PHY Covert Channels: Can you see the Idles?

Ki Suh Lee, Han Wang, and Hakim Weatherspoon, *Cornell University*

https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/lee

This paper is included in the Proceedings of the
11th USENIX Symposium on Networked Systems
Design and Implementation (NSDI '14).

April 2–4, 2014 • Seattle, WA, USA

ISBN 978-1-931971-09-6

PHY Covert Channels: Can you see the Idles?

Ki Suh Lee, Han Wang, Hakim Weatherspoon
Computer Science Department, Cornell University
kslee,hwang,hweather@cs.cornell.edu

Abstract
Network covert timing channels embed secret messages in legitimate packets by modulating interpacket delays. Unfortunately, such channels are normally implemented in higher network layers (layer 3 or above) and easily detected or prevented. However, access to the *physical layer* of a network stack allows for timing channels that are virtually invisible: Sub-microsecond modulations that are undetectable by software endhosts. Therefore, covert timing channels implemented in the physical layer can be a serious threat to the security of a system or a network. In fact, we empirically demonstrate an effective covert timing channel over nine routing hops and thousands of miles over the Internet (the National Lambda Rail). Our covert timing channel works with cross traffic, less than 10% bit error rate, which can be masked by forward error correction, and a covert rate of 81 kilobits per second. Key to our approach is access and control over every bit in the physical layer of a 10 Gigabit network stack (a bit is 100 picoseconds wide at 10 gigabit per seconds), which allows us to modulate and interpret interpacket spacings at sub-microsecond scale. We discuss when and how a timing channel in the physical layer works, how hard it is to detect such a channel, and what is required to do so.

1 Introduction

Covert channels are defined as channels that are not intended for information transfer, but can leak sensitive information [21]. In essence, covert channels provide the ability to hide the transmission of data within established network protocols [37], thus hiding their existence. Covert channels are typically classified into two categories: Storage and timing channels. In *storage channels*, a sender modulates the value of a storage location to send a message. In *timing channels*, on the other hand, a sender modulates system resources over time to send a message [10].

Network covert channels send hidden messages over *legitimate* packets by modifying packet headers (storage channels) or by modulating interpacket delays (timing channels). Because network covert channels can deliver sensitive messages across a network to a receiver multiple-hops away, they impose serious threats to the security of systems. Network *storage* channels normally exploit unused fields of protocol headers [20, 28, 34, 35], and, thus, are relatively easy to detect and prevent [14, 19, 26]. Network *timing* channels deliver messages by modulating interpacket delays (or arrival time of packets). As a result, arrivals of packets in network timing channels normally create patterns, which can be analyzed with statistical tests to detect timing channels [11, 12, 16, 32], or eliminated by network jammers [17]. To make timing channels robust against such detection and prevention, more sophisticated timing channels mimic legitimate traffic with spreading codes and a shared key [24], or use independent and identically distributed (i.i.d) random interpacket delays [25].

In this paper, we present a new method of creating a covert timing channel that is *high-bandwidth*, *robust* against cross traffic, and *undetectable* by software endhosts. The channel can effectively deliver 81 kilobits per second with less than 10% errors over nine routing hops, and thousands of miles over the National Lambda Rail (NLR). We empirically demonstrate that we can create such a timing channel by modulating interpacket gaps at sub-microsecond scale: A scale at which sent information is preserved through multiple routing hops, but statistical tests cannot differentiate the channel from legitimate traffic. Unlike approaches mentioned above, our covert timing channel, *Chupja*[1], is implemented in the physical layer of a network protocol stack. In order to hide the existence of the channel, we mainly exploit the fact that statistical tests for covert channel detection rely on collected interpacket delays, which can be highly inaccurate in a 10 Gigabit Ethernet (GbE) network, whereas access to the physical layer provides fine-grained control over interpacket delays at nanosecond scale [15, 22]. As a result, a network monitoring application needs to have the capability of fine-grained timestamping to detect our covert channel. We argue that nanosecond level of resolution is key to do so.

The contributions of this paper are as follows:

- We discuss how to design and implement a covert timing channel via access to the physical layer.
- We demonstrate that a covert timing channel implemented in the physical layer can effectively deliver secret messages over the Internet.
- We empirically illustrate that we can quantify perturbations added by a network, and the quantified

[1] *Chupja* is equivalent to *spy* in Korean

perturbation is related to bit error rate of the covert timing channel.
- We show that in order to detect *Chupja*, fine-grained timestamping at nanosecond scale is required.

2 Network Covert Channels

Network covert channels are not new. However, implementing such a channel in the physical layer has never been tried before. In this section, we briefly discuss previous approaches to create and detect network covert channels, and why access to the physical layer can create a covert channel that is hard to detect. Although our focus of this paper is covert timing channels, we discuss both covert storage channels and covert timing channels in this section.

In a network covert channel, the *sender* has secret information that she tries to send to a *receiver* over the Internet. The sender has control of some part of a network stack including a network interface (L1∼2), kernel network stack (L3∼4) and/or user application (L5 and above). Thus, the sender can modify protocol headers, checksum values, or control the timing of transmission of packets. The sender can either use packets from other applications of the system or generate its own packets. Although it is also possible that the sender can use packet payloads to directly embed or encrypt messages, we do not consider this case because it is against the purpose of a covert channel: *hiding the existence of the channel*. The *adversary* (or warden), on the other hand, wants to detect and prevent covert channels. A *passive* adversary monitors packet information to detect covert channels while an *active* adversary employs network appliances such as network jammers to reduce the possibility of covert channels.

In network *storage* channels, the sender changes the values of packets to secretly encode messages, which is examined by the receiver to decode the message. This can be easily achieved by using unused bits or fields of protocol headers. The IP Identification field, the IP Fragment Offset, the TCP Sequence Number field, and TCP timestamps are good places to embed messages [20, 28, 34, 35]. As with the easiness of embedding messages in packet headers, it is just as easy to detect and prevent such storage channels. The adversary can easily monitor specific fields of packet headers for detection, or *sanitize* those fields for prevention [14, 19, 26].

In network *timing* channels, the sender controls the timing of transmission of packets to deliver hidden messages. The simplest form of this channel is to send or not send packets in a pre-arranged interval [12, 31]. Because interpacket delays are perturbed with noise from a network, synchronizing the sender and receiver is a major challenge in these on/off timing channels. However, synchronization can be avoided when each interpacket delay conveys information, i.e. a long delay is zero, and a short delay is one [11]. JitterBugs encodes bits in a similar fashion, and uses the remainder of modulo operation of interpacket delays for encoding and decoding [36]. These timing channels naturally create patterns of interpacket delays which can be analyzed with statistical tests for detection. For example, *regularity* tests [11, 12], *shape* tests [32], or *entropy* tests [16] are widely used for covert timing channel detection. On the other hand, to avoid detection from such statistical tests, timing channels can mimic patterns of legitimate traffic, or use random interpacket delays. Liu et al., demonstrated that with spreading codes and a shared key, a timing channel can be robust against known statistical tests [24]. They further developed a method to use independent and identically distributed (i.i.d) random interpacket delays to make the channel less detectable [25].

Access to the physical layer (PHY) allows the sender to create new types of both storage and timing channels. The sender of a covert storage channel can embed secret messages into special characters that only reside in the physical layer, which are discarded before the delivery of packets to higher layers of a network stack. As a result, by embedding messages into those special characters, higher layers of a network stack will have no way to detect the existence of the storage channel. In fact, idle characters (`/I/`s), which are used to fill gaps between any two packets in the physical layer, would make for great covert channels if they could be manipulated. The IEEE 802.3 standard requires that at least twelve `/I/` characters must be inserted after every packet [3]. Therefore, it is possible to create a high-bandwidth storage channel that cannot be detected without access to the PHY. Unfortunately, this covert storage channel can only work for one hop, i.e. between two directly connected devices, because network devices discard the contents of idle characters when processing packets. However, if a *supply chain attack* is taken into account where switches and routers between the sender and the receiver are compromised and capable of forwarding hidden messages, the PHY storage channel can be very effective. We have implemented a PHY covert storage channel and verified that it is effective, but only for one hop. To prevent the PHY storage channel, all special characters must be sanitized (or zeroed) at every hop.

Our focus, however, is not a PHY covert storage channel. Instead, we demonstrate that sophisticated covert timing channels can be created via access to the PHY. The idea is to control (count) the number of `/I/`s to encode (decode) messages. i.e. to modulate interpacket gaps in nanosecond resolution.

Any network component that has access to the PHY, and thus `/I/`s, can potentially detect PHY covert channels. Indeed, routers and switches have the capability to access the physical layer (i.e. `/I/`s). Unfortunately, they are not normally programmable and do not provide an interface for access to `/I/`s. Instead, anyone that wants to detect PHY covert timing channels would need to apply statistical tests on interpacket delays (discussed earlier in this section). Of course, interpacket delays needs to be captured precisely before doing so.

In other words, PHY covert timing channels could be potentially detected if the time of packet reception could be accurately timestamped with fine-grained resolution (i.e. nanosecond precision; enough precision to measure interpacket gaps of 10 GbE networks). However, commodity network devices often lack precise timestamping capabilities. Further, although high-end network monitoring appliances [2, 9] or network monitoring interface cards [1, 6] are available with precise timestamping capabilities, deploying such high-end appliances in a large network is not common due to the volume of traffic they would need to process (they have limited memory/storage) and cost.

Given that programmatic access to the PHY and accurate timestamping capabilities in high-speed networks are not readily available, we assume a passive adversary who uses commodity servers with commodity network interface cards (NIC) for network monitoring. An example adversary is a network administrator monitoring a network using `pcap` applications. This implies that the adversary does not have access to the PHY of a network stack.

Can a passive adversary built from a commodity server and NIC detect a PHY timing channel? We will demonstrate that it cannot (Section 4.4). In particular, we will show how to exploit inaccurate timestamping of network monitoring applications in order to hide the existence of such a channel. It has been shown that access to the PHY allows very precise timestamping at sub-nanosecond resolution, whereas endhost timestamping is too inaccurate to capture the nature of 10 GbE [15, 22]. In particular, an endhost relies on its own system clock to timestamp packets which normally provides microsecond resolution, or hardware timestamping from network interface cards which provides sub-microsecond resolution. Unfortunately, packets can arrive much faster than the endhost can timestamp them. Therefore, inaccurate timestamping at an endhost can lead to an opportunity to create a timing channel. In this paper, we will discuss how to create a timing channel by modulating interpacket gaps precisely in a way that network monitoring applications cannot detect any regularities from them.

3 *Chupja*: PHY timing channel

In this section, we discuss the design and implementation of our physical layer (PHY) covert timing channel, *Chupja*. Since *Chupja* is implemented via access to the physical layer, we briefly discuss the IEEE 802.3 10 Gigabit Ethernet standard first, and present the design goal of *Chupja*, how to encode and decode secret messages, how *Chupja* is implemented, and other considerations.

3.1 10 GbE Physical Layer

According to the IEEE 802.3 standard [3], when Ethernet frames are passed to the PHY, they are reformatted before being sent across the physical medium. On the transmit path, the PHY encodes every 64 bits of an Ethernet frame into a 66-bit *block*, which consists of a two bit *synchronization header* (syncheader) and a 64-bit *payload*. As a result, a 10 GbE link actually operates at 10.3125 Gbaud ($10G \times \frac{66}{64}$). The PHY also scrambles each block before passing it down the network stack to be transmitted. The entire 66-bit block is transmitted as a continuous stream of *symbols* which a 10 GbE network transmits over a physical medium. As 10 GbE always sends 10.3125 gigabits per second (Gbps), each bit in the PHY is about 97 picoseconds wide. On the receive path, the PHY descrambles each 66-bit block before decoding it.

Idle characters (`/I/`) are special characters that fill any gaps between any two packets in the PHY. When there is no Ethernet frame to transmit, the PHY continuously inserts `/I/` characters until the next frame is available. The standard requires at least twelve `/I/`s after every packet. An `/I/` character consists of seven or eight bits, and thus it takes about 700$\sim$800 picoseconds to transmit one `/I/` character. `/I/`s are typically inaccessible from higher layers (L2 or above), because they are discarded by hardware.

3.2 Design Goal

The design goal of our timing channel, *Chupja*, is to achieve *high-bandwidth*, *robustness* and *undetectability*. By high-bandwidth, we mean a covert rate of many tens or hundreds of thousands of bits per second. Robustness is how to deliver messages with minimum errors, and undetectability is how to hide the existence of it. In particular, we set as our goal for robustness to a bit error rate (BER) of less than 10%, an error rate that is small enough to be compensated with forward error correction such as Hamming code, or spreading code [24]. We define BER as the ratio of the number of bits incorrectly delivered from the number of bits transmitted.

In order to achieve these goals, we precisely modulate the number of `/I/`s between packets in the physical layer. If the modulation of `/I/`s is large, the channel can effectively send messages in spite of noise or perturbations from a network (robustness). At the same time, if the modulation of `/I/`s is small, an adversary will not be able to detect regularities (undetectability). Further, *Chupja* embeds one timing channel bit per interpacket gap to achieve high-bandwidth. Thus, higher overt packet rates will achieve higher covert timing channel rates. We focus on finding an optimal modulation of interpacket gaps to achieve high-bandwidth, robustness, and undetectability (Section 4).

3.3 Model

In our model, the sender of *Chupja* has control over a network interface card or a compromised switch[2] with access to and control of the physical layer. In other words, the sender can easily control the number of `/I/` characters of outgoing packets. The receiver is in a network multiple hops away, and taps/sniffs on its network with access to the physical layer. Then, the sender modulates the number of `/I/`s between packets destined to the receiver's network to embed secret messages.

Our model includes an adversary who runs a network monitoring application and is in the same network with the sender and receiver. As discussed in Section 2, we assume that the adversary is built from a commodity server and commodity NIC. As a result, the adversary does not have direct access to the physical layer. Since a commodity NIC discards `/I/`s before delivering packets to the host, the adversary cannot monitor the number of `/I/`s to detect the possibility of covert timing channels. Instead, it runs statistical tests with captured interpacket delays.

3.4 Terminology

We define *interpacket delay* (IPD) as the time difference between the first bits of two successive packets, and *interpacket gap* (IPG) as the time difference between the last bit of the first packet and the first bit of the next packet. Thus, an interpacket delay between two packets is equal to the sum of transmission time of the first packet and the interpacket gap between the two (i.e. IPD = IPG + packet size). A *homogeneous* packet stream consists of packets that have the same destination, the same size and the same IPGs (IPDs) between them. Furthermore, the variance of IPGs and IPDs of a homogeneous packet stream is always zero.

3.5 Encoding and Decoding

Chupja embeds covert bits into interpacket gaps of a homogeneous packet stream of an *overt* channel. In order to create *Chupja*, the sender and receiver must share two parameters: G and W. G is the number of `/I/`s in the IPG that is used to encode and decode hidden messages, and W (*Wait time*) helps the sender and receiver synchronize (Note that interpacket delay $D = G$ + packet size). Figure 1 illustrates our design. Recall that IPGs of a homogeneous packet stream are all the same (=G, Figure 1a). For example, the IPG of a homogeneous stream with 1518 byte packets at 1 Gbps is always 13738 `/I/`s; the variance is zero. To embed a secret bit sequence $\{b_i, b_{i+1}, \cdots\}$, the sender encodes 'one' ('zero') by increasing (decreasing) the IPG (G) by ε `/I/`s (Figure 1b):

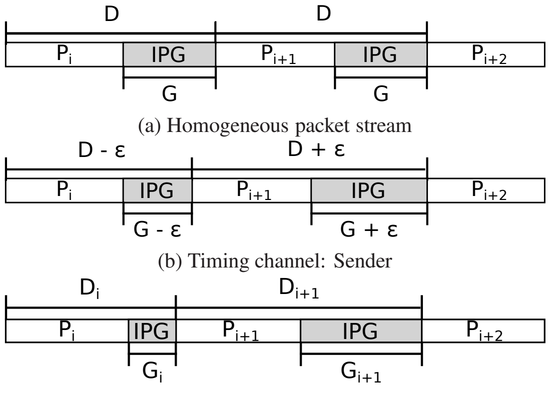


Figure 1: *Chupja* encoding and decoding.

$$G_i = G - \varepsilon \text{ if } b_i = 0$$
$$G_i = G + \varepsilon \text{ if } b_i = 1$$

where G_i is the ith interpacket gap between packet i and $i+1$. When G_i is less than the minimum interpacket gap (or 12 `/I/` characters), it is set to twelve to meet the standard requirement.

Interpacket gaps (and delays) will be perturbed as packets go through a number of switches. However, as we will see in Section 4.3, many switches do not significantly change interpacket gaps. Thus, we can expect that if ε is large enough, encoded messages will be preserved along the path. At the same time, ε must be small enough to avoid detection by an adversary. We will evaluate how big ε must be with and without cross traffic and over multiple hops of switches over thousands of miles in a network path (Section 4).

Upon receiving packet pairs, the receiver decodes bit information as follows:

$$b'_i = 1 \text{ if } G_i \geq G$$
$$b'_i = 0 \text{ if } G_i < G$$

b'_i might not be equal to b_i because of network noise. We use BER to evaluate the performance of *Chupja* (Section 4.2).

Because each IPG corresponds to a signal, there is no need for synchronization between the sender and the receiver [11]. However, the sender occasionally needs to pause until the next covert packet is available. W is used when there is a pause between signals. The receiver considers an IPG that is larger than W as a pause, and uses the next IPG to decode the next signal.

3.6 Implementation

We used SoNIC to implement and evaluate *Chupja*. SoNIC [22] allows users to access and control every bit

[2]We use the term *switch* to denote both bridge and router.

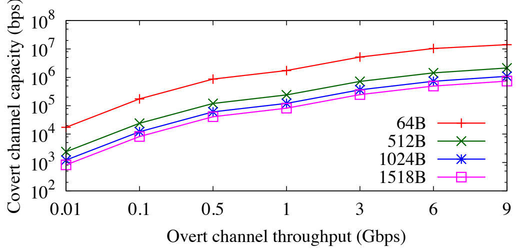


Figure 2: Maximum capacity of PHY timing channel

of 10 GbE physical layer, and thus we can easily control (count) the number of /I/s between packets. We extended SoNIC's packet generation capability to create a timing channel. Given the number of /I/ characters in the original IPG (G), the number of /I/s to modulate (ε) and a secret message, IPGs are changed accordingly to embed the message. On the receiver side, it decodes the message by counting the number of /I/s between packets in realtime. The total number of lines added to the SoNIC implementation was less than 50 lines of code.

The capacity of this PHY timing channel is equal to the number of packets being transmitted from the sender when there is no pause. Given a packet size, the maximum capacity of the channel is illustrated in Figure 2. For example, if an overt channel sends at 1 Gbps with 1518 byte packets, the maximum capacity of the covert channel is 81,913 bits per second (bps). We will demonstrate in Section 4.2 that *Chupja* can deliver 81 kilobits per second (kbps) with less than 10% BER over nine routing hops and thousands of miles over National Lambda Rail (NLR).

3.7 Discussion

Chupja uses homogeneous packet streams to encode messages, which creates a regular pattern of IPGs. Fortunately, as we will discuss in the following section, the adversary will be unable to accurately timestamp incoming packets when the data rate is high (Section 4.4). This means that it does not matter what patterns of IPGs are used for encoding at above a certain data rate. Therefore, we chose the simplest form of encoding for *Chupja*. The fact that the PHY timing channel works over multiple hops means that a non-homogeneous timing channel will work as well. For instance, consider the output after one routing hop as the sender, then the PHY timing channel works with a non-homogeneous packet stream. If, on the other hand, the sender wants to use other patterns for encoding and decoding, other approaches can easily be applied [12, 24, 25, 36]. For example, if the sender wants to create a pattern that looks more random, we can also use a shared secret key and generate random IPGs for encoding and decoding [25]. However, the focus of this paper is to demonstrate that even this simplest form of timing channel can be a serious threat to a system and not easily be detected.

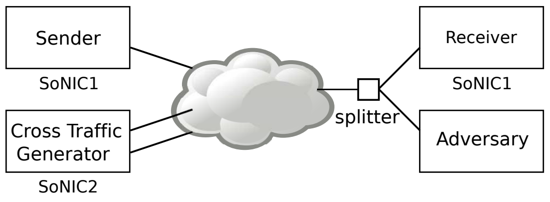


Figure 3: Network topology for evaluation. All lines are 10 gigabit fiber optic cables

Finally, note that a *Chupja* sender and receiver do not need to be endpoints of a network path, but could actually be within the network as middleboxes. Such a covert timing channel middlebox would require constant overt traffic in order to manipulate interpacket gaps.

4 Evaluation

In this section, we evaluate *Chupja* over real networks. We attempt to answer following questions.

- How *robust* is *Chupja* (Section 4.2)? How effectively can it send secret messages over the Internet?
- Why is *Chupja robust* (Section 4.3)? What properties of a network does it exploit?
- How *undetectable* is *Chupja* (Section 4.4)? Why is it hard to detect it and what is required to do so?

In Section 4.2, we first demonstrate that *Chupja* works effectively over the Internet, and achieves a Bit Error Rate (BER) less than 10% which is the design goal of *Chupja* (Section 3). In particular, we evaluated *Chupja* over two networks: A small network that consists of multiple commercial switches, and the National Lambda Rail (NLR). We discuss what is the optimal interpacket gap (IPG) modulation, ε, that makes *Chupja* work. Then, in order to understand why *Chupja* works, we provide a microscopic view of how network devices behave in Section 4.3. We conducted a sensitivity analysis over commercial switches. We mainly show how network devices preserve small interpacket delays along the path even with and without the existence of cross traffic. Lastly, we discuss how to detect a sophisticated timing channel such as *Chupja* in Section 4.4.

4.1 Evaluation Setup

For experiments in this section, we deployed two SoNIC servers [22] each equipped with two 10 GbE ports to connect fiber optic cables. We used one SoNIC server (SoNIC1) to generate packets of the sender destined to a server (the adversary) via a network. We placed a fiber optic splitter at the adversary which mirrored packets to SoNIC1 for capture (i.e. SoNIC1 was both the timing channel sender and receiver). SoNIC2 was used to generate cross traffic flows when necessary (Figure 3). Throughout this section, we placed none or multiple commercial switches between the sender and the adversary (the cloud within Figure 3).

Table 1 summarizes the commercial switches that we

	Type	40G	10G	1G	Full bandwidth	Forwarding
SW1	Core	0	8	0	160 Gbps	SF
SW2	ToR	4	48	0	1280 Gbps	CT
SW3	ToR	0	2	48	136 Gbps	SF
SW4	ToR	0	2	24	105.6 Gbps	SF

Table 1: Summary of evaluated network switches. “SF” is store-and-forward and “CT” is cut-through.

used. SW1 is a core / aggregate router with multiple 10 GbE ports, and we installed two modules with four 10 GbE ports. SW2 is a high-bandwidth 10 GbE top-of-rack (ToR) switch which is able to support forty eight 10 GbE ports at line speed. Moreover, it is a cut-through switch whose latency of forwarding a packet is only a few microseconds. SW3 and SW4 are 1 GbE ToR switches with two 10 GbE uplinks. Other than SW2, all switches are store-and-forward switches.

We used a Dell 710 server for the adversary. The server has two X5670 2.93GHz processors each with six CPU cores, and 12 GB RAM. The architecture of the processor is Westmere [4] that is well-known for its capability of processing packets in a multi-threading environment [13, 18, 27]. We used one 10 GbE Dual-port NICs for receiving packets. No optimization or performance tuning such as `irq` balancing, or interrupt coalescing, was performed except New API (NAPI) [7] which is enabled by default.

Packet size [Bytes]	Data Rate [Gbps]	Packet Rate [pps]	IPD [ns]	IPG [`/I/`]
1518	9	737028	1356.8	170
1518	6	491352	2035.2	1018
1518	3	245676	4070.4	3562
1518	1	81913	12211.2	13738
64	6	10416666	96.0	48
64	3	5208333	192.0	168
64	1	1736111	576.0	648

Table 2: IPD and IPG of homogeneous packet streams.

ε (`/I/`s)	16	32	64	128	256	512	1024	2048	4096
ns	12.8	25.6	51.2	102.4	204.8	409.6	819.2	1638.4	3276.8

Table 3: Evaluated ε values in the number of `/I/`s and their corresponding time values in nanosecond.

For most of the evaluation, we used 1518 byte and 64 byte packets for simplicity. We define packet size as the number of bytes from the first byte of the Ethernet header to the last byte of the Ethernet frame check sequence (FCS) field (i.e. we exclude seven preamble bytes and start frame delimiter byte from packet size). Then, the largest packet allowed by Ethernet is 1518 bytes (14 byte header, 1500 payload, and 4 byte FCS), and the smallest is 64 bytes. In this section, the data rate refers to the data rate of the overt channel that *Chupja* is embedded. Interpacket delays (IPDs) and interpacket gaps (IPGs) of homogeneous packet streams at different data rates and with different packet sizes are summarized

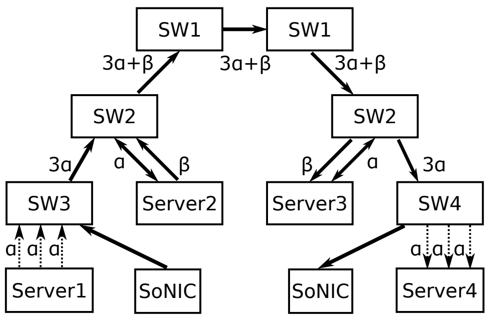


Figure 4: A small network. Thick solid lines are 10G connections while dotted lines are 1G connections.

in Table 2. Table 3 shows the number of `/I/`s (ε) we modulate to create *Chupja* and their corresponding time values in nanosecond. We set ε starting from 16 `/I/`s (= 12.8 ns), doubling the number of `/I/`s up to 4096 `/I/`s (= 3276.8 ns). We use a tuple (s, r) to denote a packet stream with s byte packets running at r Gbps. For example, a homogeneous stream with (1518B, 1Gbps) is a packet stream with 1518 byte packets at 1 Gbps.

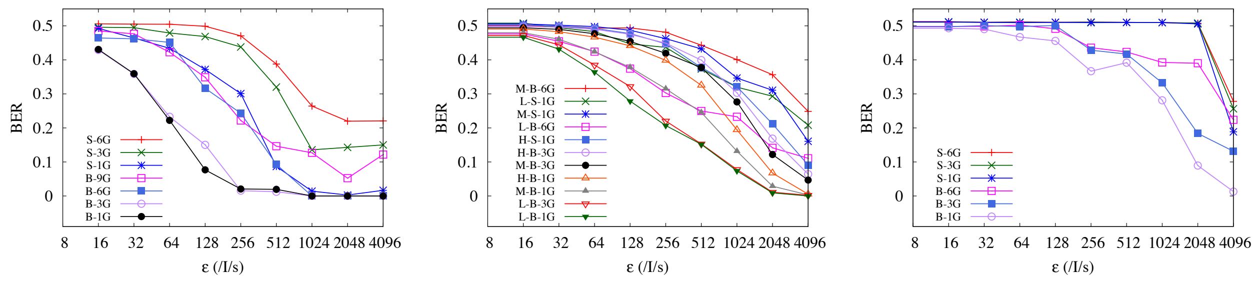
4.2 Efficiency of *Chupja*

The goal of a covert timing channel is to send secret messages to the receiver with minimum errors (robustness). As a result, Bit Error Rate (BER) and the achieved covert bandwidth are the most important metrics to evaluate a timing channel. Our goal is to achieve BER less than 10% over a network with high bandwidth. In this section, we evaluate *Chupja* over two networks, a small network and the Internet (NLR), focusing on the relation between BER and the number of `/I/`s being modulated (ε).

4.2.1 A small network

We created our own network by connecting six switches, and four servers (See Figure 4). The topology resembles a typical network where core routers (SW1) are in the middle and 1 GbE ToR switches (SW3 and SW4) are leaf nodes. Then, SoNIC1 (the sender) generates packets to SW3 via one 10 GbE uplink, which will forward packets to the receiver which is connected to SW4 via one 10 GbE uplink. Therefore, it is a seven-hop timing channel with 0.154 ms round trip time delay on average, and we measured BER at the receiver.

Before considering cross traffic, we first measure BER with no cross traffic. Figure 5a illustrates the result. The x-axis is ε modulated in the number of idle (`/I/`) characters (see Table 3 to relate `/I/`s to time), and the y-axis is BER. Figure 5a clearly illustrates that the larger ε, the smaller BER. In particular, modulating 128 `/I/`s (=102.4 ns) is enough to achieve BER=7.7% with (1518B, 1Gbps) (filled in round dots). All the other cases also achieve the goal BER except (64B, 6Gbps) and (64B, 3Gbps). Recall that Table 2 gives the capacity of the covert channel. The takeaway is that when there is



(a) Without cross traffic over a small network (b) With cross traffic over a small network (c) Over National Lambda Rail

Figure 5: BER of *Chupja* over a small network and NLR. X-Y-Z means that workload of cross traffic is X (H-heavy, M-medium, or L-light), and the size of packet and data rate of overt channel is Y (B-big=1518B or S-small=64B) and Z (1, 3, or 6G).

no cross traffic, modulating small number of `/I/s` (128 `/I/s`, 102.4 ns) is sufficient to create a timing channel. In addition, it is more efficient with large packets.

Now, we evaluate *Chupja* with cross traffic. In order to generate cross traffic, we used four servers (Server1 to 4). Each server has four 1 GbE and two 10 GbE ports. Server1 (Server4) is connected to SW3 (SW4) via three 1 GbE links, and Server2 (Server3) is connected to SW3 via two 10 GbE links. These servers generate traffic across the network with Linux `pktgen` [30]. The bandwidth of cross traffic over each link between switches is illustrated in Figure 4: 1 GbE links were utilized with flows at α Gbps and 10 GbE links at β Gbps. We created three workloads where $(\alpha, \beta) = (0.333, 0.333)$, $(0.9, 0.9)$, and $(0.9, 3.7)$, and we call them Light, Medium and Heavy workloads. Packets of cross traffic were always maximum transmission unit (MTU) sized. Then SoNIC1 generated timing channel packets at 1, 3, and 6 Gbps with 1518 and 64 byte packets. Figure 5b illustrates the result. At a glance, because of the existence of cross traffic, ε must be larger to transmit bits correctly compared to the case without cross traffic. There are a few takeaways. First, regardless of the size of workloads, timing channels with (1518B, 1Gbps) and (1518B, 3Gbps) work quite well, achieving the goal BER of less than 10% with $\varepsilon \geq 1024$. On the other hand, channels at a data rate higher than 6 Gbps are not efficient. In particular, $\varepsilon = 4096$ is not sufficient to achieve the goal BER with (1518B, 6Gbps). Second, creating timing channels with small packets is more difficult. Generally, BER is quite high even with $\varepsilon = 4096$ except H-S-1G case (BER=9%).

4.2.2 National Lambda Rail

In this section, we evaluate *Chupja* in the wild over a real network, National Lambda Rail (NLR). NLR is a wide-area network designed for research and has significant cross traffic [29]. We set up a path from Cornell university to NLR over nine routing hops and 2500 miles one-way (Figure 6). All the routers in NLR are Cisco 6500 routers. We used a SoNIC server to generate and capture *Chupja* packets at each end. The average round trip time of the path was 67.6 ms, and there was always cross traffic. In particular, many links on our path were utilized with 1∼2 Gbps cross traffic during the experiment. Cross traffic was not under our control, however we received regular measurements of traffic on external interfaces of all routers.

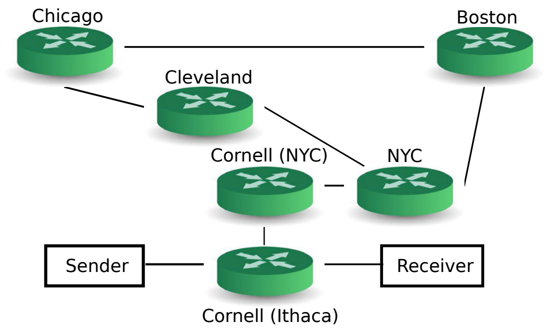


Figure 6: Our path on the National Lambda Rail

Figure 5c illustrates the results. Again, we changed the size and the data rate of overt packets. In NLR, it becomes more difficult to create a timing channel. In particular, only (1518B, 1Gbps) achieved BER less than 10% when ε is larger than 2048 (8.9%). All the other cases have higher BERs than our desired goal, although BERs are less than 30% when ε is 4096. Creating a channel with 64 byte packet is no longer possible in NLR. This was because more than 98% of IPGs were minimum interpacket gaps, i.e. most of bit information was discarded because of packet train effects [15].

We demonstrated in this section (Figure 5c) that we can create an effective covert timing channel with (1518B, 1Gbps), and with large enough ε over the NLR. The capacity of this channel can be as high as 81 kbps (Table 2). In general, large packets at slower data rate is desirable to create a timing channel. In the following section, we will investigate why this is possible by closely analyzing the behavior of network devices with respect to IPG modulations, ε.

4.3 Sensitivity Analysis

Network devices change interpacket gaps while forwarding packets; switches add randomness to interpacket gaps. In this section, we discuss how *Chupja* can deliver secret messages via a PHY timing channel in spite of the randomness added from a network. In particular, we discuss the following observations.

- A single switch does not add significant perturbations to IPDs when there is no cross traffic.
- A single switch treats IPDs of a timing channel's encoded 'zero' bit and those of an encoded 'one' bit as uncorrelated distributions; ultimately, allowing a PHY timing channel receiver to distinguish an encoded 'zero' from an encoded 'one'.
- The first and second observations above hold for multiple switches and cross traffic.

In other words, we demonstrate that timing channels can encode bits by modulating IPGs by a small number of `/I/` characters (hundreds of nanoseconds) and these small modulations can be effectively delivered to a receiver over multiple routing hops with cross traffic.

In order to understand and appreciate these observations, we must first define a few terms. We denote the interpacket delay between packet i and $i+1$ with the random variable D_i. We use superscript on the variables to denote the number of routers. For example, D_i^1 is the interpacket delay between packet i and $i+1$ after processed by one router, and D_i^0 is the interpacket delay before packet i and $i+1$ are processed by any routers. Given a distribution of D, and the average interpacket delay μ, we define I_{90} as the smallest ε that satisfies $P(\mu-\varepsilon \le D \le \mu+\varepsilon) \ge 0.90$. In other words, I_{90} is the interval from the average interpacket delay, μ, which contains 90% of D (i.e. at least 90% of distribution D is within $\mu \pm I_{90}$). For example, I_{90} of a homogeneous stream (a delta function, which has no variance) that leaves the sender and enters the first router is zero; i.e. D^0 has $I_{90} = 0$ and $P(\mu-0 \le D \le \mu+0) = 1$ since there is no variance in IPD of a homogeneous stream. We will use I_{90} in this section to quantify perturbations added by a network device or a network itself. Recall that the goal of *Chupja* is to achieve a BER less than 10%, and, as a result, we are interested in the range where 90% of D observed by a timing channel receiver is contained.

First, *switches do not add significant perturbations to IPDs when there is no cross traffic.* In particular, when a homogeneous packet stream is processed by a switch, I_{90} is always less than a few hundreds of nanoseconds, i.e. 90% of the received IPDs are within a few hundreds of nanoseconds from the IPD originally sent. Figure 7 displays the received IPD distribution measured after packets were processed and forwarded by one switch: The x-axis is the received IPD and the y-axis is the cumulative distribution function (CDF). Further, different lines represent different switches from Table 1. The characteristics of the original sender's homogeneous packet stream was a data rate of 1 Gbps with 1518B size packets, or (1518B, 1Gbps) for short, which resulted in an average IPD of 12.2 us (i.e. $\mu = 12.2$ us). We can see in Figure 7 that when μ is 12.2 us and ε is 0.1 us, $P(12.2-0.1 < D < 12.2+0.1) \ge 0.9$ is true for all switches. In general, the range of received IPDs was always bounded by a few hundreds of nanoseconds from the original IPD, regardless of the type of switch.

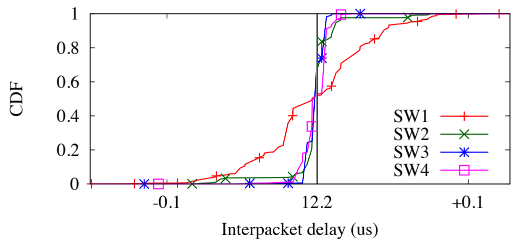


Figure 7: Comparison of IPDs after switches process a homogeneous packet stream (1518B, 1Gbps)

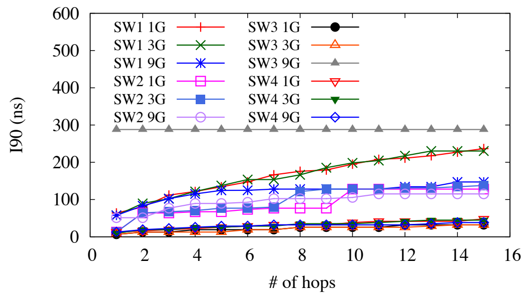
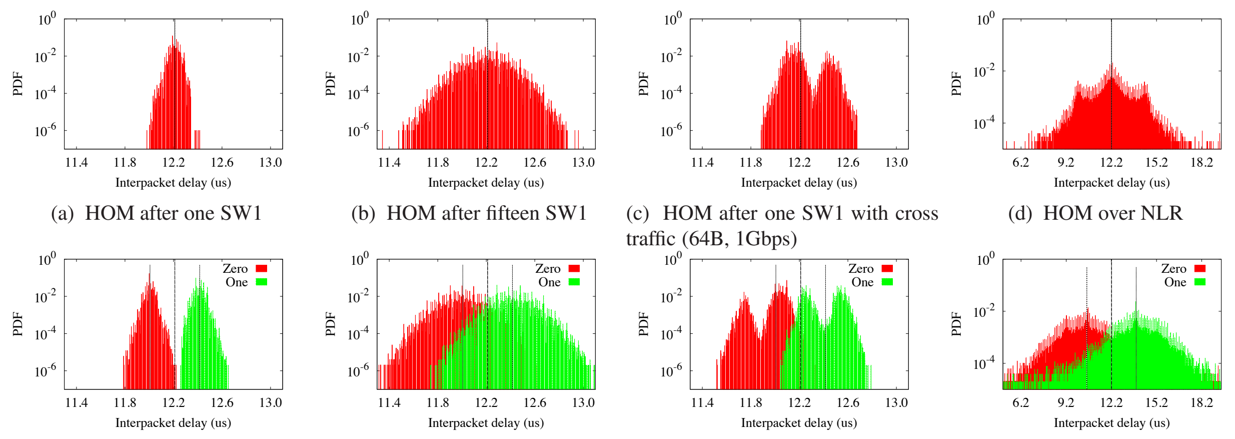


Figure 8: I_{90} comparison between various switches

Moreover, when a packet stream is processed by the same switch type, but for multiple hops, I_{90} increases linearly. Each packet suffers some perturbation, but the range of perturbation is roughly constant at every hop over different packet sizes [23] resulting in a linear increase in I_{90}. In Figure 8, we illustrate I_{90} for different switches, at different data rates (1, 3, and 9G), and as we increase the number of hops: The x-axis is the number of routing hops, y-axis is measured I_{90}, and each line is a different type of switch with a different packet stream data rate. Packet size was 1518B for all test configurations. One important takeaway from the graph is that I_{90} for the same switch shows similar patterns regardless of data rates, except SW3 9 Gbps. In particular, the degree of perturbation added by a switch is not related to the data rate (or the average interpacket delay). Instead, IPD perturbation is related to the number of hops, and the size of packet. Further, a second takeaway is that I_{90}



(a) HOM after one SW1 (b) HOM after fifteen SW1 (c) HOM after one SW1 with cross traffic (64B, 1Gbps) (d) HOM over NLR

(e) $\varepsilon = 256$ after one SW1 (f) $\varepsilon = 256$ after fifteen SW1 (g) $\varepsilon = 256$ after one SW1 with cross traffic (64B, 1Gbps) (h) $\varepsilon = 2048$ over NLR

Figure 9: Comparison of homogeneous streams and covert channel streams of (1518B, 1Gbps)

values after one hop are all less than 100 ns, except SW3 9 Gbps, and still less than 300 ns after fifteen hops. 300 ns is slightly greater than 256 /I/s (Table 3). Unfortunately, we do not have a definitive explanation on why I_{90} of SW3 9 Gbps is larger than any other case, but it is likely related to how SW3 handles high data rates.

Second, *switches treat IPDs of an encoded 'zero' bit and those of an encoded 'one' bit as uncorrelated distributions.* After encoding bits in a timing channel, there will be only two distinctive IPD values leaving the sender: $\mu + \varepsilon$ for 'one', and $\mu - \varepsilon$ for 'zero'. Let a random variable $D+$ be the IPDs of signal 'one', and $D-$ be those of signal 'zero'. We observed that the encoded distributions after one routing hop, D^1+ and D^1-, looked similar to the unencoded distribution after one routing hop, D^1. The similarity is likely due to the fact that at the sender the encoded distributions, D^0+ and D^0-, are each homogeneous packet streams (i.e. D^0, D^0+, and D^0- are all delta functions, which have no variance). For instance, using switch SW1 from Table 1, Figure 9a illustrates D^1 (the unencoded distribution of IPDs after one routing hop) and Figure 9e illustrates D^1+ and D^1- (the encoded distribution after one routing hop). The data rate and packet size was 1Gbps and 1518B, respectively, with $\varepsilon = 256$ /I/s for the encoded packet stream. We encoded the same number of 'zeros' and 'ones' randomly into the packet stream. Note the similarity in distributions between D^1 in Figure 9a and D^1+ and D^1- in Figure 9e. We observed a similarity in distributions among D^1, D^1+, and D^1- throughout different data rates and switches. We can conjecture that $D+$ and $D-$ are uncorrelated because the computed correlation coefficient between $D+$ and $D-$ is always very close to zero.

Because the distributions of D^1+ and D^1- are uncorrelated, we can effectively deliver bit information with appropriate ε values for one hop. If ε is greater than

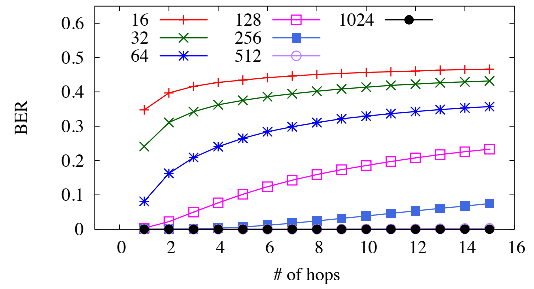


Figure 10: BER over multiple hops of SW1 with various ε values with (1518B, 1Gbps)

I_{90} of D^1, then 90% of IPDs of D^1+ and D^1- will not overlap. For example, when I_{90} is 64 ns, and ε is 256 /I/s (=204.8 ns), two distributions of D^1+ and D^1- are clearly separated from the original IPD (Figure 9e). On the other hand, if ε is less than I_{90} of D^1, then many IPDs will overlap, and thus the BER increases. For instance, Table 4 summarizes how BER of the timing channel varies with different ε values. From Table 4, we can see that when ε is greater than 64 /I/s, BER is always less than 10%. The key takeaway is that BER is *not* related with the data rate of the overt channel, rather it is related to I_{90}.

ε (/I/s)		16	32	64	128	256	512	1024
BER	1G	0.35	0.24	0.08	0.003	10^{-6}	0	0
	3G	0.37	0.25	0.10	0.005	10^{-5}	0	0
	6G	0.35	0.24	0.08	0.005	0.8×10^{-6}	0	0
	9G	0.34	0.24	0.07	0.005	0.0005	0.0004	0.0005

Table 4: ε and associated BER with (1518B, 1Gbps)

Third, *switches treat IPDs of an encoded 'zero' bit and those of an encoded 'one' bit as uncorrelated distributions over multiple switches and with cross traffic.* In particular, distributions D^n+ and D^n- are uncorrelated regardless of the number of hops and the existence of

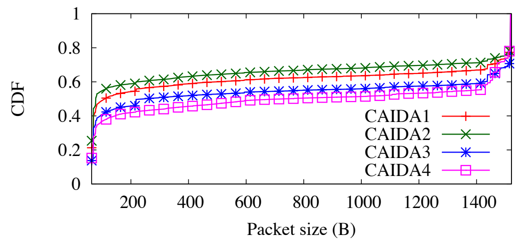


Figure 11: Packet size distributions of CAIDA traces

cross traffic. However, I_{90} becomes larger as packets go through multiple routers with cross traffic. Figures 9b and 9f show the distributions of D^{15}, $D^{15}+$, and $D^{15}-$ without cross traffic (Note that the y-axis is log-scale). The data rate and packet size was 1 Gbps and 1518B, respectively, with $\varepsilon = 256$ `/I/`s for the encoded packet stream. We conjecture that the distributions are still uncorrelated without cross traffic and after multiple hops of routers: The computed correlation coefficient was close to zero. Further, the same observation is true with the other switches from Table 1. Figure 10 shows BER over multiple hops of SW1. When ε is greater than 256 `/I/`s (=204.8 ns), BER is always less than 10% after fifteen hops. Recall that I_{90} of D after 15 hops of SW1 is 236 ns (Figure 8).

Figures 9c and 9g show the distributions after one routing hop when there was cross traffic: Distributions D^1, D^1+, and D^1- using overt data rate and packet size 1 Gbps and 1518B, respectively. The cross traffic was (64B, 1Gbps). We can see in the figures that there is still a similarity in D^1+ and D^1- even with cross traffic. However, I_{90} becomes larger due to cross traffic when compared to without cross traffic. Table 6 summarizes how I_{90} changes with cross traffic. We used five different patterns of cross traffic for this evaluation: 10-clustered (10C), 100-clustered (100C), one homogeneous stream (HOM), two homogeneous streams (HOM2), and random IPD stream (RND). A N-clustered packet stream consists of multiple clusters of N packets with the minimum interpacket gap (96 bits, which is 12 `/I/` characters) allowed by the standard [3] and a large gap between clusters. Note that a larger N means the cross traffic is *bursty*. For the RND stream, we used a geometric distribution for IPDs to create bursty traffic. In addition, in order to understand how the distribution of packet sizes affect I_{90}, we used four CAIDA traces [8] at different data rates to generate cross traffic (Table 5). With packet size and timestamp information from the traces, we reconstructed a packet stream for cross traffic with SoNIC. In the CAIDA traces, the distribution of packet sizes is normally a bimodal distribution with a peak at the lowest packet size and a peak at the highest packet size (Figure 11).

	Data Rate [Gbps]	Packet Rate [pps]	I_{90} [ns]	BER $\varepsilon = 512$	BER $\varepsilon = 1024$
CAIDA1	2.11	418k	736.0	0.10	0.041
CAIDA2	3.29	724k	848.1	0.148	0.055
CAIDA3	4.27	723k	912.1	0.184	0.071
CAIDA4	5.12	798k	934.4	0.21	0.08

Table 5: Characteristics of CAIDA traces, and measured I_{90} and BER

We observe that I_{90} increases with cross traffic (Table 6). In particular, bursty cross traffic at higher data rates can significantly impact I_{90}, although they are still less than one microsecond except 100C case. The same observation is also true using the CAIDA traces with different data rates (Table 5). As a result, in order to send encoded timing channel bits effectively, ε must increase as well. Figure 9d and 9h show the distributions of IPDs over the NLR. It demonstrates that with external traffic and over multiple routing hops, sufficiently large ε can create a timing channel.

Data Rate [Gbps]	Packet Size [Byte]	I_{90} 10C	100C	HOM	HOM2	RND
0.5	64	79.9	76.8	166.45	185.6	76.8
	512	79.9	79.9	83.2	121.6	86.3
	1024	76.8	76.8	80.1	115.2	76.8
	1518	111.9	76.8	128.0	604.7	83.2
1	64	111.9	108.8	236.8	211.2	99.3
	512	115.2	934.4	140.8	172.8	188.9
	1024	111.9	713.5	124.9	207.9	329.5
	1518	688.1	1321.5	64.0	67.1	963.3

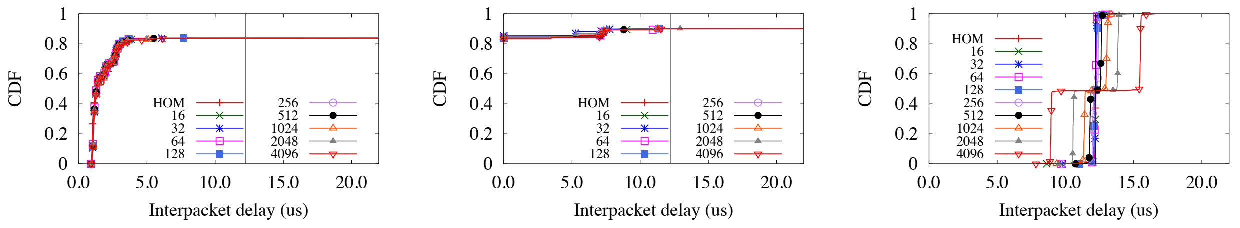
Table 6: I_{90} values in nanosecond with cross traffic.

Summarizing our sensitivity analysis results, I_{90} is determined by the characteristics of switches, cross traffic, and the number of routing hops. Further, I_{90} can be used to create a PHY timing channel like *Chupja*. In particular, we can refine the relation between I_{90} and ε^* (the minimum ε measured to achieve a BER of less than 10%). Let I_{90}^+ be the minimum ε that satisfies $P(D > \mu - \varepsilon) \geq 0.90$ and let I_{90}^- be the minimum ε that satisfies $P(D < \mu + \varepsilon) \geq 0.90$ given the average interpacket delay μ. Table 7 summarizes this relationship between $\varepsilon*$, I_{90} and $\max(I_{90}^+, I_{90}^-)$ over the NLR (Figure 6) and our small network (Figure 4).

Network	Workload	I_{90}	$\max(I_{90}^+, I_{90}^-)$	ε^* (ns)
Small network	Light	1065.7	755.2	819.2
Small network	Medium	1241.6	1046.3	1638.4
Small network	Heavy	1824.0	1443.1	1638.4
NLR		2240.0	1843.2	1638.4

Table 7: Relation between ε, I_{90}, and $\max(I_{90}^+, I_{90}^-)$ over different networks with (1518B, 1Gbps). Values are in nanosecond.

In our small network, BER is always less than 10% when ε is greater than $\max(I_{90}^+, I_{90}^-)$. On the other hand, we were able to achieve our goal BER over the NLR when ε^* is slightly less than $\max(I_{90}^+, I_{90}^-)$. Because we



(a) Kernel timestamping. (b) Zero-copy timestamping. (c) Hardware timestamping.

Figure 12: Comparison of various timestampings. Each line is a covert channel stream of (1518B, 1Gbps) with a different ε value.

do not have control over cross traffic in the NLR, I_{90} varied across our experiments.

4.4 Detection

In order to detect timing channels, applying statistical tests to captured IPDs is widely used. For example, the adversary can use regularity, similarity, shape test, and entropy test of IPDs in order to detect potential timing channels [11, 12, 16, 32]. The same strategies can be applied to *Chupja*. Since our traffic is very *regular*, those algorithms could be easily applied to detect *Chupja*. However, we argue that none of these algorithms will work if the IPG modulations cannot be observed at all. In particular, endhost timestamping is too inaccurate to observe fine-grained IPG modulations whereas *Chupja* modulates IPGs in hundreds of nanoseconds to create a timing channel. In fact, the accuracy of endhost timestamping is at most microsecond resolution. Specialized NICs can provide a few hundreds of nanosecond resolution. In this section, we demonstrate that endhost or hardware timestamping is not sufficient to detect *Chupja* timing channels. We focus on measuring and comparing an endhost's ability to accurately timestamp arrival times (i.e. accurately measure IPDs) since the ability to detect a PHY timing channel is dependent upon the ability to accurately timestamp the arrival time of packets. As a result, we do not discuss statistical approaches further.

There are mainly three places where packets can be timestamped at an endhost: Kernel, userspace, and hardware (NIC). Kernel network stacks record arrival times of packets upon receiving them from a NIC. Since so many factors are involved during the delivery of a packet to kernel space, such as DMA transaction, interrupt routines, and scheduler, kernel timestamping can be inaccurate in a high-speed network. As a result, userspace timestamping will also be inaccurate because of delays added due to transactions between kernel and userspace. To reduce the overhead of a network stack between kernel and userspace and between hardware and kernel, a technique called zero-copy can be employed to improve the performance of userspace network applications. An example of a zero-copy implementation is Netmap [33]. In Netmap, packets are delivered from a NIC directly to a memory region which is shared by a userspace application. This zero-copy removes expensive memory operations and bypasses the kernel network stack. As a result, Netmap is able to inject and capture packets at line speed in a 10 GbE network with a single CPU. Therefore, detection algorithms can exploit a platform similar to Netmap to improve the performance of network monitoring applications. We call this *zero-copy timestamping*. In hardware timestamping, a NIC uses an external clock to timestamp incoming packets at a very early stage to achieve better precision. The accuracy of timestamping is determined by the frequency of an external clock. Unfortunately, hardware timestamping is not often available with commodity NICs. However, we did include in our evaluation a specialized NIC, the Sniffer 10G [5], which can provide 500 ns resolution for timestamping.

In order to compare *kernel*, *zero-copy*, and *hardware* timestamping, we connected a SoNIC server and a network monitor directly via an optical fiber cable, generated and transmitted timing channel packets to a NIC installed in the network monitor, and collected IPDs using different timestampings. The network monitor is a passive adversary built from a commodity server. Further, we installed Netmap in the network monitor. Netmap originally used the `do_gettimeofday` for timestamping packets in kernel space, which provides only microsecond resolution. We modified the Netmap driver to support nanosecond resolution instead. For this evaluation, we always generated ten thousand packets for comparison because some of the approaches discarded packets when more than ten thousand packets were delivered at high data rates.

Figure 12 illustrates the results. Figure 12a demonstrates the effectiveness of *kernel timestamping* of a timing channel with various IPG modulation (ε) values. The data rate of the overt channel was 1 Gbps and the packet size was 1518 bytes. The x-axis is interpacket delays (IPDs) in microsecond and y-axis is a cumulative distribution function (CDF). The vertical line in the middle is the original IPD (=12.2 us) of *Chupja*. In order to detect *Chupja*, the timestamping CDF would be cen-

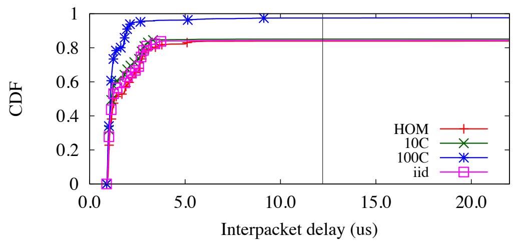


Figure 13: Kernel timestamping with (1518B, 1Gbps).

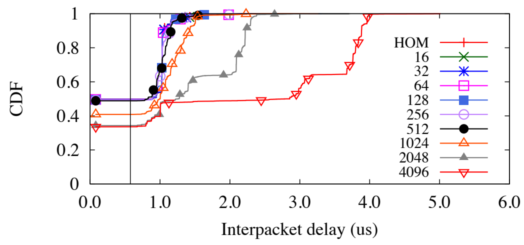


Figure 14: Hardware timestamping with (64B, 1Gbps)

tered around the vertical line at ≈12.2 us. Instead, as can be seen from the graph, all measured kernel timestamps were nowhere near the vertical line regardless of ε values (ε varied between ε=0 [HOM] to ε=4096 `/I/s`). As a result, kernel timestamping cannot distinguish a PHY covert channel like *Chupja*. In fact, even an i.i.d random packet stream is inseparable from other streams (Figure 13). Unfortunately, *zero-copy timestamping* does not help the situation either (Figure 12b). Netmap does not timestamp every packet, but assigns the same timestamp value to packets that are delivered in the one DMA transaction (or polling). This is why there are packets with zero IPD. Nonetheless, Netmap still depends on underlying system's timestamping capability, which is not capable.

On the other hand, *hardware timestamping* using the Sniffer 10G demonstrates enough fidelity to detect *Chupja* when modulation (ε) values are larger than 128 `/I/s` (Figure 12c). However, hardware timestamping still cannot detect smaller changes in IPDs (i.e. modulation, ε, values smaller than 128 `/I/s`), which is clear with a timing channel with smaller packets. A timing channel with 64 byte packets at 1 Gbps is not detectable by hardware timestamping (Figure 14). This is because packets arrive much faster with smaller packets making IPGs too small for the resolution of hardware to accurately detect small IPG modulations.

The takeaway is that to improve the possibility of detecting *Chupja*, which modulates IPGs in a few hundreds of nanoseconds, a network monitor (passive adversary) must employ hardware timestamping for analysis. However, using better hardware (more expensive and sophisticated NICs) still may not be sufficient; i.e. for much finer timing channels. Therefore, we can conclude that a PHY timing channel such as *Chupja* is invisible to a software endhost. However, a hardware based solutions with fine-grained capability [1] may be able to detect *Chupja*.

5 Countermeasures

So far, we demonstrated that covert timing channels implemented in the physical layer can leak secret information without being detected. Such channels are great threats to a system's security, and should be prevented or detected. However, as we discussed, detecting a PHY timing channel is not easy with commodity components. As a result, a network administrator who is worried about information leaks from the network must employ *capable* network appliances for prevention or detection: *PHY*-enhanced network jammers or monitoring appliances could potentially prevent or detect the existence of a covert channel.

6 Conclusion

In this paper, we presented *Chupja*, a PHY covert timing channel that is high-bandwidth, robust and undetectable. The covert timing channel embeds secret messages into interpacket gaps in the physical layer by modulating interpacket gaps at sub-microsecond scale. We empirically demonstrated that our channel can effectively deliver 81 kilobits per second over nine routing hops and thousands miles over the Internet, with a BER less than 10%. As a result, a *Chupja* timing channel works in practice and is undetectable by software endhosts since they are not capable of detecting such small modulations in interpacket gaps employed by *Chupja*. Now that we have demonstrated that a PHY covert timing channel is a security risk, future directions include efficient methods to prevent or detect such covert channels.

7 Availability

The *Chupja* and SoNIC source code is published under a BSD license and is freely available for download at `http://sonic.cs.cornell.edu`

8 Acknowledgements

This work was partially funded and supported by an IBM Faculty Award received by Hakim Weatherspoon, DARPA (No. D11AP00266), DARPA MRC, NSF CAREER (No. 1053757), NSF TRUST (No. 0424422), NSF FIA (No. 1040689), NSF CiC (No. 1047540), and NSF EAGER (No. 1151268). We would like to thank our shepherd, Shyamnath Gollakota, and the anonymous reviewers for their comments. We further recognize the engineers who helped establish and maintain the network infrastructure on which we performed these experiments: Eric Cronise (Cornell Information Technologies); Scott Yoest, and his team (Cornell Computing and Information Science Technical Staff). We also thank Danial Freedman, Ethan Kao, Erluo Li, and Chiun Lin Lim for reviewing and comments.

References

[1] Endace dag network cards. `http://www.endace.com/endace-dag-high-speed-packet-capture-cards.html`.

[2] Endaceprobes. `http://www.endace.com/endace-high-speed-packet-capture-probes.html`.

[3] IEEE Standard 802.3-2008. `http://standards.ieee.org/about/get/802/802.3.html`.

[4] Intel Westmere. `http://ark.intel.com/products/codename/33174/Westmere-EP`.

[5] Myricom Sniffer10G. `http://www.myricom.com/sniffer.html`.

[6] Napatech. `http://www.endace.com/endace-high-speed-packet-capture-probes.html`.

[7] NAPI. `http://linuxfoundation.org/en/Net:NAPI`.

[8] The CAIDA UCSD Anonymized Internet Traces. `http://www.caida.org/datasets/`.

[9] Wildpackets. `http://www.wildpackets.com/products/network_recorders`.

[10] Trusted computer system evaluation criteria. Tech. Rep. DOD 5200.28-STD, National Computer Security Center, December 1985.

[11] BERK, V., GIANI, A., AND CYBENKO, G. Detection of covert channel encoding in network packet delays. Tech. Rep. TR2005-536, Department of Computer Science, Dartmouth College, November 2005.

[12] CABUK, S., BRODLEY, C. E., AND SHIELDS, C. IP Covert Timing Channels: Design and Detection. In *Proceedings of the 11th ACM conference on Computer and Communications Security* (2004).

[13] DOBRESCU, M., EGI, N., ARGYRAKI, K., CHUN, B.-G., FALL, K., IANNACCONE, G., KNIES, A., MANESH, M., AND RATNASAMY, S. RouteBricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009).

[14] FISK, G., FISK, M., PAPADOPOULOS, C., AND NEIL, J. Eliminating steganography in internet traffic with active wardens. In *Revised Papers from the 5th International Workshop on Information Hiding* (2003).

[15] FREEDMAN, D. A., MARIAN, T., LEE, J. H., BIRMAN, K., WEATHERSPOON, H., AND XU, C. Exact temporal characterization of 10 gbps optical wide-area network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (2010).

[16] GIANVECCHIO, S., AND WANG, H. Detecting covert timing channels: an entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security* (2007).

[17] GILES, J., AND HAJEK, B. An information-theoretic and game-theoretic study of timing channels. *IEEE Transactions on Information Theory 48*, 9 (Sept. 2002), 2455–2477.

[18] HAN, S., JANG, K., PARK, K., AND MOON, S. Packetshader: a gpu-accelerated software router. In *Proceedings of the ACM SIGCOMM 2010 Conference* (2010).

[19] HANDLEY, M., KREIBICH, C., AND PAXSON, V. Network intrusion detection: Evasion, traffic normalization. In *Proceedings of the 10th USENIX Security Symposium* (2001).

[20] KUNDUR, D., AND AHSAN, K. Practical internet steganography: Data hiding in ip. In *Proceedings of Texas workshop on Security of Information Systems* (2003).

[21] LAMPSON, B. W. A note on the confinement problem. *Communications of the ACM 16*, 10 (October 1973), 613–615.

[22] LEE, K. S., WANG, H., AND WEATHERSPOON, H. SoNIC: Precise Realtime Software Access and Control of Wired Networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation* (2013).

[23] LIM, C. L., LEE, K. S., WANG, H., WEATHERSPOON, H., AND TANG, A. Packet clustering introduced by routers: Modeling, analysis and experiments. In *Proceedings of the 48th Annual Conference on Information Sciences and Systems* (2014).

[24] LIU, Y., GHOSAL, D., ARMKNECHT, F., SADEGHI, A.-R., SCHULZ, S., AND KATZENBEISSER, S. Hide and Seek in Time: Robust Covert Timing Channels. In *Proceedings of the 14th European Conference on Research in Computer Security* (2009).

[25] LIU, Y., GHOSAL, D., ARMKNECHT, F., SADEGHI, A.-R., SCHULZ, S., AND KATZENBEISSER, S. Robust and Undetectable Steganographic Timing Channels for i.i.d. Traffic. In *Proceedings of the 12th International Conference on Information Hiding* (2010).

[26] MALAN, G. R., WATSON, D., JAHANIAN, F., AND HOWELL, P. Transport and application protocol scrubbing. In *Proceedings of IEEE Conference on Computer Communications* (2000).

[27] MARIAN, T., LEE, K. S., AND WEATHERSPOON, H. Netslices: Scalable multi-core packet processing in user-space. In *Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (2012).

[28] MURDOCH, S. J., AND LEWIS, S. Embedding covert channels into tcp/ip. In *Proceedings of 7th Information Hiding workshop* (2005).

[29] NLR. National Lambda Rail. `http://www.nlr.net/`.

[30] OLSSON, R. pktgen the linux packet generator. In *Proceeding of the Linux symposium* (2005).

[31] PADLIPSKY, M. A., SNOW, D. W., AND KARGER, P. A. Limitations of end-to-end encryption in secure computer networks. Tech. Rep. ESD-TR-78-158, Mitre Corporation, August 1978.

[32] PENG, P., NING, P., AND REEVES, D. S. On the secrecy of timing-based active watermarking trace-back techniques. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy* (2006).

[33] RIZZO, L. Netmap: a novel framework for fast packet I/O. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference* (2012).

[34] ROWLAND, C. H. Covert channels in the tcp/ip protocol suite. *First Monday* (July 1997).

[35] RUTKOWSKA, J. The implementation of passive covert channels in the linux kernel. In *Proceedings of Chaos Communication Congress* (2004).

[36] SHAH, G., MOLINA, A., AND BLAZE, M. Keyboards and covert channels. In *Proceedings of the 15th conference on USENIX Security Symposium* (2006).

[37] ZANDER, S., ARMITAGE, G., AND BRANSH, P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials 9*, 3 (October 2007), 44–57.