Towards Automatic Network Side-Channel Discovery Through Abstraction & Active Network Measurement

Ben Mixon-Baca
University of New Mexico
1 University Blvd. 87131
Albuquerque, New Mexico, USA
bmixonb1@cs.unm.edu

ABSTRACT

Active network measurement techniques are critical to security researchers, computer network researchers, digital rights activists, and many others. A class of these techniques uses side-channels, and while incredibly powerful, are non-trivial to implement. They often require researchers to manually inspect the source code or reverse engineer an operating system binary to discover the underlying data structures and components that produce the side-channel. We use previous work on this class of measurement techniques to model common elements. The model allows us to design experiments to find possible side-channels present in operating systems. We implemented these experiments for a real computer network, the IPv4 Internet and released the sourced code. We found that roughly 1.4% of the Internet possesses a possible side-channel. We validate our result using Nmap, a well regarded tool in industry and academia. We show our result is consistent with previous work by comparing our result to reported percentages of specific side-channel used in other work.

Keywords

Computer Network, Network Measurement, Side-channel

1. INTRODUCTION

Active network measurements allow various communities to measure important characteristics of their computer networks. Computer network researchers use such measurements to estimate latency, bandwidth, and topology. Security researchers use them to measure trust relationships between hosts or portions of a computer network, operating system (OS) signatures, or network topology. Finally, digitial rights activists benefit by having tools to determine whether a country or Internet service provider (ISP) is manipulating communications of Internet users within its domain of control. While many measurement tools and techniques exist to perform active network measurements[3, 26, 2, 11, 6, 20], some of the most powerful techniques, leverage information leaks to perform measurements. Unfortunately, these measurement techniques are laborious to discover and implement. The cost is high because the information leaks used to perform measurements depend on OS specific details. We expect that the presence of these information leaks will likely increase over time as mechanisms to combat attacks are introduced. Examples of such mechanisms include SYN cookies [8], which are used to mitigate denial-of-serice attacks and TCP in-window attack mitigations [8]. This work moves toward a system for discovering such information leaks automatically. We do so by modeling common elements of existing techniques and use the model to design experiments that discover these information leaks.

Side-channels are features of a system that unintentionally leak information. Side-channels are most commonly associated with cryptography, however, they have also proven useful in active network measurements and are the focus of our work. Side-channels get their name because, unlike a normal channel that a machine explicitly uses to communicate (e.g., port number), side-channels are unintentional and unexpected. Side-channels are used to infer information flow between computers being measured. The information could be the number of hosts behind a firewall [27], the number of packets a host sent [16], the round-trip time between two hosts [1], whether state-level censorship is being performed [9], or even connection resetting or data injection attacks [5]. Fortunately for OS designers, the cost of finding side-channels useful for network measurement is high.

To date, only one method has been proposed to automatically find possible side-channels for network measurements [10]. Ensafi et al. explored automatic side-channel discovery through model-checking (i.e., by using formal methods) to infer whether an OS observed non-interference properties. We take an analogous approach by using a model to express the problem of discovering side-channels for network measurement. Our work differs from the ealier work by inferring violations of non-interference properties using empirical methods.

Non-interference properties are a model in computer security that describes the conditions necessary for a computer system to leak information between unauthorized users. When non-interference is violated, a user of a computer system has the ability to infer information about another user sharing the system. For example, if one user modifies the computer's state, that user's effects will leak information to other users who are not explicitly permitted to view this information. When non-interference properties are observed, one user's actions can not leak information to any unautherized users. In the literature, shared, limited resources, such as finite buffers, counters, or retransmission timers are the only reported cause of non-interference violations. Because we can not speak to what has *not* been discovered, we cannot claim this list to be exhaustive. We did find through our work that for now, the list is sufficient for the purposes of validating our approach.

To move towards automated side-channel discovery, our work consists of three phases:

Phase 1. Design a model of existing side-channels based on current measurement techniques.

Phase 2. Develop experiments based on the model.

Phase 3. Implement the experiments for the IPv4 Internet and validate the results against existing techniques.

Our model is inspired by existing side-channel based measurement techniques. We use our model to reason about the conditions necessary for a side-channel and how the side-channel might be detected. We observe that all computers measured in the literature have specific data types associated with their side-channels. We also note that these data structures have a scope associated with them. The data structure's scope determines whether non-interference is violated. We observe three phases common to all existing techniques, which correspond to a three-phase experiment we designed to detect side-channels. We use the experiment to test the following hypothesis:

- *Computers on the IPv4 Internet have side-channels.*

To test this hypothesis, we ask the following three research questions[1]:

RQ 1. *What fraction of computers on the IPv4 Internet respond to randomly initialized packets?*

RQ 2. *What fraction of computers on the IPv4 Internet send multiple responses to randomly initialized packets?*

RQ 3. *Of all computers that respond in Phase 1, what fraction use non-per-flow data structures?*

The first question is important because it gives us a general idea of the fraction of computers that will even respond to our packets. We must know which computers respond before testing whether they have a side-channel. The second question is useful in actually determining the fraction of computers on the Internet that maintain a buffer and have retransmission timers. Because buffers and timers have been shown to violate non-interference properties in several existing techniques [3, 10], knowing this fraction gives us a lower-bound estimate of the number of hosts that have a buffer and a timer. Though we cannot determine whether the buffer or timer creates a side-channel, this is a useful first step in determining the fraction of IPv4 Internet computers that may have buffer and timer based side-channels. The third question is important because it gives a lower-bound estimate on the number of computers that have a possible non-per-flow based side-channel. Non-per-flow refers to the concept of scope. Buffers, counters, and timers all have some associated scope. The most restictive, and thus, least useful scope for measurements is per-flow; ruling out this case by detecting non-per-flow scope is important. The details of per-flow scope are covered in Section 4, but to make things concrete, a non-per-flow scope indicates that all hosts communicating with some computer do not have seperate buffers, counters, our timers. These estimates are lower-bound because there may be cases our experiments do not address.

To answer our three research questions, we ran 114 randomized experiments. Each experiment measured 1000 random IPv4 addresses. We found that, on average, 3.75% of hosts respond to our packets. We found that, on average, 0.30% of Internet hosts retransmitted packets. Finally, we found that, on average, 1.40% of computers on the Internet possess some form of incremental counter. Finally, we found the number of incremental counters we discovered to be within 0.4% of a known side-channel, which is consistent with at least the Chinese Internet as of 2014 [9]. Each of these findings is based on a 95% confidence level. This suggests that side-channels are still viable for network measurement and that researchers maybe find our tool useful.

This is the first work we are aware of to use empirical methods to infer violations of non-interference properties in computer OS network code. We make the following contributions:

1. The first empirically based approach for inferring non-interference property violations in computer network side-channel discovery.
2. Demonstrate the viability of automatic side-channel discovery using emperical methods.
3. A tool[2] to support the discovery of side-channels on hosts connected to the Internet.
4. A lower bound estimate on the fraction of computers that have side-channels.

The remainder of our paper is as follows: Section 2 gives background information, Section 3 covers related work, Section 4 describes our model. We describe our experimental design and implementation in Section 5. We report our results in Section 6. We discuss the results in Section 7 and cover future work in Section 8.

2. BACKGROUND

Our work requires understanding three components:

1. Active Network Measurement
2. Non-interference properties
3. Side-channels

2.1 Active Network Measurement

Active network measurement is a set of techniques used to discover characteristics of a computer network. A typical scenario involves one computer, the measurement machine, sending packets to some other computer, collecting the responses, and analyzing the exchange. We denote the measurement machine as MM and any host to which MM sends packets as H. When there are multiple H, subscripts are used. A schematic is provided for reference in Figure 1

[1]The first phase of the experiment addresses the first two research questions, and the second and third phases address the third question.

[2]Source code is available at https://github.com/beenjaminmb/SK-ythe. It is under active development but the version used to generate data for these experiments is available for download.

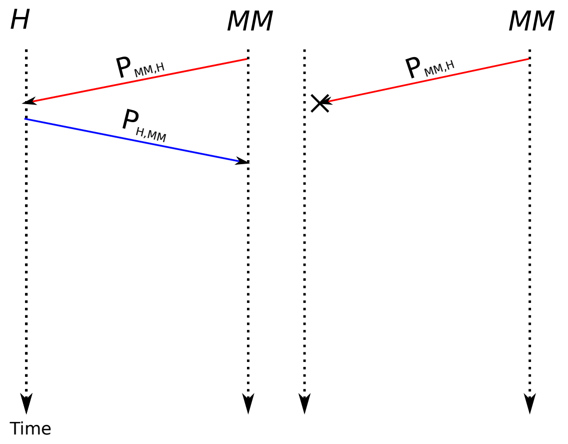

Figure 1: On the left, a typical active network measurement scenario. H is reachable from MM. On the right, MM attempts to measure some inaccessible computer.

We denote packets as P. We denote packets sent from MM to H as $P_{MM,H}$, when other subscripts are used, the meaning is clear from context. We use the terms *packet* and our notion, P, synonymously.

Tools to perform active network measurement have been used in both academia and production for many years [17, 12, 7, 4]. Each tool sends packets from MM to H. The scans implemented by these tools are similar and typically assume some well known protocol is in use. These tools program logic specific to a given protocol into the tool itself to interact with H. One tool, Nmap, implements one scan in particular that uses a side-channel to perform measurements. This makes Nmap the closest production program to ours. However, none of these tools, including Nmap, explicitly search for general side-channels. This distinguishes our work from existing tools and techniques. Our tool attempts to discover possible side-channels and gives a lower-bound percentage for different types of side-channels using a unique methodology based on existing techniques in the literature.

2.2 Non-interference

Non-interference properties are a component of a model in computer security first proposed by Goguen and Meseguer in 1982 [14]. Their model is a strict, multilevel security policy. It is strict in the sense that a user modifying the state of the system is unable to learn information about other users of the system, unless sufficient permission is granted. It is multi-level in that users of some computer system may have different access to different resources. Access should be enforced by the system. When non-interference properties are observed, no user is allowed to learn anything about the system or other users unless explicitly allowed. On the other hand, if non-interference properties are violated, some user's modification of the system's state is observable by another user lacking appropriate permissions.

In side-channel based network measurement techniques, non-interference property violations arise because of shared, limited resources in the OS of H. In the literature, these shared, limited resources are typically finite buffers, various types of counters, or timers that drive the system on which the timer is running, to change state as some function of time. H has only finite memory to maintain these data structures. Because these resources are constrained, OS designers attempt to share these resources among parties in a way that is both secure, and efficient. If the implementation for sharing resources is not designed properly, non-interference properties are violated and information is leaked.

Ensafi et al. were the first to express the problem of discovering side-channels for active network measurement using non-interference properties [10]. They used model checking to discover the existence of side-channels in a computer OS's networking code. Our approach differs from theirs because, while we use a similar model, we do not use formal methods to prove non-interference properties of the model. Instead, we use the model as a device to aid in designing experiments used to automatically discover side-channels by inferring whether or not non-interference properties are violated.

2.3 Network Measurement & Side-channels

We now give a simple example of how a side-channel is used to infer a non-interference property violation and how the violation can be mitigated.

Suppose a researcher, performing measurements from MM wants to determine whether two computers, H_1 and H_2, are exchanging packets. Suppose that whenever H_1 sends a packet to any other computer, it encodes an identification number, referred to as, $IPID$, into each packet. Further suppose that MM knows this a priori. Assume initially, that $IPID = 0$. MM must first determine the initial value of $IPID$. MM sends a packet to H_1. H_1 sends a response to MM with $IPID$ encoded in the response then increments $IPID$, resulting in $IPID = 1$. After this, let H_2 send a packet to H_1. H_1 sends a response to H_2, resulting in $IPID$ being incremented to 2. MM sends a final packet to H_1 who sends a response. MM records this final $IPID$ encoded in this response. Because H_2 sent a packet to H_1 in between MM's first and second packets, the difference is greater than 1, and MM can infer that H_1 is communicating with a computer in addition to MM. In other words, H_1 violated a non-interference property.

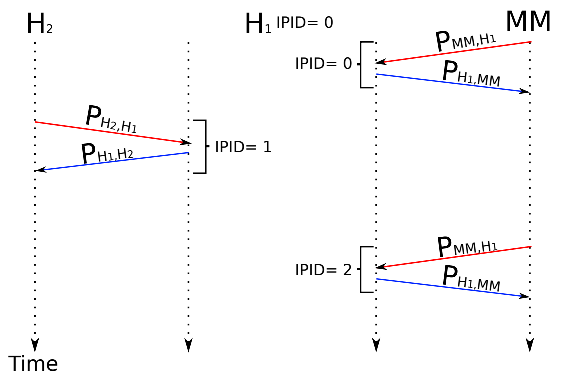

Figure 2: Incremental counter ($IPID$) side-channel: H_1 has a shared $IPID$ for MM and H_2. Non-interference is violated.

In this case, non-interference is clearly violated because H_1 has only a single *global* counter, that it encodes in all packets. This permits MM to infer that H_1 and H_2 can communicate. More subtle is the fact that MM is inferring violations by observing how H_1's $IPID$ changes. In this way, information about H_1 and H_2's communication flows between MM and H_1 due to the $IPID$ side-channel. Figure 2 illustrates this situation.

The cause of the $IPID$ side-channel arises from the fact that H_1's $IPID$ is *global* to all parties communicating with H_1. A mitigation to this side-channel is to restrict the scope of the $IPID$. Suppose that instead of a global $IPID$, H_1 has one $IPID$ for H_2 and one for MM. Now, whenever MM sends its final packet to H_1, the $IPID$ specific to MM is only affected by MM's first packet, and not by H_2's packet. In this way, MM infers that H_1 does not violate a non-interference property. Figure 3 depicts this situation.

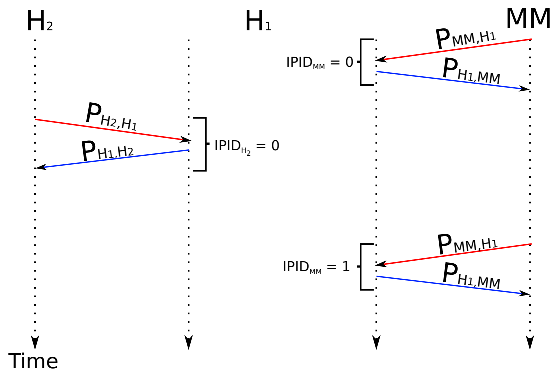

Figure 3: Schematic when H_1 has a separate $IPID$ for MM and H_2. Non-interference properties are observed.

3. RELATED WORKS

Before we describe our model, we review work that inspired ours. The first reported side-channel based active measurement technique came in 1998 on the bugtraq mailing list [3]. Antirez's technique allowed an attacker to impersonate another computer on a network and send packets to a target computer the attacker wished to learn information about. The target, not realizing it was communicating with the attacker, would attempt to respond. This induced response disclosed its existance to the attacker. This disclosure was possible because of a shared, limited resource (i.e., side-channel). The details are almost identical to the *global IPID* example in the previous section.

Ensafi et al. [10], found two more side-channels automatically by using model checking. The first arose from interactions between a port on a host and a global buffer in the host's OS, used to store packets it receives. The second was the result of interactions between a host's port and a global counter used to limit the number of times the computer can resend specific packets. Enasfi et al. subsequently developed a technique to detect the direction in which packets are dropped. In this work, they did not claim automatic discovery of side-channels. They used a combination of Antirez's technique and the retransmission counter in a Linux OS. This had a profound effect on the general network measurement community and was used in later studies to explore how the Chinese government implements access control[9].

Almost simultaneously to Ensafi et al.'s work, Gilad and Herzberg explored side-channels to interrupt communication and take over connections of computers behind firewalls [13]. Their work uses a threat model not consistent with ours. They assume that the machine they want to measure has a piece of malicious code already running on it. We do not make this assumption.

Knockel and Crandall found a global buffer in the Linux OS's Internet Protocol code that could be used to detect whether two parties were communicating, and it could count the number of packets being sent between parties [16]. Their technique is interesting, because unlike all of the other techniques, which often require TCP to be used with IP, their technique uses ICMP messages. Also, unlike the example, Antirez's, and Ensafi at al.'s techniques, which require the $IPID$ field to change in a very specific way, Knockel and Crandall's technique does not have this requirement.

Zhang et al. used retransmission behavior similar to Ensafi et al.'s initial discovery to infer whether computers were behind a firewall[27]. Their research gave insight into how simple changes in a packet's fields can lead to interesting behavior. They observed that a computer they sent TCP/IP packets to would send two distinct responses depending on how they set the TCP sequence number. TCP sequence numbers are used to ensure that data sent over TCP is properly ordered. They sent a packet with some sequence number, say N, and then sent another packet that was exactly the same, except for the sequence number, which was set to $N + 1$. Depending on the internal state of H, they received distinct responses, allowing them to infer whether a machine was behind a firewall. The insight we gleaned was that a simple increment operation to a field could result in a potentially useful side-channel.

Alexander and Crandall used yet another global buffer and retransmission timer in combination to infer the round trip time between two arbitrary H on the Internet[1]. They observed that a Linux OS attempted to save space in a buffer used for new TCP connections by evicting packets if the buffer become more than half full. This meant that if they could send packets at a fast enough rate between MM to H in a specific way, they could determine whether packets were being removed from the buffer. Depending on whether packets were evicted, they would estimate the round-trip-time between the two computers as the rate at which packets were sent. In this case, a non-interference property was violated because the buffer storing the packets was global and shared by all packets that H receives.

Zhang et al. later improved on Alexander and Crandall's original technique by altering the way packet rate is computed [27].

Finally, Qian et al. found a side-channel in the Linux OS based on a global packet retransmission rate-limiting counter [5]. RFC5961 actually makes this attack possible and at the time [25], Linux had this RFC implemented. They were were able to guess TCP sequence numbers which are recommended to be statistically hard to guess. This field is usually generated randomly because if an attacker can guess the number, then it can reset the connection or even inject data into the TCP stream. This has obvious security implications and Qian et al. demonstrated the feasibility of these attacks. In this case, a non-interference property was violated because the global counter could be controlled by an attacker to induce a different state in the target, H.

These techniques possess several commonalities. The first is that the side-channels have always come from a buffer, counter, or timer. The second is that these data structures always have some kind of scope associated with them. If the scope is global (i.e., shared between all communicating parties), then a non-interference property violation definitely occurs. Even if the scope is not global, because of the way operating systems are implemented, it is still possible to infer non-interference property violations. Our third observation is that all these techniques implement a similar algorithm when measurements are performed. These observations lead us to the model described in section 4.

4. MODEL OVERVIEW

The work discussed in Section 3 guided our choice of a three-phase experiment. To design an experiment general enough to capture the high-level details of many side-channel network measurements, while still being concrete enough to support inference of violations, we consider three questions:

1. What are the minimum details required to describe any host being measured?
2. How should a side-channel be modeled?
3. How similar are the algorithms for sending packets?

The first question was motivated by the need to describe sufficient elements that H possesses that make a side-channel measurement possible. The second question was motivated by the observation that MM has a very limited view of the network on which it is performing measurements, yet is able to determine information about how two other parties communicate. The third was motivated by the observation that common components in the packet sending algorithm might be useful for designing experiments.

4.1 Host Model

We make several assumptions about H. First, we assume the resources that comprise its state are finite. Second, we assume that such resources can be exhausted if an adversary, MM, sends packets to H at a steady rate meeting or exceeding 1 Gbps. If packets are sent too fast, then we might accidentally affect timer and counter[3] based side-channels. These side-channels have relied on packet rates to make inferences in the past, so accidentally modulating them could interfere with experiments not explicitly targeting such counter and timer based side-channels.

We now specify the components of a machine's state and use this to aid in answering the three questions asked at the beginning of this section. To answer the first question, we looked at features common to all H being measured in the literature. We found several data types that have been used as side-channels. We found that real-world Hs always have a set of buffers, B, counters, C, timers, T, and a transition function, τ. Together, these components describe a computer in our model, expressed as:

$$H = \{B, C, T, \tau\}$$

Each of these data types also has some associated *scope* for each $B, C, T, \tau \in H$. In the literature, there are three scopes most often cited. They are:

[3]Rate-limiting counters to be specific.

1. Global
2. Per-flow
3. Per-host

Global scope is when H maintains a single buffer, counter, or timer for all users of the system. For example, if $H.B$ has global scope, then a non-interference property will definitely be violated because every user that interacts with $B \in H$ can be observed by other users of H. Global scope is illustrate in the example in Section 2.3 as well as Figure 2. Per-host scope is when H maintains separate copies of its data structures for each computer that communicates with it; it is a more restrictive scope. This scope often accompanies increased resource consumption in H. An example and accompanying diagram are provided in Section 2.3, Figure 3. Per-flow scope is the most restrictive and is when H maintains seperate data structures for individual communication flows for each party communicating with H. Our experiments are concerned with differentiating this scope from the other two since it is the most restrictive, and therefore, least likely to violate non-interference properties. This permits researchers to investigate hosts whose side-channels are easier to detect.

Our model includes additional details compared to Goguen and Meseguer's model because we specify the three data types. This is necassary for reasoning generally about side-channels and designing experiments.

4.2 Side-channel Model

In network measurement, the side-channel arises due to competition for H's resources among other computers with whom H is communicating. In the literature, MM must be able to use packet exchanges between itself and H to tease out information about H and the parties H is communicating with. We model the channel with the following function:

$$SC : H \times H \mapsto H$$

SC takes two Hs as arguments and returns one H. Communication implies at least two Hs but MM is interested only in violations for a particular H. The communication these hosts engage in transforms their internal states according to their respective τ functions. MM's goal is to infer whether the H of interest violates non-interference properties. We assume we can only communicate with the H we are measuring and that it is the one returned.

As an example, imagine two computers, H_i and H_j, communicating. We model this exchange as:

$$H'_i = \tau(H_i) = SC(H_i, H_j)$$

MM, does not directly know the internal state of H_i or H'_i, but seeks to measure the state transition from H_i to H'_i under τ based only on the exchanged packets. If MM measures a state transition, it infers a non-interference property violation and the measured host is further considered for side-channels.

To detect state transitions, MM must look at the set of packets, P, exchanged between itself and H_i. MM applies a function d to P, in order to map fields inside P to some data structure in H_i. Side-channels *must*, if present, be detectable based purely on these exchanged packets. We model the relationship between the packet fields and the state-transition of H_i, induced by τ as:

$$d(P) \propto H_i = \tau(H_i) = SC(H_i, H_j)$$

The details of d depend on the particular side-channel. This is the most important component of our model because it is where we define a function to detect side-channels.

4.3 Measurement Model

To answer the third question at the beginning of this section, we looked for similarities between currently existing techniques' measurement algorithms and found that each implemented three distinct phases:

Phase 1. Preparation

Phase 2. Modulation

Phase 3. Non-interference Inference

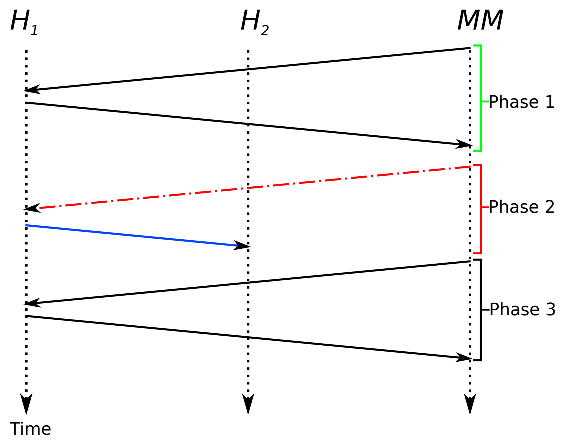

Figure 4: A typical scenario of active measurement using side-channel.

Figure 4 demonstrates the three phases used in most scans. Some scans do not require that MM actively send packets in Phase 3, however, all techniques inspect response packets at the end of their algorithms, so we generalize with a third phase.

Existing techniques assume MM knows a side-channel exists on H_1 before phase 1 begins[4]. We do not assume this for our experiments since the point of our experiments is discovering side-channels.

When Phase 1 begins, MM sends packets to H_1. The details of how packets are configured and sent is complex and often specific to a particular measurement technique. To handle this complexity, we treat Phase 1 as some process of initializing and sending packets to H_1. Phase 1 is thought of as a preparation phase, where MM attempts to manipulate some B, C, T, or combination thereof, within H_1. In our model, this is expressed as follows:

$$H_1' = \tau_{H_1}(H_1) = SC(H_1, MM)$$

[4]MM would have validated this before performing the measurement

We assume, MM, can directly measure its own internal state transition, $\tau_{MM}(MM)$. Therefore, it does not matter that SC only returns H_1'. Since we define $\tau_{MM}(MM)$ as the state-transitions of MM under communication with other computers, this fully captures the process MM uses to generate the packets it sends to H_1 in Phase 1.

In Phase 2, MM sends packets that force H_1 and H_2 to communicate in some way. In Figure 4, MM impersonates H_2 to H_1. But MM could impersonate H_1 to H_2; either case is valid. The purpose of Phase 2 is to try to force a violation of non-interference properties in one of H_1's data structures.

In Phase 3, MM attempts to measure how the internal state of H_1 was affected by the induced communication between H_1 and H_2 in Phase 2. Depending on the specifics of the technique, MM may need to send a final round of packets, as in the diagram, but this is not a strict requirement; MM may instead simply collect responses from H_1.

5. EXPERIMENTAL DESIGN

Based on the previous analysis of measurement algorithms in the literature, our experiment reflects their three-phase approach as follows:

Phase 1. Host Discovery

Phase 2. Candidate Side-channel Modulation

Phase 3. Side-channel Detection

We focused on the Internet because all previous work on side-channel based measurements have been used to measure Internet characteristics. We focus on a subset of protocols used in Internet communication:

1. Internet Protocol (IP) [23]
2. Transmission Control Protocol (TCP) [24]
3. User Datagram Protocol (UDP) [21]
4. Internet Control Message Protocol (ICMP) [22]

All experiments were conducted using an Ubuntu GNU/Linux 16.04 LTS Virtual Machine using VirtualBox 5.0.24. The machine had 32 GB RAM, and 4 processors. The network on which the measurements were conducted was a 10 GB Ethernet link connected to an unfiltered up-link. This allowed us to send any type of packet required for our experiments.

5.1 Host Discovery

This phase is designed to answer the first two research questions asked in Section 1, and is required to bootstrap Phase 2 of our experiment. Bootstrapping is achieved by collecting responses from packets we send in this phase and using them as packets to send in Phase 2. We are not concerned with the specific types of responses in this phase, we simply aim to elicit a response from some random H, based on packets derived from the process described in the next paragraph. To relate back to our model, this phase is the preparation phase and is equivalent to $\tau_{MM}(MM)$.

Initially, MM generates 1000 random IP addresses. IP addresses are used to uniquely identify some H on the Internet. We maintain a blacklist of IP addresses that have opted out of our experiments to ensure we do not scan unwilling participants and is important ethically to ensure we do not violate other hosts' acceptable use policies. Even though we cannot scan the full IPv4 address space, we are still able to measure 89% of the roughly 4 billion IPv4 address. For each H, MM generates a single packet. The packet is TCP 50% of the time, UDP 25% of the time, ICMP 15% of the time, and random bits the remaining 10% of the time.

TCP packets are composed of many fields. The two most important fields are the TCP flags field and the port numbers. TCP flags is an eight-bit vector used to signal various state transitions in the TCP protocol. This field is weighted so that 50% of packets have only a single TCP flag set. 25% have two flags set, and the remaining 25% sets three or more flags. This scheme was chosen because side-channel measurements using TCP usually have either one or two bits set but interesting behavior may be observable in other cases as well. Port numbers are used to identify specific programs running on computers that wish to communicate. Two port numbers are required, the *source* port identifies the program running on the sender (i.e., MM), and a *destination* port, identifies the program running on the computer being measured, H. The source port is initialized uniformly between 1 and 65536, inclusive. The destination port is chosen as a well-known[5] port with 50% probability and randomly with 50% probability. One of 22, 23, 53, 80, or 443 is chosen uniformly as the well-known port. These ports are used most often, according to the literature [19]. The remaining fields in the TCP header are chosen uniformly between 0 and N_i, where N_i is the length of the i^{th} field. The checksum is computed correctly for each packet.

UDP packets have only four fields: source and destination ports, checksum, and payload length. We use the same port assignment scheme as above. The checksum is correctly computed and the payload is intialized as with TCP.

ICMP packets contain a type field, a control code, a checksum, and the remainder of the packet. The control code is set to be an ICMP echo request message 50% of the time, and something else, the remaining 50% of the time. The checksum is set as in TCP and UDP, and the remainder of the packet is set to all zeros. The length of the remainder uses the same scheme used for TCP and UDP packet.

If it is some other protocol not one of the above, the IP payload is initialized to random bits.

Each packet generated is then sent to its respective H. The source IP address of the packet was assigned to a non-existing computer on the same sub-network as our MM. We chose this for two reasons. First, because of the way our MM's OS is implemented, if MM sends packets with its own IP address, it will respond to packets sent to it. This would introduce noise into the data and corrupt our experiments. Instead of addressing the packet directly as MM's IP address, we use an IP address that we know does not have a computer associated with it. Second, the only other way of achieving the same affect is if MM has a firewall configured. This would also introduce noise and corrupt our experiments so we did not use this approach.

A single packet generated in this phase corresponds 1-to-1 with an IP address. These 1000 packets are sent in 1 Mb bursts at a rate of 500 Mbps. After all packets are sent, MM waits two minutes and simply collects the responses. These responses are then used to bootstrap Phase 2. Figure 5 is a schematic of Phase 1.

[5]Well-known ports are those specified by IANA as reserved for specific Internet services.

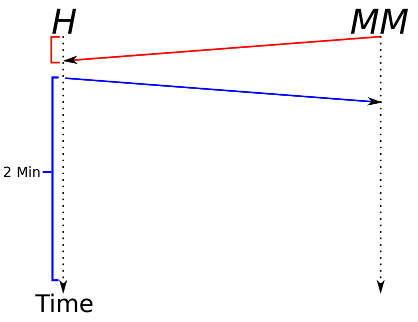

Figure 5: Phase 1 of our experiment. One host being measured but 1000 are used an actual experiment.

5.2 Side-channel Candidate Modulation

This phase is designed to answer the third research question question from Section 1 After the two minutes at the end of Phase 1, we assume all responses, $P_{H,MM}$, have been collected and continue. The state of H under consideration is assumed to be settled from any perturbations due to Phase 1 measurements. Because we know the packet, $P_{MM,H}$, that generated $P_{H,MM}$, we can use $P_{MM,H}$ as a control packet to further interrogate H. We get initial response behavior from H by sending five copies of $P_{MM,H}$ and waiting two minutes.

We continue by applying a function, f, to $P_{MM,H}$, to get $P'_{MM,H} = f(P_{MM,H})$. The function simply increments the source port of $P_{MM,H}$ by one. The intuition is that a flow corresponds to a source port number. The simplicity of using the increment function was inspired by Zhang et al. 's [27] observation that simple functions applied to fields in a packet can produce interesting results in the responses received. Our notion of flow as corresponding to a 4-tuple {source port, source address, destination port, destination address} is consistent with semantics of port numbers and IP addresses in networking and also with the literature [18, 15]. We also record an initial response, $P'_{H,MM}$, for $P'_{MM,H}$ by sending five copies of it to H and waiting two minutes. Finally, MM attempts to induce a response from H that is measurable by sending five copies of $P_{H,MM}$ and immediately after, sending five copies of $P'_{H,MM}$.

5.3 Side-channel Detection

The final phase simply collects and analyzes responses generated in Phase 2. In the literature, Phase 3 occasionally involves sending additional packets and collecting responses, we do nothing more than collect and analyze responses. In our model, this would be still be equivalent since MM would simply undergo a different state transition. This is the most important phase because we attempt to find a correspondence between the control and port modulated packets in Phase 2 and the responses received in this phase. In our model, we would be attempting to find an appropriate d to satisfy the following relationship:

$$d(P_{MM,H_i}) \propto H_i = \tau(H_i) = SC(H_i, MM)$$

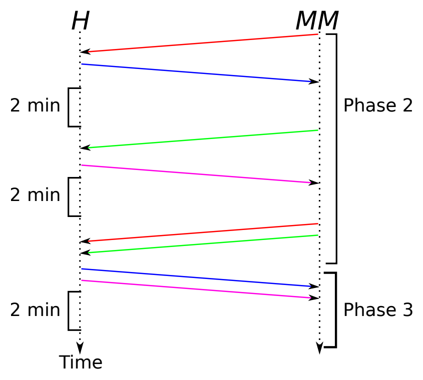

Figure 6: Phase 2 of our experiment. One host is shown, but 1000 are used in the actual measurement.

5.4 Measurement Synopsis

Our experiments consisted of 114 replications of a single, three-phase experiment. Each experiment selects 1000 random IP address from the IPv4 address space, corresponding to $H = \{H_1, H_2, ..., H_{1000}\}$ in our model. A packet set, $P_{MM,H}$, is generated by creating a packet, P_{MM,H_i} for each H_i. Each packet, $P_{MM,H_i} \in P$ is sent it its respective H_i. Some unknown subset, $P'_{MM,H}$, of $P_{MM,H}$ will generate a subset of responses, $P_{H,MM}$, used to bootstrap Phase 2.

Phase 2 uses $P_{MM,H_i} \in P'_{MM,H}$ as a control packet to send to H_i. 5 control packets, P_{MM,H_i} are sent to H_i, completely unmodified and a wait period of 2 minutes is observed. This ensures that the internal state of H_i is allowed to settle from previous perturbations due to our 5 P_{MM,H_i}'s. We then apply a function, f, to P_{MM,H_i} to change the source port of P_{MM,H_i} and send 5 of these modified copies and wait 2 minutes.

In Phase 3, we take the responses H_i sent as a result of receiving P_{MM,H_i} and f, and apply a function to each of P_{MM,H_i}'s fields to determine whether the fields are incrementing. We chose to look for incrementing fields because this is a common way hosts have updated their counters in the literature and we would like to find counters for H. We are especially interested in how counters associated with the responses from control and source port modified packets compare. Recall that the point of modifying the source port was to attempt to determine the scope of the associated field. This test attempts to find the scope of counters for H_i. Because of how the test works, we can only determine whether the host uses per-flow counters or not. If a counter is per-flow, then it is likely not useful to a researcher looking for side-channels, otherwise, it is worth investigating further.

6. RESULTS

We now present the experimental results. Phases 1 is presented by itself, followed by Phases 2 and 3 because of their interdependence.

6.1 Phase 1

Recall that Phase 1 was concerned with answering the first two research questions; *what fraction of Internet hosts respond to randomly initialized packets, and what fraction sends multiple responses to a single, randomly initialized packet?*

Respond	Retransmit
3.9%	0.3%

Figure 7: From left to right: Percent of IPv4 computers that respond to randomly initialized packets, Percent of IPv4 computers that retransmit packets

The answer to the question concerning multiple responses is particularly important because it gives a lower-bound estimate on the fraction of possible buffer and timer-based side-channels. We infer that if more than one response was received, then H must be storing packets in some finite buffer and that it has a timer to periodically resend the packet. Unfortunately, we can not say anything about the scope of the buffer. We found that 0.3% of IPv4 Internet computers had a possible buffer or timer with a confidence interval of ± 0.1 at a confidence level of 95%. A larger number is better because it implies that a larger fraction of the Internet can be used in side-channel based network measurements. Because this is only a lower bound, it is possible that a larger fraction of the Internet is actually useful for side-channel based measurements. Even at only, 0.3%, 1.4×10^9 computers are still candidates for future measurements.

6.2 Phase 2 & 3

The purpose of Phases 2 and 3 was to answer research question three; *Of all computers that respond in Phase 1, what fraction use non-per-flow data structures?* The results are presented in Figure 8

Non-per-flow Counters	Nmap estimate
1.4 %	1.0%

Figure 8: From left to right: Fraction of IPv4 Internet with non-per-flow counter, Percent of global IPIDs found by Nmap

Figure 8 provides a lower bound estimate of 1.4% of IPv4 Internet hosts that posses some kind of non-per-flow counter. Nmap found 1.0% of our samples had a known side-channel, that is, the globally incrementing IPID. These results are within $\pm 0.1\%$ of the population mean and we can be sure with 95% confidence. This demonstrates our model and methodology agree within 0.5% of a well regarded and established tool indicating the potential for success in future work.

7. CONCLUSION

We demonstrated how to map fields in packets back to their data structures. We did this by answering (*RQ 2*) in Phase 1 and (*RQ 3*) in Phase 2. For (RQ 2), we were able to infer that H had a timer and buffer based on whether it sent multiple responses. We found that, on average, at least 0.3% of hosts have some finite buffer and associated timer. We were unable to determine whether this fraction of computers possessed a buffer or timer-based side-channel but it is a promising first step towards automatically discovering such side-channels. For (*RQ 3*), we were able to find a counter field by performing a simple analysis of the Phase 3 packets, P_{MM,H_i} and $P_{H_i,MM}$, to find fields that incremented. We validated our detection approach by using an Nmap script on the raw data. Our technique found, on average, at least 1.4% of Internet hosts had non-per-flow incrementing IPIDS while Nmap found 1% of globally incrementing IPIDs. We can be 95% sure that these observations where not due to random chance. This is consistent with the fraction of IPv4 computers in China that possessed the IPID side-channel, as of 2014 [9].

For the side-channel research community, we provide a model and methodology for discovery possible side-channels with lower overhead the manually inspecting code. We also developed a tool for researchers to instrument new experiments. For the security community, the tool also is useful for performing network reconnaissance. For the digital rights activist community, the tool is useful for designing experiments that can be used to infer state-level censorship.

8. FUTURE WORK

Moving forward, we plan to devise new techniques for differentiating scope for buffers and timers. There are ethical issues we must consider for these data structures because the most basic techniques used so far lead to disruptions to other users of the computer being measured. We will implement new techniques to detect more specific details about buffers, such as size, and the function used to generate the key a packet gets inserted into the buffer with. We may need to refine our model in order to account for these considerations. We will modify different fields instead of just the source port in Phase 2 of our experiments. We will also explore the modification function we use when modulating the Phase 2 control packets. We are currently devising new ways of measuring fields other than trying to find only incrementing counters. At the beginning of our paper we conjectured that side-channels will likely increase over time, we plan to put this claim to the test by performing longitudential measurements studies. We will measure the average number and diversity of side-channels to determine whether the percentage of computers on the Internet that have side-channels increased over time.

9. ACKNOWLEDGMENTS

We thank Dr. Jedidiah R. Crandall for his insight into modeling side-channels. We thank Dr. Stephanie Forrest for reviewing and guiding portions of this work and for providing help with revisions. We thank Jeffery Knockel for reviewing our work for consistancy and validity in our methodology and experimental design.

10. REFERENCES

[1] G. Alexander and J. R. Crandall. Off-path round trip time measurement via tcp/ip side channels. ACM, 2014.

[2] Anonymous. Towards a comprehensive picture of the Great Firewall's DNS censorship. USENIX, 2014.

[3] Antirez. "new tcp scan method", 1998. [Posted on the bugtraq mailing list on December 18th, 1998.].

[4] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 153–158, New York, NY, USA, 2006. ACM.

[5] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel. Off-path tcp exploits: Global rate limit considered dangerous. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 209–225, Austin, TX, Aug. 2016. USENIX Association.

[6] J. R. Crandall, D. Zinn, M. Byrd, E. Barr, and R. East. ConceptDoppler: A weather tracker for Internet censorship. In *Computer and Communications Security*, pages 352–365. ACM, 2007.

[7] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 605–620, Berkeley, CA, USA, 2013. USENIX Association.

[8] W. Eddy and Verizon. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987 (INFORMATIONAL), Aug. 2007.

[9] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall. Detecting intentional packet drops on the internet via TCP/IP side channels: Extended version. *CoRR*, abs/1312.5739, 2013.

[10] R. Ensafi, J. C. Park, D. Kapur, and J. R. Crandall. Idle port scanning and non-interference analysis of network protocol stacks using model checking. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[11] B. M. et al. An analysis of china's "great cannon". USENIX Association.

[12] A. Filastò and J. Appelbaum. OONI: Open observatory of network interference. In *Free and Open Communications on the Internet*. USENIX, 2012.

[13] Y. Gilad and A. Herzberg. Off-path tcp injection attacks. *ACM Trans. Inf. Syst. Secur.*, 16(4):13:1–13:32, Apr. 2014.

[14] J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20, 1982.

[15] J. Rajahalme, A. Conta, B. Carpenter ,S. Deering. IPv6 Flow Label Specification. RFC 3697 (IPv6 Flow Label Specification), 2004.

[16] J. Knockel and J. R. Crandall. Counting packets sent between arbitrary internet hosts. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, Aug. 2014. USENIX Association.

[17] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[18] N. Brownlee, C. Mills, G. Ruth. Traffic Flow Measurement: Architecture. RFC 2722 (Traffic Flow Measurement: Architecture), 1999.

[19] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 27–40, New York, NY, USA, 2004. ACM.

[20] V. E. Paxson. Measurements and analysis of end-to-end internet dynamics, 1997. Ph.D. Thesis.

[21] J. Postel. USER DATAGRAM PROTOCOL. RFC 792 (USER DATAGRAM PROTOCOL), 1980.

[22] J. Postel. INTERNET CONTROL MESSAGE PROTOCOL. RFC 792 (INTERNET CONTROL MESSAGE PROTOCOL), Sept. 1981.

[23] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), Sept. 1981. Updated by RFCs 1349, 2474, 6864.

[24] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), Sept. 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[25] A. Ramaiah, R. Stewart, and M. Dalal. Improving TCP's Robustness to Blind In-Window Attacks. RFC 5961 (PROPOSED STANDARD), 2010.

[26] J.-P. Verkamp and M. Gupta. Inferring mechanics of web censorship around the world. USENIX.

[27] X. Zhang, J. Knockel, and J. R. Crandall. Original syn: Finding machines hidden behind firewalls. In *2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26 - May 1, 2015*, pages 720–728. IEEE, 2015.