TCP Covert Timing Channels: Design and Detection

Xiapu Luo, Edmond W. W. Chan and Rocky K. C. Chang
Department of Computing, The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong, SAR, China
{csxluo|cswwchan|csrchang}@comp.polyu.edu.hk

Abstract

Exploiting packets' timing information for covert communication in the Internet has been explored by several network timing channels and watermarking schemes. Several of them embed covert information in the inter-packet delay. These channels, however, can be detected based on the perturbed traffic pattern, and their decoding accuracy could be degraded by jitter, packet loss and packet reordering events. In this paper, we propose a novel TCP-based timing channel, named TCPScript to address these shortcomings. TCPScript embeds messages in "normal" TCP data bursts and exploits TCP's feedback and reliability service to increase the decoding accuracy. Our theoretical capacity analysis and extensive experiments have shown that TCPScript offers much higher channel capacity and decoding accuracy than an IP timing channel and JitterBug. On the countermeasure, we have proposed three new metrics to detect aggressive TCPScript channels.

1 Introduction

Network covert channels pose a serious threat to the Internet security, and their use for concealing malicious activities is on the rise [4]. Most early network timing channels were designed for local area networks, such as manipulating frame collisions [19]. Only recently have several practical network timing channels emerged. Several of them are based on the timing information implicitly carried in the packets. An IP timing channel (IPTime) [3], for example, decodes an IP packet arrival during a *timing interval* as `1` and regards the absence of it as `0`. Since the IP timing channel uses absolute time as the reference, it needs time synchronization between encoder and decoder. The recently proposed ICMP timing channel [2] and JitterBug [6] do not have this constraint, because they embed messages into different values of inter-packet delays.

Although these timing channels are relatively simple to implement, they suffer from two major problems. First, to embed covert messages, these channels perturb IP packets' transmission patterns and inter-packet intervals, which can disrupt the normal traffic behavior of the higher-layer protocols, such as TCP. Therefore, these timing channels could be detected by analyzing the traffic pattern at a higher layer. Second, the decoding accuracy of these channels could be degraded significantly by jitter, packet loss, and packet reordering events that are prevalent in the Internet.

In this paper, we propose *TCPScript*, a novel approach of embedding a covert channel in a TCP flow. Unlike other timing channels, TCPScript retains TCP's bursty transmission behavior and is much more reliable than other timing channels under various network conditions. Our contributions can be summarized below:

1. We have designed TCPScript to encode stealthy information into the TCP data bursts. Since the encoding is not based on the inter-packet delay, TCPScript is more resilient to delay jitters. Moreover, we have exploited clustering algorithms and TCP's rich protocol features to further increase TCPScript's reliability.
2. We have analytically computed the channel capacity for TCPScript, IPTime, and JitterBug. To our best knowledge, this is the *first* information-theoretic analysis performed for these network timing channels. The capacity analysis is important for evaluating a covert channel's impact and for designing effective defense systems.
3. We have implemented TCPScript and conducted extensive experiments on a test bed and the PlanetLab platform. Both the information-theoretic analysis and the experiment results showed that TCPScript enjoys a higher data rate and decoding accuracy than IPTime and JitterBug under various network conditions.
4. In terms of countermeasures, we have proposed three new metrics for detecting TCPScript. The experiment results have shown that our new metrics can uncover aggressive TCPScript channels.

A roadmap We first review in §2 previous works on network timing channels. In §3, we present TCPScript's encoding and decoding algorithms. To evaluate TCPScript's performance, we present an information-theoretic analysis in §4 and an experimental evaluation in §5. We consider the countermeasures in §6, which will present new metrics for detecting TCPScript and experiment results. We finally conclude this paper in §7.

2. Related work

The works on network timing channels can be traced back to the work by Venkatraman et al. [1] who pointed out that temporal covert channels could exploit five characteristics of communications; however, they did not detail how to implement these covert channels. Some network timing channels require time synchronization between encoder and decoder. For example, Padlipsky et al. proposed a timing channel where one bit is conveyed through transmitting data or not [14], and Cabuk et al. [3] applied a similar idea to IP packets.

Other network timing channels have been designed without the constraint of time synchronization. Girling first suggested hiding information into the delays between successive transmissions from a user [7]. Building on this idea, Shah et al. [6] proposed JitterBug that encodes binary bits into packet inter-arrival times. A JitterBug encoder modulates the outgoing packets by the value of inter-packet delay modulo a timing window (w): bit `0` is encoded by a small value and bit `1` by a large value. JitterBug distinguishes itself from other timing channels in that it does not introduce new packets. As another example, Berk et al. [2] proposed using inter-packet delay of ICMP packets to encode single-bit and multiple-bit messages. However, the decoding accuracy of this class of *inter-packet delay channels* is susceptible to network conditions. Moreover, they lack mechanisms for detecting and correcting errors.

A few detection algorithms have been proposed for inter-packet delay channels by identifying anomalous statistics in the inter-packet delay. For example, Cabuk et al. proposed two new metrics for detection: variance in the inter-packet delays and ϵ-similarity between neighboring inter-packet delays [3]. Shah et al. outlined a detection method for JitterBug based on regularities of the inter-packet arrival times induced by a JitterBug channel [6]. Berk et al. proposed two methods for detecting the ICMP timing channels [2]. The first method is catered for a smart attacker who is assumed to possess the highest capacity, and the second is similar to the one proposed in [6]. Recently, Gianvecchio and Wang have proposed using the Shannon entropy and the corrected conditional entropy of inter-packet delays to detect IPTime and Jitterbug [17]. On preventive measures, Kang and Moskowitz proposed the network pump that can significantly suppress a network timing channel's throughput [13].

Besides, there are many steganographic techniques [11] that can be used to embed covert messages into the content of packets. Although such covert channels usually enjoy a higher capacity, they can be defeated by active wardens [5] and other steganoanalysis methods [11].

3. The TCPScript

3.1. The model

There are four parties in the model considered in this paper: a TCPScript encoder (or just encoder), a TCPScript decoder (or just decoder), a TCP server (e.g., a web server), and a warden. The encoder and warden are in a close proximity in terms of communication delay. The warden can monitor and analyze all the incoming and outgoing network traffic, including those from the encoder's machine. 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 connection with the server outside her network and embeds messages in the TCP data channel. The decoder, on the other hand, can eavesdrop the segments sent from the encoder to the server. However, it does not have to eavesdrop the reverse traffic.

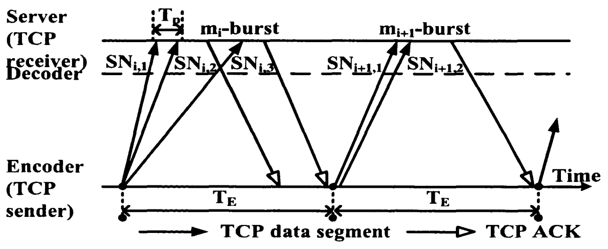
3.2. A basic encoding algorithm

Assume that the encoder is given a sequence of multibit covert messages (m_i, $i = 1, \dots$) to transmit to the decoder. Let each multibit covert message be represented by a positive integer: $m_i \in [1, M]$, $\forall i$, where M is a constant pre-agreed by the encoder and decoder. In TCPScript, each m_i is encoded by a burst of m_i back-to-back TCP data segments. We call such a burst a m_i-burst. To facilitate correct decoding, two adjacent messages (data bursts) are separated by an appropriate time interval. As a result, the covert messages are encoded in a sequence of TCP data bursts that resembles the typical pattern for a TCP bulk transfer.

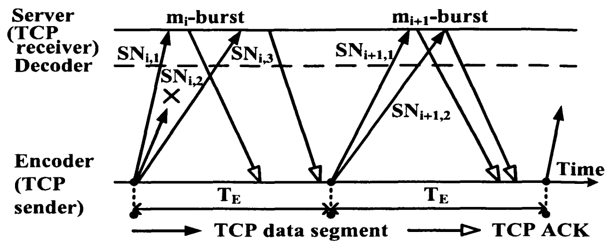
Figure 1(a) depicts the encoding process for two adjacent messages $m_i = 3$ and $m_{i+1} = 2$. The encoder sends a burst of three segments to the server for m_i. By eavesdropping the three segments, the decoder can decode m_i. Moreover, the packets arrive at the server with a time dispersion of T_p between two adjacent segments, and the server sends back two TCP acknowledgements (ACKs). The encoder considers the transmission of m_i successful after receiving the ACK for the segments within an *encoding period* T_E which begins from the time of transmitting the data burst. At the end of T_E, the encoder may send a new data burst for the next message.

To give a more precise description, we let the size of all TCP data segments be the encoder's *maximum segment size* (MSS) in bytes (the data size could be set to other values compatible for the TCP-based application in use). Denote the TCP sequence number (SN) of the jth data segment in the ith burst (or covert message) by $SN_{i,j}$, where $j = 1, \dots, m_i$. Considering the transmission of message m_i, if the encoder receives the ACK for the m_i data segments within T_E, the next data segment to be sent for m_{i+1} will have a sequence number of $SN_{i,m_i} + MSS$.

However, it is possible that not all the ACKs will arrive during T_E due to loss, delay jitter, or reordering experienced by the data segments or ACKs. As we will show next, the corresponding messages may still be decoded correctly. However, the encoder does not have enough information to ascertain the decoding result. Instead, it will continue the transmission of the subsequent messages. In our design, the encoder will always send the oldest unacknowledged data segment at the beginning of a new data burst. For example, Figure 1(b) shows



(a) There are no packet loss and reordering events, and all ACKs arrive within T_E.



(b) The second segment (with $SN_{i,2}$) is lost; however, the decoding result is still correct.

Figure 1. Two scenarios for encoding messages $m_i = 3$ and $m_{i+1} = 2$ in TCPScript, and the messages are decoded correctly in both scenarios.

that the second segment in the m_i-burst is lost. The encoder therefore retransmits the second and third segments in the next burst (i.e., $SN_{i+1,1} = SN_{i,2}$ and $SN_{i+1,2} = SN_{i,3}$). As we will see next, the repeated observation of the third segment will not affect the decoding accuracy.

3.3. A basic decoding algorithm

Prior to the covert communication, the encoder and decoder will agree on two parameters: the number of bursts (denoted by N) and the value of M. There are several methods of decoding the messages. In the following we present an off-line decoding algorithm based on packet clustering which comprises two sequential stages:

STAGE 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 covert message. Note that this clustering technique can mitigate the impact of delay jitter experienced by other clustering approaches using the inter-packet delay. We have selected the hierarchical clustering algorithm [15] which constructs the hierarchical cluster tree based on the centroid linkage (i.e., the Euclidean distance between two clusters' centroids). Our experiment results have shown that this algorithm achieves better performance than other popular clustering algorithms [15].

STAGE 2 Consider the set of packets identified by the clustering algorithm as part of the m_i-burst. The decoder first obtains the maximum and minimum SNs in the set, denoted by $SN_{i,max}$ and $SN_{i,min}$, respectively. Then the decoded message $\tilde{m}_i$ is simply given by

$$\tilde{m}_i \longleftarrow \frac{SN_{i,max} - SN_{i,min}}{MSS} + 1. \quad (1)$$

Although the decoding algorithm is very simple, it possesses the following attractive properties:

(1) Even if some data segments are missing in the m_i-burst due to packet loss, for example, the decoding result will still be correct, provided that the first and last segments are captured and classified into the m_i-burst.

(2) If there are missing packets at the head of the m_{i+1}-burst but the previous burst m_i is decoded correctly, then the decoding result for m_{i+1} will still be correct by setting

$$SN_{i+1,min} = min\{SN_{i+1,min}, SN_{i,max} + MSS\}. \quad (2)$$

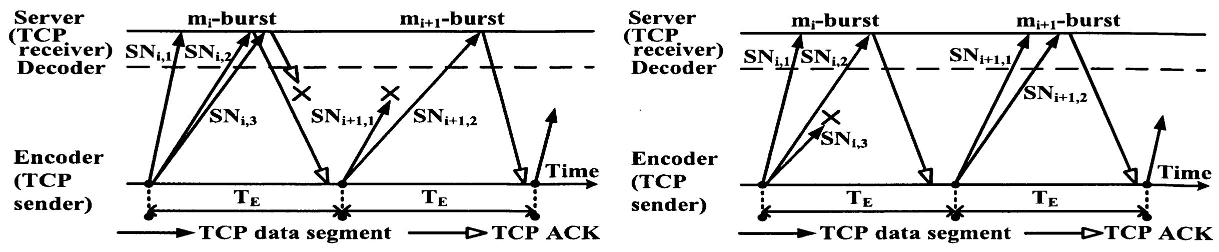
Since m_i is decoded correctly, the last segment with $SN_{i,max}$ must be captured. Therefore, $SN_{i+1,min}$ should be given by $SN_{i,max} + MSS$. For the m_1-burst, we could set $SN_{1,min}$ to $min\{SN_{1,min}, SN_{0,max} + 1\}$, where $SN_{0,max}$ is the SN of the encoder's TCP SYN segment.

(3) However, there are two scenarios where decoding errors will occur. The first scenario is illustrated in Figure 2(a), where all three data segments are received (i.e., $SN_{i,max} = SN_{i,3}$), but the ACK for the segment with $SN_{i,3}$ is lost. According to the encoding algorithm, the encoder will retransmit the third segment (i.e., $SN_{i+1,1} = SN_{i,3}$) at the beginning of the next burst. If this segment is lost again, then Eq. (2) will give $SN_{i+1,min} = SN_{i,3} + MSS$, thus resulting in a decoding error for m_{i+1} ($\tilde{m}_{i+1} = m_{i+1} - 1$). The second case, as shown in Figure 2(b), occurs when the last segment in a burst is missing (here $SN_{i+1,1} = SN_{i,3}$).

3.4. The choices of T_E and M

The choice of T_E obviously represents a trade-off between the decoding accuracy and channel 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. As a TCP sender, the encoder is already equipped with the 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 size.

The range of M is determined jointly by the TCP server's maximum receive window size (denoted by $RWND_{max}$) and the MSS: $1 \leq M \leq \frac{RWND_{max}}{MSS}$. If the window scale option



(a) A packet loss scenario that causes a decoding error for m_{i+1}.

(b) A packet loss scenario that causes a decoding error for m_i.

Figure 2. Two packet loss scenarios that will cause decoding errors in TCPScript's basic decoding algorithm.

is not used, then $RWND_{max} \leq 64$ KB. Taking a common setting of MSS = 1460 bytes, $M = \lfloor \frac{64KB}{1460} \rfloor = 44$. Moreover, depending on the desired behavior of the cover traffic, the encoder may select M from $[10, 20]$ to mimic the behavior of a long-lived TCP flow and from $[3, 8]$ to mimic the behavior of a short-lived TCP flow [18].

3.5 A loss-resilient TCPScript

To improve the reliability against packet losses, we have also proposed a *loss-resilient TCPScript* (LR-TCPScript) which is not included in this paper due to the page limit. Its basic idea is to recover the packet burst (i.e. message) for which the encoder fails to receive all the expected ACKs within T_E, before sending the next message. There are two main mechanisms in LR-TCPScript. An LR-TCPScript's encoder first uses a timeout-retransmission scheme to recover the message. This mechanism, however, is inadequate to handle two special scenarios: the lost of all data segments and the lost of all ACKs in a burst. We therefore employ a *rollback* mechanism to address these two scenarios.

4. An information-theoretic analysis

Determining a covert channel's capacity is very useful for assessing its impact and for designing the defense systems [8]. However, very few works studied the capacity of network timing channels [21, 2]. 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 [21]. Berk et al. used the Arimoto-Blahut algorithm to estimate the capacity of inter-packet delay channels [2].

However, all the previous works do not consider the effect of packet loss on the channel capacity. In this section, we fill this gap by modeling the capacity of TCPScript, IPTime, and JitterBug in the presence of packet loss. We will compare their information capacity (in terms of bits) derived from the channel models under different *packet loss probability*, denoted by P_{loss}. Here we do not consider the impact of severe jitters, because all three channels could effectively mitigate such effects by selecting proper parameters (e.g., setting the time interval to 80ms in IPTime [3] and the window size to 100ms in Jitterbug [6]). On the other hand, all of them are susceptible to packet losses.

We model the three channels as discrete memoryless channels (DMCs) with the input symbols and output symbols denoted as 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:

$$C = \max_{p(x)}\{H(X) - H(X|Y)\} = \max_{p(x)}\{H(Y) - H(Y|X)\}, \tag{3}$$

where $H(X)$ (or $H(Y)$) is the entropy of X (or Y), and $H(X|Y)$ and $H(Y|X)$ are conditional entropies [22].

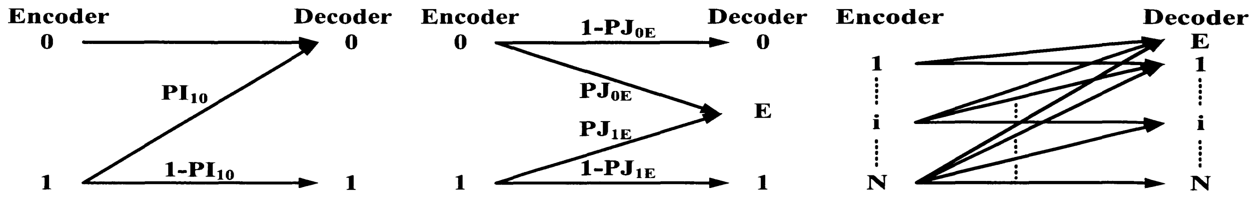
According to each channel's characteristics, we model IPTime, JitterBug, and TCPScript using Z-Channel, Binary Erasure Channel, and Error and Erasure Channel, respectively, as shown in Figure 4.

IPTIME There are two input symbols to the channel: nonarrival of a packet (0) and arrival of a packet (1). Therefore, symbol 1 will be decoded as 0 due to a packet loss (with probability PI_{10}), but symbol 0 is always decoded as 0. From Figure 3(a), we obtain the probability transition matrix as $Q(Y|X) = \begin{pmatrix} 1 & 0 \\ PI_{10} & 1 - PI_{10} \end{pmatrix}$, where $P_{loss} = PI_{10}$. By applying the known capacity result for a Z-Channel [9], we have

$$\begin{aligned} C = \quad & \mathbb{F}[p(x)(1 - P_{loss}), 1 - p(x)(1 - P_{loss})] \\ & -p(x)\mathbb{F}[(1 - P_{loss}), P_{loss}], \end{aligned} \tag{4}$$

where $\mathbb{F}[\alpha, \beta] = -\log_2(\alpha^\alpha \beta^\beta)$, and $p(x) = (P_{loss}^{\frac{P_{loss}}{1-P_{loss}}})/(1 + (1 - P_{loss})P_{loss}^{\frac{P_{loss}}{1-P_{loss}}})$.

JITTERBUG Each symbol is represented by the time interval between two consecutive packets. Therefore, if at least



(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.

Figure 3. Modeling TCPScript, IP timing channel, and JitterBug as discrete memoryless channels.

one packet is lost, it will become a deletion channel whose channel capacity is difficult to estimate [23]. 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 the input symbol of `0` and `1`, respectively. From in Figure 3(b), we obtain the probability transition matrix as $Q(Y|X) = \begin{pmatrix} 1-PJ_{0E} & PJ_{0E} & 0 \\ 0 & PJ_{1E} & 1-PJ_{1E} \end{pmatrix}$, where $PJ_{0E} = PJ_{1E} = 1-(1-P_{loss})^2$. Then we can obtain $C = (1-P_{loss})^2$.

TCPSCRIPT Recall from §3 that packet losses at a burst's tail will result in decoding errors, but packet losses at a burst's head may or may not. 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; that is, 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.

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, the probability transition matrix is:

$$\bar{p}(x_i|x_j) = \begin{cases} (1-P_{loss})^j P_{loss}^{i-j} & \text{if } i>j \\ 1-\sum_{k=0}^{i-1} p(x_i|x_k) & \text{if } i=j \\ 0 & \text{if } i<j, \end{cases} \tag{5}$$

where $i = 1,\ldots,N$ and $j = 0,\ldots,N$. 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 in either the head or tail of a burst. Therefore, the probability transition matrix is:

$$\underline{p}(x_i|x_j) = \begin{cases} P_{loss}^i & \text{if } j=0 \\ (i-j+1)(1-P_{loss})^j P_{loss}^{i-j} & \text{if } i>j\neq 0 \\ 1-\sum_{k=0}^{i-1} p(x_i|x_k) & \text{if } i=j \\ 0 & \text{if } i<j, \end{cases} \tag{6}$$

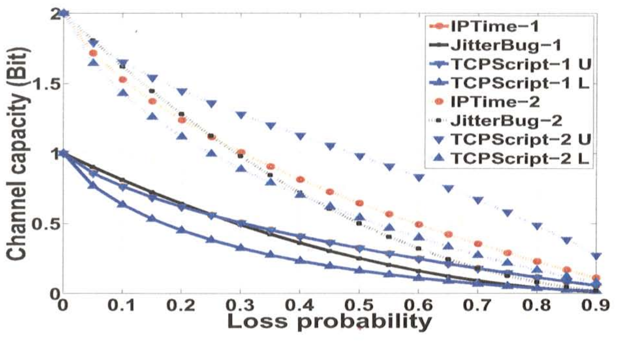
where $i = 1,\ldots,N$ and $j = 0,\ldots,N$. Based on Eq. (5) and Eq. (6), we can therefore estimate the lower bound and upper bound for TCPScript's capacity by applying the Blahut-Arimoto algorithm [22].

For a fair comparison, we multiply the capacity of IPTime and JitterBug by log_2M (i.e., equivalent to log_2M parallel channels). We use IP Time-log_2M to refer to the IPTime capacity for M symbols and similarly for JitterBug and TCPScript. We use *TCPScript U*(or *TCPScript L*) to indicate the upper (or lower) bound of TCPScript's capacity in Figure 4. We first discuss the results for $M = 2$ (i.e., single-bit encoding) which are shown in Figure 4(a). TCPScript's upper bound is overlapped with IPTime's (that is why the IPTime graph cannot be seen from the figure). When $M = 2$, TCPScript's channel model is reduced to a Z-Channel if we group the erasure state with state 1 on the decoder side. Moreover, we have observed that JitterBug slightly outperforms the other two at a low P_{loss}; however, the trend is reversed at a higher P_{loss}. These results are consistent with the fact that any packet loss will reduce JitterBug into a deletion channel.

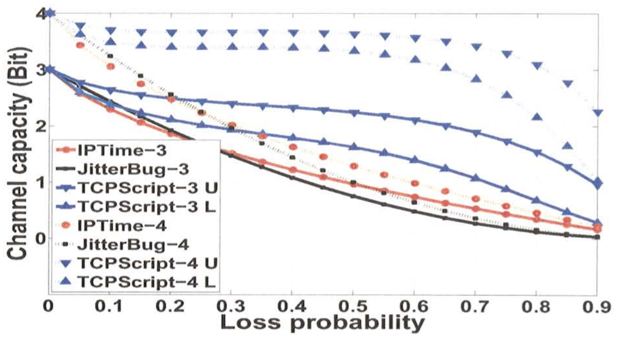
Figures 4 also shows the results for $M = 4, 6, 8$ (i.e., multi-bit encoding). When $M = 4$, TCPScript's upper bound is higher than the other two channels' capacity; however, the lower bound is still below the other two at low values of P_{loss}. When M is increased to 6 and 8, Figure 4(b) clearly illustrates the advantage of TCPScript, because both upper and lower bounds are higher 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 loss will result in decoding errors for IPTime and JitterBug.

5. Performance evaluation

In this section, we report the performance evaluation results for TCPScript, JitterBug, and IPTime. We have implemented the TCPScript encoder as an HTTP client that uses the `HTTP POST` method to transmit data to a web server. We use Linux



(a) Channel capacity with 2 and 4 input symbols.



(b) Channel capacity with 8 and 16 input symbols.

Figure 4. Comparing the channel capacity for TCPScript, IP timing channel, and JitterBug.

raw sockets to customize the outgoing packets. Note that the encoder could also be easily implemented in other TCP-based applications (e.g., FTP and P2P applications). For the TCPScript decoder, we utilize `libpcap` to capture the traffic sent from the encoder to the web server. Moreover, we have implemented IPTime's and JitterBug's encoding and decoding algorithms. Both use a fixed timing interval (or timing window) of w. The JitterBug encoder adopts the default tolerance parameter $\varepsilon = w/4$ [6].

We have conducted extensive experiments in both a test bed and the Internet using PlanetLab nodes. The test bed allows us to study the impact of arbitrarily network conditions on the channel performance, whereas the PlanetLab platform provides a real network environment for evaluation. For the purpose of evaluation and comparison, we are interested in their empirical *channel accuracy* α and empirical *channel goodput* ζ defined as

$$\alpha = 1 - p_e \text{ and } \zeta = \frac{\alpha N \log_2 M}{T_d}, \quad (7)$$

where p_e is the channel's bit error rate, and T_d is the total time required for delivering the N covert messages. We employ the Levenshtein (or Edit) distance to compute p_e, which is given by the minimum number of insertions, deletions, and substitutions required to convert a decoded message into the original message.

5.1. Test-bed experiments

The test bed has a dumbbell topology. An encoder is located on one end of the topology, whereas the decoder and a web server are located on the other. The two ends are connected by two routers R_1 and R_2. We have deployed Dummynet in both routers to emulate various network conditions. All the network links are full duplex with capacity of 100Mbit/s. The encoder embeds covert messages into the TCP packets destined to the web server which can be eavesdropped by the decoder.

We have conducted test-bed experiments with the following parameter settings. The RTT between the encoder and the web server is 30ms. The TCPScript encoder uses $T_E = \{40, 60\}$ms to transmit 100 codewords ($N = 100$) with $M = \{4, 16\}$. Each experiment was repeated 30 times; each time a different set of 100 codewords was generated randomly. We report the average values of ζ and α over the 30 results, which are denoted by $\overline{\zeta}$ and $\overline{\alpha}$, respectively. The JitterBug's and IPTime's encoders use $w = \{40, 60\}$ms to transmit the same 30 sets of codewords bit by bit in a single flow of modulated UDP packets. The size of all IP packets sent by the encoder is 1500 bytes.

Effects of packet losses Table 1 summarizes the average channel accuracy and goodput under different *packet loss rates* (PLRs), ranging between 0% and 5%, and $T_E = w = 40$ms. In each cell, the two leftmost values are the 95% confidence intervals (CIs) for $\overline{\zeta}$, whereas the values inside the parentheses are the 95% CIs for $\overline{\alpha}$. All the results are obtained without using any error correction codes. The table shows that both TCPScript and JitterBug achieve 100% accuracy in a lossless environment, whereas IPTime achieves only up to 95% accuracy due to the queueing delay introduced by the routers. Moreover, TCPScript's goodput is at least 1.5 times higher than that of JitterBug for $M = 4$. As the PLR increases, the decoding errors start increasing for all timing channels. However, even when the PLR reaches 5%, TCPScript still maintains a relatively low p_e of at most 4%, while JitterBug's and IPTime's p_e are at most 7% and 8%, respectively.

Effects of packet reordering We have evaluated the channel performance under two different packet reordering scenarios referred to as A and B. In each scenario, we configured R_1 with different numbers of pipes and entrance probabilities, such that the mean RTT between the encoder and decoder was equal to 30ms. As shown in Table 2, we do not observe decoding errors for TCPScript for both reordering scenarios. It also achieves a higher goodput than the other two timing channels. Comparing with the results for PLR $= 0\%$ in Table 1,

Table 1. The values of $\overline{\zeta}$ and $\overline{\alpha}$ for different PLRs and $T_E = w = 40$ms.

PLR	TCPScript	JitterBug	IPTime
	95% CIs of $\overline{\zeta}$ (95% CIs of $\overline{\alpha}$) 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)

we find that TCPScript still maintains similar goodputs even under the two reordering scenarios. However, both JitterBug and IPTime suffer from significant throughput degradation and decoding error rates. Their measured p_e can be up to 21% and 14%, and their goodputs drop by 21% and 9%, respectively. Since both JitterBug and IPTime encode in the packet inter-arrival times, their decoding accuracy is very sensitive to delay jitter. On the other hand, TCPScript does not directly rely on the packet inter-arrival times. Decoding errors occur only when the packet reordering causes a packet to be clustered into a wrong burst.

EFFECTS OF TRAFFIC SHAPING To study the effects of traffic shaping on TCPScript, we deployed NetPath [16] in R_1 to emulate a traffic shaper with different bandwidth limits $B = \{0.5, 1, 2, 5, 10\}$Mbps and a fixed RTT of 100ms between the encoder, and decoder and the web server. We used Iperf to generate background TCP traffic, and the encoder used $M = 4$ and $T_E = \{120, 140, 160, 180\}$ms to transmit the same set of 100 codewords. As shown in Table 3, no decoding errors are observed until the bandwidth limit drops to 0.5Mbps, and in the worst case p_e is only 3.8%. Moreover, a smaller T_E will suffer more decoding errors, because the inter-burst gaps may not be accurately identified by the decoder.

5.2. PlanetLab experiments

For the Internet experiments, we have selected eight geographically diversed PlanetLab nodes to serve as the encoders that are listed in Table 4. The Linux Hierarchical Token Bucket (HTB) scheduler has been implemented in each PlanetLab node to provide the default traffic shaping function [10]. While this function delays outgoing packets to meet certain bandwidth shares, it can also smooth out bursty traffic. The web server and the decoder, on the other hand, are located at our campus network. The decoder extracts covert messages from the packet flows sent from the encoders to the web server. We have obtained a total of 3,744 samples for each encoder during the experiment period. In Figures 5-6, the PlanetLab nodes on the x-axis are sorted in an increasing order of their RTTs.

Figure 5 reports the average channel accuracy for the timing channels with $T_E = w = \{2\text{RTT}, 2.5\text{RTT}, 3\text{RTT}\}$. Note that TCPScript generally attains higher channel accuracy than IPTime and JitterBug. Moreover, TCPScript's channel accuracy is consistently higher than 98% even in the presence of the traffic shaping function in the encoders' outgoing links. For those paths that exhibit large jitters (i.e., TW, CN, JP, and CA), all timing channels attain lower decoding accuracy. But TCPScript still enjoys higher accuracy than the other two channels. For IPTime, Figure 5(c) shows that a larger w will generally increase its decoding accuracy, because a larger timing window could mitigate the impact of time desynchronization. However, we do not observe this relationship for TCPScript and Jitterbug. An analysis of the network traces reveals that packet loss, instead of the large delay jitter, is the main factor responsible for their decoding errors.

Figure 6 shows the average channel goodput. Even using $M = 4$, TCPScript's goodput is around *two* times of the other two channels' goodput. TCPScript's goodput advantage will become even more significant for a larger M. As the path quality deteriorates in terms of the RTT, the measured goodputs also decrease.

6. Detecting TCPScript

TCPScript can evade the existing detection schemes designed for inter-packet delay channels. Therefore, we propose in this section three new metrics for detecting TCPScript: a deviation score for the burst size, entropy of burst size, and inter-ACK-data delay. We have also employed a simple threshold-based algorithm to evaluate their detection capabilities. We used the LBNL's traces [20] (around 11GB of uncompressed header traces) to build the traffic profile and evaluate the *false positive rate* (FPR). This trace set represents typical traffic characteristics for enterprise networks, and the network setting is also similar to our model described in §3.1.

6.1. Detection based on the burst size

Since TCPScript embeds information into the burst size, an obvious metric is the burst size itself. The idea is that the burst size is usually controlled by TCP congestion control algorithm; hence, the neighboring burst sizes are expected to exhibit the corresponding pattern. However, we do not expect the same for the burst sizes modulated by TCPScript, because the burst size depends only on the covert messages. Although the idea is simple, we have to tackle two challenging issues: how to identify the bursts and how to design criteria for distinguishing between normal TCP bursts and the bursts modulated by TCPScript. Both of them are nontrivial to tackle because of a wide range of normal network phenomena and a diversity in the implementations of TCP congestion control algorithms [12].

Table 2. The values of $\overline{\zeta}$ and $\overline{\alpha}$ for two packet reordering scenarios A and B and $T_E = w = 40$ms.

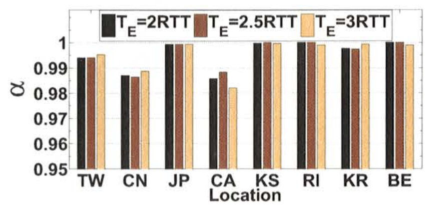
	TCPScript	JitterBug	IPTime
	95% CIs of $\overline{\zeta}$ (95% CIs of $\overline{\alpha}$) for 100 codewords with $M = 4$		
A	48.85,48.88(1.00,1.00)	25.05,25.05(0.80,0.82)	21.94,21.94(0.91,0.92)
B	48.87,48.88(1.00,1.00)	24.66,24.66(0.79,0.81)	20.68,20.68(0.86,0.87)

Table 3. The values of $\overline{\zeta}$ and $\overline{\alpha}$ for TCPScript with $M = 4$ and $T_E = \{120, 140, 160, 180\}$ms under different bandwidth limits imposed by a traffic shaper: $B = \{0.5, 1, 2, 5, 10\}$Mbps.

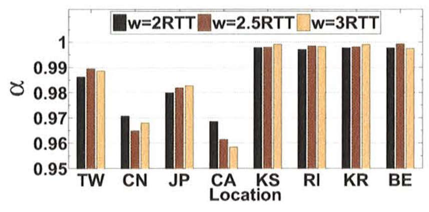
T_E	Bandwidth Limit B				
	0.5Mbps	1Mbps	2Mbps	5Mbps	10Mbps
120ms	11.89,13.98 (0.90,1)	15.53,15.70(1,1)	15.52,15.71(1,1)	15.47,15.71(1,1)	16.03,16.15(1,1)
140ms	11.32,12.44(0.95,1)	13.46,13.58(1,1)	13.46,13.59(1,1)	13.51,13.64(1,1)	13.83,13.93(1,1)
160ms	10.82,11.73(0.95,1)	11.86,12.00(1,1)	12.04,12.12(1,1)	12.14,12.18(1,1)	12.18,12.19(1,1)
180ms	9.84,10.58(0.96,1)	10.61,10.73(1,1)	10.76,10.83(1,1)	10.81,10.89(1,1)	10.83,10.91(1,1)

Table 4. Path characteristics from the encoders (on PlanetLab nodes) to the decoder (on our campus).

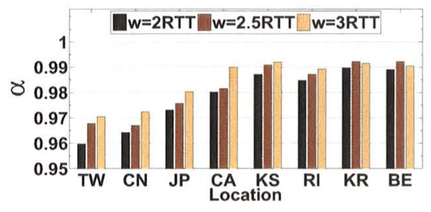
Locations	No. hops	Mean RTT and the 95% CIs	Locations	No. hops	Mean RTT and the 95% CIs
Taipei, Taiwan (TW)	10	.0525 ± .0021	Kansas, U.S. (KS)	16	.2263 ± .0003
Shenyang, China (CN)	13	.0812 ± .0026	Rhode Island, U.S. (RI)	13	.2335 ± .0003
Tokyo, Japan (JP)	16	.1165 ± .0023	Gwangju, Korea (KR)	18	.2381 ± .0018
California, U.S. (CA)	14	.2007 ± .0022	Flanders, Belgium (BE)	16	.3158 ± .0002



(a) TCPScript with $M = 4$.

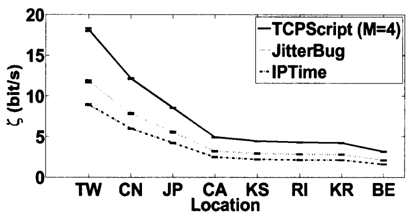


(b) JitterBug.

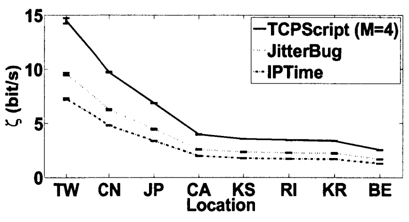


(c) IPTime.

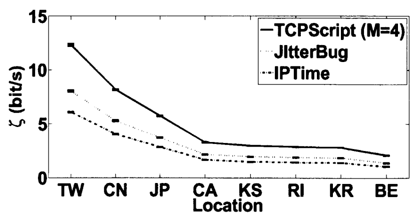
Figure 5. Measured average channel accuracy from the PlanetLab experiments.



(a) $T_E = w = 2$RTT.



(b) $T_E = w = 2.5$RTT.



(c) $T_E = w = 3$RTT.

Figure 6. Measured goodput from the PlanetLab experiments.

Since the warden does not know how many bursts have been delivered, she cannot use clustering algorithms to separate the data bursts. Therefore, we adopt a heuristic algorithm to identify a series of bursts. We let TP_i, $i = 1, 2, \ldots$, denote the timestamp values when the warden captures the ith data packet. Let $IPT_i = TP_{i+1} - TP_i$ denotes the inter-arrival time between ith and $(i+1)$th packets. We consider that TP_i marks the end of a burst if the ratio of IPT_i and the average inter-arrival time exceeds a threshold.

After obtaining a sequence of bursts from a TCP data flow, we employ two metrics for the detection. Let $\hat{m}_i$ be the number of packets in the ith burst estimated by the warden. The first metric is a *deviation score* (DS) that captures abnormal changes from $\hat{m}_i$ to $\hat{m}_{i+1}$. We have proposed a set of rules to update the DS which is omitted in this paper due to the limited space. The second metric is the entropy of the burst size, denoted by $\mathcal{E}$. It is motivated by the observation that TCPScript's burst size will be uniformly distributed, if the covert messages also follow a uniform distribution. Given N_b bursts, we then compute $\mathcal{E}$ as $-\sum_{i=1}^{N_b} p(m_i)\log p(m_i)$, where $p(m_i)$ is the probability of observing m_i. Our extensive experiments have shown that $\mathcal{E}$ detects more TCPScript channels than DS does. Therefore, we discuss only the results using $\mathcal{E}$ in the remaining of this section.

Figure 7 shows the CDFs of $\mathcal{E}$ for the LBNL traces and TCPScript traces. Around 95% of the entropy values for the LBNL traces are less than 3, but almost all entropy values for TCPScript with $M = 16$ are larger than 3. However, most entropy values for TCPScript with $M = 4$ are around 2. Therefore, $\mathcal{E}$ can detect TCPScript channels with a large M but may miss others for $M = 4$ or an even lower M.

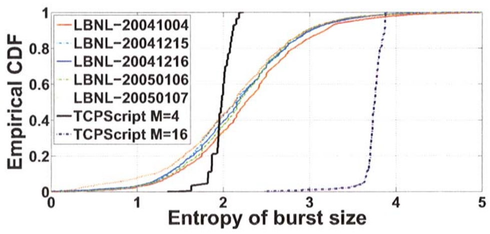


Figure 7. Entropy for the LBNL traces and TCPScript traces.

6.2. Detection based on inter-ACK-data delay

To enhance the detection capability of the above metrics, we propose another new metric (denoted as T_{AD}) which 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 acknowledgement number. That is, the transmission of Pkt_{Data} is very likely induced by the arrival of Pkt_{ACK}. The motivation of using T_{AD} is that if a TCP sender has data to send and the receive window is larger than one MSS, the sender will dispatch new data packets after receiving a new ACK. However, since the TCPScript encoder transmits another burst of packets only at end of T_E, the resulted T_{AD} is expected to be larger than the normal values. Therefore, we use T_{AD}'s mode (i.e., the peak of its density function) to detect TCPScript.

Figure 8 shows the CDF of T_{AD} for the LBNL traces and TCPScript traces. Since the same T_E is adopted for TCPScript with $M = 4, 16$ in our experiments, their CDFs almost overlap. We can see that most of the T_{AD} values in LBNL traces are very small; for example, around $78\% - 90\%$ of the T_{AD} values for the LBNL traces are less than 0.02. On the contrary, TCPScript will often result in large T_{AD} values; for example, more than 94% of the T_{AD} values for TCPScript is larger than 0.02. Therefore, this metric is effective for detecting TCPScript for both values of M.

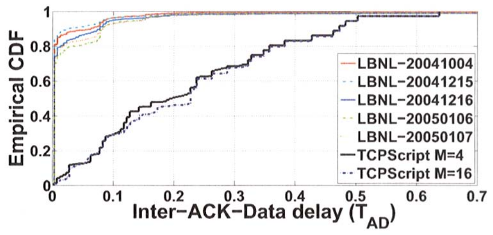


Figure 8. The CDFs of the inter-ACK-data delay in the LBNL traces and TCPScript traces.

6.3. Evaluating the new metrics

We have employed a simple threshold-based algorithm [3] to evaluate the performance of the two new metrics: $\mathcal{E}$ and T_{AD}. To build the normal profiles for them, we use the samples collected from three sets of LBNL traces (LBNL-20041004, LBNL-20050106 and LBNL-20050107) [20]. These sample values are less than the Q quantile values in the corresponding training data sets in order to mitigate the impact of outliers. Based on these two sets of data, we compute their means ($\theta_{\mathcal{E}}$ and $\theta_{T_{AD}}$) and their standard deviations ($\delta_{\mathcal{E}}$ and $\delta_{T_{AD}}$). Moreover, we define their thresholds as $\Gamma_E = \theta_{\mathcal{E}} + d \times \delta_{\mathcal{E}}$ and $\Gamma_{T_{AD}} = \theta_{T_{AD}} + d \times \delta_{T_{AD}}$, respectively, where d is a parameter for controlling the thresholds.

We have used real TCPScript traces collected from the PlanetLab experiments to evaluate the detection rates. We have tested different combinations based on $Q \in [0.8\ 0.85\ 0.9\ 0.95]$ and $d \in [1\ 1.5\ 2\ 2.5]$. Due to the limited space, we tabulate only part of the experiment results in Table 5, where $Q = 0.95$ and $d = 1.5, 2.5$. We show the detection rates using only $\mathcal{E}$, only T_{AD}, and both of them (labeled by "Both"). The last column also gives their respective FPRs that are computed from the remaining two sets of the LBNL traces, assuming that TCPScript channels do not exist in them.

From Table 5 and other experiment results that are not shown here, we have found that when Q and d increase, the average detection rate based on a single metric may decrease, because such settings will relax the constraint on the TCPScript's behavior. However, the detection rate based on both metrics may not decrease, because a union of these two orthogonal

Table 5. Detection rates and FPR using $\mathcal{E}$ and T_{AD} under different parameter settings.

		BE(M=4)	BE(M=16)	KR(M=4)	KR(M=16)	CA(M=4)	CA(M=16)	JP(M=4)	JP(M=16)	FPR
$Q = 0.95$	$\mathcal{E}$	0	1	0	0.9944	0	0.9708	0	0.9663	0.0985
	T_{AD}	0.8944	0.8156	0.8389	0.8371	0.838	0.8304	0.8362	0.7921	0.1021
$d = 1.5$	Both	0.8944	1	0.8389	0.9944	0.838	0.9883	0.8362	0.9888	0.1947
$Q = 0.95$	$\mathcal{E}$	0	1	0	0.9775	0	0.8012	0	0.9157	0.0245
	T_{AD}	0.8778	0.7709	0.8333	0.809	0.6704	0.6725	0.7627	0.7640	0.0662
$d = 2.5$	Both	0.8778	1	0.8333	0.9831	0.6704	0.9181	0.7627	0.9775	0.0901

metrics' detection ranges does not shrink. On the other hand, the FPR will increase along with Q and d. Moreover, comparing the two metrics, T_{AD} is more effective than $\mathcal{E}$ for $M = 4$, because $\mathcal{E}$ could not effectively detect TCPScript with small M which is already shown in Figure 7. On the other hand, also shown in Figure 7, $\mathcal{E}$ is more effective than T_{AD} when $M = 16$. Combining the two metrics could therefore improve the detection performance for some cases, but with the expense of increasing the FPR.

7. Conclusions

In this paper, we have presented TCPScript which represents a new approach to designing effective network timing channels. By encoding covert messages into the bursts of TCP data packets, the modulated traffic retains TCP's normal burstiness patterns. Besides TCP, this approach can be easily applied to other protocols, such as SCTP, because it uses only three fundamental mechanisms that are available in other reliable end-to-end protocols: sliding-window algorithm, sequence number, and acknowledgements. Moreover, we have performed an information-theoretic analysis to estimate the channel capacity for TCPScript, the IP timing channel, and JitterBug. The analytical results and extensive experiments show that TCPScript achieves higher goodput and decoding accuracy than the two other channels under various network conditions. On the countermeasure, we have proposed three new metrics to detect TCPScript and the experiments show that they are effective in detecting aggressive TCPScript channels.

Acknowledgments

This work was partially supported by a grant from The Hong Kong Polytechnic University (project code U386). The authors thank Prof. Wenke Lee, Prof. Wing-Cheong Lau and anonymous reviewers for their useful comments.

References

[1] B. Venkatraman and R. Newman-Wolfe. Transmission schedules to prevent traffic analysis. In *Proc. ACSAC*, 1993.

[2] V. Berk, A. Giani, and G. Cybenko. Detection of covert channel encoding in network packet delays. Technical report, Department of Computer Science, Dartmouth College, 2005.

[3] S. Cabuk, C. Brodley, and C. Shields. IP covert timing channels: Design and detection. In *Proc. ACM CCS*, 2004.

[4] E. Casey. Next-generation cyber forensics: Investigating sophisticated security breaches. *Comm. ACM*, Feb. 2006.

[5] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil. Eliminating steganography in Internet traffic with active wardens. In *Proc. Information Hiding Workshop*, 2002.

[6] G. Shah, A. Molina, and M. Blaze. Keyboards and covert channels. In *Proc. USENIX Security*, 2006.

[7] C. Girling. Covert channels in LAN's. *IEEE Trans. Software Engineering.*, Feb. 1987.

[8] V. Gligor. A guide to understanding covert channel analysis of trusted systems (light pink book). Technical Report NCSC-TG-030, National Computer Security Center, Nov. 1993.

[9] S. Golomb. The limiting behavior of the Z-Channel. *IEEE Trans. on Infor. Theory*, 26(3), May 1980.

[10] M. Huang. Bandwidth limits — PlanetLab. https://www.planet-lab.org/doc/BandwidthLimits, 2006.

[11] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker. *Digital Watermarking and Steganography*. Morgan Kaufmann, 2nd edition, 2007.

[12] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proc. IEEE INFOCOM*, 2004.

[13] M. Kang, I. Moskowitz, and D. Lee. A network Pump. *IEEE Trans. Software Engineering*, May 1996.

[14] M. Padlipsky, D. Snow, and P. Karger. Limitations of end-to-end encryption in secure computer networks. Technical report, The MITRE Corporation, 1978.

[15] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[16] S. Agarwal and J. Sommers and P. Barford. Scalable network path emulation. In *Proc. IEEE MASCOTS*, 2005.

[17] S. Gianvecchio and H. Wang. Detecting covert timing channels: An entropy-based approach. In *Proc. ACM CCS*, 2007.

[18] S. Shakkottai, N. Brownlee, and k. claffy. A study of burstiness in TCP flows. In *Proc. PAM*, 2005.

[19] T. Handel and M. Stanford. Hiding data in the OSI network model. In *Proc. Information Hiding Workshop*, 1996.

[20] V. Paxson. LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing/Overview.html, 2005.

[21] B. Venkatraman and R. Newman-Wolfe. Capacity estimation and auditability of network covert channels. In *Proc. IEEE Symp. Security and Privacy*, 1995.

[22] R. Yeung. *A First Course in Information Theory*. Kluwer Academic, 2002.

[23] Z. Wang and R. Lee. New constructive approach to covert channel modeling and channel capacity estimation. In *Proc. ISC*, 2005.