at the head of a burst, we modify the algorithm above by setting $SN_{i,min}$ to:

$$SN_{i,min} = min\{SN_{i,min}, SN_{i-1,max} + MSS\}. \tag{5.2}$$

Assume that the decoder captures the entire m_{i-1}-burst. If the first packet in the m_i-burst is not missing, then $SN_{i,min} = SN_{i-1,max} + MSS$; otherwise, $SN_{i,min} > SN_{i-1,max} + MSS$. Thus, selecting $SN_{i,min}$ according to Eq. (5.2) can still decode correctly in the presence of missing data segments at the head of the m_i-burst. For the m_1-burst, we can set $SN_{1,min} = min\{SN_{1,min}, SN_{0,max} + 1\}$, where $SN_{0,max}$ is the SN of the TCP SYN segment sent by the encoder.

(c) However, there remains a scenario where packet losses at the head of a burst still give an incorrect decoding, which is illustrated in Figure 5.2(a).

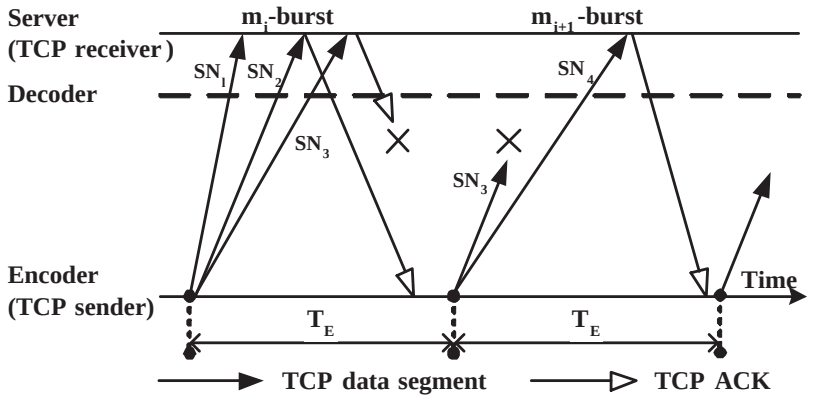
As shown, the ACK for the segment with SN_3 is lost; therefore, $SN_{i,max} = SN_3$. Moreover, according to the encoding algorithm, SN_1 of the m_{i+1}-burst is given by SN_3. Based on Eq. (5.2), $SN_{i+1,min} = SN_3 + MSS$ which causes a decoding error: $\tilde{m}_{i+1} = m_{i+1} - 1$.

(d) When data segments are missing at the tail of a burst, there is no way to decode the message correctly under this simple scheme; Figure 5.2(b) shows one such scenario.

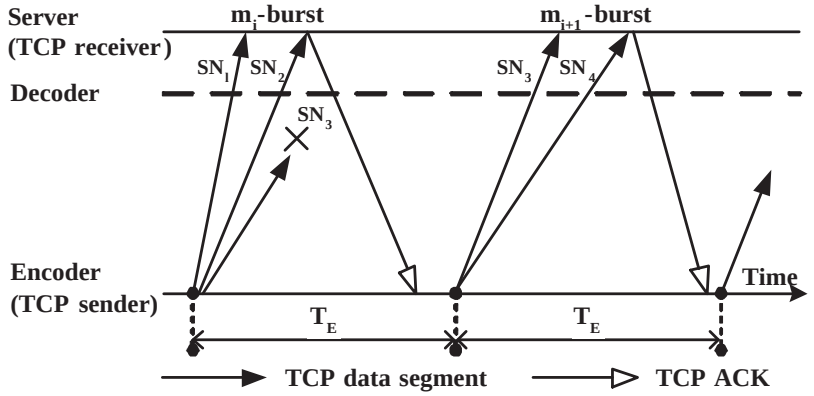
Note that TCPScript does not necessarily require all packets to have the same size and use the explicit number of packets to convey messages. Instead, the encoder and decoder could agree on a size unit, denoted as U_{size}, and then the covert message $\tilde{m}_i$ is equal to $\lceil \frac{SN_{i,max} - SN_{i,min}}{U_{size}} \rceil + 1$. By doing so, TCPScript could further imitate the behavior of some TCP applications whose packets are not always of the same size.

Selecting the value of T_E The choice of T_E obviously represents a trade-off between the decoding accuracy and channel's throughput. Assuming that the TCP server does not use delayed ACK, the encoder can set the value of T_E to at least $RTT + (m_i - 1)T_p$ for m_i, where RTT is the estimated round-trip time between the encoder and the server, and T_p is the time dispersion of adjacent data segments. A TCP sender is already equipped with an RTT estimate. To estimate the value of T_p, a simplest approach is to use the time interval between adjacent ACKs as an approximation for T_p, because the ACKs unlikely experience a large time dispersion because of their small sizes [185].

Selecting the value of M The range of M is determined jointly by the server's maximum receive window size, denoted by $RCV.WND_{max}$, and MSS: $1 \leq M \leq \lfloor \frac{RCV.WND_{max}}{MSS} \rfloor$. If the window scale option is not used, then $RCV.WND_{max} \leq 64$ KB. Take a common setting of MSS = 1460 bytes; then $M = \lfloor \frac{64KB}{1460} \rfloor = 44$. Moreover, depending on the desired behavior of the cover traffic, the encoder may select the value of M from $[10, 20]$ to mimic the behavior of a long-lived TCP flow or from $[3, 8]$ to mimic the behavior of a short-lived TCP flow [186].



(a) Packet loss at the head of the m_{i+1}-burst will cause a decoding error.



(b) Packet losses at the tail of a burst will cause a decoding error, such as the lost segment in the m_i-burst.

Fig. 5.2. Two packet loss scenarios that cause decoding errors in TCP-Script.

5.1.3 Impacts of adverse network conditions

In this section, we analyze the impact of three adverse network conditions on TCPScript.

Packet losses There are three data segment loss scenarios that will lead to incorrect decoding for the basic TCPScript, assuming no packet reordering and inconsequential delay jitter:

1. As discussed earlier, one or more packet losses occurred at the tail of the m_i-burst will cause a decoding error for m_i.

2. Packet losses at the head of the m_i-burst will cause a decoding error for m_i *only* for the scenario described in item 2(c) under TCPScript decoding.

3. When all the data segments in the m_i-burst are lost, the decoder will clearly miss m_i.

In other cases, say packet losses in the middle of a burst, TCPScript could still correctly obtain the message. Moreover, to further improve the decoding accuracy, we will describe in §5.3 a more robust scheme to recover m_i under any packet loss scenarios.

Delay jitter TCPScript can tolerate a small (relative to T_E) delay jitter. However, a large (relative to T_E) delay jitter could lead to an incorrect clustering of the packets. That is, the N bursts could be clustered into more than or less than N bursts, or into a different set of N bursts. Thus, a large delay jitter is considerably more damaging, because it could affect more than a couple of messages. The robust scheme to be presented in §5.3 can alleviate the problem, but it still cannot handle arbitrarily large delay jitter. An effective solution is of course to choose T_E conservatively at the expense of a lower capacity.

Packet reordering Packet reordering events are prevalent on some Internet paths [187]. Consider that a data segment belonging to m_i is reordered for the following three scenarios, assuming no packet losses and inconsequential delay jitter:

1. The reordered packet is still classified into the m_i-burst; this case clearly has no impact on the decoding.

2. The reordered packet is classified into the m_{i+1}-burst. The message m_i will be decoded incorrectly *only if* this reordered packet is originally the last packet in the burst (see item 2(d) under TCPScript decoding). Moreover, since the encoder retransmits this reordered packet in the next burst, the decoder receives

duplicate data segments in the m_{i+1}-burst which, however, does not affect its decoding.

3. The reordered packet is classified into the m_j-burst, where $j > i+1$. Its impact on m_i is the same as the second case. However, by appearing in the m_j-burst, the reordered packet will cause an incorrect decoding of m_j, because $SN_{j,min}$ is set to this packet's SN (see Eq. (5.2)).

On the other hand, a reordered ACK has no impact on the decoding accuracy because of the accumulative ACK and the encoding algorithm.

5.2 An information-theoretic analysis of TCPScript

Determining a covert channel's capacity is very useful for assessing its impact and for designing the defense systems [33, 34, 188]. Moreover, according to Shannon's channel coding theorem, the encoder and decoder could select a coding approach to reliably transmit information over covert channels at all rates up to its channel capacity guarantees [189]. However, very few works investigated the capacity of network timing channels [36, 42]. Venkatraman et al. used a noiseless model to analyze the capacity of network covert channels that exploit the spatial and temporal variation in transmission characteristics [36]. Berk et al. used the Arimoto-Blahut algorithm [123] to estimate the capacity of network timing channels. Its transition matrix is obtained by conducting a large volume of experiments and then measuring the probability that one time interval becomes another time interval. However, none of them consider the effect of packet loss on channel's capacity. In this section we conduct an information-theoretic analysis for TCPScript. For the purpose of comparison, we have also performed the analysis for IPTime and JitterBug.

Being network timing channels, TCPScript, IP timing channel and JitterBug are all vulnerable to packet losses and delay jitter. The former can cause substitution errors—the symbol sent by encoder is misinterpreted as another symbol or an error indicator—whereas the latter can cause synchronization errors (e.g., deletion errors

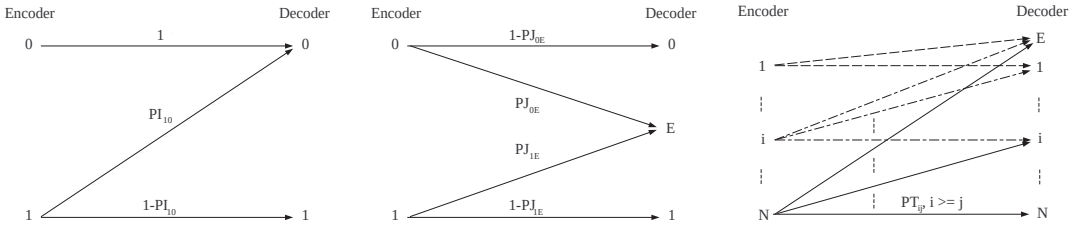
and insertion errors). Note that finding the capacity of *insertion/deletion channels* is still an active research problem. As a result, we consider only the impact of packet losses on the three network timing channels in the following analyses.

We model all three channels as discrete memoryless channels (DMCs) with the input symbols and output symbols modeled by discrete-valued random variables X and Y, respectively. Moreover, $p(x)$ and $q(y)$ are the probability mass functions for X and Y. We also denote $P(X = x_i)$ by $p(x_i)$. A DMC's capacity is defined as [123]:

$$C = \max_{p(x)} I(X;Y) = \max_{p(x)}\{H(X) - H(X|Y)\} = \max_{p(x)}\{H(Y) - H(Y|X)\}, \tag{5.3}$$

where $H(X)$ (or $H(Y)$) is the entropy of X (or Y), and $H(X|Y)$ and $H(Y|X)$ are the conditional entropies defined as:

$$H(X|Y) = -\sum_i \sum_j q(y_j)p(x_i|y_j)log_2 p(x_i|y_j) \; and \; H(Y|X) = -\sum_i \sum_j p(x_i)q(y_j|x_i)log_2 q(y_j|x_i). \tag{5.4}$$



(a) Modeling IP timing channel as a Z-Channel.

(b) Modeling JitterBug as a Binary Erasure Channel.

(c) Modeling TCPScript as an Error and Erasure Channel.

Fig. 5.3. Modeling TCPScript, IP timing channel, and JitterBug as discrete memoryless channels.

According to each channel's characteristics, we model IPTime, JitterBug, and TCPScript using Z-Channel, Binary Erasure Channel (BEC), and Error and Erasure Channel (EEC), respectively, as shown in Figure 5.4.

1. **IPTime** There are two input symbols to the channel: nonarrival of a packet (`0`) and arrival of a packet (`1`). Therefore, symbol `1` could be decoded as `0` due to a packet loss (with probability PI_{10}), but symbol `0` is always decoded as `0`.

2. **JitterBug** Each symbol is represented by the time interval between two consecutive packets. Therefore, it will become a deletion channel if at least one packet is lost. To simplify the analysis, we consider the case of transmitting two packets and model it as an *erasure channel*. The erasure state E is entered when either packet is lost with probabilities PJ_{0E} and PJ_{1E} for an input symbol of `0` and `1`, respectively.

3. **TCPScript** Recall from section 5.1 that packet losses at a burst's tail will result in decoding error, but packet losses at a burst's head may or may not incur an error. Moreover, there are two kinds of errors: $\tilde{m_i} < m_i$ or m_i is missing entirely. The first type of error enforces that $x_i > y_i$. The second type of error is the result of losing the entire burst of packets; the corresponding output symbol is an erasure state. To simplify the analysis, we derive the upper bound (burst-tail errors or burst-head errors) and lower bound (only burst-tail errors) on the probability of correct decoding which will in turn give the lower bound and upper bound on the capacity, respectively.

IPTime From the channel model in Figure 5.3(a), we can obtain the probability transition matrix as $Q(Y|X) = \begin{pmatrix} 1 & 0 \\ PI_{10} & 1-PI_{10} \end{pmatrix}$, where $PI_{10} = P_{loss}$. By applying the known capacity result for a Z-Channel [190],

$$C = \mathbb{F}[p(x)(1-PI_{10}), 1-p(x)(1-PI_{10})] - p(x)\mathbb{F}[(1-PI_{10}), PI_{10}], \ \textit{where} \quad (5.5)$$
$$\mathbb{F}[\alpha,\beta] = -\log_2(\alpha^\alpha\beta^\beta), \ \textit{and} \ p(x) = \frac{PI_{10}^{\frac{PI_{10}}{1-PI_{10}}}}{1+(1-PI_{10})PI_{10}^{\frac{PI_{10}}{1-PI_{10}}}}.$$

JitterBug From the channel model in Figure 5.3(b), we obtain the probability transition matrix of JitterBug as $Q(Y|X) = \begin{pmatrix} 1-PJ_{0E} & PJ_{0E} & 0 \\ 0 & PJ_{1E} & 1-PJ_{1E} \end{pmatrix}$, where

$PJ_{0E} = PJ_{1E} = 1 - (1 - P_{loss})^2$. By applying the known capacity result for a BEC channel [189], we obtain $C = (1 - P_{loss})^2$.

TCPScript We derive the upper bound on $p(x_i|x_j)$ (denoted by $\bar{p}(x_i|x_j)$) by obtaining the probability that packet losses occur only to the bursts' tails. Therefore, its probability transition matrix is given by:

$$\bar{p}(x_i|x_j), i = 1, \ldots, N; j = 0, \ldots, N, \quad = \quad \begin{cases} (1 - P_{loss})^j P_{loss}^{i-j} & \text{if } i > j \\ 1 - \sum_{k=0}^{i-1} p(x_i|x_k) & \text{if } i = j \\ 0 & \text{if } i < j. \end{cases} \tag{5.6}$$

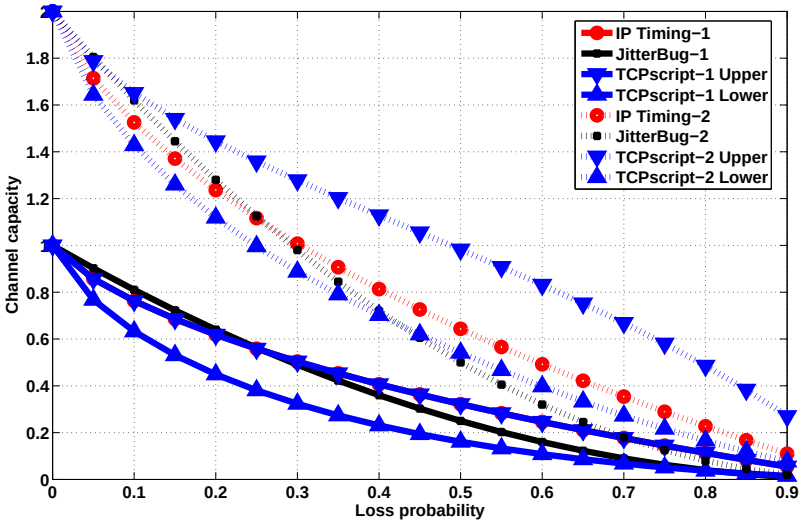
The case of $j = 0$ corresponds to the erasure state.

To derive the lower bound on $p(x_i|x_j)$ (denoted by $\underline{p}(x_i|x_j)$), we consider packet losses incurred in either the head or tail of a burst. For example, if an input symbol of 5 is decoded incorrectly to 2, there are 4 (computed by $5 - 2 + 1$) possible error combinations—$\{3, 0\}$,$\{2, 1\}$,$\{1, 2\}$,$\{0, 3\}$—where the first (or second) number is the number of packet losses at the head (or tail) of a burst. Therefore, we can obtain the probability transition matrix as:

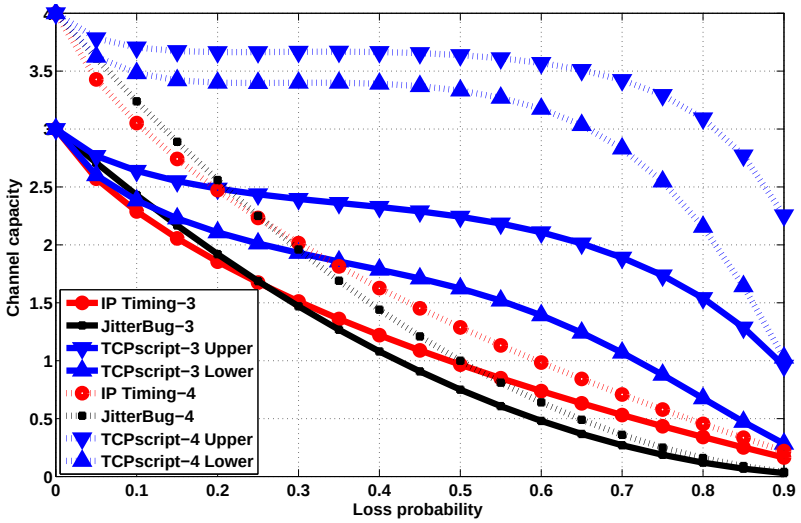
$$\underline{p}(x_i|x_j), i = 1, \ldots, N; j = 0, \ldots, N, \quad = \quad \begin{cases} P_{loss}^i & \text{if } j = 0 \\ (i - j + 1)(1 - P_{loss})^j P_{loss}^{i-j} & \text{if } i > j,\ j \neq 0 \\ 1 - \sum_{k=0}^{i-1} p(x_i|x_k) & \text{if } i = j \\ 0 & \text{if } i < j. \end{cases} \tag{5.7}$$

Based on Eq. (5.6) and Eq. (5.7), we can estimate the lower bound and upper bound for the TCPScript capacity, respectively, by applying the Blahut-Arimoto algorithm [121–123].

We compare the information capacity derived from the channel models for TCPScript, IPTime, and JitterBug under different *packet loss probability* denoted by P_{loss}. Furthermore, we consider $M = 2, 4, 6, 8$ for TCPScript. To give a fairer comparison, we multiply the capacities of IPTime and TCPScript by $log_2 M$ (i.e., equivalent to $log_2 M$ parallel channels). In terms of notations, we use IP Timing-$log_2 M$ to refer to the IPTime capacity for M symbols and similarly for JitterBug and TCPScript.



(a) Channel capacities with 2 and 4 input symbols



(b) Channel capacities with 8 and 16 input symbols

Fig. 5.4. A capacity comparison for TCPScript, IP timing channel, and JitterBug.

We first discuss the results for $M = 2$ (i.e., single-bit encoding) which are shown in Figure 5.4(a). The TCPScript upper bound is actually overlapped with IPtime (that is why you cannot see the IPTime graph from the figure). To explain this result, we go back to TCPScript's channel model in Figure 5.3(c) and set $M = 2$. Moreover, if

we group the erasure state with the state 1 on the decoder side, the resulting channel model is then the same as a Z-Channel. Besides, we have observed that JitterBug slightly outperforms the other two at a low P_{loss}; however, the trend is reversed at a higher loss probability. These results are consistent with our understanding that any packet loss could reduce JitterBug into a deletion channel.

Figure 5.4(a) and Figure 5.4(b) show the results for $M = 4, 6, 8$ (i.e., multibit encoding). When $M = 4$, the TCPScript upper bound is already larger than the other two channels' capacities; however, the lower bound is still below the other two at low P_{loss}s. When M is increased to 6 and 8, Figure 5.4(b) clearly illustrates the advantage of TCPScript, because both upper and lower bounds are larger than the other two for almost all P_{loss}s. Moreover, the advantage becomes even more outstanding as P_{loss} increases. The major reason responsible for its resilience to packet losses is that not all packet losses will incur decoding errors—only those occurred to a burst's tail and some occurred to a burst's head. In contrast, every packet error will result in decoding errors for IPTime and JitterBug.

5.3 Loss-resilient TCPScript

In this section we present a *loss-resilient TCPScript* (LR-TCPScript) that provides better reliability against packet losses by exploiting TCP's feedback channel (i.e., ACK packets). The main idea of the LR-TCPScript is to recover the packet burst (message) for which the encoder fails to receive all the ACKs within T_E before sending the next message.

LR-TCPScript encoding Recall that a TCPScript encoder considers the transmission of message m_i successful if it receives all the ACKs within T_E. For the successful case, the LR-TCPScript operates the same way as the basic TCPScript. To handle the unsuccessful case, the LR-TCPScript encoder uses the following timeout-retransmission scheme to recover the message:

1. At the time of sending the first data segment in the m_i-burst, the encoder starts a retransmit timer whose period T_O should be much larger than T_E. For example, similar to the RTO computation in TCP, we might set $T_O = 2 \sim 3 \times T_{max,E}$, where $T_{max,E}$ is an upper bound on the encoding period.

2. At the end of T_O, the encoder retransmits the data segments unacknowledged by the end of T_E; this retransmission behavior resembles the TCP timeout-based retransmissions. For example, Figure 5.5(a) shows that the third packet in the m_i-burst is lost; therefore, the ACKs do not cover the last two packets which are retransmitted after timeout. Moreover, the encoder could possibly receive new ACKs between the end of T_E and the end of T_O. To be prudent, the encoder should also retransmit the corresponding data packets, because they may have experienced a large delay jitter which could cause incorrect decoding.

3. After the retransmissions, similar as before, the encoder awaits for the ACKs during a period of T_E. If all the missing ACKs arrive during T_E, the encoder proceeds with the next packet burst. Otherwise, the retransmission procedure repeats until receiving all the ACKs or reaching a maximum number of retransmissions.

The timeout-retransmission mechanism above also works for ACK losses, but we do not have the space to describe it here.

LR-TCPScript decoding The off-line decoding algorithm for LR-TCPScript is the same as that for the basic TCPScript, except that there is a preprocessing step for the LR-TCPScript decoding which is described below.

1. Based on T_O, the decoder first identifies from the packet traces the possible timeout-and-retransmission events. Since we have set the value of T_O much larger than T_E, it is straightforward to identify these events.

2. For each such timeout-and-retransmission event, the decoder performs a two-step preprocessing on the traces:

(a) The decoder first removes the possible duplicate data segments which are due to ACK losses.

(b) The decoder then merges the set of partially received data segments before retransmissions and the retransmitted segments such that they will be likely classified into the same burst by a clustering algorithm.

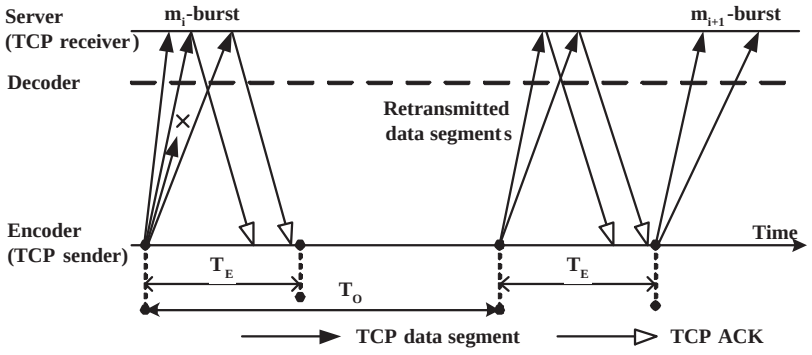
The remaining procedures for the preprocessed traces are the same as before: clustering the preprocessed traces into packet bursts, and decoding the message in each burst.

A ROLLBACK scheme The LR-TCPScript encoding and decoding work correctly for all data segment/ACK loss scenarios except for two special cases: the lost of all data segments in a burst, and the lost of all ACKs in a burst. As an example, consider that all the data segments in the m_{i+1}-burst are lost. Since the decoder has no knowledge about the m_{i+1}-burst, she will count the retransmitted segments into the m_i-burst.

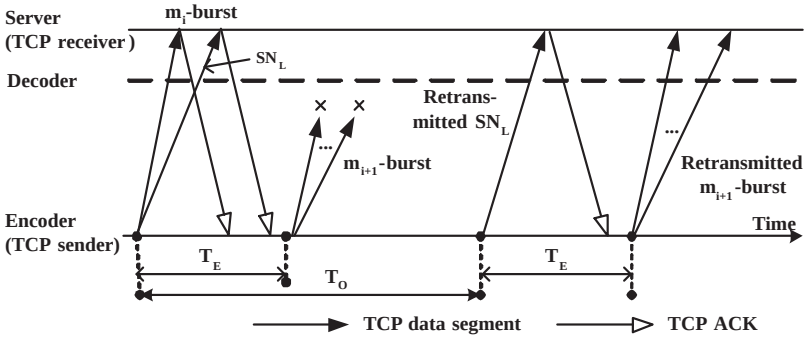
We propose ROLLBACK to address this problem which is illustrated in Figure 5.5(b). ROLLBACK works as follows. If the encoder does not receive any ACK at the end of T_O after sending the m_{i+1}-burst for the first time, it will retransmit the last data segment in the m_i-burst. This retransmitted packet (SN_L), while not affecting the message decoding, triggers a duplicate ACK. Upon receiving the duplicate ACK within T_E, the encoder retransmits all the lost data segments at the end of T_E. Since the time between receiving the segment SN_L and receiving the first retransmitted segment is considered as an inter-burst gap by the decoder, the retransmitted data segments will be correctly classified into a new burst. Note that if either SN_L or the duplicate ACK is lost, the encoder will send SN_L again at the end of another T_O.

5.4 Evaluation and comparison with other network timing channels

In this section, we report the evaluation results of the basic TCPScript (B-TCPScript) and the loss-resilient TCPScript (LR-TCPScript). We have also con-



(a) The timeout-retransmission mechanism



(b) The ROLLBACK mechanism

Fig. 5.5. LR-TCPScript's timeout-retransmission and ROLLBACK mechanisms for achieving better reliability against packet losses.

ducted experiments for JitterBug and the IPTime. The evaluation is conducted in both a controlled test bed and the PlanetLab infrastructure [44]. The test bed allows us to configure different network conditions such as delay, loss, and packet reordering, while the PlanetLab enables us to evaluate the network timing channels in a real network environment. We adopt the empirical *channel accuracy* α and the empirical *channel goodput* ζ:

$$\alpha = 1 - p_e \quad , \quad \zeta = \frac{\alpha N \log_2 M}{T_d}, \tag{5.8}$$

where p_e is the channel's bit error rate (BER) and T_d is the total time required for delivering the N covert messages. We employ the Levenshtein (Edit) Distance to com-

pute p_e, which is given by the total number of insertions, deletions, and substitutions required to convert a source message into a decoded message.

The evaluation platforms The test bed, in a dumbbell topology, consists of two routers (i.e., R_1 and R_2), a Web server, an encoder, and a decoder. All the network links are full duplex with capacity of 100Mbit/s. We employ Dummynet [164] in both routers to emulate different network conditions. The RTT between the encoder and R_1 is 5ms, whereas that between the decoder/Web server and R_2 is 25ms, unless specified otherwise. The encoder embeds covert data into the TCP packets destined to the Web server; the decoder receives the data by eavesdropping encoder's packets between R_2 and the Web server.

In the PlanetLab experiments, we have selected eight geographically diversed nodes, denoted as TW, CN, JP, CA, KS, RI, KR, and BE. These nodes serve as a TCPScript encoder and communicate with a Web server outside PlanetLab. Being located beside the Web server, the decoder extracts covert messages from the flows between the encoder and the Web server. We have measured RTTs and hop distance between the decoder and the PlanetLab nodes during the experiment period.

Table 5.1
Measured path characteristics between each PlanetLab site and the decoder machine.

Locations	Hops	RTT		
		Means	Std. Dev.	Conf. Intervals
Taipei, Taiwan (TW)	10	.0525	.0642	.0504/.0545
Shenyang, China (CN)	13	.0812	.0797	.0786/.0837
Tokyo, Japan (JP)	16	.1165	.0726	.1142/.1188
California, U.S. (CA)	14	.2007	.0682	.1985/.2029
Kansas, U.S. (KS)	16	.2263	.0077	.2260/.2265
Rhode Island, U.S. (RI)	13	.2335	.0079	.2332/.2337
Gwangju, Korea (KR)	18	.2381	.0565	.2363/.2399
Flanders, Belgium (BE)	16	.3158	.0079	.3156/.3161

TCPScript implementation We have implemented the encoder as an HTTP client that, after TCP's three-way handshaking, uses the `HTTP POST` method to transmit a large document to the server and conveys covert messages at the same time. In order to manipulate each packet, we employ Linux raw sockets to bypass the normal TCP/IP stack. The TCPScript decoder, on the other hand, utilizes `libpcap` to capture the traffic sent from the encoder to the TCP server. Based on UDP socket, we have also implemented the IPTime's and JitterBug's encoding and decoding algorithms that use a fixed timing interval (or timing window) of w. We adopt the default tolerance parameter $\varepsilon = w/4$ in the JitterBug encoder.

5.4.1 Test-bed experiments

We study the effects of packet losses and packet reordering on the channel's performance in the test bed. The B-TCPScript encoder used $T_E = \{40, 60\}$ms to transmit two sets of codewords with $M = \{4, 16\}$. Each set consisted of 100 codewords. When examining LR-TCPScript, we also fixed $T_O = \{90, 120\}$ms. For both JitterBug and IPTime, we assigned their encoders to transmit the corresponding binary codewords with a single flow of modulated UDP packets with $w = \{40, 60\}$ms. We have fixed the size of each IP packet injected by the encoder to 1,500 bytes for all experiments. We repeated each experiment for 30 times and report the average goodput $\overline{\zeta}$ and average accuracy $\overline{\alpha}$. Generally, a larger w and T_E will result in higher accuracy at the cost of lower good put.

Effect of packet losses

Table 5.3 lists the experiments results of the B-TCPScript, LR-TCPScript, JitterBug, and IPTime under random packet loss rates (PLR) ranging between 0% and 5%. In each cell, the two leftmost values correspond to the lower limit of and the upper limit of the 95% confidence intervals of $\overline{\zeta}$, and the right most values inside the parentheses correspond to the 95% confidence intervals of $\overline{\alpha}$. We observe that both

B-TCPScript and JitterBug obtained 100% accuracy during a lossless environment, while that of IPTime could only be up to 97% due to the jitter of queueing delay. Moreover, B-TCPScript's goodput was at least 1.561 times larger than that of JitterBug when $M = 4$, and was at least 3.159 times greater when $M = 6$. As PLR increased, as expected, decoder started producing errors from the decoded codewords for all the timing channels. However, even when PLR reaches 5% of the total traffic, the B-TCPScript could still maintain a relatively low p_e of at most 4%, while JitterBug's and IPTime's p_e could be at most 7% and 8%, respectively. Furthermore, we can observe that with a larger T_E or w, all the covert channels attained a lower goodput, but that may not increase their attained accuracy. It is because while increasing T_E or w can reduce the packet rate and thus the chance of packet loss due to the buffer overflow from the network router, it does not prevent packet losses due to random dropping by the router according to the assigned PLR.

On the other hand, we did not observe any error from the LR-TCPScript's decoder for all the PLR settings. However, its channel goodput decreases as PLR increases; it declines further as T_E or T_O increases. In particular, when the PLR reaches 5%, the channel goodput decreases up to 26% and 38% for $M = 4$ and $M = 16$, respectively. The reason of the decline is that the LR-TCPScript's encoder required extra time to perform retransmission to recover the loss. Moreover, comparing with B-TCPScript, LR-TCPScript generally produces relatively lower channel goodput. With PLR = 0%, we attain a similar channel goodput for both types of TCPScript for $M = 4$, but we record almost 9% of decline for $M = 16$. The reason is that LR-TCPScript needs more time to recover lost packets due to full queue caused by large burstiness. With PLR = 5%, we record even further declines of up to 26% and 42% for $M = 4$ and $M = 16$, respectively. This represents the tradeoff between the accuracy and the speed for the covert message delivery.

Table 5.2
Average goodput and average accuracy in terms of Hamming distance for the B-TCPScript, IPTime, and JitterBug obtained from the test bed under different PLRs.

PLR	TCPScript(40ms)	TCPScript(60ms)	JitterBug(40ms)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 4$		
0%	48.89,48.90(1.00,1.00)	32.93,32.94(1.00,1.00)	30.98,30.98(1.00,1.00)
1%	46.64,49.32(0.99,1.00)	32.76,32.89(0.99,1.00)	19.60,19.60(0.63,0.73)
3%	45.09,48.82(0.98,0.99)	29.25,32.78(0.92,1.00)	16.77,16.77(0.54,0.59)
5%	45.46,48.45(0.96,0.99)	32.17,32.50(0.98,0.99)	16.20,16.20(0.52,0.56)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 16$		
0%	97.67,97.77(1.00,1.00)	65.62,65.73(1.00,1.00)	30.52,30.52(1.00,1.00)
1%	97.05,97.52(0.99,1.00)	63.47,65.60(0.97,1.00)	17.83,17.83(0.58,0.66)
3%	94.47,96.19(0.97,0.98)	63.38,65.11(0.97,0.99)	16.49,16.49(0.54,0.58)
5%	89.12,96.98(0.91,0.99)	58.30,64.31(0.89,0.98)	15.97,15.97(0.52,0.55)

PLR	JitterBug(60ms)	IPTime(40ms)	IPTime(60ms)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 4$		
0%	21.09,21.09(1.00,1.00)	13.26,13.26(0.55,0.57)	9.27,9.27(0.57,0.59)
1%	13.25,13.25(0.62,0.74)	13.17,13.17(0.55,0.57)	9.25,9.25(0.57,0.59)
3%	11.94,11.94(0.56,0.62)	13.20,13.20(0.55,0.57)	9.24,9.24(0.56,0.59)
5%	11.16,11.16(0.53,0.57)	13.11,13.11(0.54,0.57)	9.22,9.22(0.56,0.59)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 16$		
0%	20.77,20.77(1.00,1.00)	12.45,12.45(0.52,0.54)	8.57,8.57(0.53,0.54)
1%	12.62,12.62(0.60,0.70)	12.51,12.51(0.52,0.54)	8.58,8.58(0.53,0.54)
3%	11.05,11.05(0.53,0.56)	12.47,12.47(0.52,0.54)	8.57,8.57(0.53,0.54)
5%	10.74,10.74(0.51,0.54)	12.50,12.50(0.52,0.54)	8.63,8.63(0.53,0.55)

Table 5.3
Average goodput and average accuracy in terms of edit distance for the B-TCPScript, IPTime, and JitterBug obtained from the test bed under different PLRs.

PLR	B-TCPScript(40ms)	JitterBug(40ms)	IPTime(40ms)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 4$		
0%	48.89,48.90(1.00,1.00)	30.98,30.98(1.00,1.00)	22.89,22.89(0.95,0.95)
1%	46.64,49.32(0.99,1.00)	30.49,30.49(0.98,0.99)	22.77,22.77(0.94,0.95)
3%	45.09,48.82(0.98,0.99)	29.76,29.76(0.96,0.97)	22.45,22.45(0.93,0.94)
5%	45.82,48.58(0.97,0.99)	29.16,29.16(0.94,0.95)	22.19,22.19(0.92,0.93)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 16$		
0%	97.67,97.77(1.00,1.00)	30.52,30.52(1.00,1.00)	22.78,22.78(0.95,0.95)
1%	97.05,97.52(0.99,1.00)	30.21,30.21(0.99,0.99)	22.64,22.64(0.94,0.95)
3%	94.94,96.11(0.97,0.98)	29.33,29.33(0.96,0.97)	22.40,22.40(0.93,0.94)
5%	94.19,95.65(0.96,0.98)	28.66,28.66(0.94,0.95)	22.20,22.20(0.93,0.93)

PLR	B-TCPScript(60ms)	JitterBug(60ms)	IPTime(60ms)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 4$		
0%	32.93,32.94(1.00,1.00)	21.09,21.09(1.00,1.00)	15.73,15.73(0.96,0.97)
1%	32.76,32.89(0.99,1.00)	20.81,20.81(0.98,0.99)	15.63,15.63(0.96,0.96)
3%	30.68,32.74(0.98,0.99)	20.38,20.38(0.96,0.97)	15.49,15.49(0.95,0.95)
5%	32.17,32.50(0.98,0.99)	19.70,19.70(0.93,0.94)	15.25,15.25(0.93,0.94)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 16$		
0%	65.62,65.73(1.00,1.00)	20.77,20.77(1.00,1.00)	15.66,15.66(0.96,0.97)
1%	64.79,65.46(0.99,1.00)	20.51,20.51(0.99,0.99)	15.58,15.58(0.96,0.96)
3%	64.39,64.98(0.98,0.99)	19.96,19.96(0.96,0.97)	15.43,15.43(0.95,0.95)
5%	63.44,64.33(0.97,0.98)	19.51,19.51(0.94,0.94)	15.25,15.25(0.94,0.94)

Table 5.4
95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) in terms of edit distance for 100 codewords with the LR-TCPScript obtained from the test bed under different PLRs.

	$T_E = 40$ms		$T_E = 60$ms	
PLR	$T_O = 90$ms	$T_O = 120$ms	$T_O = 90$ms	$T_O = 120$ms
	$M = 4$			
0%	48.37,48.73(1.00,1.00)	48.05,48.76(1.00,1.00)	32.41,32.74(1.00,1.00)	32.45,32.72(1.00,1.00)
1%	46.40,47.48(1.00,1.00)	44.66,46.57(1.00,1.00)	31.10,31.74(1.00,1.00)	30.82,31.65(1.00,1.00)
3%	41.94,43.52(1.00,1.00)	39.73,41.53(1.00,1.00)	29.67,30.43(1.00,1.00)	28.21,29.11(1.00,1.00)
5%	38.63,40.04(1.00,1.00)	35.76,37.76(1.00,1.00)	27.60,28.68(1.00,1.00)	26.14,27.67(1.00,1.00)
	$M = 16$			
0%	91.56,94.00(1.00,1.00)	89.28,92.56(1.00,1.00)	62.47,63.59(1.00,1.00)	61.97,63.20(1.00,1.00)
1%	83.71,86.31(1.00,1.00)	79.43,83.05(1.00,1.00)	58.70,60.09(1.00,1.00)	55.99,57.71(1.00,1.00)
3%	71.60,74.73(1.00,1.00)	65.68,70.27(1.00,1.00)	52.08,54.35(1.00,1.00)	49.17,51.41(1.00,1.00)
5%	62.63,66.54(1.00,1.00)	55.53,58.83(1.00,1.00)	47.30,48.88(1.00,1.00)	42.76,44.86(1.00,1.00)

Effect of packet reordering

We have evaluated channels' performance under two different packet reordering scenarios A, B. In each scenario, we configured the router R_1 with different numbers of pipes and entrance probabilities, such that the mean RTT between the encoder and the decoder was equal to 30ms. In other words, each packet entering into R_1 was randomly queued to a pipe that delays the packet for a while before passing the packet to the next hop. Therefore, if a group of packets being forwarded via pipes with different delay, it is very likely that these packets will be reordered. The encoder transmitted the same sets of packet-flow codewords for 30 times, and recorded the average goodput and the average accuracy.

The experiment results are shown in Table 5.5; we adopt the same representation as previous tables. We did not observe any loss from the B-TCPScript experiments under different packet reordering scenarios. Moreover, it has a higher goodput than the other 3 timing channels. As compared with the experiments result of PLR = 0% from the Table 5.3, we observe that B-TCPScript could still maintain similar goodput even during the two reordering scenarios. However, both JitterBug and IPTime had suffered from significant throughput degradation and decoding error rates during the two scenarios. Accordingly, their measured p_e can be up to 21% and 14%, respectively. Compared with previous experiment results, we have recorded maximum measured throughput drops by 21% and 9% for the JitterBug and IPTime, respectively. Since both JitterBug and IPTime rely on packet inter-arrival times for the message encoding/decoding, it is expected that network delay jitter, which alters packet inter-arrival times and probably disrupt the packet orderings, will significantly affect the accuracy of the two channels. On the other hand, B-TCPScript does not directly rely on the packet inter-arrival times, and produces errors only if the network delay jitter could induce packet reordering such that the packet arrived at the decoder will be misclassified as the other bursts. Furthermore, we could not attain any error with the LR-TCPScript from the two scenarios, while the goodput reduction due to packet reordering was only marginal.

5.4.2 PlanetLab experiments

Figure 5.6 reports the measured BERs for the four channels with $T_E = w = \{2\text{RTT}, 2.5\text{RTT}, 3\text{RTT}\}$, where the RTT is the roundtrip time between the PlanetLab node and the decoder. For the LR-TCPScript, we used $T_O = 5\text{RTT}$. We can observe that both B-TCPScript and LR-TCPScript in general attain higher accuracy from all the locations. The corresponding average channel accuracy is 1.040 (or 1.042) times of that of Jitterbug (or IPTime) for the LR-TCPScript, and is 1.032 (or 1.035) times for the B-TCPScript.

Table 5.5
Different channels' $\overline{\zeta}$ and $\overline{\alpha}$ under different reordering scenarios $\mathbb{R}$ ($T_E = w = 40$ms).

$\mathbb{R}$	B-TCPScript	JitterBug	IPTime
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 4$		
A	48.85,48.88(1.00,1.00)	25.05,25.05(0.80,0.82)	21.94,21.94(0.91,0.92)
B	48.87,48.88(1.00,1.00)	24.66,24.66(0.79,0.81)	20.68,20.68(0.86,0.87)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 16$		
A	97.67,97.74(1.00,1.00)	24.71,24.71(0.81,0.82)	21.92,21.92(0.91,0.92)
B	97.54,97.71(1.00,1.00)	24.14,24.14(0.79,0.80)	20.69,20.69(0.86,0.87)

$\mathbb{R}$	LR-TCPScript ($T_O = 90$ms)	LR-TCPScript ($T_O = 120$ms)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 4$	
A	47.79,48.45(1.00,1.00)	47.57,48.40(1.00,1.00)
B	47.70,48.40(1.00,1.00)	47.58,48.38(1.00,1.00)
	95% confidence intervals of $\overline{\zeta}$ ($\overline{\alpha}$) for 100 codewords with $M = 16$	
A	93.65,95.41(1.00,1.00)	92.12,93.89(1.00,1.00)
B	94.84,96.04(1.00,1.00)	92.95,94.76(1.00,1.00)

Besides, LR-TCPScript achieves 100% accuracy for the sites KS, RI, KR, and BE for all the three T_E settings, and no less than 98.8% for the others. We found that the errors observed were induced by the large network delay from the network paths, causing retransmission ambiguity from the encoder side. One scenario is that after retransmitting data packets and receiving the corresponding ACKs from the decoder, the encoder injects the next messages without knowing whether the received ACKs are actually induced by the retransmitted data packets, or by the original ones with delayed arrival at the decoder. If they are due to the latter reason and all retransmitted packets are actually lost, the decoder could misinterpret that the $(i+1)$th burst as the retransmitted part of the ith burst.

For IPTime, Figure 5.6(e) illustrates that in each PlanetLab node a larger w will generally result in a smaller p_e. It is because IPTime depends on strict synchronization

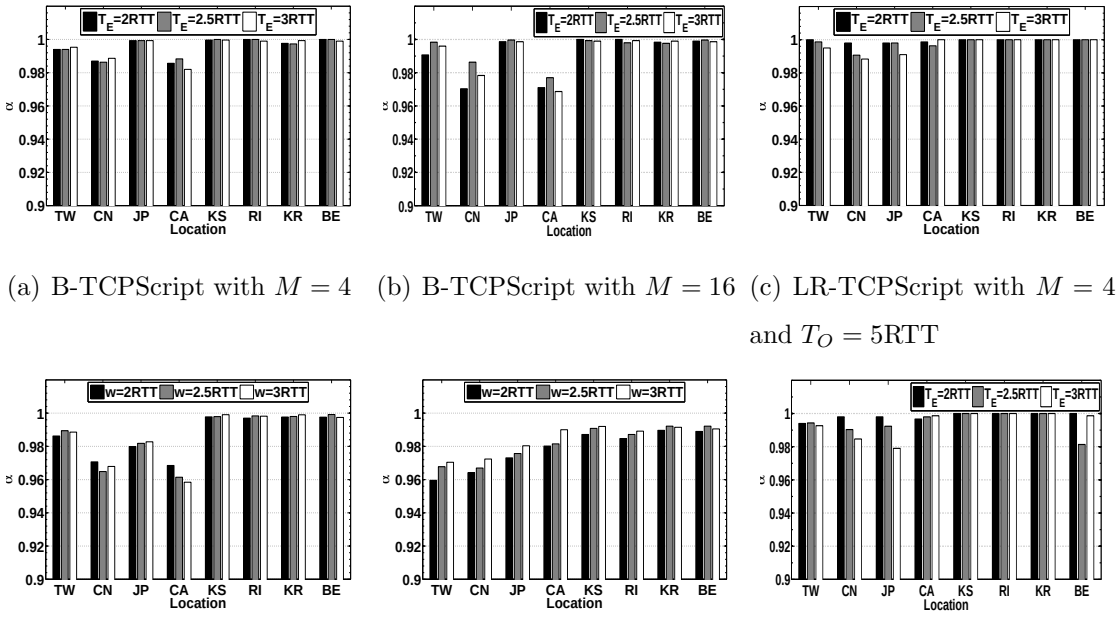
and a larger time interval will mitigate the impact of desynchronization induced by delay jitter. However, we did not observe this relationship in Figure 5.6(a) and Figure 5.6(d) for B-TCPScript and Jitterbug, respectively. Based on network traces, we learnt that the packet loss instead of large delay jitter dominates B-TCPScript's and Jitterbug's errors. Moreover, we found that RTT has no obvious effect on channel accuracy. It is due to the fact that only network delay jitter and network loss, instead of end-to-end network delay, will alter the inter-arrival time and the order of packets arriving at the decoder side.

Furthermore, for a fixed T_E (or w), the results do not exhibit a clear trend between the RTT and the channel performance either. For instance, Figure 5.6(a) shows that B-TCPScript for the site CA yields a relatively lower channel accuracy than other cases, although its RTT is in the mid-range. The reason is due to the fact that accuracy of the 3 channels is mainly based on the network delay jitter and network loss that will alter the time and sequence of packet arrival at the decoder side, but not the end-to-end network delay.

Figure 5.7 shows the performance characterization using the goodput metric. Since both B-TCPScript and LR-TCPScript can transmit multiple bits per RTT, we will expect a higher goodput than the other channels. Besides, we note that for all the sites B-TCPScript attained higher goodput than LR-TCPScript, especially for the sites TW, CN, JP and CA. It is because LR-TCPScript required extra transmission time to perform retransmission during the experiments in those sites due to worse network conditions. This also conforms with our observation from Figure 5.6 that all the channels tended to attain lower average channel accuracy from those sites.

5.5 Detecting TCPScript

In this section, we have proposed two new anomaly indicators and an anomaly-based detection scheme to identify TCPScript channels.



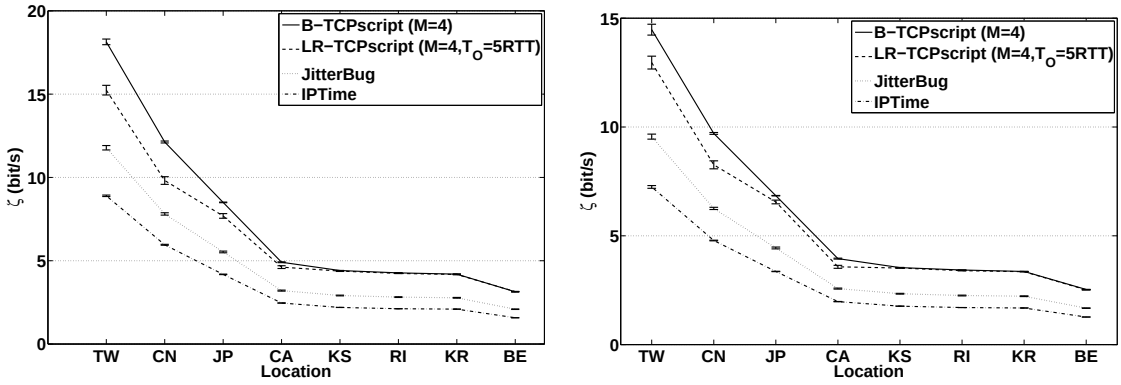
(a) B-TCPScript with $M = 4$ (b) B-TCPScript with $M = 16$ (c) LR-TCPScript with $M = 4$ and $T_O = 5\text{RTT}$

(d) JitterBug (e) IPTime (f) LR-TCPScript with $M = 16$ and $T_O = 5\text{RTT}$

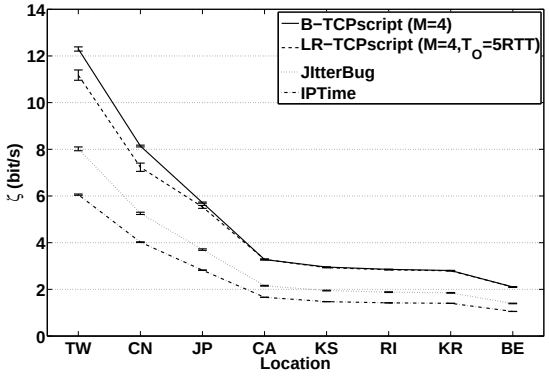
Fig. 5.6. Measured average channel accuracy from PlanetLab nodes.

The burst size The first anomaly indicator is the burst size: the number of TCP data packets in a burst. It is intuitive because the covert messages are encoded into the burst size. A potential approach to detecting TCPScript channels is based on abnormal changes from m_i to m_{i+1}. In the normal TCP traffic, the evolution of the congestion window ($cwnd$), for example, can be modeled as Markov chains [191]. Thus, the neighboring burst sizes in a normal TCP flow are expected to exhibit a certain pattern; however, we do not expect the same for the burst sizes modulated by TCPScript, because the burst sizes depend directly on the message values.

To apply this burst-size detection method, the warden has to first delimit the data packet traces captured for a TCP flow into clusters, each of which corresponds to a packet burst. There are a number of algorithms for the delimiting task, such as using a normalized distance and a threshold, but we will not further elaborate in this thesis. After obtaining a sequence of bursts estimated from a TCP data flow,



(a) $T_E = w = 2\text{RTT}$ (b) $T_E = w = 2.5\text{RTT}$



(c) $T_E = w = 3\text{RTT}$

Fig. 5.7. Measured goodput from PlanetLab nodes.

we use a *detection score* to quantify the possibility that the flow is embedded with a TCPScript channel. Let $\hat{m}_i$ be the size of the ith burst estimated by the warden. Below we propose a set of rules to update the detection score (S):

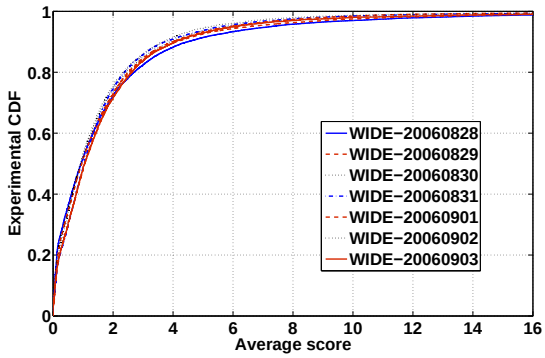
1. If $2\hat{m}_i < \hat{m}_{i+1}$, it signals a possible violation on the aggressive transmission pattern in the slow-start phase. Therefore, $S+ = C_{ss} \times (\hat{m}_{i+1} - \hat{m}_i)$.

2. If $\hat{m}_i + 1 < \hat{m}_{i+1} < 2\hat{m}_i$, it signals a possible violation on the transmission behavior in the congestion avoidance phase. Therefore, $S+ = C_{cc}(\hat{m}_{i+1} - \hat{m}_i)$.

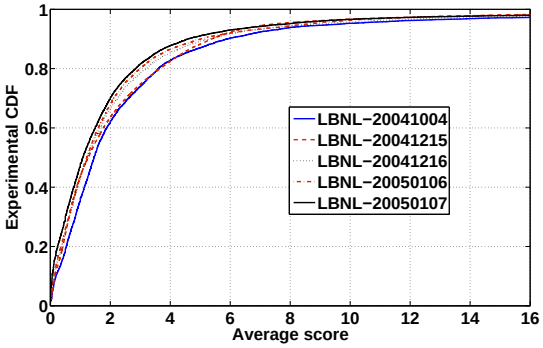
3. If $\frac{\hat{m}_i}{2} < \hat{m}_{i+1} < \hat{m}_i$, it signals a possible violation on the transmission behavior in the loss recovery phrase, because if the first packet in the $\hat{m}_i$-burst is lost, it will trigger $\hat{m}_i - 1$ duplicate packets. These duplicates will increase the *cwnd*

to $\frac{\hat{m}_i}{2} + \hat{m}_i - 1$. If the send window is determined by *cwnd*, the TCP sender can send out at most $\frac{\hat{m}_i}{2}$ packets, including the retransmitted packet. Therefore, $S+ = C_{lr} \times |(\hat{m}_{i+1} - \hat{m}_i)|$.

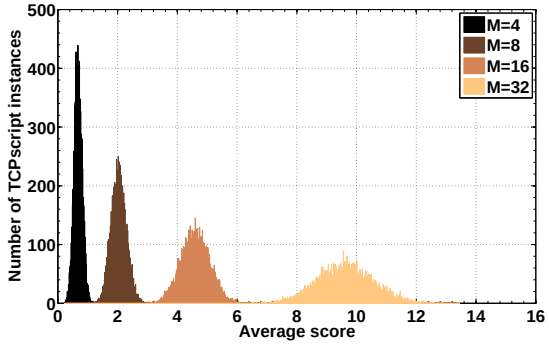
These rules are based on the fact that normal TCP congestion control algorithms, who control the burst size, will obey the general regulations in RFC2581 whereas TCPScript may not follow them. C_{ss}, C_{cc} and C_{lr} are constant weights and are set to 2, 1 and 0.5 respectively to capture more aggressive TCPScript channels, such as using a large M.



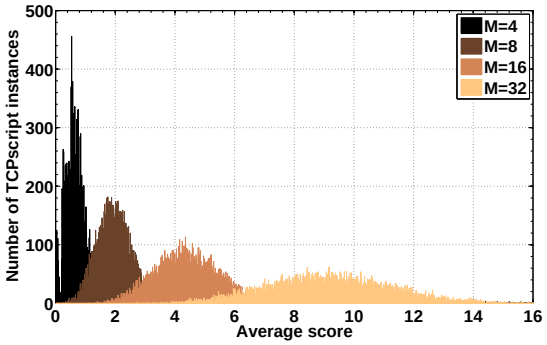
(a) Traces from the WIDE project.



(b) Traces from the LBNL institute.



(c) Simulated TCPScript (each one sends 100 messages).



(d) Simulated TCPScript (each one sends 20 messages

Fig. 5.8. The values $\bar{S}$ computed from three different sets of TCP traces.

Finally, we calculate an average score for this flow: $\bar{S} = \frac{S}{\hat{N}}$, where $\hat{N}$ is the number of bursts detected in the TCP data flow. To evaluate the discriminating power of this

indicator, we have analyzed three sets of packet traces—two sets of Internet traces from the WIDE backbone [192] (around 7.23GB of header traces) and the LBNL's internal enterprise traffic [182] (around 11GB of header traces)—and the third set comprises 10,000 simulated TCPScript traces. The burst sizes for the TCPScript traces are uniformly distributed from $[1, M]$. Figure 5.8 shows the CDFs of $\bar{S}$ for the three sets of traces. Figures 5.8(a) and 5.8(b) show that more than 90% flows captured in WIDE backbone have their $\bar{S}$s less than four, whereas more than 90% flows captured in the LBNL have their $\bar{S}$s less than six. Moreover, the daily traces for both WIDE and LBNL exhibit similar distributions of $\bar{S}$. On the other hand, Figure 5.8(c) (and Figure 5.8(d)) plots $\bar{S}$'s histograms for simulated TCPScript, each of which sends 100 messages (and 20 messages). They show a clear relation between M and $\bar{S}$: a larger M results in an increase in the detection score. On the other hand, they demonstrate that if each TCPScript instance only sends a few messages, then the detection score will decrease. As a result, we expect that the burst-size detection method will be able to uncover aggressive TCPScript channels (e.g., with $M = 16, 32$); however, more cautious channels (e.g., with $M = 4, 8$ or sending a few messages) may escape the detection.

An inter-ACK-data delay The second indicator, denoted as T_{AD}, is the inter-arrival time between an ACK (Pkt_{ACK}) and the first data packet (Pkt_{Data}) following Pkt_{ACK} and having a SN larger than Pkt_{ACK}'s value. That is, the transmission of Pkt_{Data} is very likely triggered by the arrival of Pkt_{ACK}. An example is illustrated in Figure 6.10(a) where $T_{AD} = TW_d - TW_A$. Moreover, if there are other ACKs that arrive within the period between TW_A and TW_d, the warden just ignores them.

The motivation for using T_{AD} is that if the TCP sender has data to send and the receive window is larger than one MSS, then the sender will dispatch new data packets after receiving a new ACK. On the other hand, since the TCPScript encoder transmits another burst of packets only at end of T_E, the resulted T_{AD} will be larger than the normal values. Therefore, we could base on the T_{AD} statistics to detect TCPScript

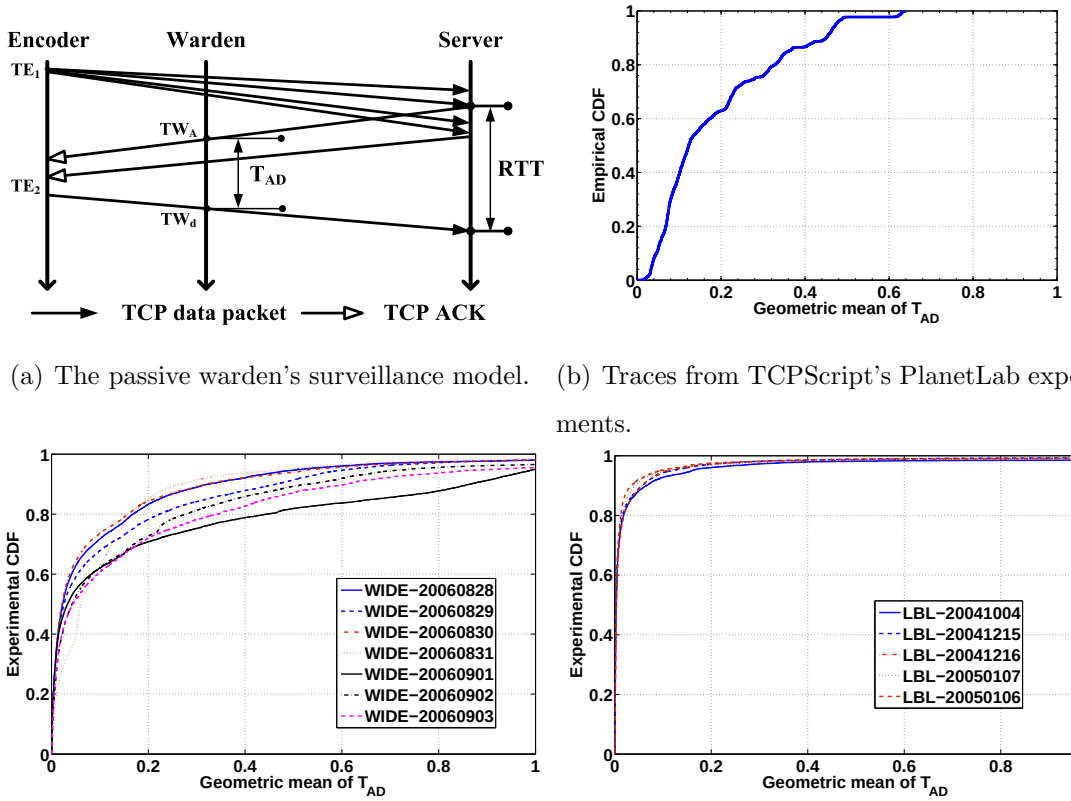
channels. In order to make the difference in the T_{AD} values more outstanding, we use geometric means to measure the T_{AD} samples [193]:

$$GeoM(x_1, x_2, \ldots, x_m) = (\prod_{j=1}^{m} x_j)^{\frac{1}{m}}, \tag{5.9}$$

where x_is are the T_{AD} samples. Thus, a TCPScript encoder is forced to use a small T_E to evade the detection.

To evaluate the discriminating power of this indicator, we again use the two sets of Internet packet traces; we show the computed CDFs of the geometric means in Figure 5.9. Figure 5.9(d) shows that most T_{AD}s in the LBNL traces are very small: more than 90% of the flows have their T_{AD}s less than 0.06 seconds. However, the T_{AD}s computed from the WIDE backbone are much larger: more than 50% of the flows have their T_{AD}s larger than 0.06s. Moreover, the distribution of T_{AD} in the LBNL traces captured on different dates are almost the same; however, there are significant variations in the range of [0.1 1] for different traces. The main reason is most likely due to the warden's location. As shown in Figure 6.10(a), since T_{AD} observed by the warden is part of the RTT between the encoder and the server (it is similar to the RTT between the encoder and the warden.), T_{AD}'s value becomes large when the warden is located far away from the encoder [194]. Since the LBNL traces are mostly internal enterprise traffic, the warden is very close to the encoder.

Figure 5.9(b) shows the CDF for TCPScript's T_{AD} samples obtained from the Internet experiments conducted in the PlanetLab. The figure shows that T_{AD} is usually quite large, since TCPScript has to wait for a period T_E to receive all the ACKs. Thus, if we build a normal profile of T_{AD} based on the LBNL traces, we could easily detect most of the TCPScript channels; however, the same cannot be said for the WIDE traces. This difference between the two sets of traces suggests that this indicator can help detect TCPScript effectively if the warden is closer to the encoder; otherwise, the additional delay induced by the TCPScript encoding cannot be detected from the T_{AD} samples.



(a) The passive warden's surveillance model.

(b) Traces from TCPScript's PlanetLab experiments.

(c) Traces from the WIDE project.

(d) Traces from the LBNL institute.

Fig. 5.9. The experimental CDF of T_{AD}'s geometric mean.

An evaluation of the two indicators We use a simple threshold-based approach [41] to evaluate the performance of the two indicators. To build a normal profile for T_{AD} (or $\bar{S}$), we use the T_{AD} (or $\bar{S}$) samples collected from the LBNL traces that are less than the $P_{T_{AD}}$ (or $P_{\bar{S}}$) quantile, where $P_{T_{AD}}$ (or $P_{\bar{S}}$) is a given parameter. Based on these two sets of data, we compute the mean $\theta_{T_{AD}}$ (or $\theta_{\bar{S}}$) and the standard deviation $\delta_{T_{AD}}$ (or $\delta_{\bar{S}}$) for T_{AD} (or $\bar{S}$). Moreover, we define thresholds for these two indicators as $\Gamma_{T_{AD}} = \theta_{T_{AD}} + d \times \delta_{T_{AD}}$ and $\Gamma_{\bar{S}} = \theta_{\bar{S}} + d \times \delta_{\bar{S}}$, respectively, where d is a parameter for controlling the thresholds. A large d may decrease both the detection rate and the number of false negatives. We have tested different combinations based on $P_{T_{AD}}, P_{\bar{S}} \in [0.8\ 0.85\ 0.9\ 0.95]$, and $d \in [0.5\ 1\ 1.5\ 2]$, and we use these thresholds

to detect TCPScript from the Internet packet traces. The encoders located in the eight PlanetLab nodes send messages to our decoder using $M = \{4, 16\}$ and $T_E = \{1, 1.25, 1.5, 2, 2.5, 3\}RTT$. Here we only tabulate the experiment results from four PlanetLab nodes in Table 5.6, where $P_{T_{AD}} = P_{\bar{S}} = 0.9$ and $d = 1, 1.5, 2$. We show the detection rates for using only $\bar{S}$, using only T_{AD} and using both (denoted by "Both").

Table 5.6
Detection rates under different parameter settings for the PlanetLab packet traces.

		BE(M=4)	BE(M=16)	KR(M=4)	KR(M=16)
$P_{T_{AD}} =$	$\bar{S}$	0	0.9665	0	0.9551
$P_{\bar{S}} = 0.9$,	T_{AD}	0.9222	0.9551	0.8889	0.9157
$d = 1$	Both	0.9222	0.9944	0.8889	0.9888
$P_{T_{AD}} =$	$\bar{S}$	0	0.6034	0	0.6011
$P_{\bar{S}} = 0.9$,	T_{AD}	0.9167	0.9106	0.8722	0.8427
$d = 1.5$	Both	0.9167	0.9497	0.8722	0.9270
$P_{T_{AD}} =$	$\bar{S}$	0	0.1006	0	0.1067
$P_{\bar{S}} = 0.9$,	T_{AD}	0.8778	0.838	0.8167	0.7753
$d = 2$	Both	0.8778	0.8492	0.8167	0.8034
		CA(M=4)	CA(M=16)	CN(M=4)	CN(M=16)
$P_{T_{AD}} =$	$\bar{S}$	0	0.9415	0	0.9663
$P_{\bar{S}} = 0.9$,	T_{AD}	0.838	0.8830	0.8333	0.8258
$d = 1$	Both	0.838	0.9825	0.8333	0.9944
$P_{T_{AD}} =$	$\bar{S}$	0	0.5731	0	0.5955
$P_{\bar{S}} = 0.9$,	T_{AD}	0.8324	0.8304	0.6667	0.6573
$d = 1.5$	Both	0.8324	0.9415	0.6667	0.8596
$P_{T_{AD}} =$	$\bar{S}$	0	0.117	0	0.1067
$P_{\bar{S}} = 0.9$,	T_{AD}	0.7989	0.8187	0.65	0.6461
$d = 2$	Both	0.7989	0.8363	0.65	0.6854

We have found that when $P_{\bar{S}}$, $P_{T_{AD}}$ and d increase, the average detection rate based on a single indicator will decrease, because such settings will relax the limitation on

the TCPScript's behavior. However, the overall detection rate may not decrease, because a union of these two orthogonal indicators' detection ranges does not shrink. Moreover, the T_{AD} indicator is more effective than the $\bar{S}$ indicator for detecting TCPScript in terms of the individual detection rate. For example, $\bar{S}$ could not detect TCPScript when $M = 4$. It corresponds to the conclusions drawn from Figure 5.8. However, neither indicator performs the best in all cases. In fact, the T_{AD} indicator fails to discover TCPScript with T_E close to RTT. Furthermore, we have found that combining them could result in a better detection performance.

5.6 Summary

In this chapter, we have proposed a new approach to designing timing channels that exploit traffic burstiness in reliable, end-to-end protocols. We have detailed the design of TCPScript in particular. By encoding into the bursts of packets, the encoded traffic retains the TCP burstiness patterns. By further exploiting TCP's rich features, TCPScript could increase its reliability and simplify the decoding procedure. We have also derived the information capacity for TCPscript. Furthermore, we have proposed two advanced TCPscript variants for improving the channel reliability. TCPScript does not suffer from traffic normalization, because it does not need to embed information into a TCP packet. Traffic shaping may cause packet losses if TCPScript selects a big M. Consequently, packet losses may decrease TCPScript's goodput. In this case, an encoder may infer the available bandwidth by using techniques like SProbe [185], abget [195], and then select a proper M.

On the defense side, we propose two new anomaly indicators and an anomaly-based detection mechanism to uncover TCPScript channels. We believe that based on these new indicators many existing detection algorithms can be adopted to identify these new covert channels. In the future work, we will evaluate various algorithms, propose new detection algorithms and design new defending mechanisms. A possible approach to defeating TCPScript is to let existing transparent proxy send back ACK

packets when it receives a packet burst and then delay the transmission of this packet burst until the arrival of next packet burst. However, the cost of this approach is high, because the proxy needs a larger buffer for storing two packet bursts for all suspicious flows. Moreover if the proxy uses such approach to handle legitimate flows, the legitimate sender may have a wrong estimation of available bandwidth and send more packets to the proxy that will not remove these packets until it receives ACKs from the remote site. Consequently, the proxy may be overwhelmed.

6. CLOAK: A TEN-FOLD WAY FOR RELIABLE COVERT COMMUNICATIONS

In this chapter, we propose Cloak, a new class of network covert channels that embed information into the combinations of TCP flows and packets. Moreover, we elaborate many design issues, such as the ten pairs of ranking and unranking algorithms and an approach of tackling the head-of-line blocking problem. Similar to TCPScript, we have conducted extensive experiments on our test bed and the PlanetLab platform to evaluate Cloak's performance. To detect Cloak channels, we propose a passive detection scheme and an active detection scheme. The former can easily capture Cloak using unbalanced codewords or aggressive flows, whereas the latter can identify more stealthy channels.

6.1 The basic idea

Network covert channels proposed in the literature so far suffer from low data rates in the presence of noises introduced from two main sources: dynamic network conditions and *active network intermediaries* (ANI) (i.e., the protocol scrubbers [40], traffic normalizer [114], and active wardens [39]). For example, the message encoding based on the inter-packet delay is very sensitive to the delay jitter; a slight change in the network delay could cause decoding errors. Furthermore, packet losses affect the integrity of both timing and storage channels. Packet losses not only affect the decoding accuracy of individual messages, they could also destroy the framing structures, if any. On the other hand, the storage channels do not suffer from this problem. However, the messages encoded in the header fields could be altered by an ANI.

In this chapter, we propose *Cloak*—a new class of timing channels which is designed to be reliable under all network conditions. That is, Cloak's decoding accuracy

is 100% even in the presence of packet losses, delay jitters, packet reordering, and packet duplications. The key elements responsible for this reliability property are using TCP data traffic as a cover (i.e., exploiting TCP's reliable transmission mechanism) and a fixed number of TCP packets (N) for encoding/decoding a message in order to avoid synchronization errors.

Another important deviation from other timing channels is that Cloak encodes a message with a unique distribution of N packets over X TCP flows, where $N, X > 1$. To our best knowledge, Cloak is the *first* network covert channel that exploits the enumerative combinatorics [196] to convey hidden messages. Moreover, the idea is general enough to be used for designing new covert channels and other steganography fields. Due to the combinatorial nature of the encoding method, Cloak's channel capacity increases quickly with (N, X). Besides, Cloak offers ten different encoding and decoding methods. Each method tradeoffs among several conflicting design goals. Although Cloak uses multiple flows for message encoding, the packet distribution over the flows can be carefully crafted to match with normal TCP behavior in an application session.

6.1.1 Encoding based on packet-flow distributions

The covert messages in Cloak are encoded by a class of combinatorial objects—each covert message is encoded with a unique distribution of N TCP packets over X TCP flows. The encoder and decoder agree on the values of N and X beforehand. Consider a simple example of $N = 5$ and $X = 3$. If the encoder and decoder could distinguish the three TCP flows, then there are a total of 21 possible ways of distributing the five packets over the three flows. Table 6.1 shows an example of encoding four-bit messages using 16 of the 21 combinations, where n_i is the number of packets sent on ith flow. Furthermore, the encoder will transmit the next message only after receiving the ACKs for the N TCP packets. On the other side of the channel, the decoder starts decoding as soon as collecting N TCP packets from the

encoder. Moreover, the encoder and decoder do not have to explicitly exchange the "codebook"; instead, the encoding and decoding can be performed using unranking and ranking functions.

Cloak is reliable in the same sense of reliability in TCP even when the messages experience packet losses, delay jitter, and other adverse conditions. First of all, Cloak's decoding accuracy is not affected by delay jitters, because the encoding is not based on the actual time. Second, since the encoder sends a covert message one at a time, it can detect whether the decoder has successfully received the last message based on the ACKs for the N TCP packets. Upon detecting an unsuccessful reception, the encoder could "partially" resend the message by retransmitting the unacknowledged TCP packets. The decoder, on the other hand, will decode only after receiving N in-sequenced TCP packets from the encoder. Therefore, if Cloak is implemented using the normal TCP stack, no additional reliability mechanism is needed to guarantee Cloak's reliability.

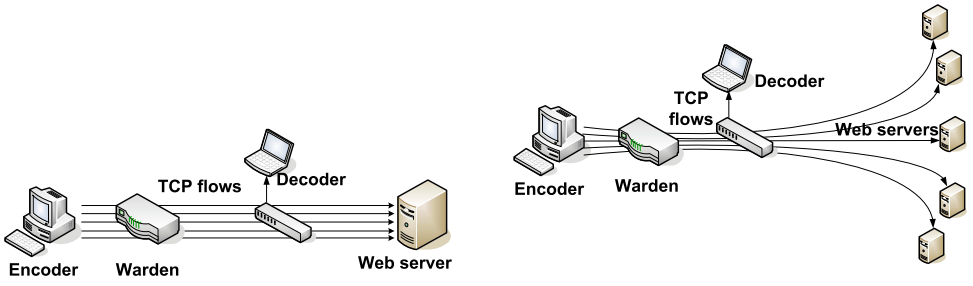
Table 6.1
An example of encoding a hexadecimal number using $N = 5$ and $X = 3$ in Cloak.

Message	n_1	n_2	n_3	Message	n_1	n_2	n_3	Message	n_1	n_2	n_3	Message	n_1	n_2	n_3
0	5	0	0	4	3	1	1	8	2	1	2	12	1	2	2
1	4	1	0	5	3	0	2	9	2	0	3	13	1	1	3
2	4	0	1	6	2	3	0	10	1	4	0	14	1	0	4
3	3	2	0	7	2	2	1	11	1	3	1	15	0	5	0

In Figure 6.1, we depict two different scenarios for the Cloak encoder and decoder to communicate. In both cases, we assume a warden on the encoder's network who guards against any network covert channels initiated from inside. The warden could be active or passive. A passive warden attempts to detect network covert channels by

analyzing all the traffic sent between the encoder and any hosts outside the network. The passive detection does not alter the traffic flows and characteristics. An active warden, on the other, attempts to interfere with any network covert channels passing through it. The strategies usually involve altering the packet contents or the traffic characteristics, such as delaying and dropping packets.

In the first scenario (Figure 6.1(a)), the encoder could establish a "normal" HTTP session with a remote server which consists of five TCP flows. The encoder encodes the messages into the TCP flows, and the decoder eavesdrops at any point of the path and decodes the messages. In order to evade a passive warden, the encoder may distribute the TCP packets over the flows such that they match with the normal TCP behavior. Moreover, the warden could not detect Cloak simply based on the presence of multiple TCP flows to the same server, because it is not uncommon to have multiple TCP flows in an HTTP session. In the second scenario (Figure 6.1(b)), the encoder could establish normal HTTP sessions with multiple servers which are dispersed in different locations. Therefore, the decoder should be located on the common routing path for all the servers. Although this approach restricts the decoder location, it can diffuse the relationship among the TCP flows.



(a) The five TCP flows connect to the same Web server.

(b) The five TCP flows connect to different Web servers.

Fig. 6.1. Two covert communication scenarios between Cloak encoder and decoder.

6.1.2 The Twelvefold Way

Besides the encoding algorithm just described, Cloak could also admit other encoding methods. In fact, Cloak offers ten different encoding methods which are based on the well-known *Twelvefold Way* [196] in the field of Enumerative Combinatorics. The Twelvefold Way refers to twelve basic counting problems that count all the possible ways of putting N balls into X urns, and their results. Let the set of balls be $\mathbb{N}$ ($|\mathbb{N}| = N$) and the set of urns be $\mathbb{X}$ ($|\mathbb{X}| = X$). Each problem can be based on whether the balls and urns are distinguishable or not (e.g., by their colors), and three possible kinds of ball distributions over the urns: (1) no restriction, (2) at most one ball per urn, and (3) at least one ball per urn. These three cases can be equivalently represented by an arbitrary function $\mathbf{f}_A : \mathbb{N} \rightarrow \mathbb{X}$, an injective function $\mathbf{f}_I : \mathbb{N} \rightarrow \mathbb{X}$, and a surjective function $\mathbf{f}_S : \mathbb{N} \rightarrow \mathbb{X}$, respectively.

The correspondence between balls and urns, and packets and flows is obvious. Table 6.2 summarizes the Twelvefold Way using flows (urns) and packets (balls) [196]. Each of the twelve results answers the corresponding counting problem (i.e., the total number of unique packet-flow distributions). Moreover, cases (11) and (12) obviously cannot be used in Cloak, therefore the ten encoding methods. We refer the ten cases to as Cloak$^c(N, X)$, $c = [1, 10]$.

According to Table 6.2, some encoding methods require distinguishable packets and/or distinguishable flows. The correspondence between the ball and urn distinguishability, and the flow and packet distinguishability is somewhat tricky. First of all, all TCP flows and packets are of course distinguishable. However, the original counting problems assume that the colors of the urns and balls do not change, but this is not the case for Cloak. For instance, the "marking information" in the flows and packets could be altered by an ANI. Therefore, the TCP flows (or packets) are considered distinguishable only if both encoder and decoder are able to identify the same flow (or packet).

Table 6.2
The Twelvefold Way and their relation to the ten (items 1-10) encoding methods in Cloak.

Elements of $\mathbb{N}$ (TCP packets)	Elements of $\mathbb{X}$ (TCP flows)	$\mathbf{f}_A$ (no restriction)	$\mathbf{f}_I$ (at most 1 packet in a flow)	$\mathbf{f}_S$ (at least 1 packet in a flow)
Distinguishable	Distinguishable	X^N (1)	$N!C_X^N$ (2)	$X!S(N,X)$ (3)
Indistinguishable	Distinguishable	C_{N+X-1}^{X-1} (4)	C_X^N (5)	C_{N-1}^{X-1} (6)
Distinguishable	Indistinguishable	$\sum_{i=1}^{X} S(N,i)$ (7)	$\begin{cases} 1 & \text{if } N \leq X \\ 0 & \text{if } N > X \end{cases}$ (11*)	$S(N,X)$ (8)
Indistinguishable	Indistinguishable	$\sum_{i=1}^{X} P(N,i)$ (9)	$\begin{cases} 1 & \text{if } N \leq X \\ 0 & \text{if } N > X \end{cases}$ (12*)	$P(N,X)$ (10)

where

– $C_X^N = \frac{X!}{N!(X-N)!}$ and $S(N,X) = \frac{1}{X!}\sum_{j=1}^{X}(-1)^{X-j}C_X^j j^N$.

– $P(N,X)$ is the number of partitions of N into X parts, which can be solved using recursion.

6.1.3 Concise proofs for the Twelvefold Way

In this section, we briefly explanation the derivation of the results in Table 6.2. Detail information can be found in [196]. To simplify the explanation, we use balls and urns to represent TCP packets and TCP flows, respectively.

(1) Since each ball could be put into any urn and therefore has X choices, the total number of combinations is X^N.

(2) If $N > X$, the number is 0. Otherwise, the encoder could first select N balls from X balls and then permutate them because the balls are distinguishable. Hence, the result is $N!C_X^N$, $N \leq X$.

(3) Partitioning a set of N distinguishable balls into X parts has $S(N, X)$ different combinations. Since urns are also distinguishable, the encoder could further permutate them and get the final result as $X!S(N, X)$. We will explain $S(N, X)$ in (8).

(4) The difference between this problem and the 6th problem is whether an urn could be empty. Let $n_i \geq 0$ ($\sum_{i=1}^{X} n_i = N$) denote the number of balls in ith urn. If we let $m_i = n_i + 1$,$i = 1, \ldots, X$, then the number of m_i' combinations is equal to C_{N+X-1}^{X-1}. Since m_i and n_i are bijective, C_{N+X-1}^{X-1} is the solution to this problem.

(5) If $N > X$, the number is 0; otherwise, the encoder selects N urns from X potential ones to accommodate a ball. Hence, the result is C_X^N.

(6) Laying out N balls in a row, the encoder could select $X - 1$ spaces from $N - 1$ spaces between two balls and then put delimiters to divide the balls into X urns. Therefore, the result is C_{N-1}^{X-1}.

(7) Since the encoder could partition a set of N distinguishable balls into i, $i = 1, \ldots, X$ parts, the result is $\sum_{i=1}^{X} S(N, i)$.

(8) The encoder has two methods to partition the balls. First, it could put one ball in an urn and partition left $N - 1$ balls into the remaining $X - 1$ urns. Second, it could put $N - 1$ balls into X urns and then put the remaining ball to one of X urns. Therefore, we obtain $S(N, X) = S(N-1, X-1) + XS(N-1, X)$ with initial conditions: $S(N, 0) = 0$, $S(N, 1) = 1$, $S(N, 2) = 2^{N-1} - 1$, and other features (i.e., $S(N, N) = 1$, $S(N, N-1) = C_N^2$, and $S(N, X) = 0$ if $X > N$).

(9) Since the encoder could partition N into i parts, where $i = 1, \ldots, X$, the result is $\sum_{i=1}^{X} P(N, i)$. Note that $P(N, X) \neq S(N, X)$. We will explain $P(N, X)$ in (10).

(10) This is a classic integer partition problem. The encoder could partition the balls using two methods. First, it could let one urn contain only one ball and partition the remaining $N-1$ balls into the remaining $X-1$ urns. Second, it could let each urn have one ball and then partition the remaining $N-X$ balls into X urns. Therefore, we have $P(N,X) = P(N-1,X-1) + P(N-X,X)$ with initial conditions: $P(N,1) = 1$ and condition $P(N,N) = 1$.

(11) If $N > X$, the solution is 0. Otherwise, the result is 1, because each urn could hold at most one ball, and these urns are indistinguishable (i.e., exchanging balls between urns does not increase the number of possible combinations).

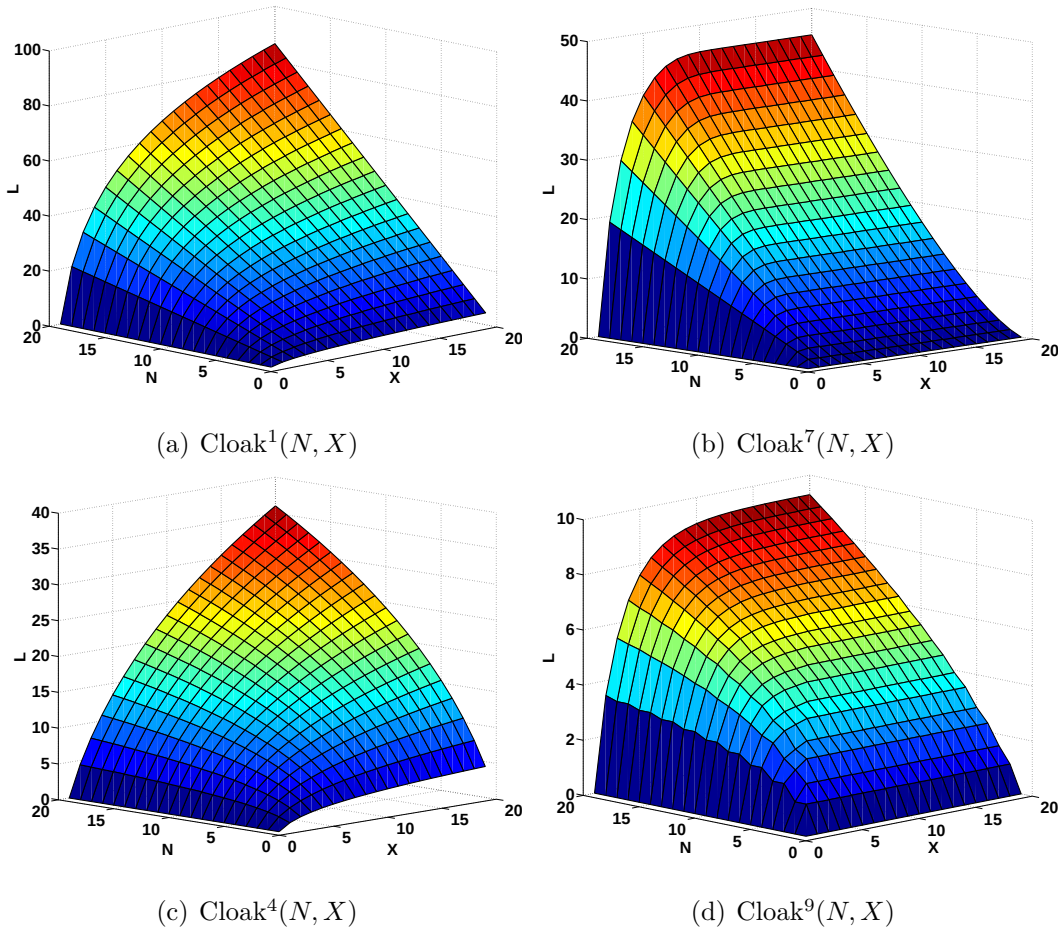
(12) If $N > X$, the solution is 0. Otherwise, the solution is 1, because each urn could hold at most one ball, and both urns and balls are indistinguishable.

6.1.4 The ten-fold way in Cloak

In this section, we discuss the differences among the ten encoding methods and explain why we need all of them. The first important difference among them is their channel capacity. By modeling a Cloak channel as a classical information channel, we can obtain the capacity of a Cloak$^c(N,X)$ channel in bits/symbol based on mutual information [123]. Since Cloak is reliable and there is only one set of covert messages, the channel capacity can be increased only by increasing the size of the covert message set. By denoting the Twelvefold Way result for Cloak$^c(N,X)$ by $T^c(N,X)$, a higher value of $T^c(N,X)$ therefore gives a higher channel capacity. Furthermore, each unique packet-flow distribution can encode an L-bit word, where $1 \leq L \leq \lfloor \log_2 T^c(N,X) \rfloor$.

In the following, we explain the relationship between channel capacity and flow and packet distinguishability. First, making the flows distinguishable increases the channel capacity (e.g., $T^1(N,X) > T^7(N,X)$ and $T^4(N,X) > T^9(N,X)$). Similarly, making the packets distinguishable also increases the channel capacity (e.g., $T^1(N,X) > T^4(N,X)$ and $T^7(N,X) > T^9(N,X)$). Based on the last two items, it is easy to obtain that $T^1(N,X) > T^7(N,X) > T^4(N,X) > T^9(N,X)$. Therefore,

packet distinguishability increases the channel capacity faster than the channel distinguishability does (e.g., $T^7(N,X) > T^4(N,X)$). We have charted the values of Ls in Figure 6.2. Note their differences on how their Ls increase with N and X. Finally, for each row in Table 6.2, the channel capacity for $\mathbf{f}_A$ is the largest (e.g., $T^1(N,X) > T^3(N,X)$ and $T^7(N,X) > T^8(N,X)$). Based on the channel capacity, we define *data rate* in bits per second as $\frac{C}{T_s}$, where T_s is the time for transmitting a message. The minimal time for transmitting a message in Cloak (i.e., N packets in X flows) is one RTT between the encoder and decoder. To achieve a reasonable channel capacity, we consider $X > 1$ and $N > 1$ in the rest of this thesis.



(a) Cloak$^1(N,X)$ (b) Cloak$^7(N,X)$

(c) Cloak$^4(N,X)$ (d) Cloak$^9(N,X)$

Fig. 6.2. Maximum values of L for Cloak$^c(N,X)$, $c = 1, 7, 4, 9$.

Besides the channel capacity, the ten encoding methods differ also in three other important aspects. The first one concerns the channels that require distinguishable packets (i.e., $c = 1, 3, 7, 8$). For these channels, the encoder usually adds "markers" to the TCP packets in order to make them distinguishable. The additional markers, however, could be "modified" when the packets traverse an active warden, which could result in decoding errors. In other words, there is a tradeoff between achieving a higher channel capacity by making the packets distinguishable and the decoding accuracy. Similar problems occur also to the channels with flow distinguishability.

The second one is connected to a head-of-line blocking (HoLB) problem. To explain the issue, consider $c = 1, 2$; the difference between them is that the second method caps the number of packets distributed to a flow to one. Therefore, in terms of the packet distribution, the flows for $c = 2$ differ at most by one packet, but that for $c = 1$ is N (i.e., all the packets could be distributed to a single flow). The latter case may require several RTTs to complete a message's transmission; thus, this HoLB problem, as we shall see later, could reduce the actual data rate very significantly.

The last issue is that some flows for the methods under $\mathbf{f}_A$ and $\mathbf{f}_I$ may become idle for a prolonged period of time, which may cause the remote servers to close the connection. However, those methods under $\mathbf{f_S}$ mitigate this problem by insisting each flow to carry at least one packet for each message. We will discuss how to tackle these issues in section 6.3.

6.2 Ranking and unranking algorithms

Both `Rank()` and `Unrank()` have to be computationally efficient. Therefore, a table-lookup approach will not work because of the potentially huge packet-flow encoding space. Instead, we have designed efficient ranking and unranking algorithms for the ten encoding methods. Our strategy is to first design the algorithms for five primitive components in $T^c(N, X)$: X^N, C_X^N, $S(N, X)$, $P(N, X)$, and $\lambda!$, where $\lambda = N, X$. We employ the existing ranking and unranking functions for the last four

components: C_X^N [197], $S(N,X)$ [198], $P(N,X)$ [197], and $\lambda!$ [199]. Therefore, we can directly apply these algorithms to $c = 5, 8, 10$.

6.2.1 Algorithms for $c = 1$

We start by observing that an integer R, where $0 \leq R \leq 2^L - 1$ and $L = \lfloor \log_2(X^N) \rfloor$, has a unique representation of [200]:

$$R = a_0 X^0 + a_1 X^1 + \cdots + a_{N-1} X^{N-1}, \quad a_i \in \{0, \ldots, X-1\}. \tag{6.1}$$

Consider that R is the rank, N is the total number of TCP packets, and X is the number of TCP flows. To unrank R, the idea is to covert R into the form in Eq. (6.1) and then to place a packet pkt_j, $0 \leq j < N$, into flow k, $0 \leq k < X$, if and only if $a_j = k$. To rank a given packet-flow distribution specified in $\Lambda[i]$, $0 \leq i < X$, we compute the right hand side of Eq. (6.1). The detailed algorithm is given in **Function 2** in which Idx_{pkt} is the index of packet pkt, and $\Lambda[i]$ contains the indices of the packets sent on flow i. To rank a given packet-flow distribution specified in Λ, we compute the right hand side of Eq. (6.1). That is, the ranking algorithm, as shown in **Function 1**, basically converts a base X number into a nonnegative decimal number.

Function 1 Rank1(Λ, X) : *integer*

```
R ← 0;
for i = 0 to X − 1 do
    for j = 0 to Size(Λ[i]) − 1 do
        R ← X^{Λ[i].get(j)} × i + R;
    end for
end for
return R;
```

Function 2 Unrank1(R, X) : *integer*[]

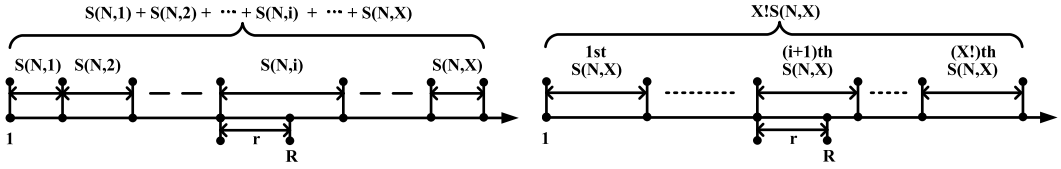
```
Idx_pkt ← 0;
while R ≠ 0 do
    i ← R mod X;
    Λ[i].add(Idx_pkt);
    R ← (R − i)/X;
    Idx_pkt ← Idx_pkt + 1;
end while
return Λ;
```

6.2.2 A general framework

As for the remaining methods ($c = 2, 3, 4, 6, 7, 9$), we are not aware of any ranking and unranking functions designed for them. Therefore, we propose a general framework that employs the divide-and-conquer strategy to design new ranking/unranking algorithms.

Based on Table 6.2, we divide $T^c(N, X)$, $c = 2, 3, 4, 6, 7, 9$, into three groups. The first group consists of $T^4(N, X)$, $T^5(N, X)$ and $T^6(N, X)$ which are very similar to C^μ_λ. The second group consists of $T^7(N, X)$ and $T^9(N, X)$ which are a sum of $S(\lambda, i)$ or $P(\lambda, i)$, $i = 1, \ldots, \mu$. The third group consists of $T^2(N, X)$ and $T^3(N, X)$. $T^2(N, X)$ is the product of $\lambda!$ and C^μ_λ, and $T^3(N, X)$ is the product of $\lambda!$ and $S(\lambda, \mu)$.

We could construct ranking/unranking algorithms for the first group of Cloak channels by substituting λ and μ with corresponding parameters. Then we get Cloak[5]'s ranking/unranking algorithms by substituting λ in C^μ_λ with X and its μ with N. Cloak[6]'s ranking/unranking algorithms are obtained by replacing λ in C^μ_λ with $N - 1$ and its μ with $X - 1$. Cloak[4]'s ranking/unranking algorithms could be derived from Cloak[6]'s algorithms by replacing its $N - 1$ with $N + X - 1$.



(a) Cloak[7]. (b) Cloak[3].

Fig. 6.3. Examples of the general framework for the design of new ranking/unranking algorithms.

For the second group of Cloak channels, we use Cloak[7] as an example to demonstrate how to design new ranking/unranking algorithms based on the basic model's algorithms. Figure 6.3(a) illustrates the result for Cloak[7] (i.e., $\sum_{i=1}^{X} S(N, i)$). To encode a value R, Cloak[7]'s unranking algorithm will first determine the number of

connections (say i) to be used by subtracting $\sum_{j=1}^{i-1} S(N,j)$ from R until the left value, say r, is not larger than $S(N,i)$. It then determines how to allocate the N distinguishable packets into i flows through $S(N,i)$'s unranking algorithm with input r. After observing N packets, the decoder will first restore r through $S(N,i)$'s ranking algorithm and then obtain the original message $R = \sum_{j=1}^{i-1} S(N,j) + r$. Using a similar approach, we could construct ranking/unranking algorithms for Cloak^9.

For the third group of Cloak channels, we use Cloak^3 as an example. Figure 6.3(b) illustrates the result for Cloak^3 (i.e., $X!S(N,X)$). To encode a value R, Cloak^3's unranking algorithm will first compute $r = R \; mod \; S(N,X)$ and $i = \frac{R-r}{S(N,X)}$. The encoder will then determine how to allocate N distinguishable packets into X flows by using $S(N,X)$'s unranking algorithm with input r. Next, the encoder will determine how to permute these flows according to a permutation of X elements, which is obtained through $\lambda!$'s unranking algorithm with input i. A number of ranking and unranking functions for permutation are available, and the time complexity of traditional unranking algorithms is $\mathcal{O}(n^2)$ [201] (where n is the number of elements in a permutation). Here we employ new ranking (**Function 3**) and unranking algorithms (**Function 4**) with time complexity $\mathcal{O}(n)$ [199]. Note that P^R is the reverse order of P. Finally, the encoder will permute Λ's elements and then transmit these N distinguishable packets through X distinguishable flows. On the decoder side, it will first compute i by using $\lambda!$'s ranking algorithm and then calculate r through $S(N,X)$'s ranking algorithm. Finally, it will restore the original message $R = iS(N,X) + r$. A similar approach could be used to design ranking and unranking algorithms for Cloak^2.

Function 3 RankPermute(N, P, P^R) : *integer*

if $N == 1$ **then**
  return 0 ;
end if
$s \leftarrow P[N-1]$;
$Swap(P[N-1], P[P^R[N-1]])$;
$Swap(P^R[s], P^R[N-1])$;
return $(s + N \times RankPermute(N-1, P, \bar{P}))$;

Function 4 UnrankPermute (R, N, P) :

if $N > 0$ **then**
  $tmp \leftarrow P[N-1]$;
  $P[N-1] \leftarrow P[r \bmod N]$;
  $P[r \bmod N] \leftarrow tmp$;
  $UnrankPermute(\lfloor r/N \rfloor, N-1, P)$;
end if

6.2.3 Algorithms for $c = 4, 5, 6$

Let C_λ^μ be the number of ways of selecting μ elements, denoted as $\Lambda = [\lambda_1, \ldots, \lambda_\mu]$, from a set having λ elements. We design Cloak^5's ranking/unranking algorithms by substituting C_λ^μ's λ with X and its μ with N. The output of the unranking algorithm with the input R is the set of TCP flows that will be used to transmit packets (i.e., λ_i contains a selected TCP flow's index, $\lambda_i \in \{1, \ldots, X\}$). After observing these distinguishable flows, the decoder could construct a set of N elements and obtain the message R through C_X^N's ranking algorithm. Without loss of generality, we arrange Λ according to the following order:

$$\lambda_1 < \lambda_2 < \ldots < \lambda_\mu, \quad let \quad \lambda_0 = 0.$$

Since $\mu = N$ and $\lambda = X$ for Cloak^5, we will use N and X to describe the ranking and unranking algorithms. To rank an *N-element* subset, we count the number of *N-element* subsets preceding the given one in the lexicographic order, which is shown in Eq. (6.2) [197].

$$Rank(\Lambda) = \sum_{i=1}^{N} \sum_{j=\lambda_{i-1}+1}^{\lambda_i - 1} C_{X-j}^{N-i}. \tag{6.2}$$

According to Eq. (6.2), we can obtain an Λ by unranking an integer value R as follows [197]:

$$\begin{aligned}
\lambda_1 &= n \Leftrightarrow \sum_{j=1}^{n-1} C_{X-j}^{N-1} \leq R < \sum_{j=1}^{n} C_{X-j}^{N-1} \\
\lambda_i &= n \Leftrightarrow \sum_{j=\lambda_{i-1}+1}^{n-1} C_{X-j}^{N-i} \leq R - \sum_{j=1}^{\lambda_{i-1}-1} C_{X-j}^{N-i} < \sum_{j=\lambda_{i-1}+1}^{n} C_{X-j}^{N-i}.
\end{aligned}$$

Based on the `kSUBSETLEXRANK` and `kSUBSETLEXUNRANK` algorithms in [197], Cloak[5]'s ranking and unranking algorithms are shown in **Function 5** and **Function 6**, respectively.

Function 5 $Rank5(\Lambda, N, X) : integer$

```
R ← 0;
Λ[0] ← 0
for i = 1 to X do
  if Λ[i − 1] + 1 ≤ Λ[i] − 1 then
    for j = Λ[i − 1] + 1 to Λ[i] − 1
    do
      R ← R + C_{X−j}^{N−i};
    end for
  end if
end for
return R;
```

Function 6 $Unrank5(R, N, X)$: $integer[]$

```
j ← 1;
for i = 1 to N do
  while C_{X−j}^{N−i} ≤ R do
    R ← R − C_{X−j}^{N−i};
    j ← j + 1;
  end while
  Λ[i] ← j;
  j ← j + 1;
end for
return Λ;
```

Cloak[6]'s ranking and unranking algorithms are obtained by replacing C_λ^μ's λ with $N-1$ and its μ with $X-1$. Recall the basic idea of counting Cloak[6]: laying out N balls in a row, the encoder could select $X-1$ out of the $N-1$ spaces between two balls and then put delimiters to divide the balls into X urns. Each λ_i contains the number of packets to be sent by the ith flow ($1 \leq i \leq X$). A sequence of auxiliary variables, named $\overline{\Lambda} = [\overline{\lambda}_1, \ldots, \overline{\lambda}_{X-1}]$, hold the indices of "space" mentioned in the basic idea. To encode message R, the encoder will let the ith TCP flow transmit

λ_i packets (let $\lambda_0 = 0$). The decoder will generate a set of $X-1$ elements (i.e., $\overline{\Lambda}$) and restore the message R through C_{N-1}^{X-1}'s ranking algorithm. Cloak^6's ranking and unranking algorithms are listed in **Function 7** and **Function 8**, respectively.

Function 7 $Rank6(\Lambda, N, X)$: *integer*

$\overline{\Lambda}[0] \leftarrow 0$;
$\overline{\Lambda}[1] \leftarrow \Lambda[1]$;
for $i = 2$ to $X-1$ **do**
  $\overline{\Lambda}[i] \leftarrow \Lambda[i] + \overline{\Lambda}[i-1]$;
end for
$R \leftarrow Rank5(\overline{\Lambda}, X-1, N-1)$;
return R;

Function 8 $Unrank6(R, N, X)$: *integer*[]

$\overline{\Lambda} \leftarrow Unrank5(R, X-1, N-1)$;
$\Lambda[0] \leftarrow 0$;
for $i = 1$ to $X-1$ **do**
  $\Lambda[i] \leftarrow \overline{\Lambda}[i] - \overline{\Lambda}[i-1]$;
end for
$\Lambda[X] \leftarrow N - \overline{\Lambda}[X-1]$;
return Λ;

Cloak^4's ranking/unranking algorithms could be derived from Cloak^6's algorithms by replacing $N-1$ with $N+X-1$. We use λ_i ($\lambda_i \geq 0$ and $\sum_{i=1}^{X} \lambda_i = N$) to denote the number of packets transmitted by ith flow and let $\widehat{\lambda}_i = \lambda_i + 1$ (i.e. $\sum_{i=1}^{X} \widehat{\lambda}_i = N+X$). To design Cloak^4's unranking algorithm, we first obtain the unranking result of Cloak^6(N+X,X), denoted as $\widehat{\Lambda} = [\widehat{\lambda}_1, \ldots, \widehat{\lambda}_X]$, and then obtain Cloak^4(N,X)'s result as $\Lambda = [\widehat{\lambda}_1 - 1, \ldots, \widehat{\lambda}_X - 1]$. Finally, the encoder will use the ith TCP flow to transmit λ_i packets if $\lambda_i > 0$. After observing the packets, the decoder could generate a set of $X-1$ elements and increase each value by 1 before inputting it to C_{N+X-1}^{X-1}'s ranking algorithm. Based on Cloak$^6(N, X)$'s ranking and unranking algorithms, we therefore the algorithms for Cloak$^4(N, X)$ in **Function 9** and **Function 10**.

Function 9 $Rank4(\Lambda, N, X)$: *integer*

for $i = 1$ to X **do**

$\Lambda[i] \leftarrow \Lambda[i] + 1$;

end for

$R \leftarrow Rank6(\Lambda, X-1, N+X-1)$;

return R;

Function 10 $Unrank4(R, N, X)$: *integer*[]

$\Lambda \leftarrow Unrank6(R, X-1, N+X-1)$;

for $i = 1$ to X **do**

$\Lambda[i] \leftarrow \Lambda[i] - 1$;

end for

return Λ;

6.2.4 Algorithms for $c = 7, 8$

Recall that $S(\lambda, \mu)$ is the number of different ways of partitioning a set with λ distinguishable elements into μ parts. Denote an arrangement as $\Psi = [\psi_1, \ldots, \psi_\lambda]$, $\psi_i \in \{1, \ldots, \mu\}$, which means that the ith element will be put into the ψ_ith set. Cloak[8]'s ranking and unranking algorithms are built by superseding $S(\lambda, \mu)$'s λ with N and its μ with X. After attaining Ψ through $S(N, X)$'s unranking algorithm, the encoder will use the ψ_ith TCP flow to send ith packet. After observing N packets, the decoder could easily construct Ψ and then input it to $S(N, X)$'s ranking algorithm to recover the message. Moreover, $S(N, X)$ can be defined recursively according to Eq. (6.3) which forms the basis for designing the ranking and unranking algorithms.

$$S(N, X) = X \times S(N-1, X) + S(N-1, X-1) \tag{6.3}$$

The Ranking and Unranking algorithms shown in **Function 11** and **Function 12** are based on the `RankSetPtns` and `UnrankSetPtns` functions from [198]. Note that for the ease of programming, the output of **Function 12** has the form: $\widetilde{\Psi} = [\widetilde{\psi}_0, \ldots, \widetilde{\psi}_{\lambda-1}]$, $\widetilde{\psi}_j \in \{1, \ldots, \mu\}, 0 \leq j < \lambda - 1$ (i.e., the subscript j starts from 0 and $\widetilde{\psi}_j = \psi_{j+1}$).

Function 11 $Rank8(\Psi, N, X)$: *integer*

if $(N == X)|(X == 0)$ **then**
  return 0;
else
  for $i = 0$ to $X - 2$ **do**
    $\overline{\Psi}[i] \leftarrow \Psi[i]$;
  end for
  $iloc \leftarrow 0$;
  while $\Psi[iloc]! = X$ **do**
    $iloc \leftarrow iloc + 1$;
  end while
  if $(\Psi[N-1] == X)$ & $(iloc == N-1)$ **then**
    $R \leftarrow Rank8(\overline{\Psi}, N-1, X-1)$;
    return R;
  else
    $tmp \leftarrow Rank8(\overline{\Psi}, N-1, X)$;
    $R \leftarrow tmp + S(N-1, X-1) + (\Psi[N-1]-1)S(N-1, X))$;
    return R;
  end if
end if

Function 12 $Unrank8(R, N, X)$: *integer*[]

if $(N == 1)\&(X == 1)$ **then**
  $\Psi[0] \leftarrow 1$;
  return Ψ;
else if $R < S(N-1, X-1)$ **then**
  $\overline{\Psi} \leftarrow Unrank8(R, N-1, X-1)$;
  for $i = 0$ to $X - 2$ **do**
    $\Psi[i] \leftarrow \overline{\Psi}[i]$;
  end for
  $\Psi[N-1] \leftarrow X$;
  return Ψ;
else
  $V \leftarrow R - S(N-1, X-1)$;
  $P \leftarrow \lfloor \frac{V}{S(N-1,X)} \rfloor$;
  $RM \leftarrow V \bmod S(N-1, X)$;
  $\overline{\Psi} \leftarrow Unrank8(RM, N-1, X)$;
  for $i = 0$ to $X - 2$ **do**
    $\Psi[i] \leftarrow \overline{\Psi}[i]$;
  end for
  $\Psi[N-1] \leftarrow P + 1$;
  return Ψ;
end if

Listed in **Function 13**, Cloak$^7(N, X)$'s ranking algorithm will first determine how many connections, say i, have been used to convey the packets, and then use Cloak^8's ranking algorithm (**Function 11**) to compute the value r according to the distribution of packets and flows (i.e., Ψ). Finally, the rank value is obtained as $R = \sum_{j=1}^{i-1} S(N, j) + r$. Presented in **Function 14**, Cloak$^7(N, X)$'s unranking algo-

rithm will first determine how many connections, say i, will be used by subtracting $\sum_{j=1}^{i-1} S(N,j)$ from the rank value until the left value (i.e., $r = R - \sum_{j=1}^{i-1} S(N,j)$) is not larger than $S(N,i)$, and then unrank the left value r by using Cloak[8]'s unranking algorithm (**Function 12**).

Function 13 $Rank7(\Psi, N, X)$: *integer*

```
X ← max(Ψ);
R ← 0;
for j = 1 to X − 1 do
    R ← R + S(N, j);
end for
r ← Rank8(Ψ, N, X);
R ← R + r;
return R;
```

Function 14 $Unrank7(R, N, X)$: *integer*[]

```
R ← R + 1;
iloc ← 1;
while R > (isum = S(N, iloc)) do
    iloc ← iloc + 1;
    R ← R − isum;
end while
Ψ ← Unrank8(R − 1, N, iloc);
return Ψ;
```

6.2.5 Algorithms for $c = 9, 10$

$P(\lambda, \mu)$ is the number of ways of partitioning the number λ into μ parts. Let $\Lambda = [\lambda_1, \ldots, \lambda_\mu]$ denote one partition, where each λ_i contains the number of elements belong to the ith part. Cloak[10]'s ranking/unranking algorithms are obtained by replacing $P(\lambda, \mu)$'s λ with N and its μ with X. After attaining Λ through $P(N, X)$'s ranking algorithm, the encoder will let the ith TCP flow to send λ_i packets. After observing N packets, the decoder will generate Λ according to the number of packets sent by each flow and then obtain the message through through $P(N, X)$'s unranking algorithm. Although there is still no general formula for $P(N, X)$, it obeys the following rule, based on which we can calculate the $P(N, X)$ recursively, for example using the `CountPartitions` function in [198].

$$p(N, X) = p(N - 1, X - 1) + p(N - X, X), \tag{6.4}$$

where $P(N, X) = 0$ if $N \leq 0$ or $X \leq 0$ or $X > N$, and $p(1, 1) = 1$.

Function 15 $Rank10(\Lambda, N, X)$: *integer*

```
R ← 0;
while N > 0 do
  if Λ[X] == 1 then
    N ← N − 1;
    X ← X − 1;
  else
    for i = 1 to X do
      Λ[i] ← Λ[i] − 1;
    end for
    R ← R + p(N − 1, X − 1);
    N ← N − X;
  end if
end while
return R;
```

Function 16 $Unrank10(R, N, X)$: *integer*[]

```
Λ ← 0;
while N > 0 do
  if RV < p(N − 1, X − 1) then
    Λ[N] ← Λ[N] + 1;
    N ← N − 1; X ← X − 1;
  else
    for i = 1 to X do
      Λ[i] ← Λ[i] + 1;
    end for
    R ← R − p(N − 1, X − 1) ;
    N ← N − X;
  end if
end while
return Λ;
```

As described in section 6.2.2, the basic idea of Cloak$^9(N, X)$'s ranking algorithm is to first determine the number connections (say i) that have been used to transmit packets and then use Cloak^{10}'s ranking algorithm (**Function 15**) to compute r according to the distribution of packets and flows (i.e., Λ). Finally, the rank is obtained as $R = \sum_{j=1}^{i-1} p(N, j) + r$. Cloak$^9(N, X)$'s ranking algorithm is listed in **Function 17**. Cloak$^9(N, X)$'s unranking algorithm presented **Function 18** will first determine the number of connections (say i) that will be used by subtracting $\sum_{j=1}^{i-1} p(N, j)$ from the rank until the remaining value (i.e., $r = R - \sum_{j=1}^{i-1} p(N, j)$) is not larger than $p(N, i)$, and then unrank the remaining value r by using Cloak^{10}'s unranking algorithm (**Function 16**).

Function 17 $Rank9(\Lambda, N, X)$: *integer*

$R \leftarrow 0$;
$UC \leftarrow Size(\Lambda)$;
for $j = 1$ to $UC - 1$ **do**
  $R \leftarrow R + PNX(N, j)$;
end for
$r \leftarrow Rank10(N, UC, \Lambda)$;
$R \leftarrow R + r$;
return R;

Function 18 $Unrank9(R, N, X)$: *integer*[]

$R \leftarrow R + 1$;
$iloc \leftarrow 1$;
while $R > (isum = p(N, iloc))$ **do**
  $iloc \leftarrow iloc + 1$;
  $R \leftarrow R - isum$;
end while
$\Lambda \leftarrow Unrank10(R - 1, N, iloc)$;
return Λ;

6.2.6 Algorithms for $c = 2, 3$

As described in section 6.2.2, the basic idea of Cloak$^2(N, X)$'s ranking algorithm (**Function 19**) contains the following steps:

1. Let $\Phi = [\phi_1, \ldots, \phi_N]$, $\phi_i \in \{1, \ldots, X\}$ denote a set of N packets and each ϕ_i represent the index of TCP flow that will be used to send the ith packet. Since in Cloak^5 all TCP flows' indexes are sorted, we use $\widetilde{\Phi}$ to label the sorted Φ and employ $\Upsilon = [\upsilon_0, \ldots, \upsilon_{N-1}]$ to label the indexes of Φ (i.e., $\Phi[\Upsilon[j-1]] = \widetilde{\Phi}[j]$, $1 \leq j \leq N$).

2. Use Cloak^5's ranking algorithm with input $\widetilde{\Phi}$ to calculate r.

3. Use the ranking algorithm for permutations with input Υ to calculate i.

4. Obtain the rank as $R = iC_X^N + r$.

Similarly, the basic idea of Cloak$^2(N, X)$'s unranking algorithm (**Function 20**) contains the following steps:

1. Decompose the rank value R into $R = i|C_X^N| + r$.

2. Using Cloak[5]'s unranking algorithm (**Function 6**) to unrank r, we can determine Φ (i.e., the ϕ_ith the flow will send the ith packet.).

3. Using the unranking algorithm for permutations to unrank i, we get $\widetilde{\Upsilon}$, another set of indexes for Φ, and then obtain a new $\widetilde{\Phi}$ by re-arrange Φ according to $\widetilde{\Upsilon}$ (i.e., $\widetilde{\Phi}[j] = \Phi[\widetilde{\Upsilon}[j-1]]$, $1 \leq j \leq N$).

Function 19 $Rank2(\Phi, N, X)$: *integer*

$\widetilde{\Phi} \leftarrow Sort(\Phi)$;
$\Upsilon \leftarrow Getflow Idx(\widetilde{\Phi})$;
$r \leftarrow Rank5(\widetilde{\Phi}, N, X)$;
$\Upsilon^R \leftarrow Reverse(\Upsilon)$;
$i \leftarrow RankPermute(N, \Upsilon, \Upsilon^R)$;
$R \leftarrow iC_X^N + r$;
return R;

Function 20 $Unrank2(R, N, X)$: *integer*[]

$r \leftarrow R \bmod C_X^N$; $i \leftarrow \frac{R-r}{C_X^N}$;
$\Phi \leftarrow Unrank5(r, N, X)$;
$UnrankPermute(N, i, \widetilde{\Upsilon})$;
for $i = 1$ to N **do**
  $\widetilde{\Phi}[i] \leftarrow \Phi[\widetilde{\Upsilon}[i-1]]$;
end for
return $\widetilde{\Phi}$;

As described in section 6.2.2, the basic idea of Cloak$^3(N, X)$'s ranking algorithm (**Function 21**) consists of the following steps:

1. Let $\widetilde{\Phi} = [\widetilde{\phi}_0, \ldots, \widetilde{\phi}_{N-1}]$, $\widetilde{\phi}_i \in \{1, \ldots, X\}$, denote a set of N packets and each $\widetilde{\phi}_i$ $(0 \leq i < N)$ represent the index of TCP flow that will be used to send the ith packet. Let $\widetilde{\Upsilon} = [\widetilde{\upsilon}_0, \ldots, \widetilde{\upsilon}_{X-1}]$ label the indexes of TCP flows according to their appearance sequence in $\widetilde{\Phi}$.

2. Use Cloak[8]'s ranking algorithm with input $\widetilde{\Phi}$ to calculate r.

3. Use the ranking algorithm for permutations with input $\widetilde{\Upsilon}$ to calculate i.

4. Obtain the rank as $R = iS(N, X) + r$.

Similarly, the basic idea of Cloak$^3(N, X)$'s unranking algorithm (**Function 22**) consists of the following steps:

1. Decompose the rank value R into $R = iS(N, X) + r$.

2. Using Cloak[8]'s unranking algorithm (**Function 6**) to unrank r, we can determine $\widehat{\Phi}$ (i.e., the $\widehat{\phi}_i$th the flow will send the ith packet, $0 \leq i < N$).

3. Using the unranking algorithm for permutations to unrank i, we get $\widetilde{\Upsilon}$, another set of indexes for $\widehat{\Phi}$, and then obtain a new $\widetilde{\Phi}$ by re-arrange $\widehat{\Phi}$ according to $\widetilde{\Upsilon}$ (i.e., $\widetilde{\Phi}[j] = \widetilde{\Upsilon}[\widehat{\Phi}[j] - 1] + 1$, $0 \leq j \leq N$).

Function 21 $Rank3(\widetilde{\Phi}, N, X)$: *integer*

$\widetilde{\Upsilon} \leftarrow Getflow Idx(\widetilde{\Phi}) - \mathbf{1}$;
$r \leftarrow Rank8(\widetilde{\Phi}, N, X)$;
$\widetilde{\Upsilon}^R \leftarrow Reverse(\widetilde{\Upsilon})$;
$i \leftarrow RankPermute(N, \Upsilon, \Upsilon^R)$;
$R \leftarrow iS(N, X) + r$;
return R;

Function 22 $Unrank3(R, N, X) : integer[]$

$r \leftarrow R \bmod S(N, X)$; $i \leftarrow \frac{R-r}{S(N,X)}$;
$\widehat{\Phi} \leftarrow Unrank8(r, N, X)$;
$UnrankPermute(N, i, \widetilde{\Upsilon})$;
for $j = 1$ to N **do**
  $\widetilde{\Phi}[j] = \widetilde{\Upsilon}[\widehat{\Phi}[j] - 1] + 1$;
end for
return $\widehat{\Phi}$;

6.3 Design issues

In this section, we discuss a number of design elements that are central to a practical deployment of Cloak in the Internet and to Cloak's performance.

6.3.1 Message encoding and decoding

As mentioned in the last section, the encoder and decoder do not need to exchange a codebook explicitly. Instead, they use two special functions for encoding and decoding: `Rank()` and `Unrank()`. Each Cloak$^c(N, X)$ channel has its own function pair. The function `Rank()` takes in a flow-packet distribution and returns its *rank* that is the index of the flow-packet distribution in the decreasing lexicographi-

cally ordered array of all possible distributions, starting from 0. `Unrank()` does the opposite—taking in a rank and returning the corresponding flow-packet distribution.

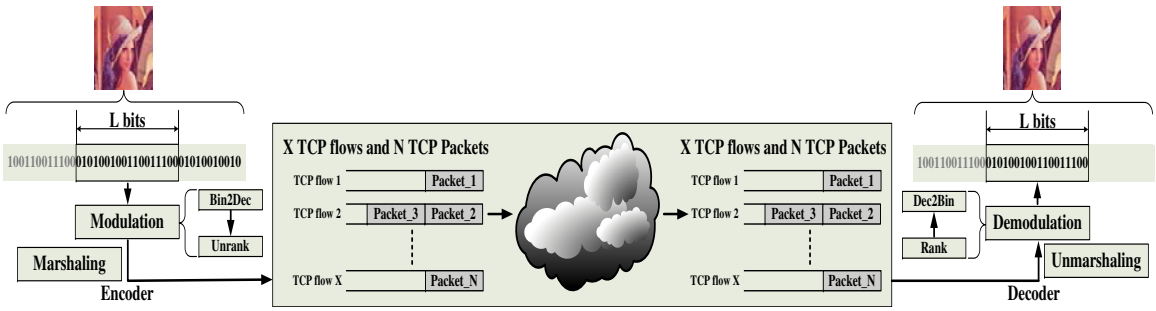


Fig. 6.4. The encoding and decoding processes in Cloak.

Figure 6.4 depicts the encoding and decoding processes in Cloak. The encoder and decoder are assumed to have agreed on (c, N, X) beforehand. They could also dynamically change (c, N, X) by exploiting the random beacons widely available in the Internet, e.g., stock indices [202]. The messages are encoded based on L-bit words, where $1 \leq L \leq \lfloor \log_2 T^c(N, X) \rfloor$. There are three major steps involved in sending a covert message. Each L-bit word is first converted to the nonnegative decimal value (through the `Bin2Dec()` function) that serves as the rank for the corresponding packet-flow distribution. Then, `Unrank()` is invoked to compute the distribution. Finally, the encoder marshals the packet-flow code into the actual TCP flows and data packets. After sending the N packets over the X flows, the encoder has to receive the ACKs for the N packets before sending the next N packets. In the case of packet losses, Cloak may rely on TCP to recover them.

The three-step process above is exactly reversed for receiving a covert message. In the first step, the decoder unmarshalls the packet-flow distribution from the flows and packets received from the encoder. That is, the decoder collects exactly N TCP packets from the X flows before moving to the next step. Moreover, since the number of flows can be distinguished based on the order of the TCP three-way handshaking performed, the decoder can count the number of data packets in each flow. Similar to

before, any TCP packet loss, duplication, or reordering can be taken care of by TCP. As soon as N packets are collected, the decoder feeds the distribution into `Rank()` which yields the corresponding rank. As a last step, the rank is converted back to the L-bit word (through the function `Dec2Bin()`).

6.3.2 Head-of-line blocking problem

In this section, we discuss a head-of-line blocking (HoLB) problem that we have encountered when conducting Internet experiments. The HoLB problem degrades the data rates of all encoding methods, except for $c = 2, 5$. To explain the problem, we consider an extreme scenario where most of the N packets are distributed to a single flow, while other flows receive at most one packet. Therefore, the total transmission time for the message is governed by the time required to transmit the packets in the most busy flow which prevents the encoder from transmitting the next message. Furthermore, since the TCP congestion window usually starts with one or two packets, it will take the busy flow's sender several RTTs to complete the transmissions, thus leading to a low data rate. The problem may worsen if there are packet losses in the most busy flow that will retransmit those packets according to the timeout mechanism or the fast retransmission/fast recovery mechanism. This problem may also occur to the flows that are connected to different servers which experience a wide range of RTTs.

A simple way of mitigating the HoLB problem is to aggressively transmit every N packets. The basic idea is that the encoder will dispatch all packets belonging to kth message after receiving ACK packets that acknowledge the data packets for the $(k-1)$th message or when the period T_E times out. If the encoder does not receive all the expected ACKs before T_E, it will retransmit unacknowledged packets and reset the timer. T_E is usually set to the estimated RTT that is computed through the EWMA of RTT samples, an approach similar to the one used in normal TCP. However, the downside is that the resulting traffic pattern will be different from the

normal TCP behavior. This has prompted us to design a new codeword scheme to be discussed next.

A D-limited codeword scheme

The D-limited codeword scheme essentially caps the maximum number of packets assigned to a flow to D; that is, it enforces $\max\{n_i\} \leq D$, where $D \geq 1$ is a constant. The choice of D should be chosen such that it is less than the encoder's TCP send window size in terms of packets. In this way, all the packets can be sent out in one RTT; otherwise, multiple RTTs will be needed for transmitting a message.

We use $c = 10$ (indistinguishable packets and flows) to illustrate how this codeword scheme works. We first define two quantities:

1. $\Upsilon(N)$ is the total number of ways to distribute N packets into TCP flows, such that each flow is given *at most* D packets.

2. $\Gamma(N, D)$ is the total number of ways to distribute N packets into D TCP flows, such that each flow is assigned at least one packet. Note that $\Gamma(N, D) = P(N, D)$ if both packets and flows are indistinguishable (i.e., $c = 10$).

Proposition 6.3.1 *If both packets and flows are indistinguishable,* $\Upsilon(N) = \sum_{i=1}^{D} P(N, i)$.

Proof We prove this proposition using the Ferrers diagram representation [196]. It is not difficult to observe that the Ferrers diagram for partitioning N into d ($1 \leq d \leq N$) parts is a conjugate to the Ferrers diagram for partitioning N into parts whose maximal sizes are d. For example, Figure 6.5 illustrates a Ferrers diagram for distributing eight TCP packets into three flows where the maximal flow size is five. However, the maximal flow size is only three in its conjugate that uses five flows. Since both packets and flows are indistinguishable, there is a bijection between these two sets of partitions. By summing up all cases for $d = 1, \ldots, D$, we obtain $\Upsilon(N) = \sum_{i=1}^{D} P(N, i)$. ■

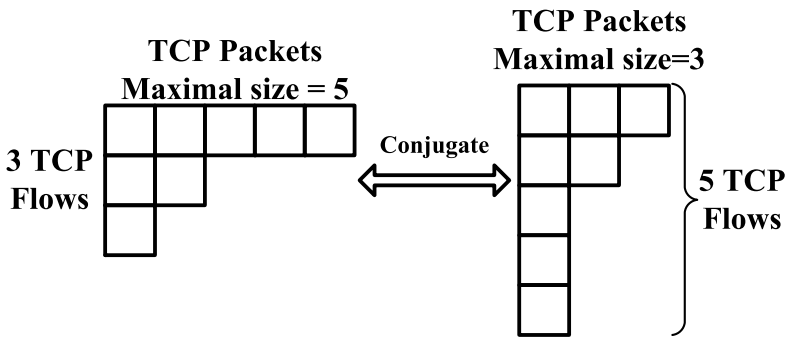


Fig. 6.5. A Ferrers diagram for the proof of Proposition 6.3.1.

Corollary 6.3.1 *To generate D-limited codewords from $P(N, D)$, we need at most $N + 1 - D$ flows to convey a message.*

Proof Among all the possible partitions of $P(N, D)$, there exists an extreme case in which one flow transmits $N + 1 - D$ packets and each of the remaining $D - 1$ flows transmits only one packet. According to the conjugation, we know that the D-limited codewords need at most $N + 1 - D$ flows. ■

Proposition 6.3.1 computes how much information this D-limited codeword scheme could transmit. Corollary 6.3.1, on the other hand, shows that if the upper bound of flows is X, then $N \leq X + D - 1$. We now use Proposition 6.3.1 and Corollary 6.3.1 directly to construct D-limited codewords for $c = 10$:

1. **Encoding** To transmit a message (a binary string), the encoder first calculates its decimal value and then uses $Cloak^{10}$'s unranking algorithm to get the corresponding packet-flow distribution, denoted by ζ. After that, the encoder computes ζ's conjugate [196], denoted by ζ', and transmits packets according to ζ'.

2. **Decoding** Upon receiving a packet-flow distribution ζ', the decoder first computes its conjugate ζ and then uses $\text{Cloak}^{10}(N, X)$'s ranking algorithm to decode the message.

To construct D-limited codewords for other encoding methods, we can adopt our general framework for the design of new ranking and unranking algorithms. That is, when the encoder receives ζ' from $Cloak^9$ or $Cloak^{10}$, it could expand ζ' by considering distinguishable packets or flows. For example, if only flows are distinguishable, we could permutate the location of flows that have different n_i and then increase the capacity in a way similar to $\lambda!$ or C_X^N. If only packets are distinguishable, we could consider how to partition them into different flows and therefore to increase the capacity in a way similar to $S(N, X)!$. If both flows and packets are distinguishable, we could permutate the location of packets that belong to different flows. The only requirement is not to change the value of n_i.

6.3.3 Packet and flow distinguishability

There are generally two approaches to make the packets and flows distinguishable. The first is a *timing approach* that is particularly useful for distinguishing flows. Consider that an encoder performs the TCP three-way handshakings for the flows *one at a time*; the order of handshakings can therefore serve as implicit markers for the flows. Moreover, if the inter-handshaking period is large enough so that the TCP SYN packets are not reordered, the decoder could therefore observe the same order of the TCP connections established (i.e., the flows are distinguishable). Note that even when the TCP packet headers are modified by an ANI, the flow distinguishability is still preserved using this approach. However, the flow distinguishability may be destroyed if the ANI reorders the TCP SYN packets, or if it collects the SYN packets and then sends them out at the same time. Note that after the connections have been established, the encoder is not necessary to re-generate these flows, unless it wants to change the coding scheme. Then it will avoid generating lots of TCP connections.

The second is a *storage approach* that can be used to distinguish packets and flows. However, since the timing approach may not be applicable to packet distinguishability, the storage approach is most useful for packets. This method embeds

a unique *marker* into a packet in order to make it distinguishable. To make the N TCP data packets distinguishable, we need a $\lceil \log_2 N \rceil$-bit marker. The key point is that by using only $\lceil \log_2 N \rceil$ bits to make packet distinguishable, the user may gain much more capacity for transferring covert messages through the combinations of flows and packets. Besides well-known methods such as IPID, TCP timestamp and initial sequence number [37,109,131], other possible ways include partial ACK, receive window size, and packet length. According to our experience, the partial ACK seems to be a promising approach. Based on our measurement study performed from more than 100 popular security Web sites, the partial ACKs can traverse the firewalls in all cases. On the other hand, using the receive window size and packet length for the markers runs into a much higher risk of being detected, because their distributions deviate from the normal ones.

6.4 Experiment results

In this section, we evaluate how Cloak's data rate is affected by the RTT, router hop distance, geographical locations, and various adverse network conditions. Besides, we evaluate the effect of the HoLB problem on Cloak, and its performance with the D-limited codeword scheme. We also compare Cloak's performance with other timing channels: IP timing channel (IPTime) [41] and JitterBug [43], wherever we find appropriate. We conducted experiments in the real Internet environment with the PlanetLab platform, and our test bed which allows controlled experiments with various network conditions.

We measure the data rate of the timing channels in terms of their *goodput* defined as:

$$G = (1 - p_e)\frac{M \times L}{T_d}, \tag{6.5}$$

where T_d is the total time required for delivering M L-bit covert messages, and p_e is the channel's bit error rate (BER). The BER is computed based on the *Levenshtein distance* which is given by the number of insertions, deletions, and substitutions

needed to convert a source message into a decoded message. Since Cloak is reliable, its p_e is 0.

6.4.1 Implementation

We have implemented Cloak's encoder and decoder as a TCP client and a TCP listener, respectively, including the ten `Rank()` and `Unrank()` functions. We have implemented `Bin2Dec()`, `Dec2Bin()`, `Rank()`, and `Unrank()` as offline functions. That is, the encoder pre-computes all required packet-flow combinations, and the decoder starts decoding only after capturing all N packets from X flows.

Cloak

For the Cloak's encoder, we have implemented two types of transmission functions based on the TCP socket (Cloak(STREAM)) and the raw socket (Cloak(RAW)). In Cloak(STREAM), system's TCP stack guarantees the transmission reliability and let its traffic pattern resemble normal TCP flows. However, it may take several RTTs to complete a single codeword transmission; thus limiting its data rate. Cloak(RAW), on the other hand, applies the aggressive transmission mechanism discussed in section 6.3.2 to improve its data rate. We have also implemented a separate capturing thread in the encoder to monitor the ACK arrivals, in order to determine if the other side has received all the N packets.

We have implemented the Cloak's decoder with `libpcap` v0.9.5 library for sniffing TCP packets. Moreover, we use a snaplen of 96 bytes to reduce the overhead during the packet capturing operation; we did not observe any packet drops throughout the experiments.

IPTime and JitterBug

We have implemented both IPTime's and JitterBug's encoding and decoding schemes as plug-in modules in the Cloak encoder and decoder, respectively. We employ the UDP socket (i.e., `SOCK_DGRAM`) since the packet transmission in these two timing channels does not require reliability. During the encoding process, the plug-ins invoke the modulation function in the Cloak encoder to let the codeword bypass `Bin2Dec()` and `Unrank()`, and to marshal the binary stream directly into a flow of modulated UDP packets. Moreover, the encoder generates the modulated sequences complying with the specifications of IPTime and JitterBug. Both the IPTime's encoder and JitterBug's encoder use a fixed timing interval (or timing window) of w. The JitterBug's encoder, in addition, has a default tolerance parameter of $\varepsilon = w/4$. The corresponding plug-ins in the decoder perform the reverse procedures for decoding. Moreover, we did not implement any framing and error correction mechanism for Cloak, IPTime, and JitterBug.

6.4.2 Experiment platforms

PlanetLab setup

As shown in Figure 6.6(a), we locate the encoders in nine geographically diverse PlanetLab nodes, and the decoder and a Web server in a campus network. The encoders send packets to the Web server, and the decoder eavesdrops the packets and decoded them.

We have obtained a total of 17,545 RTT samples between the decoder and each PlanetLab node during the experiment period. Table 6.3 shows the nine PlanetLab nodes with the router hop counts from the encoder to them and the RTT statistics with a 95% confidence interval (i.e., C.I.). Note that the average RTTs range between

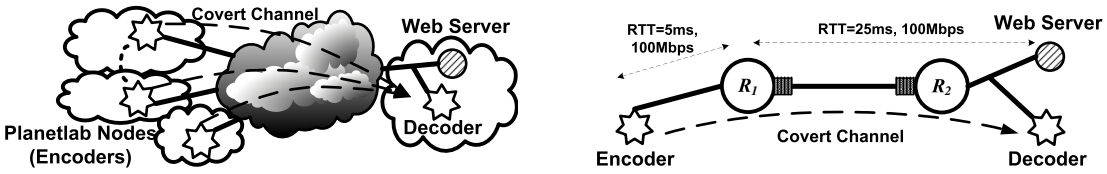
0.0652 seconds and 0.3418 seconds; the RTT measurements for the nodes JP, KR, and CA have higher variations than the others.

Table 6.3
Measured path characteristics between each PlanetLab site and the decoder machine.

Locations	Hops	RTT		
		Means	Std. Dev.	95% Conf. Intervals
Shenyang, China (CN)	13	.0652	.0060	.0651/.0653
Tokyo, Japan (JP)	16	.0992	.0244	.0988/.0996
California, U.S. (CA)	14	.1767	.0230	.1763/.1770
Kansas, U.S. (KS)	16	.2176	.0056	.2175/.2177
Rhode Island, U.S. (RI)	13	.2267	.0074	.2266/.2268
Gwangju, Korea (KR)	18	.2343	.0356	.2338/.2348
Ghent, Belgium (BE)	16	.3075	.0048	.3074/.3075
London, UK (UK)	19	.3124	.0061	.3123/.3124
Lisbon, Portugal (PT)	17	.3418	.0171	.3415/.3420

Test-bed setup

As illustrated in Figure 6.6(b), the test bed consists of two soft routers shared by an encoder, a decoder, and a Web server. All the links have a capacity of 100Mbit/s. We deploy Dummynet [164] in both routers to emulate various network conditions: packet losses, network delay, and packet reordering. The RTT between the encoder and R_1 is 5ms, whereas that between the decoder/Web server and R_1 is 25ms, unless specified otherwise. By sniffing all network traffic between R_2 and the Web server, the decoder is able to receive covert messages from the encoder.



(a) The PlanetLab setup.

(b) The test-bed setup.

Fig. 6.6. The two experiment platforms: PlanetLab and a controlled test bed.

6.4.3 PlanetLab experiments

Experiment design

For the Cloak's experiments, we only report the experiment results of Cloak$^1(N, X)$ due to the page limit. To study the effect of N, we fix X to 20 to give a large enough number of flows, and $N = \{5, 9, 10, 11, 15, 20, 30, 40, 50\}$ which covers a reasonable range of channel capacity. Similarly, to study the effect of X, we fix N to 20 and consider $X = \{4, 6, 8, 10, 12, 14\}$.

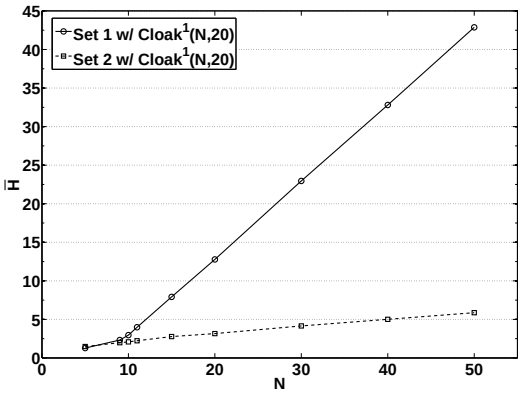
To study the adverse effects of the HoLB problem, we have generated two sets of codewords (datasets 1 and 2) for each N in Cloak$^1(N, 20)$. Each dataset consists of 100 L-bit ($M = 100$ and $L = \lfloor \log_2 X^N \rfloor$) codewords. Moreover, we assign each packet in dataset 2 to the 20 flows with equal probability; however, we intentionally assign more packets in dataset 1 to flow 1. We measure the *degree of HoLB* of a codeword by $H = \max_{0 \le i < X} n_i$. Figure 6.7(a) plots the values of $\overline{H}$, the mean values of H for different values of N. As shown, the rate of increase in $\overline{H}$ for dataset 1 is about ten times higher than that for dataset 2 when N is beyond 10. Moreover, we have generated other sets of codewords (datasets 3 and 4) for each X in Cloak$^1(20, X)$. The codewords for datasets 3 and 4 are generated the same ways as for datasets 1 and 2, respectively. Figure 6.8(a) shows that the values of $\overline{H}$ for the two datasets diverge as X increases.

Experiment results

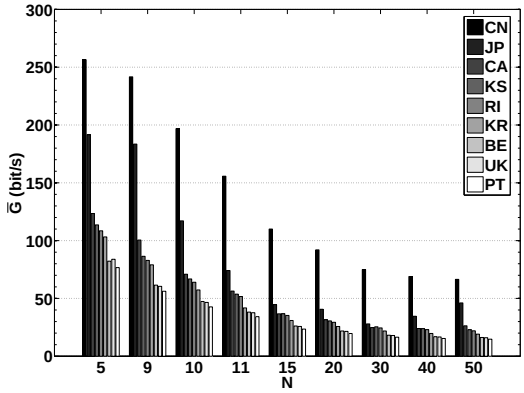
Figures 6.7(b), 6.7(c), 6.8(b), and 6.8(c) plot the average goodput for the nine PlanetLab nodes with the four datasets of codewords. We compute the average goodput for each (N,X) tuple by performing 30 measurements. For each N or X, the nine nodes in the figures are sorted in the ascending order of their measured mean RTTs given in Table 6.3. We first report the results for datasets 2 and 4 (Figure 6.7(c) and Figure 6.8(c)) for which the packets are assigned uniformly to the 20 flows. Among all the nodes, CN achieves a maximum channel goodput of around 450 bit/s in Figure 6.7(c). Both figures also show that the average goodput $\overline{G}$ for the two smallest RTTs (nodes CN and JP) are the highest. However, the goodput does not necessarily decrease with the RTTs. That is, although the goodput is inversely proportional to the RTT, there are other factors, such as packet losses, that could disturb the goodput. Moreover, the increase seems to be more drastic for the case of increasing X. For example, the JP node's goodput is increased by more than four times as X increases from 4 to 12. On the other hand, the rates of increases for other nodes with longer RTTs are smaller. That is, when RTT is large, it will reduce the gain obtained from the increase in the channel capacity.

Next, we evaluate the effects of the biased packet distributions on the average goodput. We first compare the results for datasets 1 and 2 (Figure 6.7(b) and Figure 6.7(c)). The comparison reveals that they show opposite trends as N increases: the goodput decreases with N in Figure 6.7(b). It is important to point out that the scales of the two figures are actually different, therefore the goodput in Figure 6.7(c) is all greater than the respective cases in Figure 6.7(b), except for $N = 5$. Since $\overline{H}$ increases with N as shown in Figure 6.7(a), it will take flow 1 a longer time to complete its packet transmission as N increases. For the comparison of datasets 3 and 4 (Figure 6.8(b) and Figure 6.8(c)), the goodput in Figure 6.8(c) is all greater than the respective cases in Figure 6.8(b). However, unlike the previous cases, the goodput in Figure 6.8(b) slightly improves as X increases, but the goodput stops growing as

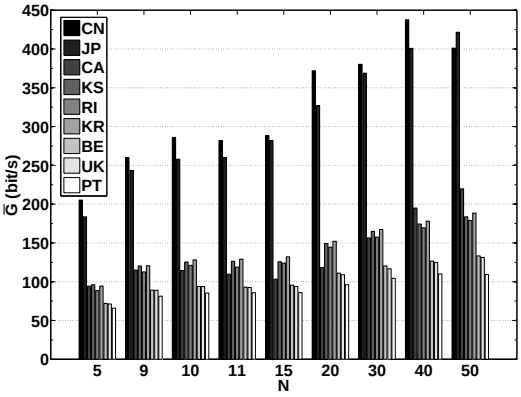
X reaches 10. An increase in X in fact alleviates the HoLB problem, because flow 1 will become less busy; as a result, it is not surprising to see some improvement in the goodput as X increases.



(a) The values of $\overline{H}$ for datasets 1 and 2 as a function of N.



(b) The average goodput for the PlanetLab nodes with dataset 1.

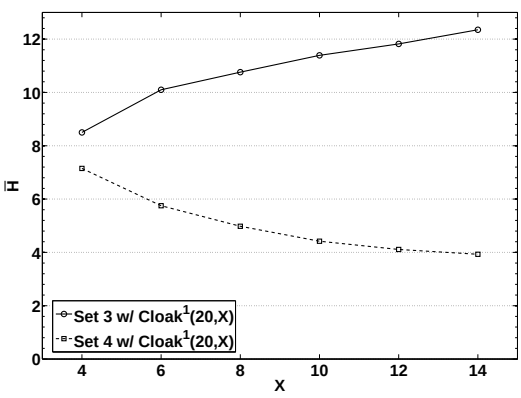


(c) The average goodput for the PlanetLab nodes with dataset 2.

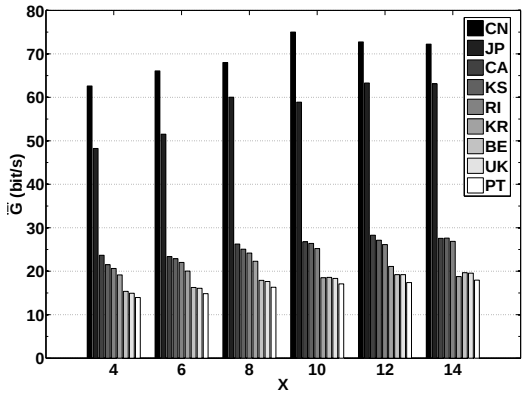
Fig. 6.7. Results for PlanetLab nodes: the average goodput verses N for Cloak$^1(N, 20)$ with datasets 1 and 2.

Evaluation of the D-limited codewords

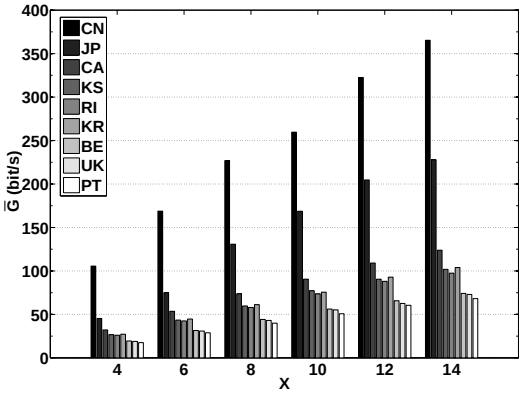
To measure the performance of the D-limited codewords, we have selected five (JP, CA, KS, KR, and BE) out of the nine PlanetLab nodes to measure the average



(a) The values of $\overline{H}$ for datasets 3 and 4 as a function of X.



(b) The average goodput for the PlanetLab nodes with dataset 3.



(c) The average goodput for the PlanetLab nodes with dataset 4.

Fig. 6.8. Results for PlanetLab nodes: the average goodput verses X for Cloak$^1(20, X)$ with datasets 3 and 4.

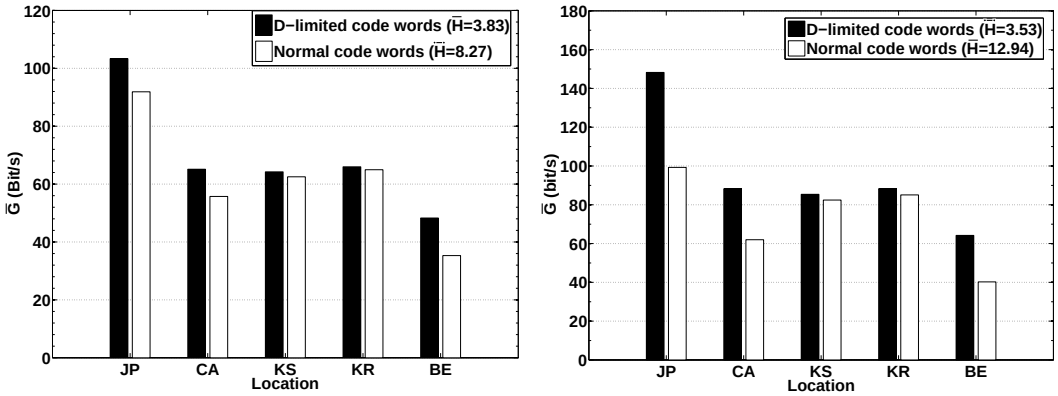
goodput of Cloak. Similar to the last section, we have generated a set of 100 L-bit binary codewords for each (N, X) tuple for Cloak$^1(N, X)$, where $X = 6$ and $N = \{12, 16, 20\}$. We use Cloak(STREAM) to encode them into two distinct sets of codewords: one generated by the D-limited codewords scheme with $D = 6$ and the other by the normal codewords. The average goodput is again based on 30 measurements.

Figure 6.9 compares the average goodput of the two codewords for the five nodes. The figures show that the D-limited codeword always gives a higher goodput than the normal scheme for all nodes and for all three (N, X) tuples. Each figure also gives the average degrees of HoLB for the two codewords. The average degrees for the D-limited codewords are quite stable in all three cases, whereas the degree for the normal codewords is the highest in Figure 6.9(c), followed by Figure 6.9(b) and then by Figure 6.9(a).

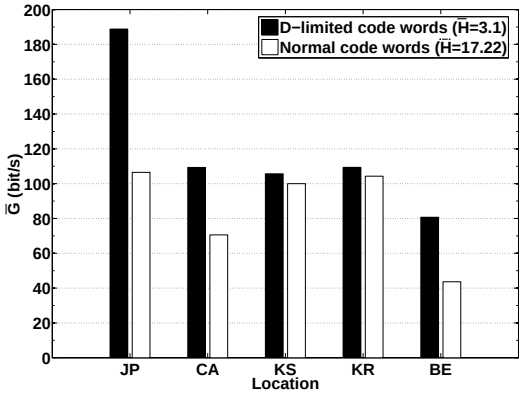
As a result, the percent of improvement of using the D-limited codewords also follows the same decreasing order for nodes JP, CA, and BE in Figures 6.9(a)-6.9(c). In particular, we have noticed a maximum gain of 77% from the JP node with Cloak$^1(20, 6)$. On the other hand, the nodes KS and KR attain much less gains; for example, the gain is only 1.6% for the KR node with Cloak$^1(12, 6)$. By examining the traffic traces, we have found that the packet loss rates at these two nodes are much lower than the others. Therefore, the normal scheme has already achieved a very high goodput; the additional benefit of adopting the D-limited scheme becomes marginal. We also evaluate the performance of the aggressive transmission scheme and find that it could significantly increase the Cloak's goodput.

Comparing Cloak, JitterBug, and IPTime

We have also conducted experiments on JitterBug and IPTime in the five PlanetLab nodes. In this set of experiments, we have generated another 100 packet-flow codewords using the normal Cloak$^1(20, 4)$ encoder with $\overline{H} = 5.86$. Each node uses both Cloak(RAW) and Cloak(STREAM) to transmit the codewords. We set Cloak(RAW)'s T_E to the measured mean RTTs. For the JitterBug and IPTime experiments, the encoder marshals each respective binary codeword directly into a flow of modulated UDP packets with $w = \{\text{RTT}, 1.5\text{RTT}\}$. Both the average goodput and average BER are computed based on 30 samples.



(a) Cloak$^1(12, 6)$ (b) Cloak$^1(16, 6)$



(c) Cloak$^1(20, 6)$

Fig. 6.9. Comparing the average goodput for the normal codewords and the 6-limited codewords.

We summarize the experiment results in Table 6.4. In each cell, the two leftmost values correspond to the lower limit of and the upper limit of the 95% confidence intervals for the same average goodput, and the rightmost value inside the parentheses corresponds to the measured average BER. We first point out that it is difficult to conduct a fair comparison among the three channels, because, for example, Cloak uses multiple flows whereas the other two use only one. Therefore, the comparison is based on how their goodputs are affected by the RTTs. Recall that the five nodes are sorted in an ascending order of their mean RTTs. For both Cloak channels, we do not find any general relationship between their goodputs and the RTTs, except

that the lowest goodputs for both cases are given by the highest RTT (i.e., BE). On the other hand, the goodputs for JitterBug and IPTime show downward trends as the RTT increases. The magnitude of the goodput degradation is rather significant, which is between three to four times when comparing the goodput for JP and BE. Their average BERs also show similar downward trends except for a couple of points.

Table 6.4
Average goodput and average BER for Cloak, IPTime, and JitterBug obtained from five PlanetLab nodes.

	95% confidence intervals of average goodput (average BER)		
	Cloak(`RAW`)	Cloak(`STREAM`)	JitterBug(RTT)
JP	203.86/216.28 (0)	55.17/58.57 (0)	13.01/13.04 (.0155)
CA	85.66/88.49 (0)	68.75/71.75 (0)	7.33/7.35 (.0265)
KS	90.77/91.00 (0)	66.57/69.26 (0)	6.13/6.14 (.0018)
KR	106.52/107.90 (0)	68.68/71.51 (0)	5.67/5.68 (.0012)
BE	63.88/64.20 (0)	44.69/45.61 (0)	4.36/4.36 (.0011)

	95% confidence intervals of average goodput (average BER)		
	JitterBug(1.5RTT)	IPTime(RTT)	IPTime(1.5RTT)
JP	8.77/8.78 (.0164)	9.48/9.50 (.0363)	6.49/6.51 (.0209)
CA	4.76/4.81 (.0521)	5.43/5.47 (.0282)	3.68/3.69 (.0158)
KS	4.10/4.10 (.0010)	4.50/4.51 (.0112)	3.02/3.02 (.0088)
KR	3.81/3.81 (.0010)	4.17/4.18 (.0126)	2.80/2.81 (.0081)
BE	2.91/2.91 (.0007)	3.21/3.21 (.0076)	2.14/2.15 (.0066)

6.4.4 Test-bed experiments

We have conducted test-bed experiments to study the effects of packet losses and packet reordering on Cloak's performance. We found that both packet loss and packet

reordering will decrease covert channels' goodput. For Cloak, the degradation is due to the prolonged delay for packet retransmission. For IPTime and JitterBug, the adverse network conditions result in high BER. Extensive experiment results suggest that Cloak is more robust than IPTime and JitterBug because Cloak could still keep zero BER and relatively high goodput.

We study Cloak(STREAM) and Cloak(RAW) with $T_E = 60$ms by transmitting two sets of codewords from Cloak$^1(20, 10)$ with $\overline{H} = \{4.42, 11.39\}$. Each set consists of 100 66-bit codewords. For both JitterBug and IPTime, their encoders transmit the corresponding binary codewords using a single flow of modulated UDP packets, where $w = \{30, 60\}$ms. Each experiment repeats for 30 times and we report the average goodput and average BER. On the other hand, we configure the Dummynet to drop packets with a packet loss rate (PLR) ranging between 0% and 5%.

Table 6.5 tabulates the experiment results. Clearly, the packet losses degrade the goodput for all cases. However, Cloak is reliable; therefore, the results still show that its average BER is 0%. JitterBug and IPTime, on the other hand, suffer from relatively high average BERs; their highest average BERs recorded are 6.4% and 8.7%, respectively. Moreover, the effects of the packet losses are different for Cloak(STREAM) and Cloak(RAW). Since Cloak(STREAM) follows the TCP congestion control algorithm, packet losses have more profound impact on its goodput; its goodput degrades by five times when the PLR increases from 0 to 5%. However, with the same change in the PLR, the goodput for Cloak(RAW) drops only by 33%. For JitterBug and IPTime, the value of w has similar effects on their goodputs. When w is doubled, their goodputs are approximately decreased by 50%. Finally, it is clear that their goodputs are not affected by $\overline{H}$.

We have also studied the effect of packet reordering on covert channels' goodput. Specifically, we have evaluated the average goodput of each channel with 3 different packet reordering scenarios A, B, C. In each scenario, R_1 was configured with different numbers of pipes and entrance probabilities while the mean RTT between the encoder and the decoder was equal to 30ms. In other words, each packet entering into R_1 was

Table 6.5
The average goodput and average BER for Cloak, IPTime, and JitterBug obtained from the test bed under different PLRs.

PLR	Cloak(RAW)	Cloak(STREAM)	JitterBug(30ms)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 4.42$		
0%	992.92/1010.87 (0)	513.54/515.85 (0)	41.72/41.72 (0)
1%	881.20/899.64 (0)	188.32/204.30 (0)	41.15/41.23 (.0128)
3%	748.13/769.16 (0)	92.24/97.94 (0)	40.07/40.18 (.0383)
5%	665.97/679.63 (0)	61.09/66.11 (0)	39.04/39.17 (.0626)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 11.39$		
0%	948.51/964.63 (0)	194.87/195.01 (0)	41.09/41.09 (0)
1%	872.19/893.67 (0)	114.38/123.04 (0)	40.54/40.61 (.0125)
3%	748.97/768.56 (0)	53.56/62.15 (0)	39.51/39.65 (.0368)
5%	680.55/693.69 (0)	33.84/40.52 (0)	38.52/38.65 (.0611)

PLR	JitterBug(60ms)	IPTime(30ms)	IPTime(60ms)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 4.42$		
0%	21.76/21.77 (0)	29.32/29.32 (.0623)	15.62/15.62 (.0320)
1%	21.48/21.52 (.0123)	29.15/29.18 (.0671)	15.53/15.55 (.0368)
3%	20.91/20.97 (.0381)	28.84/28.88 (.0769)	15.35/15.38 (.0479)
5%	20.34/20.44 (.0635)	28.51/28.58 (.0871)	15.19/15.22 (.0579)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 11.39$		
0%	21.43/21.43 (0)	29.32/29.32 (.0625)	15.62/15.62 (.0322)
1%	21.13/21.16 (.0133)	29.17/29.19 (.0670)	15.54/15.55 (.0368)
3%	20.59/20.66 (.0374)	28.85/28.90 (.0767)	15.38/15.40 (.0464)
5%	20.09/20.15 (.0609)	28.57/28.62 (.0858)	15.21/15.25 (.0565)

randomly queued to a pipe that delays the packet for a while before passing the packet to the next hop. Therefore, if a group of packets being forwarded via pipes with different delay, it is very likely that these packets will be reordered. The encoder transmitted the same sets of packet-flow codewords for 30 times, and recorded the average goodput and average BERs.

The experiment results are shown in Table 6.6 and we adopt the same representation as previous tables. The results show that Cloak is resilient to packet reordering and enjoy a higher goodput than the other 2 timing channels. In particular, both Cloak versions are resilient to packet reordering without any decoding error. As compared with the experiments result of PLR $= 0\%$ from the Table 6.5, we observe that Cloak(`RAW`) could still maintain high goodput of at least 967.6 bit/s on average even during the three reordering scenarios, whereas Cloak(`STREAM`) had suffered from the packet reordering events with the throughput penalty of at most 25.7% compared with the previous results. Nonetheless, its measured goodput is still at least 4.3 times and 5.3 times greater than the maximum measured goodput of JitterBug and IPTime, respectively. Furthermore, both JitterBug and IPTime had suffered from significant throughput degradation and decoding error rates in the three scenarios. Accordingly, their measured BERs can be up to 29.2% and 18%, respectively. Compared with previous experiment results, we have recorded maximum measured throughput drops of 29.2% and 12.6% for the JitterBug and IPTime, respectively.

Table 6.6
Average goodput of three covert channels under different reordering scenarios $\mathbb{R}$.

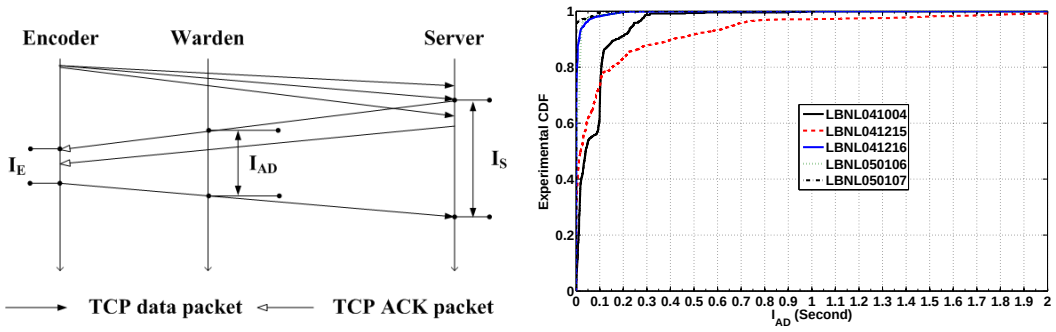
$\mathbb{R}$	Cloak(RAW)	Cloak(STREAM)	JitterBug(30ms)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 4.42$.		
A	1013.31/1027.43 (0)	463.87/468.55 (0)	29.44/29.62 (.2923)
B	1005.78/1022.96 (0)	401.32/405.63 (0)	30.98/31.13 (.2558)
C	1012.77/1028.17 (0)	358.63/363.82 (0)	33.37/33.57(.1977)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 11.39$.		
A	960.54/974.64 (0)	166.58/168.98 (0)	29.46/29.62 (.2812)
B	962.00/976.84 (0)	144.04/145.82 (0)	30.89/31.04 (.2468)
C	965.14/981.96 (0)	148.81/151.20 (0)	32.90/33.12 (.1965)

$\mathbb{R}$	JitterBug(60ms)	IPTime(30ms)	IPTime(60ms)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 4.42$.		
A	18.48/18.56 (.1496)	27.52/27.62 (.1181)	15.14/15.16 (.0613)
B	15.42/15.51 (.2896)	25.58/25.69 (.1801)	14.45/14.51 (.1026)
C	18.73/18.81 (.1378)	26.54/26.65 (.1493)	14.09/14.15 (.1250)
	95% confidence intervals of average goodput (average BER) for 100 codewords with $\overline{H} = 11.39$.		
A	18.32/18.40 (.1432)	27.55/27.66 (.1175)	15.14/15.18 (.0607)
B	15.39/15.46 (.2802)	25.64/25.74 (.1785)	14.46/14.50 (.1029)
C	18.52/18.59 (.1338)	26.61/26.70 (.1477)	14.14/14.20 (.1221)

6.5 Detecting Cloak

In this section, we explore an effective algorithm for detecting Cloak. In particular, we propose using a new detection metric that measures the time between an ACK arrival (Pkt_{ACK}) and a new data packet arrival (Pkt_{Data}) at the warden, which is denoted as I_{AD} in Figure 6.10(a). Moreover, if there are other ACKs arriving within

the period between Pkt_{ACK} and Pkt_{Data}, the warden will consider only the last duplicate ACK and ignore other ACK packets.



(a) Passive warden's surveillance model.

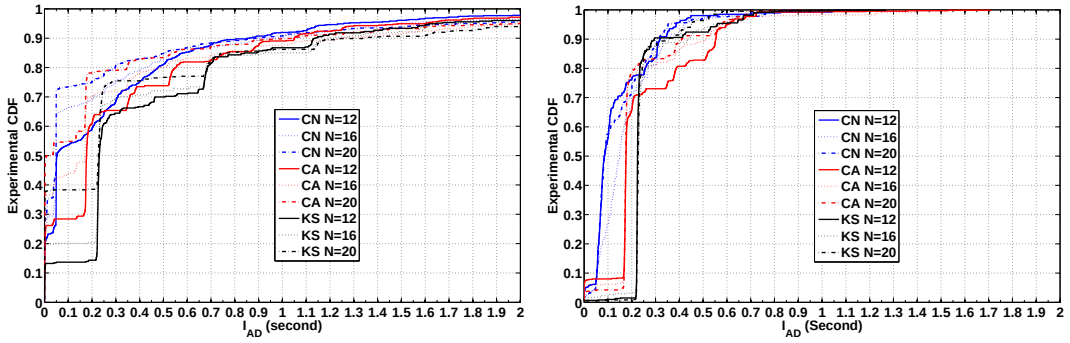
(b) The LBNL traces.

Fig. 6.10. The metric I_{AD} for detecting Cloak channels and the LBNL traces.

In the following we explain the motivation for using I_{AD} in the detection of Cloak. In a normal TCP connection, if a sender has data to send, it will dispatch new packets upon receiving a new ACK. However, with Cloak, some TCP flows may not immediately send new data upon receiving new ACKs, because the encoder cannot send a new message without receiving the TCP ACKs for the previous message. As a result, the value of I_{AD} tends to be larger for the TCP flows used by Cloak. Moreover, the I_{AD} value will even be higher when the N packets are unequally assigned to the X flows. However, a large I_{AD} may also be the result of normal think time (e.g., persistent connections in HTTP/1.1) [203]. Therefore, our detection scheme consists of two steps. The first step is to identify suspicious flows based on passive detection, and the second is to perform an active test to determine whether a Cloak channel exists.

For the purpose of passive detection, we have analyzed the distribution of I_{AD} from the experiments conducted between three PlanetLab nodes (CN, CA, and KS) and a host in a campus network. The Cloak parameters are given by $X = \{2, 4, 6, 8, 10\}$ and $N = \{12, 14, 16, 18, 20\}$. Each experiment setting contains 100 randomly generated

normal codes with $\overline{H} = 11.97$ and another 100 D-limited codes with $\overline{H} = 3.1$. All experiments are repeated 30 times. Figure 6.11(a) and Figure 6.11(b) show results for normal codes and D-limited codes respectively. We can observe from Figure 6.11(b) "jumps" which take place approximately at each flow's RTT. The reason is that the number of RTTs required for transmitting D-limited codes is mostly one. Similar jumps also occur in Figure 6.11(a) which take place at one RTT, two RTTs and three RTTs. The results show that the number of RTTs required for normal codes will be more than one, because some flows could not transmit its packets within a send window.



(a) I_{AD}s of Cloak$^1(N, 6)$ using normal codewords.

(b) I_{AD}s of Cloak$^1(N, 6)$ using D-limited codewords (D=6).

Fig. 6.11. Empirical CDF for I_{AD}.

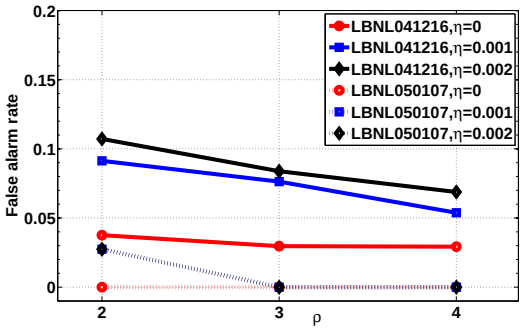
Our passive detection algorithm employs a nonparametric CUSUM algorithm [179] to identify flows that frequently have large I_{AD}s. The CUSUM algorithm assumes that the mean value of the variable under surveillance will change from negative to positive when a change occurs. Since measured I_{AD}s are larger than zero, we transform them into a new random sequences, $\tilde{I}_{AD} = I_{AD} - (\Theta_{AD} + \rho\Delta_{AD})$, where Θ_{AD} and Δ_{AD} are the mean value and standard deviation of normal I_{AD} respectively, and ρ controls the limit of allowable large I_{AD}s. Note that besides constant parameters the CUSUM algorithm only needs one value, denoted as $y_{\tilde{I}}$, for detection. This value will be

updated when a new $\tilde{I}_{AD}$ is observed. Therefore, the warden will keep one $y_{\tilde{I}}$ for each flow and raises an alarm if any $y_{\tilde{I}}$ exceeds the threshold.

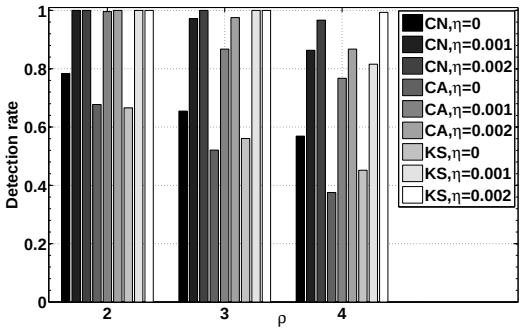
The algorithm requires the estimation of Θ_{AD} and Δ_{AD}. We use three training sets of the LBNL data [182] to estimate the parameters (the data obtained on 2004-10-04, 2004-12-15, and 2005-01-06). To mitigate the effect of outliers in the training sets, we consider samples whose values are less than the $(1-\eta)$ quantile. We then use the algorithm to compute the detection rate from the real Cloak traces, and the false alarm rate from two other sets of LBNL data (those obtained on 2004-12-16 and 2005-01-07). Figure 6.10(b) plots the experimental CDF of I_{AD} for the five sets of LBNL data. Detailed information about these traces and related analysis could be found in [204]. We have analyzed the TCP flows that belong to eight application protocols and have at least ten packets, because different applications may affect the detection result. The selected application protocols could be classified into five categories [204]: Web (HTTP, HTTPs), Email (SMTP, IMAP4 w/o SSL, POP3 w/o SSL), Bulk (FTP), Interactive (SSH, Telnet), Streaming (RTSP).

Figure 6.12(a) plots the false alarm rate that is defined as the number of legitimate flows that are mistakenly identified as Cloak's flows to the total number of flows. The parameters ρ and η affect both the false alarm rate and detection rate. Overall, the false alarm rates are quite small. On the other hand, we define the detection rate as the ratio of the number of identified Cloak's flows to the total number of Cloak's flows. Figures 6.12(b)-6.12(c) show the detection rates obtained from different locations with the normal codewords and D-limited codewords, respectively. The figures show that the algorithm still could not detect all TCP flows belonging to Cloak; however, the detection rates are quite high overall.

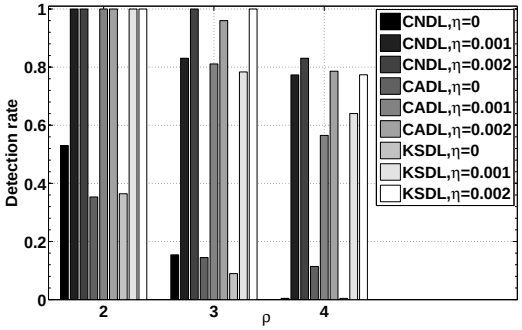
The purpose of the second step is to further increase the detection rate and decrease the number of false alarm. In this step, an active tester will first prepare a time sequence denoted as $\Phi = \{\phi_1, \phi_2, \ldots\}$. At the beginning of the second step, it will randomly select a flow from a set of suspicious flows that have been identified in the first step. The active tester then introduces delay to the selected flow by deferring



(a) False alarm rate.



(b) Detecting unbalanced codewords under different η and ρ.



(c) Detecting D-limited codewords under different η and ρ.

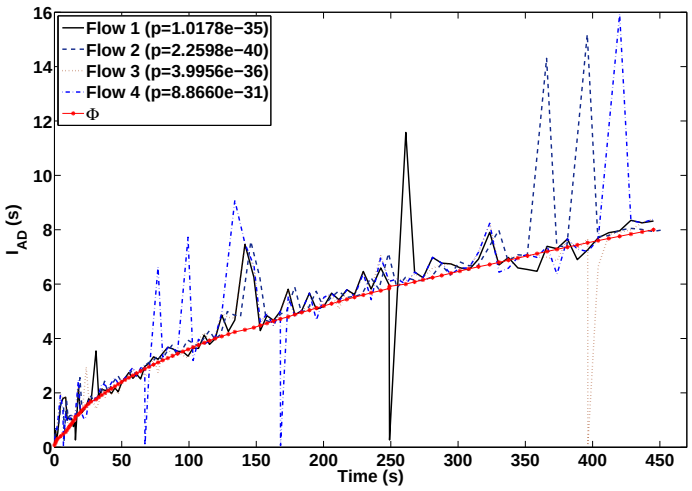
Fig. 6.12. Experiment results for the passive detection in the first step.

the transmission of the flow's packets. The delay duration is equal to ϕ_i, and the times of deferring packets are arbitrary, for example, every W_P packets or every W_T seconds. Since all flows will not send packets for a new message if at least one flow has not finished its transmission for the current message, the artificial delay caused by Φ will inflate all flows' I_{AD}s which consequently increase the detection rate of the CUSUM algorithm.

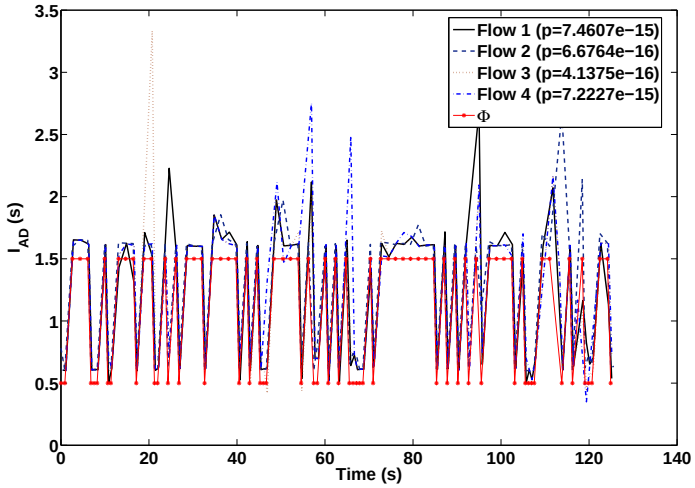
On the other hand, in order to decrease the number of false alarm, the active tester tests whether the newly captured flow is closely related to the flow that has been inserted delay. If these two flows are not closely related, then at least one flow does not belong to a Cloak channel, assuming that there is only one Cloak

channel. To measure whether these two flows are closely related, we will compute the *nonparametric* correlation coefficients between Φ and a new sequence, denoted as Ξ. Ξ is a sequence whose members are selected from the newly captured flow's I_{AD}s. The selected criteria is that the I_{AD} is the first I_{AD} observed at the end of the injected delay. Therefore, the tester will finally have two sequences, Φ and Ξ, and it will form a NULL hypothesis (i.e., there is no correlation between Φ and Ξ). In the experiments, we employ the existing *corr* function in MATLAB to compute Kendall's τ, because this function will also compute the p-value for hypothesis testing [205]. If the obtained p-value is less than 0.05, then we could reject the NULL hypothesis and regard the newly captured flow as a Cloak's puppet; otherwise, this flow is regarded as a legitimate flow.

There are various methods to generate Φ. Here, we employ two and illustrate their results in Figure 6.13. One is to construct an increasing sequence whose initial value is zero and the increasing step is equal to RTT. Figure 6.13(a) shows an example of actively testing a Cloak(14,4) channel between CN and the host in the campus. We can observe that the Ξ sequences belonging to different flows are controlled by the Φ sequence. Moreover, based on the very small p-values, we could confidently reject the NULL hypothesis. In other words, we could successfully detect the Cloak's flows because of the obvious relationship between Φ and Ξs. The other one is to use the maximum-length L level shift register sequences, or m-sequences, because of its best autocorrelation [206]. Figure 6.13(b) illustrates another example for testing a Cloak(14,4) channel between JP and the host. In this example, Φ is an m-sequence with 2 levels. We could clearly infer the close relationship between the Ξ sequences obtained from Cloak's 4 TCP flows and the Φ sequence from the figure and the related p-values. Using the active testing algorithm, we have re-conducted the detection experiments and have found that *all* flows controlled by Cloak could be identified with very small p-value (less than $1e-10$). On the downside, the active testing might degrade the legitimate flows' performance, because their RTTs will be prolonged.



(a) Monotonically increasing sequences.



(b) m-sequences with 2 levels.

Fig. 6.13. Examples of the active testing in the detection's second step.

6.6 Comparing the time cost and packet cost of IPTime, JitterBug, TCP-Script and Cloak

For the time cost, both TCPScript and Cloak require at least one RTT to deliver one symbol, depending on the design and implementation. Here, we use symbol to refer to a message unit. For example, the B-TCPScript requires one RTT to transmit

one symbol that consists of m packets ($1 \leq m \leq M$). Cloak(RAW) also needs only one RTT to transmit one symbol consisting of N packets distributed over X flows. However, the time cost may not affect TCPScript's and Cloak's performance in terms of camouflage capability and goodput because of the following reasons:

1. By imitating TCP's packet-level behavior (i.e. almost regular bursts in every RTT as shown in Fig.6.14), both TCPScript and Cloak may evade existing detection schemes for network timing channels, for example, those based on inter-packet delay [41, 42].

2. By taking the advantage of TCP's feedback control mechanism, the TCPScript can significantly increase its robustness, and Cloak can provide reliable transmission service similar to TCP's.

3. Note that a symbol in TCPScript or Cloak usually carries more than one-bit information, whereas a symbol in the IPTime channel and JitterBug deliver only one-bit information.

Other network timing channels also have time cost. For example, the IPTime channel needs an interval to delimit consecutive messages [41], and Jitterbug needs a short (or long) interval to convey messages. If there is no noise in the Internet from packet loss, delay, jitter, and reordering, their time cost is smaller than TCPScript's and Cloak's as all can transmit covert messages reliably. Unfortunately, since these noises are prevalent phenomena that will cause decoding errors, their small time costs will not increase goodput. Moreover, the IPTime channel's and Jitterbug's traffic patterns do not resemble the normal behavior of TCP, the dominant protocol supporting 90% - 95% Internet traffic [207]. Therefore, it is not difficult to identify IPTime channel and Jitterbug when they are embedded into TCP flows.

We define the packet cost, denoted by $\mathbb{C}_{pkt}$, as the number of packets needed to convey one bit of information by assuming that the covert messages are uniformly distributed (i.e. all codewords will appear with equal probability.)

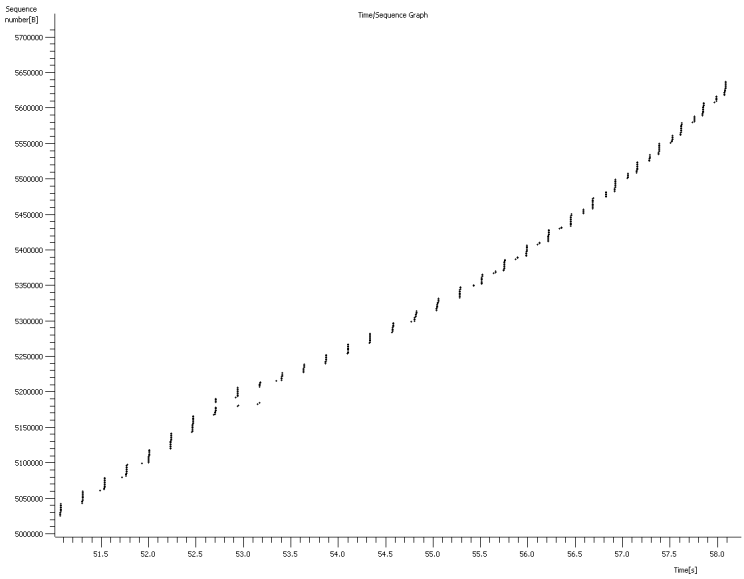


Fig. 6.14. TCP's packet-level behavior observed from a real traffic trace.

1. For the IPTime channel, its $\mathbb{C}_{pkt} = \frac{1}{2}$, because if the encoder will convey a T-bit covert message that has $\frac{T}{2}$ number of `1`, then the encoder will send only $\frac{T}{2}$ packets.

2. For Jitterbug, its $\mathbb{C}_{pkt} \approx 1$, because if the encoder will convey a T-bit covert message, then it will send $T + 1$ packets and $\lim_{T\to\infty} \frac{T+1}{T} = 1$. Figure 6.15 illustrates some of its $\mathbb{C}_{pkt}$ values.

3. For TCPScript, its $\mathbb{C}_{pkt} = \frac{1+M}{2log_2 M}$. Any m $(1 \leq m \leq M)$ packets can deliver $log_2 M$ bits of information, and the corresponding packet cost is therefore $\frac{m}{log_2 M}$. The probability of transmitting m packets is $\frac{1}{M}$. Hence $\mathbb{C}_{pkt} = \sum_{m=1}^{M} \frac{1}{M} \frac{m}{log_2 M} = \frac{1+M}{2log_2 M}$. As shown in Figure 6.16, TCPScript achieves the minimal $\mathbb{C}_{pkt} = 1.25$ when $M = 4$.

4. Among Cloak's ten variants, some of them have low packet cost, whereas others have high packet cost. Here, we investigate only the simplest one Cloak[1], whose $\mathbb{C}_{pkt} = \frac{1}{log_2 X}$, because it uses N packets to transmit $Nlog_2 X$ bits of information.

Figure 6.17 illustrates some of its $\mathbb{C}_{pkt}$ values. We can see that its packet cost decreases quickly with the number of flows (X). Therefore, by carefully choosing the parameters, Cloak can enjoy the lowest packet cost.

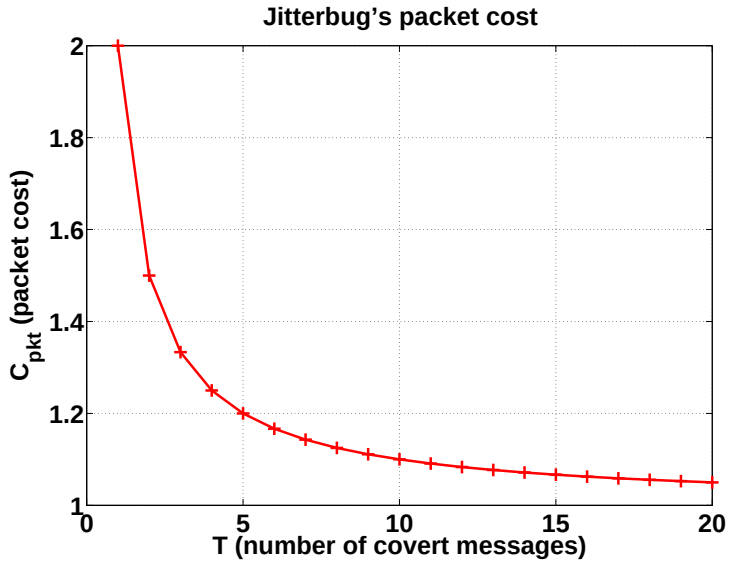


Fig. 6.15. Jitterbug's packet cost verses the number of covert messages sent.

6.7 Summary

In this chapter, we have proposed Cloak, a new class of timing channels. The major design choices responsible for Cloak's attractive properties are the use of TCP as the cloaking medium, and the exploitation of enumerative combinatorics to encode a message into multiple TCP flows and a fixed number of TCP packets. The former provides the needed reliability for free, while the latter facilitates the use of the Twelvefold Way to increase the channel data rate and avoid general decoding problems in network timing channels. Cloak does not suffer from traffic normalization if packets are indistinguishable, because they do not require modification of header field for making the packets distinguishable. On the other hand, traffic normalization

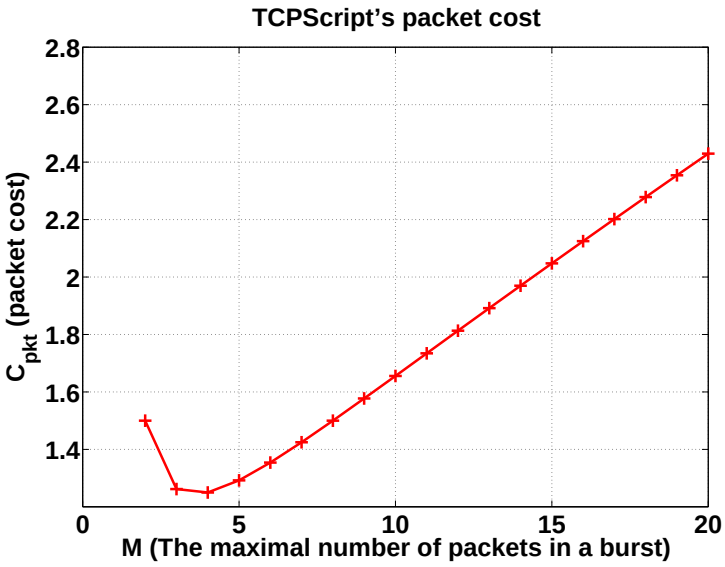


Fig. 6.16. TCPScript's packet cost under different values of M.

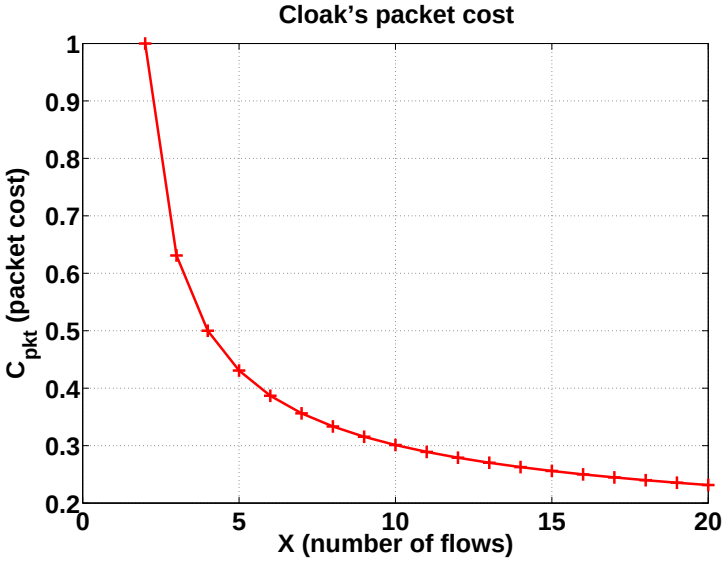


Fig. 6.17. Cloak's packet cost under different values of X.

could not completely remove the marks of distinguishable flows, e.g., be the order of connection establishment. Moreover, none of Cloak's variants will suffer from traffic

shaping, because the underlying TCP will handle this issue for Cloak. We have implemented the Cloak encoder and decoder, and evaluated its goodput under controlled environment and in the wild. Moreover, we have designed and evaluated a two-step detection algorithm for Cloak.

7. CONCLUSIONS AND FUTURE WORKS

In this thesis, we have exploited TCP's congestion control algorithms and protocol features to design more intelligent DoS attacks and effective covert channels. The basic idea of the PDoS attacks is to send intermittent false feedback signals to the TCP sender. We have shown analytically and experimentally that a sequence of properly spaced false signals is sufficient for achieving similar effects as for a traditional flooding-based attack. Moreover, this type of low-rate attack is much harder to detect. On the other hand, the TCPScript and Cloak design are fundamentally different from other network timing channels. TCPScript requires only a single TCP flow, whereas Cloak generally requires multiple TCP flows. Contrary to the common belief that unreliability is inherent in network timing channels, we have demonstrated that timing channels can be made very reliable (TCPScript) or completely reliable (Cloak). Moreover, both TCPScript and Cloak offer a much higher data rate and flexibility than other timing channels.

7.1 Pulsing denial-of-service attacks

The PDoS attacks represent a new generation of DoS attacks. Instead of exhausting the network bandwidth (which requires a lot of packets) and exhausting system resources (which requires special packets), the PDoS attacks the feedback channel of an end-to-end protocol. TCP's feedback channel is unreliable and insecure; TCP senders are not equipped for detecting false feedback signals. A PDoS attack generates a sequence of such signals by inducing packet losses at routers. By not receiving ACKs or enough duplicate ACKs, the victim TCP senders will decrease their congestion windows. Since it takes a much longer time to restore the window than that to drop it, the attack will still be very effective even if the attack pulses are spaced out.

Although we have considered only TCP in this thesis, it is not difficult to see that SCTP and DCCP are also vulnerable to PDoS attacks, because they are designed to be TCP-friendly.

To provide a unified framework for analyzing different types of DoS attacks, we have proposed polymorphic DoS (PMDoS) attacks: DoS attacks in different forms. The PMDoS attack generalizes the PDoS attacks, other low-rate attacks, such as shrew and RoQ attacks, and flooding-based attacks. We have analyzed an AIMD-based PMDoS attack for which the attack periods are modeled as an alternating renewal process. On another front, we have studied the problem of optimizing PDoS attacks. Unlike the traditional DoS attacks, PDoS attacks could be launched with different attack objectives. Some attacks may be more aggressive in intensifying the attack impact, whereas others may attempt to evade detection by reducing the attack intensity. Based on our analysis, we have characterized three different classes of attacks and obtained their optimized attack parameters.

Since the attack pulses are short in duration, it is not easy to detect their presence. In this thesis, we have proposed two new detection mechanisms: a two-stage detection system for PDoS attacks and Vanguard for PMDoS attacks. The two-stage system is designed for detecting PDoS attacks at the protected networks. The first stage processes the TCP data and ACK traffic using wavelet transforms, and the second stage detects change points which could be induced by a PDoS attack. Vanguard, on the other hand, is designed to achieve a more ambitious goal of detecting various PMDoS attacks and flooding-based attacks. It performs detection based on three traffic anomalies which, we believe, is a minimal set for maintaining low false positive and false negative rates. We have evaluated them on a test bed under various network conditions and attack scenarios; both detection systems are computationally efficient and accurate.

7.1.1 Future works

To further understand various PDoS attacks, we will focus on aperiodic PDoS attacks, because their irregular patterns may evade many existing detection schemes that assume a fixed period. We will explore new models to evaluate the impact of aperiodic PDoS attacks and design new detection schemes. We conjecture that PDoS attacks based on other kinds of stochastic process can also produce such damage to the TCP flows. In particular, we will investigate the impact of PDoS attacks characterized by long range dependent stochastic processes (e.g., fractional autoregressive moving average(FARIMA) and fractional Brownian), because they resemble normal network traffic [208].

Moreover, all the existing detection schemes are located on the receiver side or near to the bottleneck router. We will design detection mechanisms on the sender side which will allow the sender to choose another path if the working path is under attack. Besides attack detection, we will design and evaluate new defense mechanisms for the PDoS attacks. Existing aggregate-level defense schemes may also punish legitimate TCP flows, whereas flow-level defense schemes may miss those PDoS attacks that employ lots of malicious flows [83]. One possible approach is therefore to isolate and protect legitimate TCP flows and to limit other flows by using aggregate-level mechanisms. Another promising approach is the packet score mechanism [209].

Moreover, we will explore the possibility of launching PDoS attacks in wireless, ad hoc, and sensor networks. Some previous works [96, 97, 210] have already demonstrated this possibility to certain extents. We will investigate new approaches to mount the PDoS or PDoS-like attacks in those networks and evaluate their impacts from both theoretical and practical aspects. Moreover, we will consider tradeoffs between attack power and attack cost, and design countermeasures to protect these networks from attacks.

Another promising avenue is to understand PDoS attacks from control theory. We will explore how to design robust feedback control system that will protect or verify

feedback signals and consider tradeoff between system responsiveness and system's total performance. Besides, we will study the impact of PDoS-like attacks on existing feedback control systems, for example, autonomous systems that adopt different kinds of feedback controllers.

Last but not least, due to possible law violations, we have not conducted PDoS attacks in real networks and evaluated the detection schemes. Therefore, we have launched PDoS attacks only in our test bed with artificial background traffic and evaluated the detection schemes. Moreover, we used real traces from WIDE [192] and LBNL [182] to estimate the false alarm rates of the detection schemes. We are aware of the limitations of using artificial traffic and the small scale of test bed. We are also aware that some other research groups have carried out similar experiments in the Emulab and the DETER test bed [53, 55]. We would like to cooperate with them on evaluating both the impact of PDoS attacks and the performance of the detection schemes.

7.2 Network covert channels

In the second class of TCP exploits, we have shown how to camouflage covert messages in the "normal" behavior of TCP traffic. By exploiting TCP's flow concept, sliding window, and acknowledgement mechanisms, TCPScript and Cloak generally provide higher capacity, more camouflage approaches, and higher resilience against adverse network conditions. Since these mechanisms are widely adopted by modern transport protocols, the basic design approaches can also be applied to other transport protocols.

TCPScript camouflages covert messages under a TCP flow's bursty traffic. It encodes messages into the number of packets in each data burst. Moreover, TCP's acknowledgement mechanism helps TCPScript increase its reliability. To analyze its capacity, we have performed an information-theoretic analysis of TCPScript. The analytical results show that TCPScript is more robust to packet loss than other re-

cently proposed timing channels. Moreover, we have conducted extensive experiments in a test bed and the PlanetLab platform. The experiment results also support that TCPScript provides higher throughput and is more robust to adverse network conditions. To defend against TCPScript, we propose a new detection scheme based on two statistics: burst size and inter-ACK-data delay. Empirical results show that this detection scheme could successfully detect most TCPScript channels, especially the more aggressive ones.

As another new design point using multiple TCP flows, Cloak encodes covert messages into the combinations of flows and packets which could be distinguishable or indistinguishable. Cloak possesses the unique advantage of guaranteeing reliable covert communications which cannot be attained by any existing timing channel. Moreover, it offers ten variants which achieve different scales of tradeoff between capacity and concealment. Therefore, Cloak could be used for different objectives of the covert communications. We have designed ten pairs of ranking and unranking algorithms to efficiently perform encoding and decoding. We have also proposed a passive scheme and an active scheme to detect Cloak channels. The former uses inter-ACK-data delay to uncover Cloak instances that use unbalanced codes. The latter detects Cloak by inserting a special sequence of delays into the suspicious TCP flows. The disadvantage of the active approach is a possible degradation on the performance of the legitimate flows.

7.2.1 Future works

Since the features exploited by TCPScript are widely used in other protocols, we will first explore possible covert channels in other protocols. Moreover, we will propose a unified approach, accompanied by a formal analysis, for the class of newly identified covert channels. Another avenue is to model the capacity of TCPScript-like network covert channels under different warden models. On the other hand, we will derive a general combination-based framework for designing covert channels and

apply it to other steganography problems. For example, Cloak's flexibility is based on two parameters: the number of packets for a codeword (N) and the number of flows (K). A more general framework requires only N for constructing new covert channels. Removing the need of fixing the number of flows will further increase Cloak's stealthiness.

On the defense side, we will develop new detection algorithms and defense mechanisms for TCPScript, Cloak, and their variants. The detection algorithms should take into consideration both the detection performance in terms of detection rate and false alarm rate,and the implementation cost when handling a large number of TCP flows. Moreover, the new defense mechanisms will help wardens throttle the covert channel capacity while incurring only minimal overheads. We will also develop a penalty model for the scenario where a legitimate flow is mistakenly labeled as a covert channel and consequently suffers from very low throughput.

A. DISCRETE WAVELET TRANSFORM

Wavelet transform is very suitable for analyzing irregular signals, such as network traffic, because it gives a more accurate local description of signal characteristics in both time and frequency domains. Indeed, wavelet transform has been applied to analyze network traffic and identify traffic anomalies. For example, wavelet analysis has been employed to identify traffic anomalies caused by flooding-based DoS and flash crowds through a deviation score [211]. Compared with the work in [211], there are two main differences in our wavelet analysis. First, the wavelet analysis there is used to perform a postmortem analysis of trace data, whereas ours concentrates on a real-time analysis of incoming data. Second, the analysis there considers only the signal variations in the high and medium frequency bands that are not sufficient to detect the PDoS attack. Our analysis requires both high and low frequency bands.

The discrete wavelet transform (DWT) represents a signal $f(t) \in L^2(R)$ using scaling functions $\varphi_{j,k}(t)$, and a translated and dilated version of wavelet functions $\psi_{j,k}(t)$:

$$f(t) = \sum_k c_{j_0}(k)\varphi_{j_0,k}(t) + \sum_k \sum_{j=j_0} d_j(k)\psi_{j,k}(t), \tag{A.1}$$

where $\{\varphi_{j,k}(t) = 2^{j/2}\varphi(2^j t - k),\ j,k \in Z\}$ and $\{\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k),\ j,k \in Z\}$. In this expansion, the first summation describes a coarse approximation of $f(t)$, and the second summation depicts the details of $f(t)$. In practice, the coefficients $c_j(k)$ and $d_j(k)$ are calculated via the Mallat's pyramid algorithm:

$$c_j(k) = \sum_m h_0(m - 2k)c_{j+1}(m), \tag{A.2}$$

$$d_j(k) = \sum_m h_1(m - 2k)c_{j+1}(m), \tag{A.3}$$

where h_0 and h_1 are the coefficients of low-pass and high-pass filters, respectively. If the scaling functions and wavelet functions form an orthonormal basis, Parseval's theorem states that $f(t)$'s energy is equal to the energy in its scaling coefficients and wavelet coefficients [212]. That is,

$$\int |f(t)|^2 dt = \sum_k |c_{j_0}(k)|^2 + \sum_k \sum_{j=j_0} |d_j(k)|^2. \tag{A.4}$$

Since the wavelet functions operate like high-pass filters that use narrow time windows to compute differences in signals [173], they can capture the variability of the incoming traffic volumes. On the other hand, the scaling functions perform like low-pass filters; therefore, they can be used to extract the trend of the outgoing TCP ACK traffic.

In order to realize an on-line detection, we use a moving window to group W continuous samples for the computation of DWT. Let $S = s(t), t \geq 1$, be the traffic samples, and $S_W(n) = \{s(t)\}_{t=(n-1)W+1}^{n \times W}, n \geq 1$, be the sequential windows of the samples. We use S^{In} and S^{Out} to denote the traffic samples for the incoming data traffic and outgoing ACK traffic, respectively. We also use $S_W^{In}(n)$ and $S_W^{Out}(n)$ to refer to the observation periods for the two respective cases.

Since the fluctuation of the incoming traffic can be captured by its high-frequency part, we continuously process $S_W^{In}(n)$ through the DWT and obtain their wavelet coefficients $d_{j,k}^{In}$. In order to quantify the degree of variability, we define a statistic based on the signal energy:

$$E_H(n) = \frac{1}{W} \sum_k |d_{1,k}^{In}|^2, \tag{A.5}$$

where $d_{1,k}^{In}$ is the wavelet coefficient at the finest scale ($j = 1$). A similar approach was used in [213] to investigate the scaling properties of the network traffic.

On the other hand, we process $S_W^{Out}(n)$ to obtain the trend of the outgoing TCP ACK traffic. We also define a statistic based on the signal energy to represent the trend of the outgoing TCP ACK traffic:

$$E_L(n) = \frac{1}{W} \sum_k |c_{L,k}^{Out}|^2, \tag{A.6}$$

where $c_{L,k}^{Out}$ is the scaling coefficient at the highest decomposed scale ($j = L$).

B. NONPARAMETRIC CUSUM ALGORITHM FOR CHANGE-POINT DETECTION

In order to automatically locate the change point in the statistics E_H and E_L as soon as possible, we apply the nonparametric sequential detection algorithm at the end of every observation period. Here, we employ the nonparametric CUSUM algorithm for this purpose. This algorithm has also been used in other detection methods for (D)DoS attacks [214–216].

The formal definition of the nonparametric CUSUM algorithm is summarized as follows [179, 217]:

$$y(n) = (y(n-1) + x(n))^+, \quad y(0) \equiv 0, \ n = 1, 2, \ldots, \tag{B.1}$$

where $(y(n))^+$ is equal to $y(n)$ if $y(n) > 0$, and 0, otherwise. Its decision rule is:

$$d_N(\cdot) = d_N(y(n)) = I(y(n) > C_{cusum}), \tag{B.2}$$

where C_{cusum} is the threshold. $x(n)$ is defined on the probability space $(\Omega, \mathcal{F}, P)$ by the model

$$x(n) = a + h(n)I(n \geq m) + \xi(n), \tag{B.3}$$

where $\xi = \{\xi(n)\}_{n=1}^{\infty}$ is the random sequence such that its mathematical expectation $\overline{\xi(n)} \equiv 0$, and $\{h(n)\}$ is the deterministic sequence representing the profile of changes that take place at the moment m [179]. As suggested in [216, 217], we calculate the threshold C_{cusum} by

$$C_{cusum} = (\tau - m)^+(h - \|a\|), \quad if \ m \geq 1, \tag{B.4}$$

where τ is the preferred detection time.

The CUSUM method assumes that $a < 0$ and $h + a > 0$, which together imply that the mean value of $x(n)$ will change from negative to positive when a change

occurs. Therefore, it is necessary to first transform the statistics under the change-point detection to new random sequences which have negative mean values under normal conditions.

C. THE TEST BED'S TOPOLOGY

The test bed has evolved since the inception of our research on PDoS attacks and network covert channels. Currently, its topology is shown in Figure C.1. The test bed has four software Border Gateway Protocol (BGP) routers that are Linux machines running Quagga v0.99.6 to provide BGP routing service. Therefore, we could also examine the impact of PDoS attacks on BGP and routing [54]. Both the background traffic and victim TCP flows are generated by IPerf v2.0.2 between two pairs of machines. Since the minimal RTO is 200ms in Linux, we have also prepared a configuration that sets the victim TCP sender's minimal RTO to 1 second. The purpose is to evaluate the effect of Shrew attack that exploits a fixed minimal RTO value and other kinds of PDoS attacks.

We have implemented the attack program based on WinPcap [166]. We have deployed Dummynet [164] and NIST [218] in the bottleneck to emulate different network conditions and evaluate the impact of PDoS attacks on different queue management schemes. The buffer size is set to RTT * bottleneck's capacity. We capture both incoming TCP data traffic and outgoing TCP ACK traffic at router AS65003, based on which we estimate the throughput degradation caused by PDoS attacks and evaluate the detection schemes. For the experiments on network covert channels, the test bed's setting is shown in Figure C.2. We locate the decoder close to router AS65003 to sniff TCP connections between the encoder and the legitimate server.

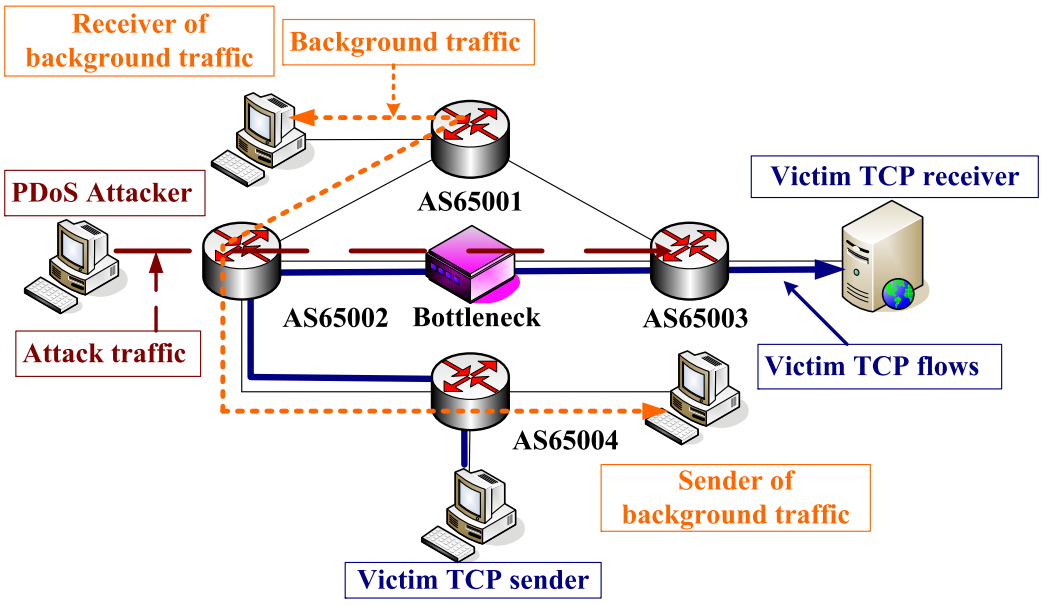


Fig. C.1. The test bed's topology for PDoS attack experiments.

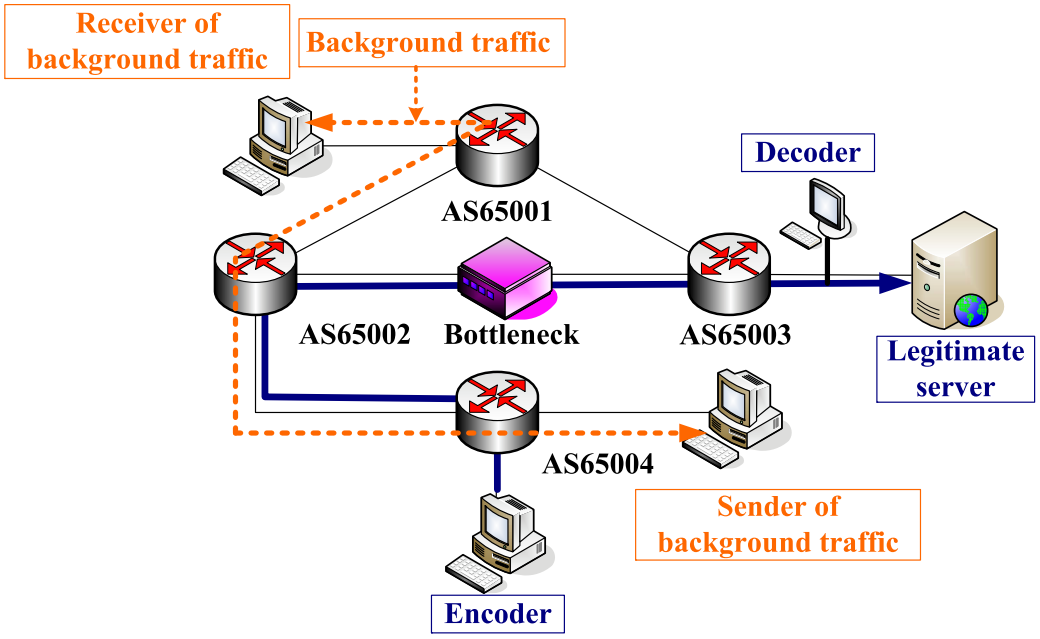


Fig. C.2. The test bed's topology for network covert channel experiments.

LIST OF REFERENCES

[1] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *ACM Computer Communication Review*, October 2005.

[2] L. Gordon, M. Loeb, W. Lucyshyn, and R. Richardson, "CSI/FBI computer crime and security survey." http://www.gocsi.com, 2004-2006.

[3] Symantec Corporation, "Symantec Internet security threat report volume xi." http://www.symantec.com/enterprise/theme.jsp?themeid=threatreport, March 2002-2007.

[4] X. Luo and R. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Proc. Network and Distributed System Security Symposium (NDSS)*, February 2005.

[5] A. Young and M. Yung, "Deniable password snatching: on the possibility of evasive electronic espionage," in *Proc. IEEE Symposium Security and Privacy*, 1997.

[6] NISCC Monthly Bulletin, "Phishing trojan plays ping-pong with captured data." www.niscc.gov.uk/niscc/docs/re-20060831-00629.pdf, August 2006.

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

[8] A. Singh, O. Nordstro, C. Lu, and A. Santos, "Malicious ICMP tunneling: Defense against the vulnerability," in *Proc. Australasian Conf. Information Security and Privacy*, 2003.

[9] S. Schechter and M. Smith, "Access for sale: A new class of worm," in *Proc. ACM Workshop on Rapid Malcode (WORM)*, 2003.

[10] F. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *Proc. European Symposium on Research in Computer Security (ESORICS)*, 2005.

[11] J. Fernandez and P. Bureau, "Optimising malware," in *Proc. International Swarm Intelligence & Other Forms of Malware Workshop (Malware'06)*, 2006.

[12] E. Casey, "Next-generation cyber forensics: Investigating sophisticated security breaches," *Communications of the ACM*, February 2006.

[13] M. Bauer, "New covert channels in HTTP: Adding unwitting Web browsers to anonymity sets," in *Proc. ACM Workshop on Privacy in the Electronic Society*, 2003.

[14] K. Borders and A. Prakash, "Web Tap: Detecting covert Web traffic," in *Proc. ACM CCS*, 2004.

[15] S. Floyd, "Congestion control principles." RFC 2914, September 2000.

[16] G. Ramachandran, S. Shalunov, A. Morton, L. Ciavattone, and J. Perser, "Transmission control protocol." RFC 793, IETF, September 1981.

[17] B. Harris and R. Hunt, "TCP/IP security threats and attack methods," *Computer Communicastions*, vol. 22, 1999.

[18] S. Bellovin, "Security problems in the TCP/IP protocol suite," *Computer Communications Review*, April 1989.

[19] S. Bellovin, "A look back at 'security problems in the TCP/IP protocol suite'," in *Proc. Annual Computer Security Applications Conference (ACSAC)*, December 2004.

[20] S. Young and D. Aitel, *The Hacker's Handbook: The Strategy Behind Breaking into and Defending Networks*. Auerbach, 2003.

[21] P. Mateti, "Security issues in the TCP/IP suite." http://www.cs.wright.edu/ pmateti/InternetSecurity/Lectures/TCPexploits/sec-tcpip.pdf, 2007.

[22] R. Chang, "Defending against flooding-based, distributed denial-of-service attacks: A tutorial," *IEEE Communications Magazine*, vol. 40, no. 10, 2002.

[23] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proc. IEEE Symposium Security and Privacy*, 1997.

[24] W. Eddy, "TCP SYN flooding attacks and common mitigations." IETF draft-ietf-tcpm-syn-flood-04, May 2007.

[25] B. Guha and B. Mukherjee, "Network security via reverse engineering of TCP code: Vulnerability analysis and proposed solutions," *IEEE Network*, vol. 11, 1997.

[26] F. Gont, "ICMP attacks against TCP." IETF draft-ietf-tcpm-icmp-attacks-02, May 2007.

[27] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control." RFC 2581, April 1999.

[28] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted denial of service attacks," in *Proc. ACM SIGCOMM*, August 2003.

[29] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the transients of adaptation for RoQ attacks on Internet resources," in *Proc. IEEE ICNP*, 2004.

[30] V. Paxson and M. Allman, "Computing TCP's retransmission timer." RFC 2988, November 2000.

[31] R. Anderson and F. Petitcolas, "On the limits of steganography," *IEEE Journal of Selected Areas in Communications*, vol. 16, no. 4, 1998.

[32] B. Lampson, "A note on the confinement problem," *Communications of the ACM*, October 1973.

[33] DoD US, "Department of defense trusted computer system evaluation criteria (orange book)," Tech. Rep. DoD 5200.28-STD, National Computer Security Center, December 1985.

[34] V. Gligor, "A guide to understanding covert channel analysis of trusted systems (light pink book)," Technical Report NCSC-TG-030, National Computer Security Center, November 1993.

[35] I. Moskowitz and M. Kang, "Covert channels – here to stay?," in *Proc. COMPASS*, 1994.

[36] B. Venkatraman and R. Newman-Wolfe, "Capacity estimation and auditability of network covert channels," in *Proc. IEEE Symposium Security and Privacy*, 1995.

[37] S. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proc. Information Hiding Workshop*, 2005.

[38] C. Girling, "Covert channels in LAN's," *IEEE Transactions on Software Engineering*, February 1987.

[39] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating steganography in Internet traffic with active wardens," in *Proc. Information Hiding Workshop*, 2002.

[40] D. Watson, M. Smart, G. Malan, and F. Jahanian, "Protocol scrubbing: Network security through transparent flow modification," *IEEE/ACM Transactions on Networking*, 2004.

[41] S. Cabuk, C. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *Proc. ACM CCS*, 2004.

[42] V. Berk, A. Giani, and G. Cybenko, "Detection of covert channel encoding in network packet delays," Technical Report TR2005536, Department of Computer Science, Dartmouth College, 2005.

[43] G. Shah, A. Molina, and M. Blaze, "Keyboards and covert channels," in *Proc. USENIX Security*, 2006.

[44] PlanetLab, "home page." http://www.planet-lab.org/.

[45] X. Luo, R. Chang, and E. Chan, "Performance analysis of TCP/AQM under denial-of-service attacks," in *Proc. IEEE MASCOTS*, 2005.

[46] X. Luo, E. Chan, and R. Chang, "Vanguard: A new detection scheme for a class of TCP-targeted denial-of-service attacks," in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006.

[47] X. Luo and R. Chang, "Optimizing the pulsing denial-of-service attacks," in *Proc. Intl. Conf. Dependable Systems and Networks (DSN)*, 2005.

[48] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream control transmission protocol." RFC 2960, October 2000.

[49] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (DCCP)." RFC 4340, March 2006.

[50] D. Wei, C. Jin, S. Low and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, February 2007.

[51] K. Dong, S. Yang, and S. Wang, "Analysis of low-rate TCP DoS attack against FAST TCP," in *Proc. Intl. Conf. on Intelligent Systems Design and Applications*, 2006.

[52] A. Shevtekar and N. Ansari, "Do low rate DoS attacks affect QoS sensitive VoIP traffic?," in *Proc. IEEE ICC*, 2006.

[53] R. Chertov, S. Fahmy, and N. Shroff, "Emulation versus simulation: A case study of TCP-targeted denial of service attacks," in *Proc. IEEE/CreateNet Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, 2006.

[54] Y. Zhang, Z. Mao, and J. Wang, "Low-rate TCP-targeted DoS attacks disrupts Internet routing," in *Proc. Network and Distributed System Security Symposium (NDSS)*, February 2007.

[55] J. Mirkovic, A. Hussain, B. Wilson, S. Fahmy, P. Reiher, R. Thomas, W. Yao, and S. Schwab, "Towards user-centric metrics for denial-of-service measurement," in *Proc. Workshop on Experimental Computer Science*, 2007.

[56] X. Luo, E. Chan, and R. Chang, "Cloak: A ten-fold way for reliable covert communications," in *Proc. European Symposium on Research in Computer Security (ESORICS)*, 2007.

[57] M. Gouda, *Elements of Network Protocol Design*. John Wiley & Sons, Inc., 1998.

[58] M. Duke, R. Braden, W. Eddy, E. Blanton, "A roadmap for transmission control protocol (TCP) specification." RFC 4614, September 2006.

[59] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control." IETF draft-ietf-tcpm-rfc2581bis-02, February 2007.

[60] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, August 1993.

[61] S. Cai, Y. Liu, and W. Gong, "Client-controlled slow TCP and denial of service," in *Proc. IEEE Conference on Decision and Control (CDC)*, 2004.

[62] R. Stewart and M. Dalal, "Improving TCP's robustness to blind in-window attacks." IETF draft-ietf-tcpm-tcpsecure-07, February 2007.

[63] J. Touch, "Defending TCP against spoofing attacks." IETF draft-ietf-tcpm-tcp-antispoof-06, February 2007.

[64] L. Joncheray, "A simple active attack against TCP," in *Proc. USENIX Security Symposium*, 1995.

[65] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.1." RFC 4346, April 2007.

[66] S. Sanfilippo, "hping." http://www.hping.org/, visited on 2007.

[67] P. Biondi, "scapy." http://www.secdev.org/projects/scapy/, visited on 2007.

[68] M. Grimes and J. Nathan, "nemesis." http://www.packetfactory.net/projects/nemesis/, visited on 2007.

[69] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz, "Known TCP implementation problems." RFC 2525, IETF, March 1999.

[70] R. Beverly, "A robust classifier for passive TCP/IP fingerprinting," in *Proc. Passive and Active Measurement Workshop (PAM)*, 2004.

[71] Fyodor, "Remote OS detection via TCP/IP fingerprinting (2nd generation)." http://insecure.org/nmap/osdetect/, visited on 2007.

[72] Fyodor, "Nmap." http://insecure.org/nmap/, visited on 2007.

[73] F. Yarochkin, M. Kydyraliev and O. Arkin, "xprobe." http://sys-security.com/blog/xprobe2/, visited on 2007.

[74] M. Zalewski, "p0f." http://lcamtuf.coredump.cx/p0f.shtml, visited on 2007.

[75] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "TCP congestion control with a misbehaving receiver," *ACM Computer Communication Review*, October 1999.

[76] R. Sherwood, B. Bhattacharjee, and R. Braud, "Misbehaving TCP receivers can cause Internet-wide congestion collapse," in *Proc. ACM CCS*, 2005.

[77] S. Kanhere and A. Naveed, "A novel tuneable low-intensity adversarial attack," in *Proc. IEEE LCN Workshop on Network Security*, 2005.

[78] S. Taghizadeh, A. Helmy, and S. Gupta, "A systematic simulation-based study of adverse impact of short-lived TCP flows on long-lived TCP flows," in *Proc. IEEE INFOCOM*, 2005.

[79] M. Guirguis, A. Bestavros, and I. Matta, "On the impact of low-rate attacks," in *Proc. IEEE ICC*, 2006.

[80] A. Shevtekar, K. Anantharam, and N. Ansari, "Low rate TCP denial-of-service attack detection at edge routers," *IEEE Communications Letters*, April 2005.

[81] Y. Kwok, R. Tripathi, Y. Chen, and K. Hwang, "HAWK: Halting anomalies with weighted choking to rescue well-behaved TCP sessions from shrew DoS attacks," in *Proc. Intl. Conf. Computer Networks and Mobile Computing*, 2005.

[82] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proc. IEEE ICNP*, 2001.

[83] Y. Xu and R. Guerin, "On the robustness of router-based denial-of-service (DoS) defense systems," *ACM SIGCOMM Computer Communication Review*, 2005.

[84] Y. Xu and R. Guerin, "A double horizon defense design for robust regulation of malicious traffic," in *Proc. SecureComm*, 2006.

[85] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," in *Proc. ACM SIGCOMM*, 2004.

[86] S. Sarat and A. Terzis, "On the effect of router buffer sizes on low-rate denial of service attacks," in *Proc. IEEE ICCCN*, 2005.

[87] H. Sun, J. Lui, and D. Yau, "Defending against low-rate TCP attack: Dynamic detection and protection," in *Proc. IEEE ICNP*, 2004.

[88] Y. Chen, K. Hwang, and Y. Kwok, "Filtering of shrew DDoS attacks in frequency domain," in *Proc. IEEE LCN Workshop on Network Security*, 2005.

[89] Y. Chen and K. Hwang, "Collaborative detection and filtering of shrew DDoS attacks using spectral analysis," *Journal of Parallel and Distributed Computing*, no. 9, 2006.

[90] G. Yang, M. Gerla, and M. Sanadidi, "Defense against low-rate TCP-targeted denial-of-service attacks," in *Proc. IEEE ISCC*, 2004.

[91] E. Barrantes and S. Forrest, "Increasing communications security through protocol parameter diversity," in *Proc. Latin-American Conference on Informatics*, 2006.

[92] G. Maciá-Fernández, J. Díaz-Verdejo, and P. García-Teodoro, "Evaluation of a low-rate DoS attack against iterative servers," *Computer Networks*, vol. 51, 2007.

[93] M. Chan, E. Chang, L. Lu and S. Ng, "Effect of malicious synchronization," in *Proc. Applied Cryptography and Network Security (ACNS)*, 2006.

[94] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of quality (RoQ) attacks on Internet end-systems," in *Proc. IEEE INFOCOM*, 2005.

[95] M. Guirguis, A. Bestavros, I. Matta and Y. Zhang, "Reduction of quality (RoQ) attacks on dynamic load balancers: Vulnerability assessment and design tradeoffs," in *Proc. IEEE INFOCOM*, 2007.

[96] I. Aad, J. Hubaux, and E. Knightly, "Denial of service resilience in ad hoc networks," in *Proc. ACM Mobicom*, 2004.

[97] L. Perrone and S. Nelson, "A study of on-off attack models for wireless ad hoc networks," in *Proc. IEEE Workshop on Operator-Assisted (Wireless Mesh) Community Networks*, 2006.

[98] S. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proc. IEEE Symposium on Security and Privacy*, 2005.

[99] X. Fu, Y. Zhu, B. Graham, R. Bettati and W. Zhao, "On flow marking attacks in wireless anonymous communication networks," in *Proc. IEEE ICDCS*, 2005.

[100] W. Yu, X. Fu, S. Graham, D. Xuan and W. Zhao, "DSSS-based flow marking technique for invisible traceback," in *Proc. IEEE Symposium on Security and Privacy*, 2007.

[101] M. Wolf, "Covert channels in LAN protocols," in *Proc. Workshop on Local Area Network Security*, 1989.

[102] T. Handel and M. Stanford, "Hiding data in the OSI network model," in *Proc. Information Hiding Workshop*, 1996.

[103] K. Szczypiorski, "HICCUPS: Hidden communication system for corrupted networks," in *Proc. International Multi-Conference on Advanced Computer Systems*, 2003.

[104] C. Kratzer, J, Dittmann, A. Lang, and T. Kuhne, "WLAN steganography: A first practical review," in *Proc. ACM Multimedia and Security Workshop*, 2006.

[105] T. Dogu and A. Ephremides, "Covert information transmission through the use of standard collision resolution algorithms," in *Proc. Information Hiding Workshop*, 1999.

[106] S. Bhadra, S. Shakkottai, and S. Vishwanath, "Covert communication over slotted ALOHA systems," in *Proc. Allerton Conference Communication, Control, and Computing*, 2004.

[107] S. Li and A. Ephremides, "A covert channel in MAC protocols based on splitting algorithms," in *Proc. IEEE WCNC*, 2005.

[108] Z. Wang, J. Deng, and R. Lee, "Mutual anonymous communications: A new covert channel based on splitting tree MAC," in *Proc. IEEE INFOCOM (Minisymposium)*, 2007.

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

[110] N. Lucena, G. Lewandowski, and S. Chapin, "Covert channels in IPv6," in *Proc. PET Workshop*, 2005.

[111] D. Kaminsky, "OzymanDNS." http://www.doxpara.com/ozymandns_src_0.1.tgz, 2005.

[112] F. Heinz and J. Oster, "NSTX (IP-over-DNS)." http://nstx.dereference.de/nstx, 2005.

[113] T. Pietraszek, "DNScat." http://tadek.pietraszek.org/projects/DNScat, 2005.

[114] M. Handley, C. Kreibich, and V. Paxson, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *Proc. USENIX Security Symposium*, 2001.

[115] W. Hu, "Reducing timing channels with fuzzy time," *Journal of Computer Security*, 1992.

[116] J. Trostle, "Modelling a fuzzy time system," in *Proc. IEEE Symposium Security and Privacy*, 1993.

[117] J. Millen, "Covert channel capacity," in *Proc. IEEE Security and Privacy*, 1987.

[118] J. Millen, "Finite-state noiseless covert channels," in *Proc. Computer Security Foundations Workshop*, 1989.

[119] I. Moskowitz and A. Miller, "Simple timing channels," in *Proc. IEEE Symposium Security and Privacy*, 1994.

[120] I. Moskowitz, S. Greenwald, and M. Kang, "An analysis of the timed Z-Channel," *IEEE Transactions on Information Theory*, 1998.

[121] S. Arimoto, "An algorithm for calculating the capacity of an arbitrary discrete memoryless channel," *IEEE Transactions on Information Theory*, vol. IT-18, pp. 14–20, 1972.

[122] R. Blahut, "Computation of channel capacity and rate-distortion functions," *IEEE Transactions on Information Theory*, vol. IT-18, pp. 460–473, 1972.

[123] R. Yeung, *A First Course in Information Theory*. Kluwer Academic, 2002.

[124] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of Internet path properties," in *Proc. ACM SIGCOMM Internet Measurement Workshop (IMW)*, 2001.

[125] C. Abad, "IP checksum covert channels and selected hash collision." http://www.gray-world.net/papers/ipccc.pdf, 2001.

[126] A. Hintz, "Covert channels in TCP and IP headers," in *DEFCON Security Conference*, 2002.

[127] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. Workshop on Multimedia Security*, 2002.

[128] daemon9, "Loki." http://gray-world.net/papers/projectloki.txt, 1996.

[129] daemon9, "Loki 2." http://gray-world.net/papers/projectloki2.txt, 1997.

[130] R. Murphy, "Covert channels using IPv6/ICMPv6." DEFCON-14, August 2006.

[131] J. Giffen, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *Proc. PET Workshop*, 2002.

[132] J. Rutkowska, "The implementation of passive covert channels in the Linux kernel," in *Proc. Chaos Communication Congress*, 2004.

[133] N. Lucena, J. Pease, P. Yadollahpour, and S. Chapin, "Syntax and semantics-preserving application-layer protocol steganography," in *Proc. Information Hiding Workshop*, 2004.

[134] E. Jones, "Legitimate sites as covert channels: An extension to the concept of reverse HTTP tunnels.." www.gray-world.net/papers/lsacc.txt, 2000.

[135] Gray-World Team, "Covert channel and tunneling over the HTTP protocol detection: GW implementation theoretical design." http://www.gray-world.net/projects/papers/cctde.txt, 2003.

[136] Gray-World Team, "How to cook a covert channel." http://www.gray-world.net/projects/papers/cooking_channels.txt, 2006.

[137] NoCrew, "GNU HTTP tunnel." http://www.nocrew.org/software/httptunnel.html.

[138] P. Padgett, "Corkscrew." http://www.agroman.net/corkscrew/, 2001.

[139] P. Starzetz, "Ambiguities in TCP/IP - firewall bypassing." http://www.gray-world.net/papers/ambiguitiesintcpip.txt, 2002.

[140] D. Pack, W. Streilein, S. Webster, and R. Cunningham, "Detecting HTTP tunneling activities," in *Proc. IEEE Annual Information Assurance Workshop*, 2002.

[141] K. Egevang and P. Srisuresh, "The IP network address translator (NAT)." RFC 1631, May 1994.

[142] G. Danezis, "Covert communications despite traffic data retention." http://homes.esat.kuleuven.be/ gdanezis/cover.pdf, 2006.

[143] X. Luo, E. Chan, and R. Chang, "Crafting web counters into covert channels," in *Proc. IFIP International Information Security Conference (SEC)*, 2007.

[144] R. Chakinala, A. Kumarasubramanian, R. Manokaran, G. Noubir, C. Pandu Rangan, and R. Sundaram, "Steganographic communication in ordered channels," in *Proc. Information Hiding Workshop*, 2006.

[145] J. Giles and B. Hajek, "An information-theoretic and game-theoretic study of timing channels," *IEEE Transactions on Information Theory*, vol. 48, pp. 2455 – 2477, September 2002.

[146] Y. Yang and S. Lam, "General AIMD congestion control," in *Proc. IEEE ICNP*, November 2000.

[147] N. Cardwell, S. Savage and T. Anderson, "Modeling TCP latency," in *Proc. IEEE INFOCOMM*, 2000.

[148] S. Fredj, T. Bonald, A. Proutiere, G. Regnie, and J. Roberts, "Statistical bandwidth sharing: a study of congestion at flow level," in *Proc. ACM SIGCOMM*, 2001.

[149] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth, "On the Lambert W Function," *Advances in Computational Mathematics*, vol. 5, 1996.

[150] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM Computer Communication Review*, July 1997.

[151] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, September 1998.

[152] D. Cox, *Renewal Theory*. Chapman & Hall, 1962.

[153] M. Hassan and R. Jain, *High performance TCP/IP Networking: Concepts, Issues, and solutions*. Pearson Education, Inc., 2004.

[154] A. Brandt, "The stochastic equation $Y_{n+1} = A_n Y_n + B_n$ with stationary coefficients," *Advances in Applied Probability*, vol. 18, no. 1, 1986.

[155] U. Horst, "The stochastic equation $y_{t+1} = a_t y_t + b_t$ with non-stationary coefficients," *Journal of Applied Probability*, vol. 38, no. 1, 2001.

[156] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," *IEEE/ACM Transactions Networking*, vol. 13, no. 2, 2005.

[157] "NLANR/DAST: Iperf 1.7.0 - The TCP/UDP bandwidth measurement tool." http://dast.nlanr.net/Projects/Iperf/.

[158] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communication Review*, July 2002.

[159] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred, "Statistical approaches to DDOS attack detection and response," in *Proc. DARPA Information Survivability Conference and Exposition*, 2003.

[160] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets," in *Proc. ACM SIGCOMM*, 2001.

[161] C. Jin, H. Wang, and K. Shin, "Hop-count filtering: an effective defense against spoofed DDoS traffic," in *Proc. ACM CCS*, 2003.

[162] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Transactions in Networking*, vol. 11, no. 4, 2003.

[163] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm." RFC 3782, April 2004.

[164] L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, 1997.

[165] D. Wei and P. Cao, "NS-2 TCP-Linux: An NS-2 TCP implementation with congestion control algorithms from linux," in *Proc. Workshop of NS-2*, 2006.

[166] "WinPcap, the free packet capture architecture for Windows." http://winpcap.polito.it.

[167] S. Walton, *Linux Socket Programming*. Sams Publishing, 2001.

[168] K. Chandrayana and S. Kalyanaraman, "Uncooperative congestion control," in *Proc. ACM SIGMETRICS*, 2004.

[169] L. Zhang and D. Clark, "Oscillating behavior of network traffic: a case study simulation," *Internetworking: Research and Experience*, December 1990.

[170] "The network simulator - ns-2." http://www.isi.edu/nsnam/ns/.

[171] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," in *Proc. ACM SIGMOD*, May 2001.

[172] M. Weeks and M. Bayoumi, "Discrete wavelet transform: Architectures, design and performance issues.," *Journal of VLSI Signal Processing*, vol. 35, 2003.

[173] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley, MA: Wellesley-Cambridge Press, 1996.

[174] A. Ohsumi, H. Ijima, and T. Kuroishi, "Online detection of pulse sequence in random noise using a wavelet," *IEEE Trans. on Signal Processing*, vol. 47, no. 9, 1999.

[175] J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the source," in *Proc. IEEE ICNP*, 2002.

[176] C. Chatfield, *The Analysis of Time Series: An Introduction*. Chapam & Hall/CRC, sixth ed., 2003.

[177] M. Stricker and M. Orengo, "Similarity of color images," in *Proc. SPIE Conf. Storage and Retrieval for Image and Video Databases III*, 1995.

[178] A. Smeulders, M. Worring, S. Santini, A. Gupta and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[179] B. Brodsky and B. Darkhovsky, *Non-Parametric Statistical Diagnosis Problems and Methods*. Kluwer Academic Publishers, 2000.

[180] A. Hussain and J. Heidemann and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proc. ACM SIGCOMM*, 2003.

[181] A. Oppenheim, A. Willsky, and S. Nawab, *Signals and Systems*. Prentice Hall, second ed., 1996.

[182] V. Paxson, "LBNL/ICSI enterprise tracing project." http://www.icir.org/enterprise-tracing/Overview.html, 2005.

[183] I. Daubechies, *Ten Lectures on Wavelets*. Society for Industrial and Applied Mathematics, 1992.

[184] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley-Interscience, 2000.

[185] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. SPIE/ACM MMCN*, 2002.

[186] S. Shakkottai, N. Brownlee, and k. claffy, "A study of burstiness in TCP flows," in *Proc. Passive and Active Measurement Conference (PAM)*, 2005.

[187] J. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, 1999.

[188] J. Mchugh, "Covert channel analysis." http://gray-world.net/papers/mchugh95covert.pdf, 1995.

[189] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley-Interscience, 2nd ed., 2006.

[190] S. Golomb, "The limiting behavior of the Z-Channel," *IEEE Transactions on Information Theory*, vol. 26, May 1980.

[191] C. Casetti and M. Meo, "A new approach to model the stationary behavior of TCP connections," in *Proc. IEEE INFOCOM*, 2000.

[192] MAWI Working Group, "Packet traces from WIDE backbone." http://tracer.csl.sony.co.jp/mawi/, 2006.

[193] L. Hoehn and I. Niven, "Averages on the move," *Mathematics Magazine*, pp. 151–156, May 1985.

[194] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP connection characteristics through passive measurements," in *Proc. IEEE INFOCOM*, 2004.

[195] D. Antoniades, M. Athanatos, A. Papadogiannakis, E. Markatos, and C. Dovrolis, "Available bandwidth measurement as simple as running wget," in *Proc. PAM*, 2006.

[196] R. Stanley, *Enumerative Combinatorics*. Cambridge University Press, 1997.

[197] D. Kreher and D. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search*. CRC press, 1998.

[198] H. Wilf, "East Side, West Side: An introduction to combinatorial families with Maple programming." http://www.cis.upenn.edu/~wilf/lecnotes.html, 2002.

[199] W. Myrvold and F. Ruskey, "Ranking and unranking permutations in linear time," *Information Processing Letters*, vol. 79, pp. 281–284, 2001.

[200] B. Sharma and R. Khanna, "On m-ary Gray codes," *Information Sciences*, pp. 31–43, 1978.

[201] E. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms - Theory and Practice*. Prentice-Hall, 1977.

[202] H. Lee, E. Chang, and M. Chan, "Pervasive random beacon in the Internet for covert coordination," in *Proc. Information Hiding Workshop*, 2005.

[203] F. Hernandez-Campos, A. Nobel, F. Smith, and K. Jeffay, "Understanding patterns of TCP connection usage with statistical clustering," in *Proc. IEEE MASCOTS*, 2005.

[204] R. Pang, M. Allman, M. Bennett, and J. Lee, V. Paxson, and B. Tierney, "A first look at modern enterprise traffic," in *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, 2005.

[205] MathWorks, Inc, "Linear or rank correlation." www.mathworks.com/access/helpdesk/help/toolbox/stats/corr.html, 2007.

[206] W. Davies, *System identification for self-adaptive control*. Wiley-Interscience, 1970.

[207] S. Floyd and E. Kohler, "Tools for the evaluation of simulation and testbed scenarios." Internet-draft draft-irtf-tmrg-tools-04, July 2007.

[208] K. Park and W. Willinger, eds., *Self-similar Network Traffic and Performance Evaluation*. Wiley, 2000.

[209] Y. Kim, W. Lau, M. Chuah and H. Chao, "Packetscore: a distributed proactive defense against DDOS attacks," in *Proc. IEEE INFOCOM*, 2004.

[210] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *Proc. ACM Mobihoc*, 2005.

[211] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proc. ACM Internet Measurement Workshop*, November 2002.

[212] C. Burrus, R. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer*. Upper Saddle River, NJ: Prentice Hall, 1998.

[213] P. Huang, A. Feldmann, and W. Willinger, "A non-intrusive, wavelet based approach to detecting network performance problems," in *Proc. ACM Internet Measurement Workshop*, 2001.

[214] R. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A novel approach to detection of denial of service attacks via adaptive sequential and batch-sequential change-point detection methods," in *Proc. IEEE Workshop on Information Assurance and Security*, June 2001.

[215] J. Baras, A. Cardenas, and V. Ramezani, "On-line detection of distributed attacks from space-time network flow patterns," in *Proc. Army Science Conference*, December 2002.

[216] H. Wang, D. Zhang, and K. Shin, "Detecting SYN flooding attacks," in *Proc. IEEE INFOCOM*, 2002.

[217] B. Brodsky and B. Darkhovsky, *Nonparametric Methods in Change-Point Problems*. Kluwer Academic Publishers, 1993.

[218] M. Carson and D. Santay, "NIST Net: a Linux-based network emulation tool," *ACM Computer Communication Review*, July 2003.