SNEAK-PEEK: HIGH SPEED COVERT CHANNELS IN DATA CENTER NETWORKS

BY

RASHID TAHIR

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Matthew Caesar

ABSTRACT

With the advent of big data, modern businesses face an increasing need to store and process large volumes of sensitive customer information on the cloud. In these environments, resources are shared across a multitude of mutually untrusting tenants increasing propensity for data leakage. With the recent spate of high-profile data exfiltration attacks and the emergence of critical vulnerabilities such as Heartbleed and Shellshock, coupled with increasing use of clouds in all aspects of our daily lives, this problem stands to grow further in severity.

In this thesis, we present a novel *network-based* covert channel that can arise in the context of shared network resources in data-center environments even in the presence of network monitors regulating flow destinations with NAC policies and VLAN-based isolation mechanisms. Through a series of experiments on diverse network hardware (including SDNs) and commercial clouds such as EC2 and Azure, we demonstrate that our network-based channel achieves orders of magnitude greater bit rates than reported in any recent literature. Furthermore, we present an information-theoretic framework to model and study the channel. Using this model we derive an upper bound on the information rate of the channel and propose a coding scheme that nearly achieves this upper bound. Additionally we introduce some techniques to make the covert channel robust to noise, and empirically study its performance in the presence of realistic cross-traffic. Finally, we discuss several avenues for mitigation, and demonstrate the effectiveness of our schemes both empirically and mathematically.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to express my sincere and utmost gratitude to my advisor, Matthew C. Caesar, for his relentless support and guidance during my research. It is an honor to have an advisor who takes such keen interest in the professional and personal development of his students and mentors them to become better researchers in particular and better human beings in general. I have benefited enormously from his immense knowledge and constant motivation and hope to grow further under his mentorship during the course of my Ph.D. study.

This work was developed in collaboration with Taha Khan, Adnan Ahmed, Xun Gong and Amiremad Ghassami and I am greatly thankful to them. This thesis is based on a paper submitted recently for publication and is currently under review. Xun Gong led the development of the information theoretic framework of the channel, and I along with Taha and Adnan developed the encoding/decoding scheme and the implementation and performance evaluation of the channel.

I am also greatly thankful to other collaborators of this project including Fareed Zaffar and Negar Kiyavash, for their valuable inputs during the course of this project.

I am also thankful to Brighten Godfrey and his students to allow us to use the OCEAN testbed and perform repeated experiments on it in isolation.

Most importantly, I would like to thank my family and friends for their constant support. In particular, I am indebted to my parents whom I can never payback, my extremely understanding and supporting wife, my brother who has been my ideal all my life and my three wonderful sisters for their understanding and encouragement to do the best in all matters of life.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Third party clouds are becoming increasingly popular for outsourcing computation and data storage. A broad spectrum of corporations now store and process sensitive data on commercial clouds such as Amazon's EC2, Microsoft's Azure or the HIPAA-compliant ClearData Cloud [22]. Companies that deal with credit ratings, such as Experian [4], health care providers, such as BioMedix [2] and Barrow Neurological Institute [2], and military organizations, such as the United States Department of Defense [3] and the CIA [1], all have migrated (or are currently in the process of migrating) their computational infrastructures to the cloud, achieving the benefits of economies of scale, reducing IT costs, and improving reliability and other desirable properties of computation.

Given the sensitivity of user data, it is imperative for cloud providers to ensure that data remains private and isolated. Unfortunately, the very nature of the cloud is multi-tenant, and hence this data protection becomes a serious challenge. Sharing the same infrastructure between multiple tenants is crucial for achieving economies of scale via cost savings arising from shared management, statistical multiplexing and efficient utilization of a limited set of resources [46]. This necessity of sharing infrastructure leads to the danger of information leakage between tenants, which can be highly detrimental and can result in major costs for businesses [9, 14, 21, 12, 10]

This problem becomes more severe given the emergence of massive cyberespionage-based malware ecosystems such as GhostNet[11], ShadowNet[6] and Axiom[18]. These global crime rings systematically compromise machines in governments and organizations, with the single and solitary objective of leaking out confidential data (often hosted on data centers), either openly or in most cases via stealthy covert channels to avoid detection and traceback. The discovery of critical vulnerabilities, such as Heartbleed [19] and ShellShock [20], on a regular basis has further exacerbated the situation by providing attackers a broad spectrum of “entry points” into target machines. The only remaining challenge for data thieves is to somehow exfiltrate the data by bypassing network-based monitors, which is why network-based covert channels, in particular, have reemerged as a major cause for concern especially in the cloud arena.

In light of these challenges, cloud operators resort to partitioning of resources using virtualization technologies, such as hypervisors, VPNs, Network Virtualization Platforms [8] etc. However, researchers recently demonstrated [46, 54, 52, 51] that despite being logically isolated at the host level, VMs sharing the same machine can still leak sensitive information via covert and side channels. Many mitigation schemes have since been proposed that can provide significant protection against these attacks either at the host level [50, 36, 7, 56, 51] or via network-based appliances that deploy a clever combination of network monitoring, access control, firewalls etc. to prevent leakage even if the machine is compromised.

In this thesis, we demonstrate how to leak out data even with all the aforementioned host *and* network-based security mechanisms deployed. Our covert channel achieves very high bit rates in the presence of real-life cross traffic whilst remaining undetectable. We test its practicality on commercial clouds such as EC2 and Azure and demonstrate orders of magnitude greater bit rates than any previous work in the data center networks domain. Furthermore, to build a more complete understanding of the problem, we construct a formal analytical model of the channel, and present an information-theoretic upper bound on the bit rate of the channel along with an optimal scheme, which nearly achieves the upper bound. We also analyze the difficulty of detecting our covert channel by estimating the Hurst exponent in the presence of cross traffic. For enshanced robustness to noise, we introduce Forward Error Correction (FEC) using Low-Density Parity-Check (LDPC) codes [27]. Our “smart” decoding algorithm maintains high bit rates by adapting to the characteristics of the cross traffic. To mitigate our covert channel, we present an approach, which leverages live migration techniques to dynamically reposition flows and VMs reducing the leakage rates substantially.

The rest of the thesis is organized as follows: We begin with Chapter 2 covering the related works, followed by a discussion of the channel construction in Chapter 3. The mathematical model of our channel, analysis on bounds, and the decoding and thresholding schemes are presented in Chapter 4. Chapter 5 describes our experimental evaluation followed by a discussion of our mitigation schemes in Chapter 6. We present some additional aspects and limitations of our work in Chapter 7, and finally we conclude in Chapter 8.

CHAPTER 2

RELATED WORKS

A large body of literature deals with the study of Inter-Packet Delay-based (IPD) covert channels over traditional protocols such as IP/TCP or HTTP with a comprehensive survey available here [55]. However, there has been very little work to date that studies covert channels specifically in the context of modern, cross-machine data center networks operating under Software Defined and traditional networks. It is not clear how well traditional IPD channels will work in the cloud domain given the fact that cloud vendors employ strict isolation mechanisms and the infrastructure is significantly more complex (e.g., load balancers and anti-DDoS network boxes) with different sharing semantics. Additionally, the distinctive nature of network traffic in clouds, high processing speeds of the network infrastructure and the use of SDNs leaves many unanswered questions, which have not yet been explored. Sadeghi et al. [47] present the design of an IPSec-based VPN, which attempts to thwart covert channels arising from sharing of resources across LANs. However, their end-host based approach does not mitigate our attack, as it does not prevent delay variations from being transmitted across flows, which is exploited by our attack. Bates et al. [24] recently proposed a network-flow watermarking scheme that borders our work. However, this work focuses on determining co-residency as opposed to covert channels. The threat model we consider is also harder in the sense that cross-tenant communication between trusted and untrusted tenants is strictly forbidden meaning that cross-Virtual Network routing is disallowed. Furthermore, we develop a formal analytical model of our channel to study its characteristics and come up with information theoretic bounds on the leakage rate, which is not considered in these prior works. Additionally our work also encompasses various network environments such as SDNs, which, to the best of our knowledge, have never been explored in any work previously. Kanuparthy et al. [34] employ the same basic principle as ours however their work is not targeted towards building high speed covert channels in data centers, rather they argue the accuracy of the approach and demonstrate some degree of success in general internet routing domains and as an application of their approach mention covert channels. Some researchers have also proposed timing-based covert channels that attempt to mimic legitimate traffic patterns [26] to blend in with the non-malicious traffic. Others have focused more on provable undetectability of covert channels in independent and identically distributed (i.i.d) traffic [48, 39] – these works are geared primarily towards reducing detectability of covert channels, which are orthogonal but could be applied to our channel.

On a related plane, researchers also demonstrated host-based covert and side channels on commercial clouds and virtualized environments [52, 46, 54, 53]. These channels were based on low level hardware, such as the processor cache and the system bus, in order to thwart any software-based isolation mechanisms deployed by the cloud vendor. Although these channels were primarily presented as side channels, an adversary may conveniently use the same mechanisms to covertly send data to colluding VMs on the same server as demonstrated by Xiao et al. [53]. Many mechanisms have since been proposed to detect and thwart these channels, such as modified cache architectures and dedicated servers [50, 36, 7, 45], however none of these schemes can thwart our channel because of the novelty of our work and the use of a medium, which is highly multiplexed across a multitude of different tenants.

CHAPTER 3

CHANNEL CONSTRUCTION

Our scheme requires the existence of a party "inside" the trusted domain to leak out information to recipients outside. We use the term *insider* to refer to such a party, which could be for example a malware (which has infiltrated a trusted machine) or a disgruntled employee that wishes to leak out information. We refer to any other entity colluding with the insider (but not part of the trusted network) as an *outsider*.

We start by proposing a *unidirectional* construction for the channel where information can only flow in one direction, from the insider to the outsider. We note however, that bidirectional channels can be established by extending the same scheme, see Section 7.3.

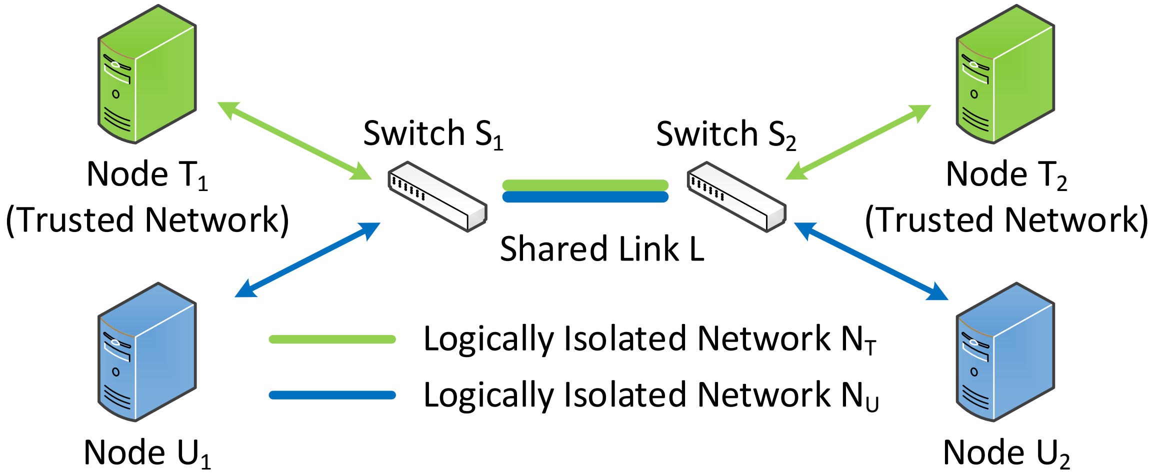


Figure 3.1 Set up for the covert channel. Nodes from the Trusted and the Untrusted Network sharing network resources.

3.1 Construction for Unidirectional Channel

Our channel is a timing-based, cross-Virtual Network (cross-VN) covert channel that relies on the underlying shared network resources in the data center to transfer data between logically isolated virtual networks. We consider the scenario where a switch or a shared network resource is handling traffic from two different flows belonging to two different logical networks with software isolation guarantees in place. We characterize these virtual networks in a manner such that nodes from a particular virtual network can only route to nodes within that network (as would be realized with, for example, VLAN-based isolation). In other words inter-VN routing is prohibited. For simplicity both networks have two nodes each as shown in Figure 3.1. Nodes T_1 (insider) and T_2 are part of a trusted network N_t, which is separated logically from network N_u, which essentially belongs to another tenant located in the same data center. Nodes U_1 and U_2 (both outsiders) are part of this other tenant's virtual network. The two networks are co-located in that they share the same network infrastructure, i.e., switch S_1 and S_2 connected by a link L. A Additionally, consider two flows in the network such that flow F_1 is a flow from T_1 to T_2 and flow F_2 is a flow from node U_1 to U_2. We call F_1 the *insider* flow and F_2 the *outsider* flow. For simplicity, switch S_1 can be modeled as an infinite buffer server that is serving jobs from two separate clients. In such a scenario, T_1 will use the insider flow to induce delays in to the outsider flow since both flows are sharing the underlying network. Node U_2 can extract the covert message by measuring the amount of latency experienced by its packets belonging to the outsider flow. This key insight, displayed in Figure 3.2 can be used by colluding nodes, on two different virtual networks to pass information between the two logically separated entities. For instance, the insider can increase or decrease its traffic through the shared link in order to signal a '1' or a '0' bit. As a result, even when nodes are not allowed to route to each other directly, an outsider, would still be able to infer the traffic pattern of the insider resulting in information leakage from the trusted to the untrusted domain. The bidirectional channels can be established by extending the same scheme.

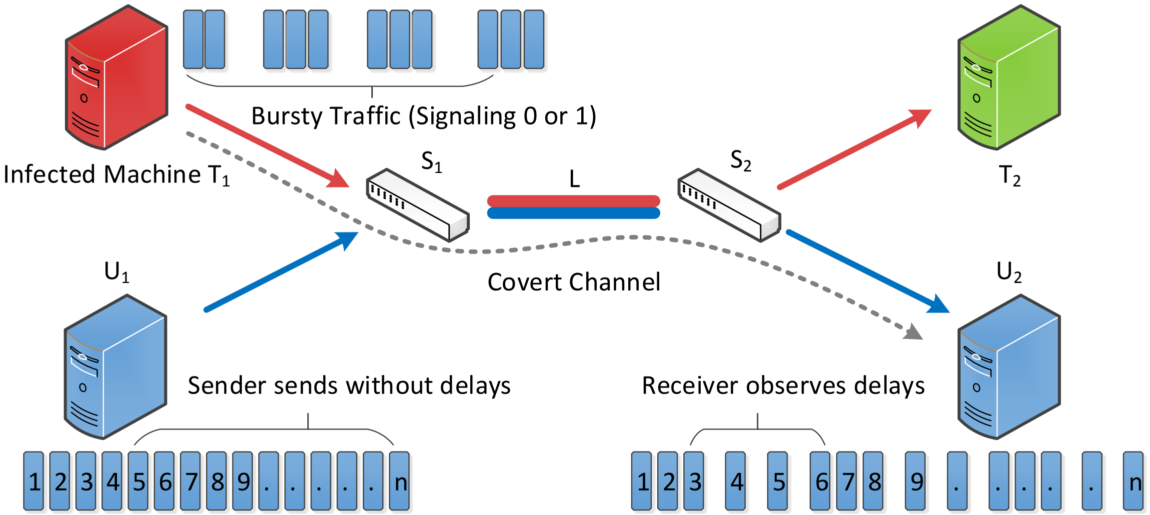


Figure 3.2 The insider (Infected Machine T_1) encodes the covert messages in the form of bursts. This in turn induces delays into the outsider flow observable by the receiver (U_2).

CHAPTER 4

MODEL AND ANALYSIS

Our timing covert channel can be modeled as a first-in-first-out (FIFO) queue shared by two packet processes initiated by users T_1 and U_1 respectively (Figure 4.1). Consider time is discretized into slots, and each packet requires one slot to service. At each time slot, both T_1 and U_1 can send at most one packet, which enters the queue at the beginning of the time slot. Though T_1 and U_1 are not allowed to communicate directly, because their packets share the same queue, T_1 can encode messages in the arrival times of its packets, which are passed onto U_2 (a partner of U_1) via queuing delays. Ideally, if U_1 sends a packet every time slot, the capacity of this covert channel can reach 1 bit per time slot; at each time slot T_1 simply idles to send a bit '0' or sends a packet to signal a bit '1'. However, this is not feasible in practice because the packets would be dropped due to the instability of the queue. Hence, we analyze the capacity and coding scheme in the timing covert channel under the restriction that the total packet arrival rate does not exceed the service rate; i.e., the queue is stable.

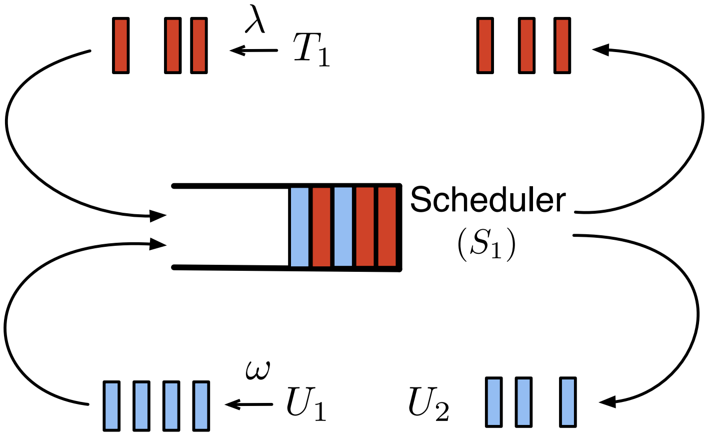


Figure 4.1 Timing covert channel in a shared queue. T_1 and U_1 both send packets to a scheduler, and communicate through a covert channel created in the shared queue; T_1 encodes a message on packet arrival times, and U_2 (a partner of U_1) decodes this information from queuing delays of her packets.

4.1 An Upper-bound on Channel Capacity

Assume T_1 wants to send one of M different messages during N time slots. It randomly picks a message $m \in \{1, \cdots, M\}$ (each message is chosen with probability $\frac{1}{M}$), and encodes it into a length N binary sequence, $\{\delta_1, \cdots, \delta_N\}$. At slot i, T_1 sends a packet if $\delta_i = 1$, and stays idle otherwise. To receive this message, U_1 sends n packets to the scheduler during the N time slots, among which n' packets leave the queue by time N. Denote U_1's arrival times by $\{A_1, \cdots, A_n\}$, and the departure times up to time N by $\{D_1, \cdots, D_{n'}\}$. Naturally, $D_{n'} \leq N$ and $n' \leq n$. U_1 and U_2 estimate T_1's transmitted message as $\tilde{m} \in \{1, \cdots, M\}$ from the queuing delays calculated from $\{A_1, \cdots, A_{n'}\}$ and $\{D_1, \cdots, D_{n'}\}$. During this encoding/decoding process, the information rate from T_1 to U_2 is characterized by $\frac{\log M}{N}$, assuming that the decoded message matches with the sent message, i.e., the error probability $P_e = \mathbb{P}(m \neq \tilde{m})$ is zero.

From Fano's inequality [28, Theorem 2.11.1], the information rate and error probability are related as follows:

$$\frac{\log M}{N} \leq \frac{1}{1 - P_e} \frac{I(m; \tilde{m})}{T}, \tag{4.1}$$

where $I(m; \tilde{m})$ is the mutual information. Mutual information characterizes the reduction in uncertainty of the original message m after observing the decoded message $\tilde{m}$. This is clear from the definition of mutual information: $I(m; \tilde{m}) = H(m) - H(m|\tilde{m})$, where $H(\cdot)$ denotes the entropy function.

Note in this encoding/decoding process, data is actually processed across the following Markov chain:

$$m \rightarrow \delta_1, \cdots, \delta_N \rightarrow A_1, \cdots, A_{n'}, D_1, \cdots, D_{n'} \rightarrow \tilde{m}, \tag{4.2}$$

where $X \rightarrow Y$ means that random variable Y is a function of random variable X. Applying data processing inequality [28, Theorem 2.8.1], which states the information contained in a random variable can not increase after processing, we have

$$I(m; \tilde{m}) \leq I(\delta_1, \cdots, \delta_N; A_1, \cdots, A_{n'}, D_1, \cdots, D_{n'}). \tag{4.3}$$

Due to the deterministic unit time FIFO service order, U_1 and U_2 know the length of shared queue at time A_k from the queuing delay $D_k - A_k$. Therefore, they can learn the number of jobs T_1 has sent between each pair of U_1's consecutive jobs, A_k and A_{k+1}. This means,

$$I\left(\delta_1, \cdots, \delta_N; A_1, \cdots, A_{n'}, D_1, \cdots, D_{n'}\right) \leq \sum_{i=1}^{n'} H\left(X_i\right), \tag{4.4}$$

where $X_i = \sum_{j=A_i+1}^{A_{i+1}} \delta_j$. Define $h\left(\mu, l\right)$ to be the maximum entropy of a random variable X, which takes values in $\{0, 1, \cdots, l\}$ and has mean $l\mu$. It can be shown that $h\left(\mu, l\right)$ is a concave function of both μ and l. Thus, from Jensen's inequality [37, (9.1.3.1)], we have

$$\frac{\sum_{i=1}^{n'} H\left(X_i\right)}{n'} \leq h\left(\lambda, \frac{1}{\omega}\right), \tag{4.5}$$

where the average packet arrival rate of T_1, $\lambda = \frac{\mathbb{E}\left[\sum_{i=1}^{N} \delta_i\right]}{N}$, and the rate of U_1, $\omega = \frac{n}{\mathbb{E}\left[A_n\right]}$, where $\mathbb{E}\left[\cdot\right]$ denotes the expectation value of a random variable.

Combining (4.1), (4.3), (4.4) and (4.5), we derive an upper-bound on the information rate of covert messages from T_1 to U_2:

$$\frac{\log M}{N} \leq \frac{\omega \log M}{n'} \leq \sup_{\lambda + \omega \leq 1} \omega h\left(\lambda, \frac{1}{\omega}\right). \tag{4.6}$$

Solving the above maximum numerically, we get the max information rate, namely the capacity of the covert channel, as 0.8377 bit per time slot, from which we can estimate the bandwidth of our covert channel; e.g., if the link bandwidth is 1 Gbps and packet size is 64 KB, an upper-bound on the bandwidth is about 1.72 Kbps.

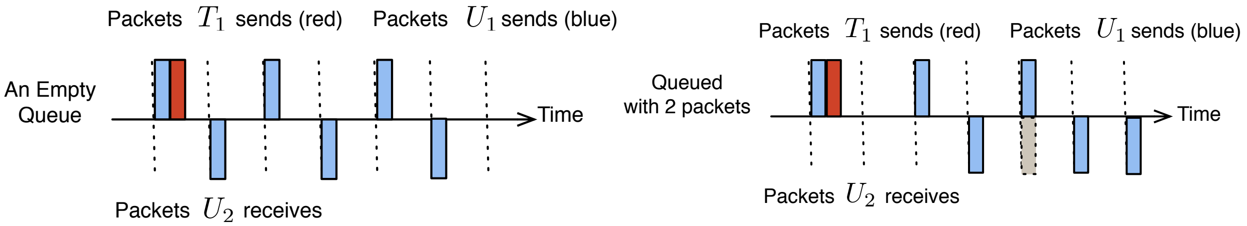
4.2 An Achievable Coding Scheme

In this section, we present a coding scheme for communication over the covert channel. As mentioned above, to achieve the highest information rate, U_1 must send a packet in every time slot, but this will result in instability of the queue and hence is not feasible. Therefore, we divide every N time slots to 2 phases. Phase 1 has K slots, and Phase 2 has the remaining $N - K$ slots.

Phase 1**:** At each time slot, U_1 sends a packet, and T_1 either sends a packet or idles with equal probability.

Phase 2**:** This phase is designed to keep the queue stable. U_1 sends a packet every s time slots, where $s \geq 2$. In every s time slots, T_1 takes one of three possible actions with equal probability. He either idles, sends a packet at the first slot, or sends packets in all s slots.

The information rate of this scheme is $\frac{K}{N} \times 1 + \frac{N-K}{N} \times \frac{\log 3}{s}$ bit per time slot, and the total arrival rate



(a) An empty queue case

(b) A non-empty queue case

Figure 4.2 Effect of queue on the covert channel. When U_1 and T_1 both send a packet in one time slot, we assume U_1's packets get serviced first. (a) Given an empty queue in the start, the packet sent by T_1 (the red block on the top) will not affect the timing pattern of packets received by U_2; hence the message sent by T_1 cannot be decoded. (b) If the queue is not empty in the start (e.g., with 2 packets buffered), U_2 can detect the packet sent by T_1 from an extra slot of delay (the grey block at the bottom), and thus decode the sent message.

to the queue is $\frac{K}{N} \times \frac{3}{2} + \frac{N-K}{N} \times \frac{s+4}{3s}$. Maximizing the information rate subject to not exceeding arrival rate 1 (keeps queue stable), we have $s = 2$ and $K = 0$ (Phase 1 is eliminated). The resulting information rate is 0.793 bit per time slot (which is very close to the upper bound), and the coding scheme is as follows: U_1 *sends a packet every* 2 *time slots. In every* 2 *time slots,* T_1 *takes one of three possible actions with equal probability; idling, sending 1 packet, or sending 2 packets.*

In order for U_2 to decode T_1's message, it should be able to distinguish between T_1's three actions. This requires the queue to be non-empty, as illustrated by the example in Figure 4.2. Ensuring that the queue is non-empty requires more coordination in terms of sending packets between T_1 and U_1, which renders the implementation of the covert channel more complex.

Another scheme, which does not require such coordination, is the following 2-phase mechanism.

Phase 1: At each time slot, U_1 sends a packet, and T_1 either sends a packet or idles with equal probability.

Phase 2: Both U_1 and T_1 idle completely.

The overall information rate using this scheme is $\frac{K}{N}$ bit per slot (the covert channel is utilized only during Phase 1), and the total packet arrival rate is $\frac{3K}{2N}$. To maintain the stability of the queue, we require that $\frac{3K}{2N} \leq 1$. This implies the maximum information rate of this scheme is 0.67 bit per time slot, achieved by picking $\frac{K}{N} = \frac{2}{3}$. The information rate of this scheme is close to the achievable scheme discussed earlier and does not require the receiver to distinguish between T_1's three actions, which in turn simplifies the encoding/decoding, hence we employ this in our experiments.

4.3 Decoding Scheme

In IPD channels like ours, to decode individual bits as either a zero or a one two problems need to be addressed. Firstly determining the threshold value is very important. Bits whose delay lies above the threshold value are decoded as one and those below are decoded as zero. Secondly, the issue of bit marking where we need to mark packets belonging to the same bit, i.e., where a particular bit starts and where it ends (syncing the sender and the receiver). We address the two issues separately below:

4.3.1 Threshold Calculation

The optimal thresholding value can be determined using a brute force scan, which requires the original string of bits to minimize the error. As the original string is not known to us, we use the mean of the dataset for simplicity. However, due to the changing nature of the cross traffic, such as a flash crowd, a solitary global mean gives poor results as shown in our evaluation in Section 5. Our scheme solves this problem by splitting the message up. Instead of performing the analysis above on a large stream of *n*-bits, the decoder splits up the stream into multiple *regions* where a region is a sequence of bits with uniform cross traffic across. This "region marking" is performed over the entire bit sequence at the decoding side with no assistance from the sender. The decoder takes the first i bits in the message and calculates their mean m_1. Then the second i bits are taken and their mean m_2 is compared against mean m_1. If both values are "close" to each other (half a standard deviation in our case) then the first $2i$ bits are marked as one big region with one threshold value to decode the bits else the two regions are marked as separate regions. The process of comparing the mean of each subsequent set of i bits with the last marked region continues until the entire message has been iterated and all the regions have been marked. Afterwards a separate threshold value is used for each region reducing the overall error by limiting the contamination of flash crowds and sudden variations in cross traffic to specific regions.

4.3.2 Bit Marking

The second major issue in IPD channels is that of bit marking. In our scheme, the duration of the on-off interval is known beforehand to both the sender and the receiver. The sender starts off by sending a preamble signal (predetermined sequence of bits), which effectively syncs the two. The preamble signal is also used by the receiver to determine the average number of packets in the on and off intervals. This gives the algorithm reference points of what latencies and numbers to expect. To identify each bit, it keeps cumulating the packet delays that it observes and if the cumulative delay crosses the duration of the on (or off) interval and the number of cumulated packets are “close” to the average number of packets observed in the on (or off) interval of the preamble sequence then the cumulated packets are all marked as belonging to the same bit. This process keeps the sender and the receiver in sync. However, since the cross traffic keeps changing, the number of packets in an interval might increase or decrease (e.g., packet drops from queue buffering) resulting in cascading errors. To cater to this change, a third parameter is introduced, which is the mean of the current region. Empirically, we observed that the mean of a region is inversely proportional to the number of packets in the on-off interval. This can be understood intuitively as well since a higher mean represents higher traffic loads and a higher probability of packet loss. The relationship between the number of packets and the mean was found to be linear in nature. Hence, before the decoding algorithm marks the bits in a region it adjusts its expectations for the number of packets by taking into account the mean of that region.

CHAPTER 5

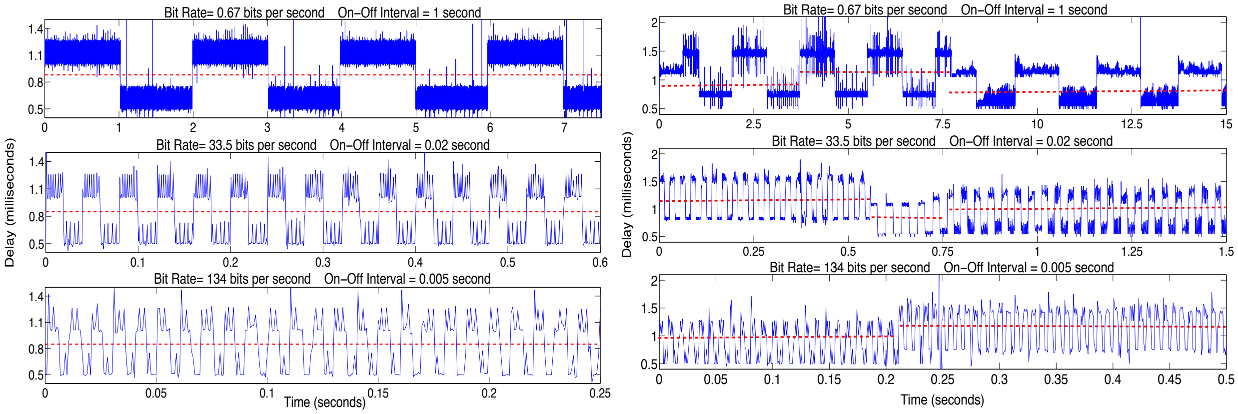
EVALUATION

Multiple distinct testbeds and commercial clouds were used for a thorough experimental evaluation. We first explain all environments here and then present the main results from a few them in the interest of space.

The first testbed was a cluster comprising six machines set up in a dumbbell topology with three machines on each end and two disjoint paths connecting the two sub-clusters together. All links were 1Gbps and the switches were 8-Port Non-Blocking Gigabit GREENnet Full-Duplex Switches (Model # TEG-S80Dg). In this paper we refer to this testbed as the *cluster*. The second testbed was our in-house cloud [5], which is an SDN-based (OpenFlow [40]) testbed. It is composed of 176 server ports and 676 switch ports, using Pica8 Pronto 3290 switches via TAM Networks (running Open vSwitch [16]), NIAGARA 32066 NICs from Interface Masters, and servers from Dell. It is called Ocean Cluster for Experimental Architectures in Networks (OCEAN) so we simply refer to it as OCEAN here. Results for the Emulab Network Emulation Testbed [13] were also gathered along with extensive simulations on Network Simulator-2 (NS-2) [15], both of which allowed us to test various topologies (Fat Tree, VL2 etc.) with varying sizes. Finally, we successfully demonstrated the practicality and seriousness of our channel by testing it on EC2 and Azure.

We implemented the coding scheme presented in Section 4.2 but did not incorporate error-correction. We discuss our modified algorithm, which performs Forward Error Correction (FEC) on the decoded bits in Section 7.4. Our results here were achieved using an alternating bit sequence for visual clarity however, we also tried arbitrary bit sequences (such as private keys) and achieved very similar results.

Wherever needed, cross-traffic was generated using network traces from actual data center traffic dumps [25], normalized for our set up so as to cater to link speeds and bandwidth availability. The generated cross-traffic turned out to be a mix of temporally-spaced TCP and UDP flows with different durations and sizes. During flow extraction, we filtered out extremely small micro flows as they did not affect the experimenta-



(a) Without Cross-Traffic

(b) With Cross-Traffic

Figure 5.1 1s are represented by an on interval and 0s are represented by an off interval. Alternating 1s and 0s and as were sent. The red line is the threshold line below which we decode a zero and above it is a one.

tion in any way and sped up the empirical analysis considerably. The volume of cross-traffic generated in all experiments was in correspondence to what was observed in the actual traffic dumps.

5.1 Waveform Experiments

Figure 5.1a shows the first set of experiments that we conducted. Following the set up shown in Figure 3.2, we used node T_1 (encoder) to send a message containing a sequence of alternating 1s and 0s. This bit sequence was encoded in the form of alternating on-off intervals for a UDP flow, which in turn induced corresponding latency variations into the constant stream of packets (also UDP) being sent from node U_1 to node U_2 (which was the decoder). Figure 5.1a shows the latency between successive UDP packets (of the outsider flow) as seen by node U_2. We conducted an additional series of experiments with actual cross traffic as shown in Figure 5.1b and still observed high bit rates and low percentage error.

The waveform for similar experiments on OCEAN running at 268 bits/s can be seen in Figure 5.2. We deployed our custom Floodlight Controller [17] and experimented with various topologies. We found that our covert channel works equally well, and in some cases better than traditional networks, in an SDN environment even for very large topologies. In fact we got higher bit rates on OCEAN because of faster packet processing at the switches and the server. The relationship between percentage error and bit rate in OCEAN, was found to be very similar to the cluster case however the spike in percentage error appeared on higher bit

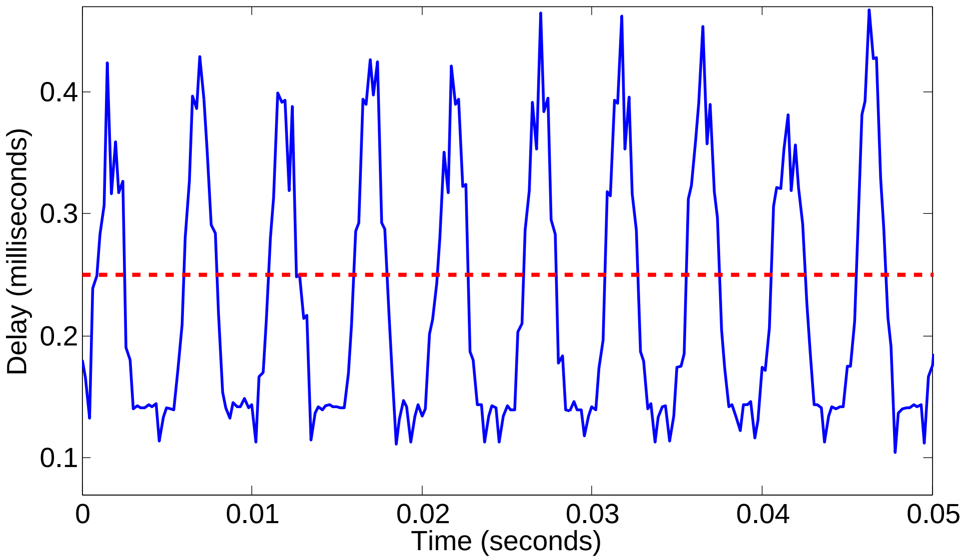


Figure 5.2 Waveform for OCEAN running at 268 bits/s.

rates because of more advanced NICs on the servers.

5.2 Distribution Functions

To measure the distribution of packet delays during the on-off intervals we plot the distribution functions. Figure 5.3 shows that during the on interval packets tend to cluster around two values, which are both above the threshold latency induced by an on interval. A similar phenomenon can also be seen for the off interval where packets primarily group around two latencies. This intra-interval latency gap does not affect our results in any way, as these values fall well above or below the threshold value used by the decoder. We believe this latency gap is a function of the packet size as we observed fluctuating values with different combinations of packet sizes. This is why we see four “steps” (instead of two) in Figure 5.4, which plots the Probability Distribution Function for our data sets.

5.3 Percentage Error vs Bit Rate and Message Splitting

Table 5.1 explores the relationship between percentage error and bit rate. The channel becomes harder to decode, as bit transitions become packed more tightly together. If we continue reducing the on-off interval

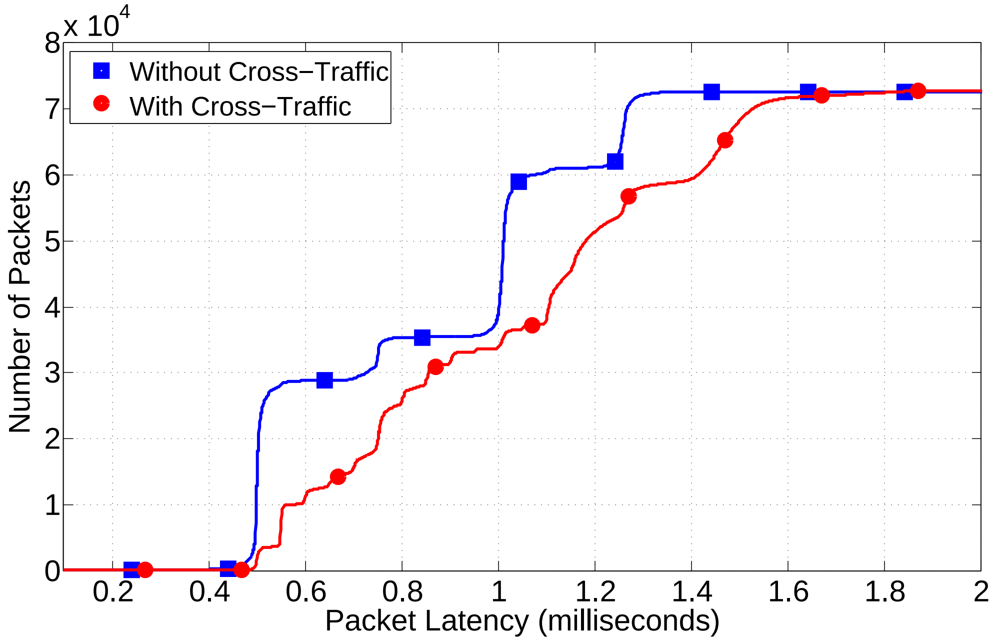


Figure 5.3 Cumulative Distribution Functions.

Bit Rate	Error without Cross Traffic	Error with Cross Traffic (No Message Splitting)	Error Cross Traffic + Message Splitting
100	0 %	3.30 %	0 %
200	0 %	42.80 %	0 %
500	0 %	Error $>$ 80 %	8.68 %

Table 5.1 Our channel performs at very high speeds in the absence of noise. However, as cross traffic increases the error rates spike but our adaptive decoding scheme coupled with message splitting almost removes the entire error.

then beyond a point (region 2) the error rate spikes. We noticed this upward spike for both cross-traffic and without cross-traffic experiments at almost the same bit rate, suggesting that in Region 2 the errors are because of the limitations of the hardware (the NIC can not send and receive packets at such small intervals). On the other hand, in Region 1 all error incurred is solely because of the cross-traffic. The table also highlights the benefits of our message splitting mechanism. The substantial difference between the error values for the decoded message with and without message splitting shows the effectiveness of our decoding scheme.

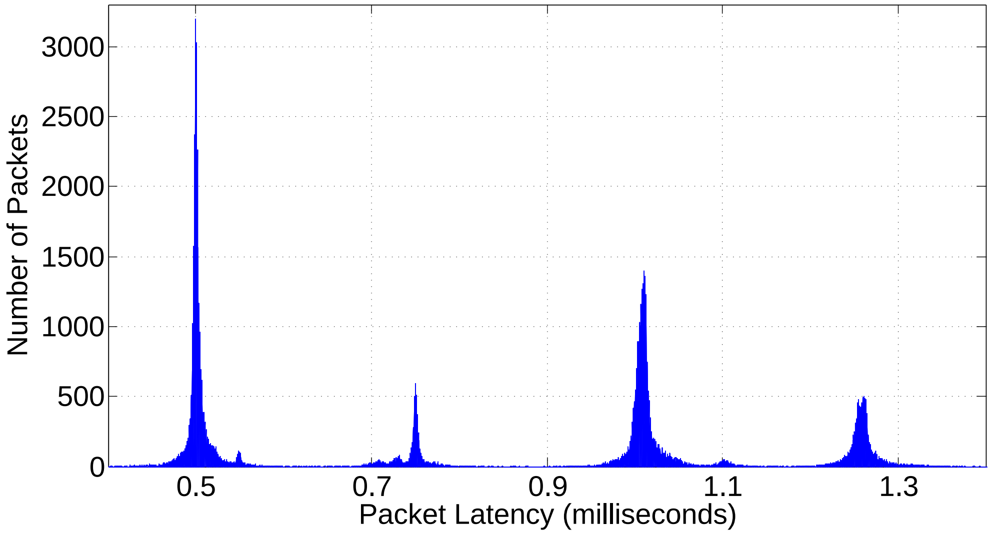


Figure 5.4 Probability Distribution Functions.

5.4 Effect of Packet Size

Since the participants of the covert channel have the ability to tune the size of packets we decided to check the effects of this parameter as well. We observed that packets that are too small fail to induce noticeable latency into the packets of the outsider flow resulting in a very high percentage of errors. On the other hand, using very large packets results in the *capture* of the channel by the insider flow, causing packets from the outsider flow to starve and eventually get dropped as the queue fills up. We found that packets with size around 16KB worked the best and induced visible latencies while keeping the percentage error to a minimum. This phenomenon is captured in Figure 5.5.

On-Off Interval	Upper-Bound	Encoding Scheme	Empirical Bit Rate
0.5	1.6754	1.34	1.34
0.1	8.377	6.7	6.633
0.02	41.885	33.5	31.49
0.01	83.77	67	61.975
0.005	167.54	134	122.61

Table 5.2 Table comparing theoretically determined bit rates to empirically achieved goodput (bit rate adjusted for error).

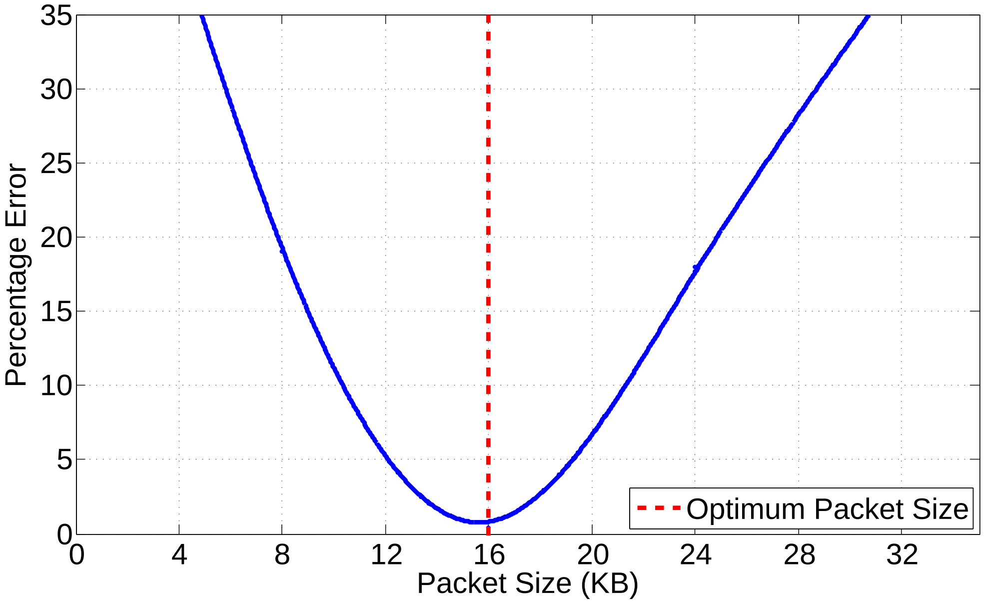


Figure 5.5 We kept the bit rate constant at 268 bits/s and ran experiments for packets sizes ranging from 4KB to 32KB with 4KB steps and observed that the lowest percentage error is achieved at 16KB packet size.

5.5 Theoretical vs Empirical Bit Rate

To measure the empirical performance of our channel, we compare the empirically measured goodput (bit rate adjusted for error) of the channel against the bit rates determined theoretically from the upper-bound and the simple scheme mentioned in Section 4. As seen in Table 5.2, the goodput was observed to be close to the theoretically calculated bit rates in all experiments, highlighting the applicability of our analytical model to practice.

5.6 Effect of Total Traffic Load/Network Condition

To measure how network conditions affect our channel we performed a series of experiments increasing link utilization by 20% before each run. We observed a constant upward shift in the average delay of packets at the receiver however, since the increase is roughly uniform it does not interfere with the decoding process. Figure 5.6 illustrates this step-wise increase. We conclude that our channel can function efficiently in varying network conditions irrespective of the degree of traffic load.

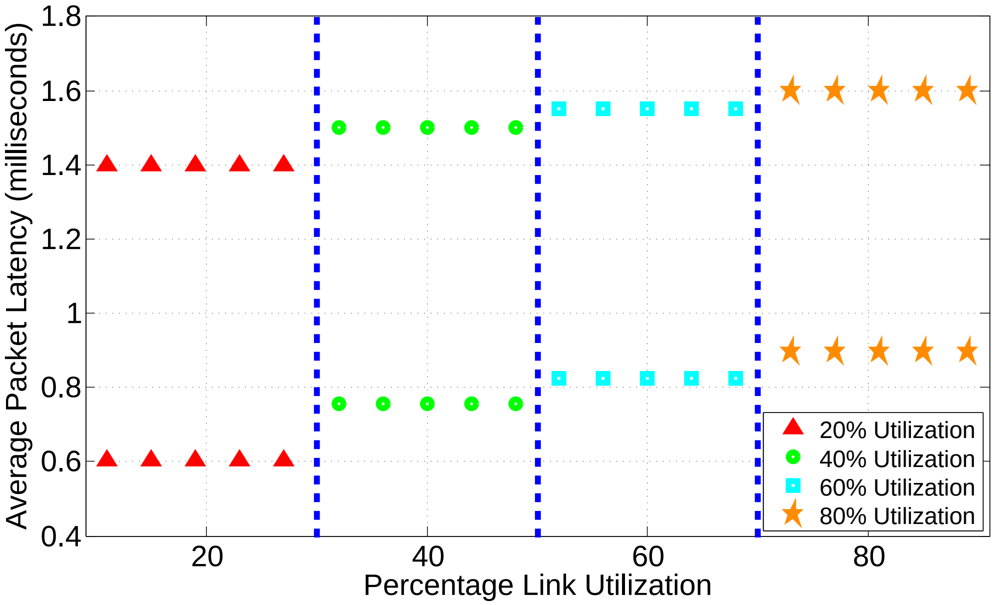


Figure 5.6 Eeffect of traffic load.

5.7 Detectability of the Covert Channel

Several studies [38, 29] have suggested that under normal conditions, network traffic exhibits self-similarity whereas this property is lost during anomalous conditions such as malicious intrusions and resource exhaustion attacks [44]. While a Cumulative Distribution Function (CDF) can be used to flag such anomalous traffic patterns, the Hurst exponent **(H)** is typically used as a measure for determining self-similarity. For a self-similar process the Hurst exponent takes a value between 0.5 and 1, and the degree of self-similarity increases as the value approaches 1. To measure how anomalous our channel seems to a network profiling tool we measured the Hurst exponent in the presence of background traffic under different bit rates. Figure 5.7 shows that even for bit rates as high as 200 bits/s our covert channel blends in nicely with the cross-traffic as the Hurst exponent stays well above the threshold value for self-similarity. It is worth mentioning here that even though it seems from our graphs that the waveform is very distinct and easily recognizable, it is only because we are transmitting a bit sequence with alternating ones and zeros, which is *not* the case with actual covert messages.

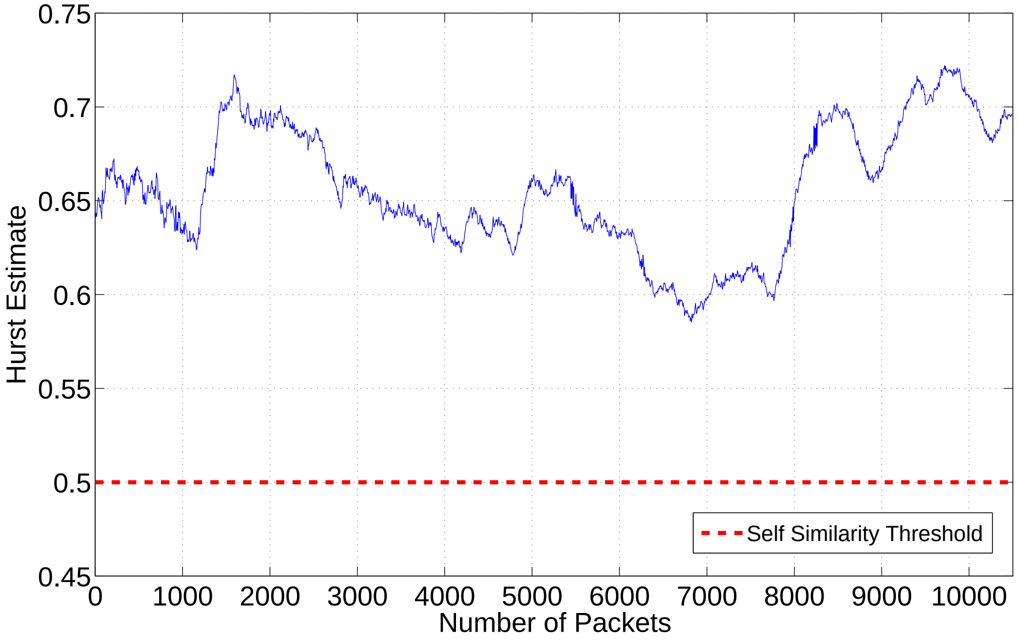


Figure 5.7 The observed value for the Hurst exponent consistently stays above the self-similarity threshold value indicating that it is hard to profile our channel.

5.8 Effect of Queuing Policy and Hypervisor

To study the sensitivity of our covert channel to different queuing schemes, we conducted simulations on NS-2, which enabled us to study a broader set of queuing policies than is possible with hardware switches, some of which do not implement or give access to configuration of multiple queuing strategies. So far, our results were based on switches using FIFO queuing. In this particular experiment we tried Stochastic Fair Queuing (SFQ) [41], Fair Queuing (FQ) [43], Drop Tail (DT), Random Early Detection (RED) [30] and Deficit Round Robbin (DRR) [49]. Figure 5.8 shows some waveform results that we obtained. Surprisingly we found that our channel works extremely well in almost all schemes that we tried. We found that RED and DT behaved remarkably similar to each other so we show one graph representing both of them.

Building on this result, we contend that the underlying hypervisor design does not affect our channel. The virtual switch (residing on the host) uses one of the aforementioned policies and as shown above they don't have any negative effect on the channel's performance. Similarly, another entity that could potentially disrupt the channel is the hypervisor scheduler. Here again, we point out that since we managed to run all our experiments on a wide variety of different hypervisors (Xen, VMWare's vSphere, VirtualBox, Amazon's customized Xen, and Azure's Hyper-V) without any modification to the code or channel semantics, we are

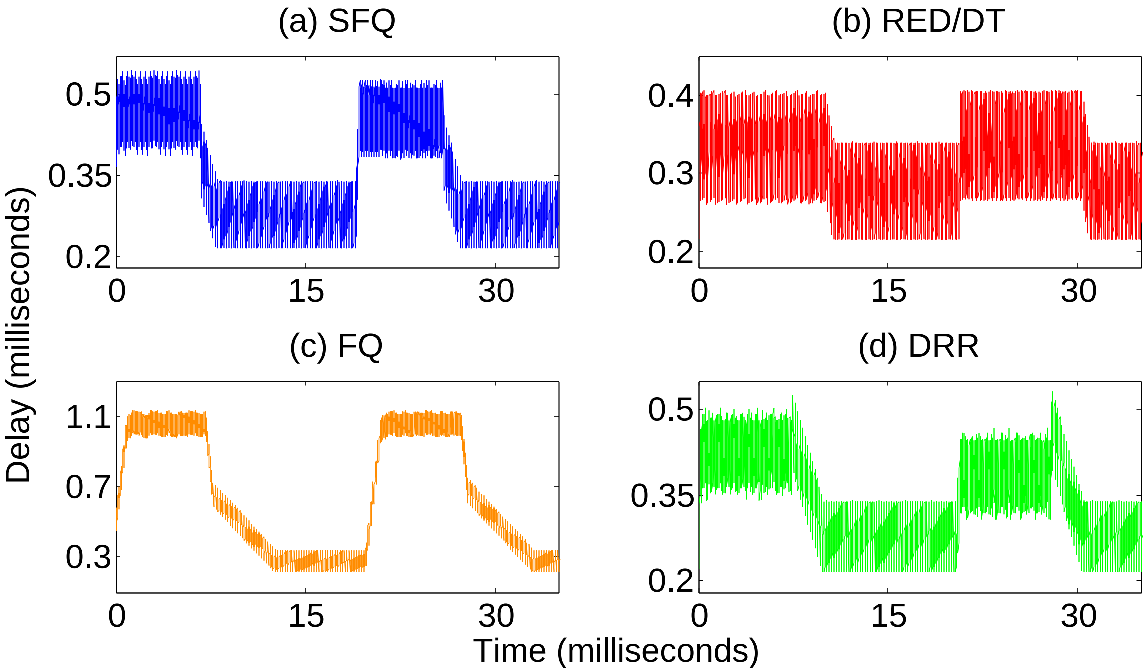


Figure 5.8 Waveform simulations for various queuing schemes.

confident that the underlying hypervisor design, particularly the scheduler, does not influence our channel in any way.

5.9 Microsoft Azure and Amazon EC2

Apart from testing our covert channel in a synthetic environment, we performed experiments on EC2 and Azure as well. To set up our channel on these commercial environments we had to successfully achieve link sharing on the insider and outsider flows. This was mainly achieved through hit and trial however, we tried to increase our chances of link sharing by provisioning senders and receivers very carefully in these clouds. Specifically, the documentation of EC2 hints at the fact that inter-availability zone traffic could share underlying resources. Hence we provisioned the senders and the receivers on neighboring availability zones while keeping each sender-receiver VM-pair on the the same Virtual Private Cloud (VPC). This greatly increased our chances of achieving coresidency on links in EC2. A similar argument holds true for achieving link-sharing in Azure but again some hit and trial is needed.

For these experiments, we performed serveral different runs at different times of the day and noticed that

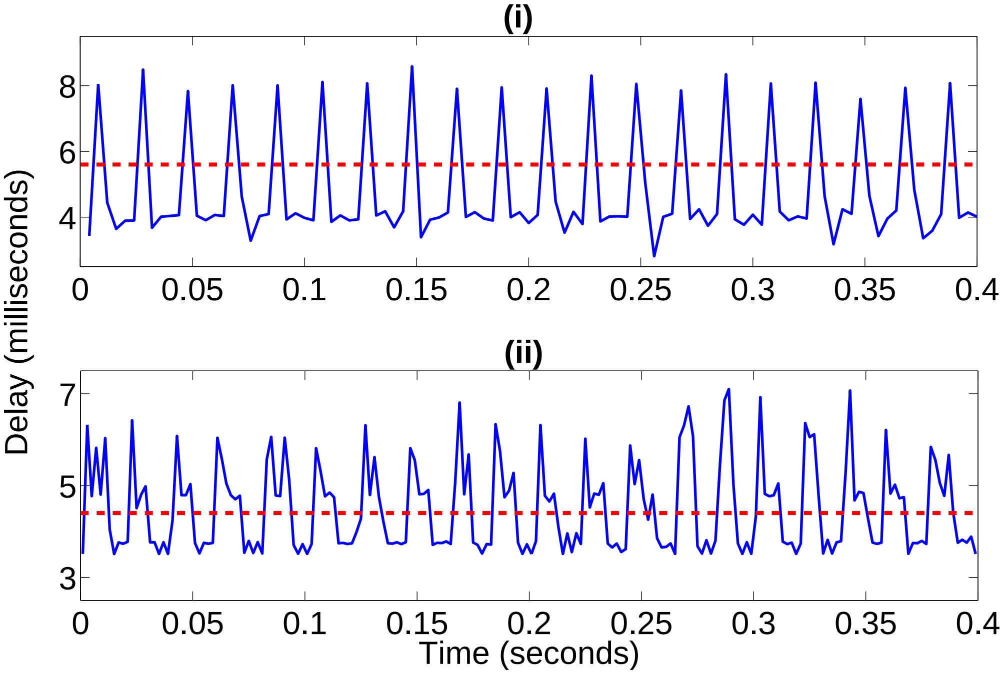


Figure 5.9 Waveforms for commercial clouds.

even though the effect of cross-traffic was noticeable, our channel performed decently. Figure 5.9(i) shows one such run at 100 bits/s for Azure and Figure 5.9(ii) shows the waveform for EC2 at 100 bits/s. We also noticed that for different types of instances our results varied. For example, the compute-optimized instances gave us better results compared to instances with very limited resources. Intuitively, it can be argued that instances with more resources, particularly higher network bandwidth and presumably better NICs, should improve our results. This is because our channel depends on how fast we can send packets and how much we can send. The aforementioned effect has been captured in Figure 5.10 for Azure. A similar trend was observed in EC2 with "larger" instances performing much better.

5.10 Emulab Testbed

We created a 3-level Fat Tree comprising 135 nodes on Emulab and tested our covert channel on it. We found that the channel worked flawlessly. The result for a particular run of the channel at 67 bps is presented in Figure 5.11. We also tried other topologies such as VL2 [32] and received encouraging results, which were similar to the Fat tree so we omit them here. These experiment and others done on OCEAN over large

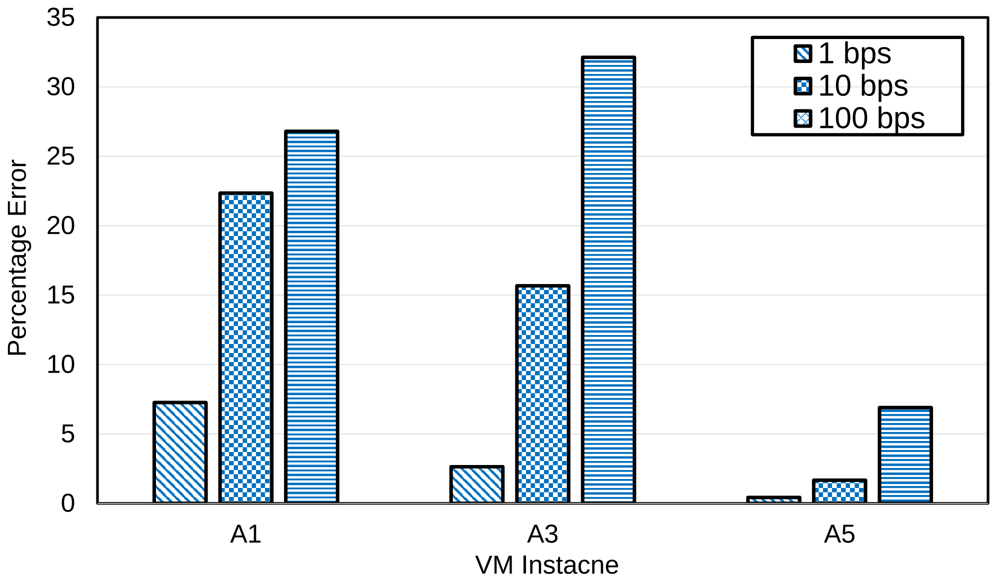


Figure 5.10 Performance of various instance types in Azure with A1 being the smallest and A5 the largest.

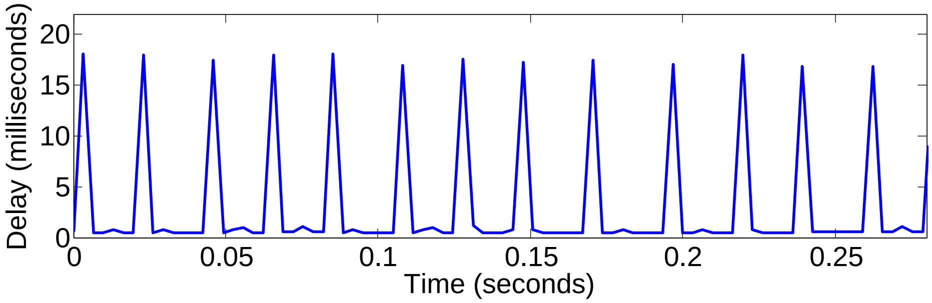


Figure 5.11 Waveform obtained from our covert channel setup on Emulab.

data center topologies highlight the fact that our covert channel operates at high bit rates irrespective of the topology and the network environment (SDNs or traditional networks).

CHAPTER 6

MITIGATION

Many researchers in the covert channel community argue that covert channels can never be eliminated from a system in their entirety [55, 33, 42] unless sharing is done away with altogether [46]. For this reason we present a mitigation scheme that primarily aims to rate-limit the capacity of the covert channel by essentially minimizing the time spent on shared resources rather than eliminating the virtualization (or multiplexing) infrastructure.

Our mitigation scheme is built on top of two key insights. Firstly, data centers are typically over-provisioned in terms of the number of paths between two communicating hosts to cater for link or switch failures [32]. Secondly, data centers often employ high-speed load balancers (hashing 5-tuple flow IDs onto different outbound interfaces) to ensure efficient link utilization and to reduce link congestion [32]. However, such static load balancing may increase the attacker's ability to establish co-location; by repeatedly modifying the 5-tuple the attacker can cycle through a set of paths until one shared with the outsider is attained. With these insights in mind, we developed a scheme that involves dynamically migrating flows in order to reduce the amount of time spent on the same underlying resource while minimizing modifications to the load balancing infrastructure. The basic idea is to select a candidate flow after it has been scheduled on a particular link and migrate it to a different path dynamically and repeatedly (if possible), ideally node-disjoint or at least edge-disjoint, based on the characteristics of a flow or the current state of the resource being shared. It is important to note here that reallocation of flows should not be done too quickly so as to maintain stability of transmission. We explain various different mechanisms that can be employed, in combination, depending on the circumstances. For instance if a trustworthy tenant is scheduled alongside an untrusted tenant, the flows of the untrusted tenant could be assigned a more aggressive hopping scheme. We present some schemes for hopping and their performance in the section below:

6.1 Path Hopping

If multiple node-disjoint paths are available between the insider and outsider's flows, changing the path of either of the flows would eliminate the sharing and severely limit the capacity of the channel. This "post-load balancing" migration of the flow can be achieved with simple modifications to the load balancing strategy. With network virtualization platforms in place, there is also an opportunity to migrate entire virtual networks around. LIME[35] is one such example where overlay networks are migrated to a distinct or partially distinct set of physical resources. In order for the load balancer to migrate a flow, we first need to decide which flow to migrate (flow selection). We then need to decide on a location where we move the flow to (flow placement). We present some alternate simple schemes for both problems below:

6.1.1 Flow Selection

Similarity-Based Selection: SDNs give us the ability to query real time traffic statistics from each switch. If two flows are found to be similar in terms of their link utilization (or other such metrics), we simply swap them with each other so as to minimize the effect of hopping on the other flows. If either of the swapped flows happens to be a participant in the covert channel, the channel will break down.

Timed Selection: In this scheme we associate a timer with each flow and whenever the timer expires we migrate the flow to another path.

Random Selection: The system has a global timer and when the timer expires a (weighted) coin is flipped for each flow. If the coin lands on its head then we migrate the flow, else we leave the flow as it is.

6.1.2 Flow Placement

Random Placement: Select the destination path randomly from a set of paths.

Anti-Social Placement: Select the destination path that is least crowded (in terms of the number of flows or the link utilization). Such live migration also has the effect of alleviating congestion.

Quick Selection: Select the destination path that gives the least downtime.

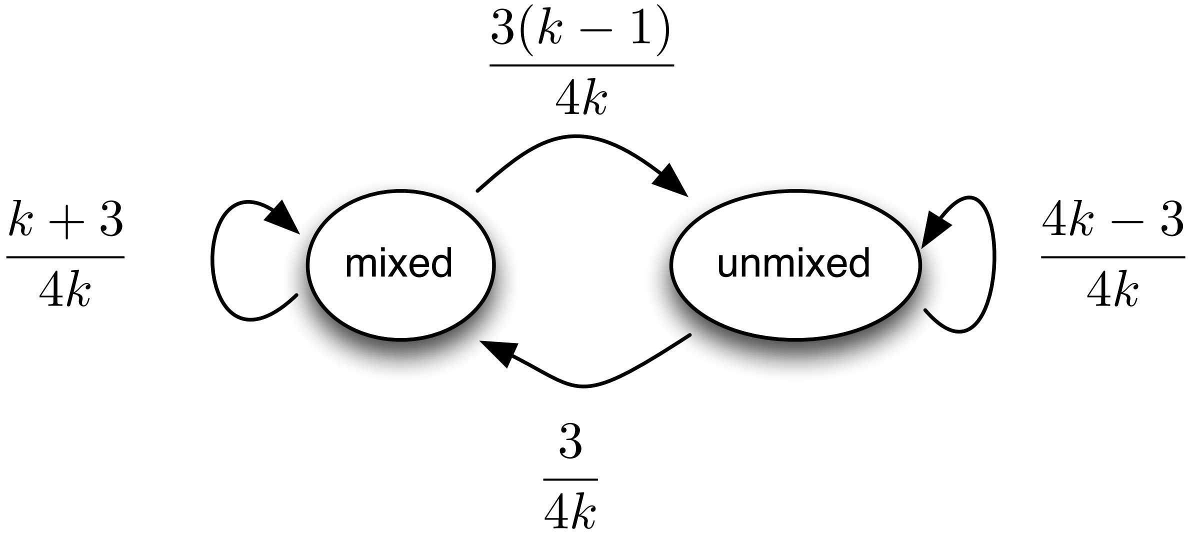


Figure 6.1 The Markov chain of state of the covert channel applying Random hopping and Random Placement.

6.2 Location Hopping

Our mitigation scheme also has the potential of mitigating cross-VM covert and side channels [46, 52, 51]. Migrating one of the VMs to a different server would eliminate any sharing. We call this process "location hopping" and all schemes mentioned in the previous section can be applied to the context of VMs directly.

6.3 Performance Analysis of the Mitigation Schemes

We analyze the performance of two of the proposed schemes below, namely Random Hopping and Timer-Based Hopping (for flow selection) joined together with Random Placement (for flow placement). Assume there are in total k paths available for selection, and define state $mixed$ to be the case when the two flows reside on the same path (the covert channel exists), and state $unmixed$ to be the case when they are scheduled onto different paths (the covert channel is eliminated).

Random Selection + Random Placement: In this mitigation scheme, every time the global timer fires, each flow is migrated with probability $\frac{1}{2}$, and the destination path of the migration is randomly selected from the k paths. As a result, the state transition of the covert channel can be described by the Markov chain in

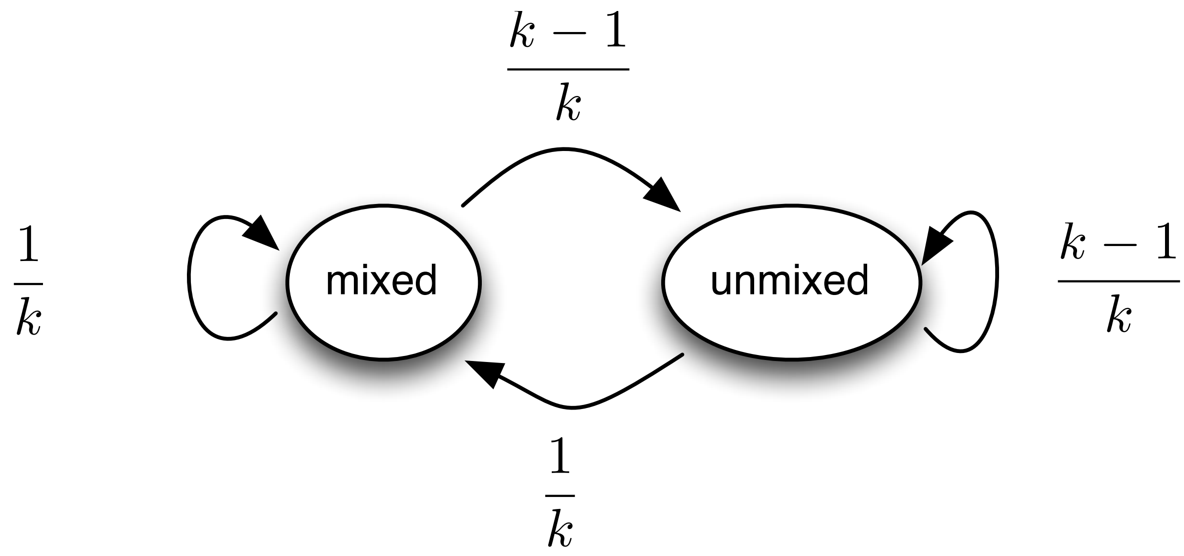


Figure 6.2 The Markov chain of state of the covert channel applying Timed Hopping and Random Placement.

Figure 6.1, from which it is easy to show on average for $\frac{1}{k}$ of time, the system stays in state $mixed$, i.e., the covert channel exists.

Timed Selection + Random Placement: In this scheme, the channel state may change whenever either flow's timer fires. Suppose the current state is $mixed$, and the timer of one flow fires, then the probability of this flow being allocated a different path rather than staying on the one shared by the other flow (and the channel state is changed to $unmixed$) is $\frac{k-1}{k}$. The same transition probability holds when the timers of both flows are fired. The Markov chain in Figure 6.2 illustrates this state transition. We can again show that the system stays in state $mixed$ for $\frac{1}{k}$ of time on average.

6.4 Results of Path Hopping on OCEAN

We implemented the schemes described above and show the performance of Random Selection + Random Placement in Figure 6.3. The region where the sharing took place can be seen in the start followed by three intervals of solitary placement for both the insider and outsider flows (although we are only showing the outsider flow here). We noticed downtimes on the orders of tens of milliseconds mostly to a couple hundred of milliseconds at times. Given these low downtimes we believe that we can hop more frequently if a tenant reputation system is in place and the tenant has a low reputation score. This would ensure that the tenant's flows keep moving around in the data center, which in turn would induce an unstable covert channel (if the insider does manage to restart sharing somehow).

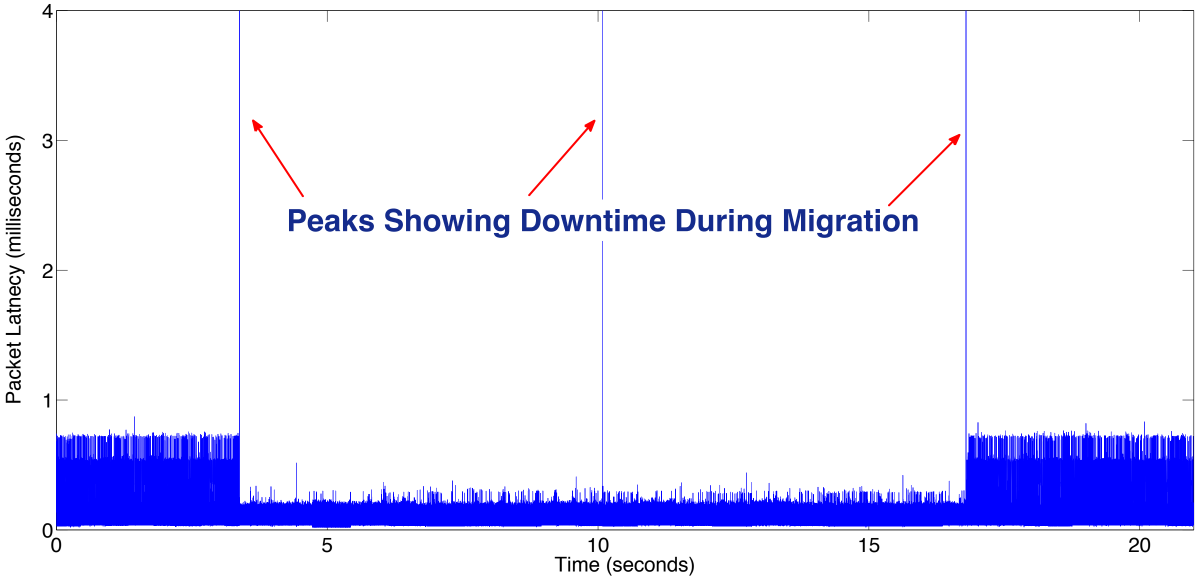


Figure 6.3 Random Selection + Random Placement Mitigation Scheme: The graph clearly shows the regions where the flows are overlapping and the covert channel is operational and how hopping reduces the bit rate. The three long peaks are the downtimes we witnessed during the hopping.

CHAPTER 7

DISCUSSION AND LIMITATIONS

We now discuss and analyze several additional aspects of our channel, and mention some limitations of our study.

7.1 Clock Synchronization

Clock de-synchronization can be a problem in timing-based covert channels. Researchers have used various mechanisms to address this issue such as with the use of self-clocking codes [52]. In order to keep things simple, we re-sync our sender and receiver after every *k*-time units by sending a predefined sequence of bits (e.g., Barker Code [23]) which is known to both the encoder and decoder before-hand. Anytime the decoder sees this pattern of strings it resets its clock as soon as the sequence terminates thereby re-syncing with the sender.

7.2 Effect of Load Balancing

Load balancing can play a pivotal role in reducing bit rates of timing-based covert channels as we demonstrated both analytically and empirically in Section 6. However traditional static load balancing (e.g., consistently hashing 5-tuple flow IDs onto outbound interfaces), which is the dominant form of load balancing in clouds today, fails to deliver a meaningful impact on covert channels as it does not directly reduce the likelihood the insider will share a network queue with the colluding outsider. In fact, static load balancing may increase the attacker's ability to establish a channel—by repeatedly modifying the 5-tuple the attacker can cycle through a set of paths until one shared with the outsider is attained.

Bit Rate	% Error Without LDPC	Code Rate =3/4	Code Rate =3/5	Code Rate =1/2
100	0	0	0	0
200	0	0	0	0
500	8.68	8.4	10	5

Table 7.1 A comparison of percentage error in the received bits with and without the use of FEC. Various values for LDPC code rates (*k/n*) are shown for reference.

7.3 Case for the Bidirectional Channel

Thus far we have focused exclusively on unidirectional covert channels, as performance and properties of the bidirectional case are very similar. We note that establishing a bidirectional channel can provide more power to the adversary. For example, the outsider can provide control information back to the insider, improving transmission speed and error tolerance (as they can precisely indicate which data blocks are lost due to transmission error, and request retransmission), and potentially reduce data transmission requirements (as they can just request certain blocks of interest, or perform interactive exploration of the private data). The construction of the bidirectional channel is quite similar to the unidirectional version, with the difference that now the system has four flows (2 insider flows and 2 outsider flows), 1 pair in each direction with each insider flow inducing delays into the outsider flow.

7.4 Dealing with Errors

A particular property of our channel is the fact that delay-based errors can only manifest themselves in the zero bits of a message sequence. Since cross-traffic, or any other type of noise, can only add to the latency we never see a one bit fall below the threshold and get flipped into a zero. Hence the one bits always stay error-free (not accounting for packet loss), which leaves us the zero bits in the string. There are several error-correction schemes that could be used, however we have chosen Low-Density Parity-Check (LDPC) codes [31] on our bit sequence to perform Forward Error Correction (FEC). LDPC codes have found renewed use in many network protocols including 802.11 standard and 10GBase-T Ethernet primarily because of high speed encoding/decoding features. In our case, Node T_1 encodes a *k-bit* message to form a *n-bit* encoded block where *n >k*. U_2 will be able to successfully decode the message using any *k* correctly received bits, implying that now U_2 can tolerate up to *n-k* losses for every block. Ideally, the redundancy should be in proportion to the prevailing error-rate on the channel and the nature of the data being encoded. The "Optimized Decoding Algorithm" that performs the decoding is presented in Algorithm 1 and the results are shown in Table 7.1. We have not tried other error-correction schemes that might be more suitable for our case so we list this as a limitation of our work.

Algorithm 1 Optimized Decoding algorithm

```
$Decoded\_Bits = \{\}$
$Corrected\_Bits = \{\}$
$Packet\_Delays$= {All observed packet delays}
$T_{min}$=min($Packet\_Delays$)
$T_{max}$=max($Packet\_Delays$)
for $i$ in $Packet\_Delays$ do
    for $j$=$T_{min}$ to $T_{max}$ do
        $Bin$=$Packet\_Delays(i, i + k)$
        # where $k$ is the length of the bit subsequence
        $[error_j, bits_j]$ =Decode($Bin$, $j$)
        $Error = error_j$
        $Bits = bits_j$
    end for
    # get index of minimum error
    $Index_{min}$=min($Error$)
    $Decoded\_Bits$=$Bits(k)$
    $i = i + Index_{min}$
end for
$Corrected\_Bits = LDPC\_Decoder(Decoded\_Bits)$
```

CHAPTER 8

CONCLUSION

Software-isolated virtual machines and networks create the illusion of a personally-owned computing machine with a secure communication channel. However, the shared physical resources underlying the virtualized isolation present the opportunity for malicious data transfer.In this paper, we highlight a growing trend in covert channels and present novel ways of compromising the private data of a user by exploiting the weaknesses that go hand-in-hand with the flexibility that virtualization has to offer. We present a discussion on cross-VN covert channels and analyze their channel capacity. We propose a defensive scheme that reduces the effects of these channels by decreasing the time spent on shared resources and complicating the ability of the attacker to "map out" the network.

REFERENCES

[1] Amazon Again Beats IBM For CIA Cloud Contract. `https://tinyurl.com/oegycn7`.

[2] ClearDATA Customers. `http://www.cleardata.com/cm/content/customers.asp`.

[3] DoD Cloud Broker. `http://www.disa.mil/Services/DoD-Cloud-Broker`.

[4] Experian runs its private cloud on VMware. `https://tinyurl.com/p8ljc57`.

[5] Ocean Cluster for Experimental Architectures in Networks (OCEAN). `http://ocean.cs.illinois.edu/`.

[6] SHADOWS IN THE CLOUD:Investigating Cyber Espionage 2.0. `https://tinyurl.com/3qwrfrc`.

[7] Virtual Private Cloud. `http://aws.amazon.com/vpc/`.

[8] VMware NSX. `http://www.vmware.com/products/nsx`.

[9] Hackers break into Pentagon's fighter jet project. *ComputerWorld*, April 2009. `https://tinyurl.com/pwl3myy`.

[10] Panetta warns of dire threat of cyberattack on U.S. *New York Times*, October 2012. `https://tinyurl.com/9rtsrr2`.

[11] Tracking GhostNet: Investigating a Cyber Espionage Network. *ProPublica*, April 2012. `https://tinyurl.com/ygakufq`.

[12] 4-year long HIPAA breach uncovered. *Healthcare IT News*, January 2014. `https://tinyurl.com/ls66a3a`.

[13] Emulab network emulation testbed. 2014. `https://www.emulab.net/`.

[14] FBI warns retailers to expect more credit card breaches. *Chicago Tribune*, January 2014. `https://tinyurl.com/nzc7jyp`.

[15] Network simulator 2. 2014. `http://nsnam.isi.edu/nsnam/index.php/User_Information`.

[16] Open vSwitch. 2014. `http://openvswitch.org/`.

[17] Project Floodlight. 2014. `http://www.projectfloodlight.org/floodlight/`.

[18] Researchers Identify Sophisticated Chinese Cyberespionage Group. *The Washington Post*, October 2014. `https://tinyurl.com/pntdm64`.

[19] The Heartbleed Bug. apr 2014. `http://heartbleed.com/`.

[20] What is Shellshock. 2014. `https://shellshocker.net/`.

[21] White House says Sony hack is a serious national security matter. *The Washington Post*, December 2014. `https://tinyurl.com/nkf8mpg`.

[22] ClearDATA Cloud. 2015. `https://www.cleardata.com/`.

[23] R. H. Barker. Group synchronizing of binary digital sequences. *Communication Theory*, page pp. 273287., 1953.

[24] Adam M. Bates, Benjamin Mood, Joe Pletcher, Hannah Pruse, Masoud Valafar, and Kevin R. B. Butler. On detecting co-resident cloud instances using network flow watermarking techniques. *Int. J. Inf. Sec.*, pages 171–189, 2014.

[25] Theophilus Benson, Aditya Akella, and David A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC*, 2010.

[26] S. Cabuk, C.E. Adviser-Brodley, and E.H. Adviser-Spafford. Network covert channels: design, analysis, detection, and elimination. 2006.

[27] Sae-Young Chung, Jr. Forney, G.D., T.J. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 db of the shannon limit. *Communications Letters, IEEE*, 2001.

[28] T M Cover and J A Thomas. *Elements of Information Theory*. Wiley, New York, 1987.

[29] Deniz Ersoz, Mazin S. Yousif, and Chita R. Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *ICDCS*, pages –1–1, 2007.

[30] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993.

[31] Robert G. Gallager. Low-density parity-check codes, 1963.

[32] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A Scalable and Flexible Data Center Network. In *ACM SIGCOMM*, 2009.

[33] A. B. Jeng and M. D. Abrams. On network covert channel analysis. In *3rd Aerospace Computer Security Conference*, 1987.

[34] Partha Kanuparthy, Constantine Dovrolis, and Mostafa Ammar. Spectral probing, crosstalk and frequency multiplexing in internet paths. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 291–304, New York, NY, USA, 2008. ACM.

[35] Eric Keller, Soudeh Ghorbani, Matt Caesar, and Jennifer Rexford. Live Migration of an Entire Network (and Its Hosts). In *HotNets-XI*, 2012.

[36] Jingfei Kong, Onur Aciimez, Jean-Pierre Seifert, and Huiyang Zhou. Hardware-software integrated approaches to defend against software cache-based side channel attacks. In *HPCA*, pages 393–404, 2009.

[37] S. G. Krantz. *Handbook of Complex Variables*. MA: BirkhŁuser, 1995.

[38] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 1994.

[39] Y. Liu, D. Ghosal, F. Armknecht, A.R. Sadeghi, S. Schulz, and S. Katzenbeisser. Hide and seek in timerobust covert timing channels. *Computer Security–ESORICS 2009*, pages 120–135, 2009.

[40] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[41] PE Meckenney. Stochastic fair queuing. *proceedings of INFCOM*, 2:733–740, 1990.

[42] Ira S. Moskowitz and Myong H. Kang. Covert Channels - Here to Stay? In *COMPASS*, 1994.

[43] J. Nagle. On packet switches with infinite storage. RFC 970, December 1985.

[44] U. Premarathne, U. Premaratne, and K. Samarasinghe. Network traffic self similarity measurements using classifier based Hurst parameter estimation. In *ICIAfS*, 2010.

[45] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced cloud services. In *NS Simulator for Beginners'09*, pages 77–84, 2009.

[46] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *ACM CCS*, pages 199–212, 2009.

[47] Ahmad-Reza Sadeghi, Steffen Schulz, and Vijay Varadharajan. The Silence of the LANs: Efficient Leakage Resilience for IPsec VPNs. In *ESORICS*, pages 253–270, 2012.

[48] S.H. Sellke, C.C. Wang, S. Bagchi, and N. Shroff. Tcp/ip timing channels: Theory to implementation. In *IEEE INFOCOM*, pages 2204–2212, 2009.

[49] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. *SIGCOMM Comput. Commun. Rev.*, 25(4):231–242, October 1995.

[50] Zhenghong Wang and Ruby B. Lee. Covert and Side Channels Due to Processor Architecture. In *ACSAC*, 2006.

[51] JingZheng Wu, Liping Ding, Yanjun Wu, Nasro Min-Allah, Samee Ullah Khan, and Yongji Wang. C^2detector: a covert channel detection framework in cloud computing. *Security and Communication Networks*, 2014.

[52] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In *USENIX Security Symposium*, 2012.

[53] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. A covert channel construction in a virtualized environment. In *ACM Conference on Computer and Communications Security'12*, pages 1040–1042, 2012.

[54] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An Exploration of L2 Cache Covert Channels in Virtualized Environments. In *ACM CCSW*, pages 29–40, 2011.

[55] Sebastian Zander, Grenville J. Armitage, and Philip Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials*, pages 44–57, 2007.

[56] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *IEEE SSP*, pages 313–328, 2011.