Roya Ensafi

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Jadidiah Crandall , Chairperson

Stefan Savage

Terran Lane

Michalis Faloutsos

Advanced Network Inference Techniques Based on Network Protocol Stack Information Leaks

by

Roya Ensafi

B.E., Computer Engineering, Ferdowsi University of Mashhad, 2006
M.S., Computer Science, University of New Mexico, 2011,

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

December, 2014

Dedication

This dissertation is dedicated to my Mom, Touba Hassan Pour, who supported me nonstop since I can remember. She is my hero. I wish her years and years of happiness to come.

Acknowledgments

First I would like to thank my advisor Jedidiah Crandall. Since the first day I arrived at UNM, he and I have had many creative discussions that helped to develop this research from scratch. I would also like to thank many others for their mentorship and efforts. Many thanks to Stefan Savage, Terran Lane, and Michalis Faloutsos for serving on my dissertation committee. Thanks to Deepak Kapur, Stephanie Forrest, Terran Lane, and Cris Moore for sharing their insights into the nature of computer science. Thanks to Vern Paxson for his amazing publications which have inspired me and given me joy when reading his papers back in Iran.

I would like to thank my parents Mohammad Hossein and Touba, my one and only grandfather, and my siblings for their love and support. Thanks to Roshan Rammohan, Wendy Patterson, Kasra Manavi, and Philipp Winter for being there for me unconditionally. Without your support I would not have been able to deal with the difficult situations that have arisen in my life. Special thanks to Lourdes McKenna, Lynne Jacobsen, and George Luger who have been extremely kind to me throughout my time at UNM.

Thanks to Philipp Winter not only for his contribution on a new SYN backlog idle scan presented in Section 5.4, but also for his collaboration related to Chapter 5. Our collaboration gave me the joy of feeling respected as a researcher and whose collaborations effort is truly appreciated. Also, I would like to thank my co-author Jeffrey Knockel for his contribution of the ARMA modeling in Section 4.4.

Last, but certainly not least, thanks to the mountains, trees, and springs of KelateBala for the humility and wonderment.

Advanced Network Inference Techniques Based on Network Protocol Stack Information Leaks

by

Roya Ensafi

B.E., Computer Engineering, Ferdowsi University of Mashhad, 2006
M.S., Computer Science, University of New Mexico, 2011,

Ph.D. Computer Science, University of New Mexico, 2014

Abstract

Side channels are channels of implicit information flow that can be used to find out information that is not allowed to flow through explicit channels. This thesis focuses on network side channels, where information flow occurs in the TCP/IP network stack implementations of operating systems. I will describe three new types of idle scans: a SYN backlog idle scan, a RST rate-limit idle scan, and a hybrid idle scan. Idle scans are special types of side channels that are designed to help someone performing a network measurement (typically an attacker or a researcher) to infer something about the network that they are not otherwise able to see from their vantage point.

The thesis that this dissertation tests is this: *because modern network stacks have shared resources, there is a wealth of information that can be inferred off-path by both attackers and Internet measurement researchers.* With respect to attackers, no matter how carefully the security model is designed, the non-interference property

is unlikely to hold, *i.e.*, an attacker can easily find side channels of information flow to learn about the network from the perspective of the system remotely. One suggestion is that trust relationships for using resources be made explicit all the way down to IP layer with the goal of dividing resources and removing sharendess to prevent advanced network reconnaissance. With respect to Internet measurement researchers, in this dissertation I show that the information flow is rich enough to test connectivity between two arbitrary hosts on the Internet and even infer in which direction any blocking is occurring.

To explore this thesis, I present three research efforts:

- First, I modeled a typical TCP/IP network stack. The building process for this modeling effort led to the discovery of two new idles scans: a SYN backlog idle scan and a RST rate-limited idle scan. The SYN backlog scan is particularly interesting because it does not require whoever is performing the measurements (*i.e.*, the attacker or researcher) to send any packets to the victim (or target) at all.

- Second, I developed a hybrid idle scan that combines elements of the SYN backlog idle scan with Antirez's original IPID-based idle scan. This scan enables researchers to test whether two arbitrary machines in the world are able to communicate *via* TCP/IP, and, if not, in which direction the communication is being prevented. To test the efficacy of the hybrid idle scan, I tested three different kinds of servers (Tor bridges, Tor directory servers, and normal web servers) both inside and outside China. The results were congruent with published understandings of global Internet censorship, demonstrating that the hybrid idle scan is effective.

- Third, I applied the hybrid idle scan to the difficult problem of characterizing inconsistencies in the Great Firewall of China (GFW), which is the largest

firewall in the world. This effort resolved many open questions about the GFW.

The result of my dissertation work is an effective method for measuring Internet censorship around the world, without requiring any kind of distributed measurement platform or access to any of the machines that connectivity is tested to or from.

Contents

[1]Since this research was performed with the help of collaborators, I have adopted using "we" instead of "I" in Chapters 2 through 6.

List of Figures

List of Tables

Chapter 1

Introduction

A fundamental limitation of most network and Internet measurement techniques is that, in order to receive the answers in response to probes and measure something about the network, it is necessary for whomever is carrying out the measurements to use their own return IP address for all probes sent. Otherwise, the responses to the probes will not be seen by the measurement machine. For attackers (or, penetration testers) this means that they must reveal the origin of their scans and it also means that they can only learn about the network from their own vantage point. For Internet measurement researchers, this means that we can only learn about the Internet from the perspective of virtual private services (VPS) or distributed research environments (*e.g.* DIMES [62], MLab [63], or PlanetLab [77]) or, in the case of Internet censorship studies, informed volunteers that are eager to participate in these studies. In this dissertation, I propose a different approach: to learn something about the network in between A and B, cause A and B to send packets to each other by sending probes with spoofed return IP addresses, and then use side channels in their TCP/IP network stacks to infer something about their communication with each other. Thus, A and B can be virtually any machines on the Internet and need not be under the researchers' control. This is important, because the most important

regions of the Internet to measure regarding Internet freedom issues (*e.g.*, Asia, the Middle East, Africa, Eastern Europe, and South America) are precisely the same regions where VPSes and research infrastructure vantage points such as PlanetLab, MLab, or DIMES are not widely available.

The first main chapter of this dissertation focuses on idle scan attacks. An idle scan is a type of network scan. Network scans can be used to learn valuable information about a network, such as what targets are available and what services they offer. Idle port scanning uses side-channel attacks to bounce scans off of a bystander host to stealthily scan a target IP address or infer IP-based trust relationships between the bystander and the target IP address. Idle scans can be used to hide the origin of a scan, to infer trust relationships and firewall rules, and to reveal hidden networks and hosts. After Antirez proposed the first idle scan in 1998 [12], my dissertation work was the first study to model idle scans, as well as the first to discover new idle scans and apply them to real world problems.

The second and third main chapters of this dissertation focus on applying idle scans for Internet measurement research. This poses new challenges, because while idle scan attacks need not respect the network resources of others, idle scans designed for Internet measurment use by ethical researchers must not fill buffers or otherwise use resources in a way that would deny service to other users. Furthermore, the Internet is extremely noisy: Internet hosts are rarely completely idle and packet loss in intercontinental paths is normal.

At a fundamental level, this thesis is about programming a "weird machine" [19] that was not intended to be programmed. TCP/IP channels are a unique kind of "weird machine" because information flow must be caused to occurr with carefully crafted sequences of packets. My contribution to the field of computer science touches on one of the most basic research questions of computer science: *how to program machines that are challenging to program.* For early machines and today's

novel computing paradigms such as biological or quantum computers, the challenges in programming machines arise from the physical aspects of the machine. For "weird machines" in the context of computer security and network measurement research, the challenge arises because the machines that are being programmed (*e.g.*, TCP/IP network stacks) are not intended to be programmed in the way that my work programs them (*e.g.*, by sending a carefully crafted sequence of packets to perform a measurement and cause the answer to be sent back to the measurement machine).

There are three main research challenges that I address in this thesis: (1) Can a host on the Internet be scanned without sending (spoofed) packets to it? (see Chapter 3); (2) Is it possible to find out whether packets are being dropped between two arbitrary IP addresses without having access to either one of them? (see Chapter 4); (3) To demonstrate the power of our hybrid idle scan: can we detect Internet censorship in China for Tor relays *via* idle scans and identify geographic correlations? (see Chapter 5)

1.1 Dissertation Overview

In Chapter 2, I begin by describing our threat model and provide the technical background of IPID idle scans.

Next, in Chapter 3, I proceed by presenting our results from building a transition system model of a network protocol stack for an attacker, a victim, and a zombie. I describe two new idle scans which resulted from our modeling effort, based on TCP RST rate limiting and SYN backlogs, respectively. Through experimental verification of these attacks, I show that it is possible to scan victims which the attacker is not able to route packets to, meaning that protected networks or ports closed by firewall rules can be scanned [34].

In Chapter 4, I describe a new form of idle scan known as the *hybrid idle scan*, which can be used to remotely detect intentional packet drops on the Internet. That is, given two arbitrary IP addresses on the Internet that meet some simple requirements, our proposed technique can discover packet drops (*e.g.*, due to censorship) between the two remote machines, as well as infer in which direction the packet drops are occurring [33].

After discovering and verifying our novel and practical idle scan, I used it to characterize observed inconsistencies in the Great Firewall of China (GFW). Past work has revealed that the firewall sometimes fails. In other words, sometimes clients in China are able to reach blacklisted servers outside China. This phenomenon has not yet been characterized because it is infeasible to find a large and geographically diverse set of clients in China from which to test connectivity. In Chapter 5, I overcome this challenge using our novel hybrid idle scan technique that is able to measure the connectivity between a remote client and an arbitrary server, neither of which are under the control of the researcher performing measurements [35].

After having presented the main contributions of this thesis, Chapter 6 gives an overview of relevant related work in the fields of network security and Internet measurement. Chapter 7 discusses ethical implications of our experiments.

Finally, Chapter 8 concludes this thesis by discussing our results and future work in the field.

Chapter 2

Background and Context[1]

2.1 Interaction Modeling Efforts

Throughout the process of developing this research, we adapt an interaction model consisting of three hosts: attacker, victim, and zombie [2]. We define a host to be at the edge of the Internet, *i.e.*, an end host. Hosts exhibit internal state, such as a SYN backlog, the IP identification field variable (IPID), and receive buffers for incoming network traffic. Hosts also have ports which can be open, closed, or filtered and their status does not change during measurements. An open port is a TCP port for which the host will accept incoming TCP connections. For UDP, open ports simply drop packets and closed ports respond with ICMP error messages. Filtered ports behave just like a typical host but for simplicity only ports that are either open or closed and never filtered are considered in our model. Hosts reply to packets based on rules that model a typical Linux or FreeBSD network stack. Our model is based on the

[1] Since this research was performed with the help of collaborators, I have adopted using "we" instead of "I" in Chapters 2 through 6.

[2] Antirez [12] who proposed the first type of idle scan used attacker, victim, and zombie that we also use in Chapter 2 and 3. However, we use *measurement machine* as an attacker, *server* as a victim and *client* as a zombie in Chapter 4 and 5.

IP protocol and includes TCP (but only up to the point of half-open connections), ICMP, and UDP.

The SYN backlog of a host is a backlog for half-open TCP connections for which a SYN/ACK has been sent and the host is waiting for an ACK to complete the three-way handshake. The SYN backlog drops duplicate SYNs for the same IP address and port pairs. In our model packets are only removed from the SYN backlog when a TCP RST is received from the source IP address and port of the original SYN packet (because we only model half-open TCP/IP connections, so there is no ACK for the third part of a three-way TCP handshake). When the SYN backlog is full, the host replies with a SYN cookie and drops the SYN. A SYN cookie is a method for sending an initial sequence number in the SYN/ACK that, when ACKed by the remote host, contains enough information to complete the connection so that no state about the half-open connection needs to be kept in memory [16].

The IP identification field (IPID) is a 16-bit value that is assigned to every IP packet created by a host in order to uniquely identify fragments of an original IP packet. Traditionally, this value is sequentially incremented whenever a packet is sent by the host. However, many operating systems changed the implementation of the IPID to hold randomly generated values instead. Although in practice, as shown in [40], many operating systems still employ incrementally increasing IPIDs. An incrementally increasing IPID is required for the IPID idle scan to work which will be explained in the following section.

Figure 2.1 shows the basic definition of an idle scan that we use for our model. The diagram contains four boxes, three of which are the attacker, the zombie, and the victim host. The fourth box describes the absence of a host so all packets to it are dropped. A solid arrow denotes that the source host can send packets to the destination using its own return IP address. A dashed arrow indicates that the source host can send a packet to the destination using any return IP address other than its

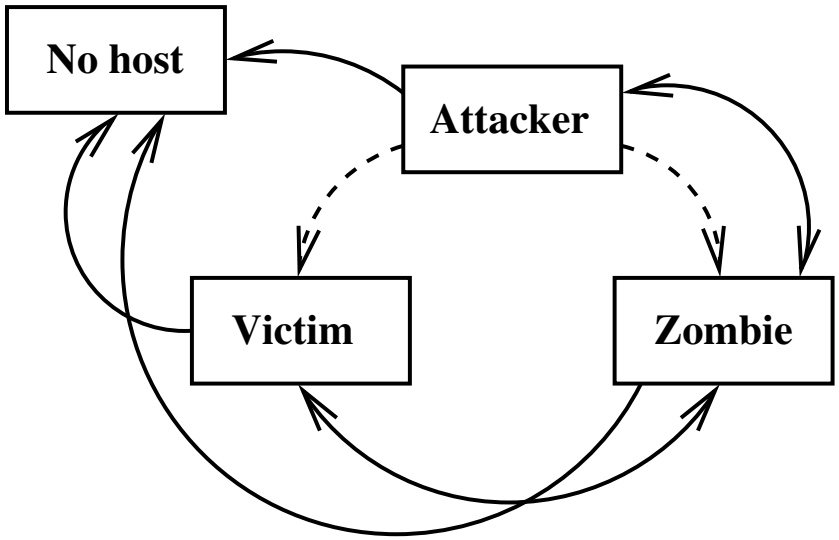


Figure 2.1: Basic definition of an idle scan. Dashed lines represent IP communication with spoofed source IP addresses whereas solid lines represent IP communication with unspoofed IP addresses.

own. The salient feature of this definition of an idle scan is that the attacker cannot send packets to the victim using its own return IP address. This entails that the victim never sends any packets to the attacker, and that the attacker therefore only ever receives packets from the zombie, since the victim and zombie only ever reply to packets using their real IP address as the return address.

Our goal is to ensure that the network satisfies the *non-interference property*, *i.e.*, *an attacker cannot gain any information if this property holds.* A network that satisfies the non-interference property is defined as: for any possible sequence of packets that the attacker can send to the victim and zombie, the sequence of packets the attacker receives in response is identical regardless of whether the victim's port is open or closed. This definition models the desired behavior that the attacker cannot gain any information about the target victim's port.

We reach our goal by implementing two possible scenarios faced by the attacker which the attacker is attempting to distinguish and thus gain information about the

victim. In each scenario, there is a victim and a zombie whose behavior and initial state are identical (except for either the status of the target port on the victim or censorship between victim and zombie, of course), but whose behavior and internal state over time can differ between scenarios through certain sequences of events due to the port status of the target port. The attacker sends identical packets in both scenarios.

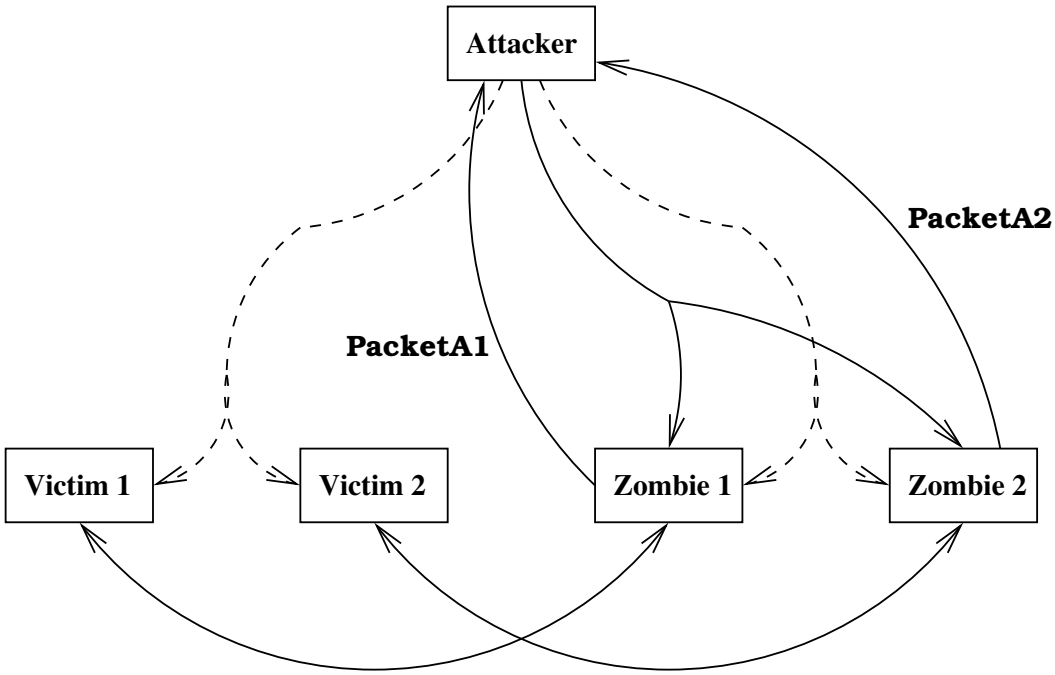


Figure 2.2: Overview of our model (the IP address with no host that drops all packets is excluded from this figure for clarity).

Figure 2.2 gives an overview of our interaction model for testing non-interference properties of network stacks for idle scans. The status of the target port or the existence of censorship between zombie and victim is modeled as two different scenarios. Victim 1 and Zombie 1, for example, exist in scenario 1 where the target port on Victim 1 is open. Victim 2 and Zombie 2 exist in scenario 2 where the target port on Victim 2 is closed. The attacker can forge any arbitrary sequence of packets,

but it must forge identical packets in both scenarios. The hosts in the different scenarios can respond differently and contain different internal state. `PacketA1` and `PacketA2` are the sequence of packets the attacker receives in scenario 1 and scenario 2, respectively.

In our model, the attacker can choose any arbitrary sequence of packets non-deterministically that do not violate the definition of an idle scan. Furthermore, the attacker does not need to reply to packets; the fact that the model allows the attacker to send arbitrary packets covers all possibilities for replies. For the destination and return IP addresses of a packet, the attacker can choose among its own IP address, that of the victim or the zombie, or an IP address with no live host (that simply drops all packets). The only constraint is that the attacker cannot send a packet to the victim with its own IP address as the return IP address as this violates the definition of an idle scan.

The attacker can distinguish between TCP segments employing SYN cookies and segments using regular SYN/ACKs that it receives in our model. This assumption holds in practice because of the statistical properties of the initial sequence numbers of SYN cookies and the fact that SYN cookies are never retransmitted whereas regular SYN/ACK segments are.

The attacker can further choose any value for the IP protocol (TCP, UDP, or ICMP), TCP flags, source and destination ports, validity of checksums, and so on. Every packet that the attacker forges is forwarded to the appropriate host in both scenarios.

In Chapter 3, we demonstrate the existence of two novel idle scans and explore their properties by model checking for a non-interference property over this model. One idle scan is based on a host's SYN backlog and the other on RST or ICMP rate limitations. The latter requires relatively high packet rates or the use of the

frequently filtered ICMP protocol, so for Chapter 4 and Chapter 5 we focus on the SYN backlog idle scan combined with IPID idle scan. We also demonstrate both attacks using real machines and give more technical details about the protocol stacks of various versions of Linux, FreeBSD, and Windows.

2.2 IPID Idle Scans

Similar to virtually all side channel attacks, idle scans are associated with shared, limited resources. Figure 2.3 illustrates an example of the original idle scan discovered by Antirez in 1998 [12] and presented on the bugtraq mailing list. The technique is described in more detail by Lyon [58].

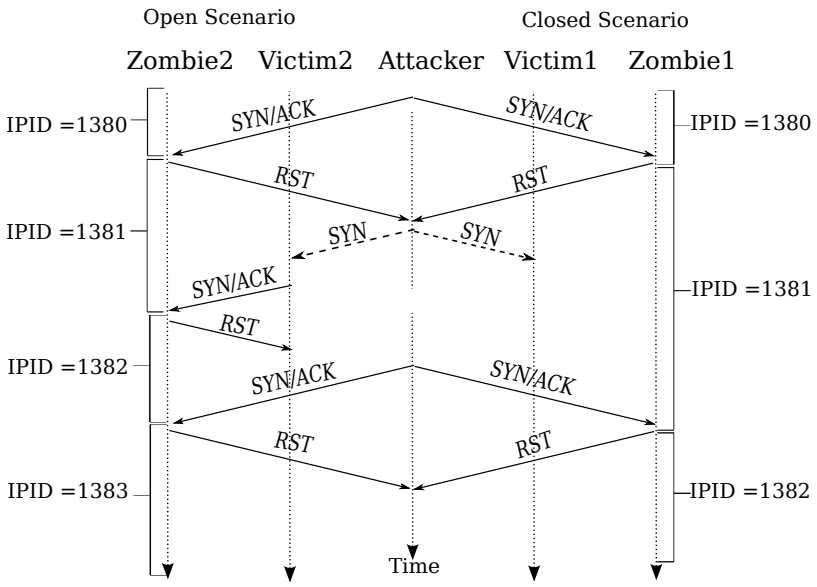


Figure 2.3: The original IPID idle scan discovered by Antirez in 1998. The attacker is able to scan a victim's port without directly communicating with it.

In the original form of the idle scan (shown in Figure 2.3), the attacker queries the zombie for IP packet responses and observes the sequence of IPIDs in the zombie's responses. The attacker then sends one or more SYN packets to the victim on the target port to be scanned with the return IP address pointing to the zombie and the return port pointing to a closed port on the zombie. If the victim replies to the SYN with a SYN/ACK, meaning the victim's port is open, then the zombie will reply to the victim with a TCP reset (RST) and the attacker will observe a discontinuity in the sequence of IPIDs that it receives from the zombie. If the victim's port is closed, the SYN is dropped or replied to by the victim with a RST, which the zombie simply drops and no discontinuity is observed by the attacker. Thus, the attacker is able to infer the port status of the victim without revealing their return IP address to the victim. Furthermore, the attacker is able to infer trust relationships between the victim and the zombie. For example, the attacker might infer that the victim only accepts connections from a particular trusted subnetwork by using a zombie on that subnetwork.

Chapter 3

Idle Scanning and Non-interference Analysis of Network Protocol Stacks

As mentioned in Chapter 1, by scanning the network, the attacker is able to gain valuable information about the hosts that exist and the services they offer, infer IP-based trust relationships between hosts that are enforced by firewall rules and router tables, and collect other information that they can use in the next stage of attack. In this chapter, we show that model checking can be a useful framework for predicting and mitigating attacker capabilities. In idle scans, an attacker scans a victim without sending packets to that victim using its own return IP address. The model of idle scans that we describe in this chapter led to the discovery of two new forms of idle scan.[1] One of these, based on SYN backlog structures that are common to all modern network stacks, gives an attacker capabilities beyond the one

[1]These counterexamples were discovered during the process of deciding what details to include in the model, so resulted from the modeling effort but were not unexpected results from the model checker itself.

previously known form of idle scan in the literature. The other one is based on RST rate limiting that is limit a host to only send a finite number of RST per second. Finally, we demonstrate that if a distinction between trusted and untrusted hosts were made explicit in the lower layers of the network protocol stack, then separate RST rate limitations and a split SYN backlog structure eliminates these attacks in our model of network stacks, which is complex enough to model all of the details of each attack.

3.1 Introduction

The two new forms of idle scan that have resulted from the model checking effort presented in this chapter are based on RST rate limiting and SYN backlogs, respectively. These were discovered during the process of building the model and manifest as counterexamples to a non-interference property that are produced by the model checker. In the RST rate limiting counterexample, the zombie in this case is a FreeBSD machine that limits the number of RST packets that it will send in a given time period. The attacker can infer the port status of the victim by testing the rate at which the zombie will reply with RST packets, the details of this are in Section 3.4.

The SYN backlog counterexample is different from the existing IPID-based idle scan, which is described in Chapter 2.2, in that the attacker never sends (spoofed) packets to the victim. Instead the attacker forges SYN packets from the victim to the zombie, and the zombie sends a SYN/ACK to the victim and places these SYN packets in its SYN backlog (a data structure for holding half-open TCP connections for which a SYN/ACK has been sent but an ACK response has yet to arrive). Because RSTs and ICMP errors from the victim will cause this SYN backlog entry to be removed, the attacker can effectively perform a SYN/ACK scan of the victim without needing the ability to route packets to the victim. The attacker does this

by testing the state of the SYN backlog by sending SYNs with its own return IP address and viewing the SYN/ACK responses. The replies of the victim probes can be inferred from the attacker's ability to get SYN backlog entries for its own SYNs. This makes possible testing for the liveness of IP addresses on protected networks with a rudimentary form of OS detection, and even port scanning on certain types of hosts on a port that is entirely blocked by a firewall. More details are given in Section 3.4.

Like virtually all side-channel attacks, idle scans are associated with shared, limited resources. Because these resources generally cannot be made unlimited, we recommend in light of our results that trust relationships between hosts be made explicit to those hosts all the way down to the IP layer. Currently the only distinction at the TCP and IP layers is subnetworks, which do not necessarily correspond to the IP-based trust relationships between hosts that are enforced by firewall rules and routing tables. Trusted hosts can be hosts protected by the same firewall or that have special trust relationships in the packets they can route to each other. By making a distinction between trusted and untrusted hosts non-interference can be achieved by statically dividing shared resources, effectively eliminating idle scans. We verify non-interference for our model with separate RST rate limitations using symbolic model checking. Then we demonstrate that our split SYN backlog structure using bounded model checking to a depth of 1000 transitions has no violations of non-interference. This means that no practical attack for this counterexamples exists within the constraints of our model.

This chapter is organized as follows. Our model is described in Section 3.3, followed by a description of the counterexamples discovered during the process of building the model and some experimental results from their implementation in Section 3.4. We demonstrate that non-interference is achievable by distinguishing between trusted and untrusted hosts in Section 3.4.3. This is followed by discussion

and future work in Section 3.5. Chapter 6 provides more background and related works.

3.2 Background

Non-interference [41] is a widely used concept of information flow security that has seen wide application for proving security properties of programs. The works that are most related to ours in this space are those that treat non-interference as two or more separate scenarios that must produce the same result from the attacker's view for non-interference to be demonstrated, *e.g.*, TightLip [112] or the work of McCamant and Ernst [66, 67]. We apply non-interference to network stacks in this chapter. Non-interference proved to be a very fruitful model of information flow in this context, but for future work that might consider packet loss, packet delay, and other such factors, alternatives such as non-deducibility [94] may be necessary. For the modeling effort presented in this work, which is based on an abstracted model of real networks that does not include packet loss and delay, non-interference proved to be a very useful property because it can be specified with Linear Temporal Logic (LTL). Treating the problem as a covert channel problem and studying object storage [49] and timing channels [108] is an attractive approach, but covert channel models assume collusion of the sender and receiver of information and do not capture in their models the sequences of events necessary to describe idle scans in a natural way.

The model checker that we chose for our study is the Symbolic Analysis Laboratory (SAL) [15]. SAL provides a SAT-based bounded model checker that allows for counterexamples to be easily interpreted as a trace through the states of the model, or, in our case specifically, a sequence of packets. SAL also provides a BDD-based symbolic model checker. Model checking has been applied to many properties of

network protocols and their implementations where specific bugs lead to security vulnerabilities or availability issues, (*e.g.*, [31, 81, 43]). We have particularly patterned our analysis following Rushby's tutorials for modeling the Needham-Schroeder protocol [83] to identify Lowe's bug and the fault-tolerant algorithm for maintaining interactive consistency (Byzantine agreement) [84] as the transition systems for these problems seem similar to the ones for modeling port scanning and side-channel attacks in a protocol stack. Our results demonstrate that model checking is also useful for studying information flow on networks, particularly in this chapter within the context of idle port scans.

3.3 Formalizing Non-interference Analysis of Idle Scans

After establishing our network stack model that is described in Section 2.1 , in this section we describe more details and its implementation in SAL, and finally we list simplifying assumptions of the model.

3.3.1 SAL for Modeling, Counterexamples, and Verification

We model the network stack as a transition system. At an informal, high level, a transition system specifies computation as a sequence of transitions in a state machine. A state is given by the values of the local variables used to describe transitions. A transition system has an initial state. For every transition, there is an optional guard, which when true in the current state, leads the computation from the current state to the next state. For a nondeterministic transition system, multiple transitions may be triggered and one transition is randomly selected. This is repeated and the computation terminates if no guard is true in the current state.

Dijkstra's guarded command language [27] is an example of a formalism for specifying transitions.

We used SAL (Symbolic Analysis Laboratory) for specifying the transition system and analyzing its properties. SAL is a language and a tool kit for specifying transition systems and analyzing them using model checking. SAL provides support for a suite of tools which have been successfully used for analyzing protocols and distributed algorithms (see [86]).

Figure 3.1 shows the outline of our SAL code for the model. Ellipses indicate where detailed code has been omitted, the full model is 895 lines of SAL code.

For a transition step, a nondeterministic choice is made between the attacker, victim, or zombie. If the attacker is chosen, it forges a nondeterministic packet, which can be a "drop" packet that has no effect. This packet is placed in the receive queue of the destination IP address. If the victim or zombie is chosen, it removes the next packet from its FIFO receive queue and replies based on its internal state and configuration. The functions `ProperReply` and `UpdateSynbacklog` are responsible for choosing the packet to reply with, if any, and any updates to the host's internal state (specifically the RST counter and SYN backlog). Note that these are pure functions and do not update any state themselves.

Figures 3.2 and 3.3 show how the `ProperReply` and `UpdateSynbacklog` functions are used. A transition has a guard, *e.g.*, "`(z1.fullness /= 0 AND z2.fullness /= 0 ) -->`", which is a quantifier-free formula specifying a condition on the current state and must hold before the transition is executed, and then a formula relating the current state with the next state. An example of such a formula is "`z1'.fullness = z1.fullness - 1`", where `z1'` is the variable in the next state and `z1` is the variable in the current state. In this example the variable will be decremented by 1 in the next state.

When the guard on a transition for the zombie or victim fires, that host must remove a packet from its queue and then reply and update its state in both scenarios. `UpdateSynbacklog` returns not only the new state of the SYN backlog, but also a variable called `.synPIsThere` which can take on the values `put`, `notexist`, or `exist`. This return value is passed to `ProperReply`, which needs to know if a SYN packet was put in an entry in the SYN backlog, no entry was found for it because the SYN backlog is full, or it already existed in the SYN backlog. In this way `ProperReply` knows whether to send a SYN/ACK, send a SYN cookie, or drop the packet, respectively, if the packet is a SYN.

For example, if the zombie in one of the scenarios receives a SYN packet, it calls `UpdateSynbacklog` to determine the new state of the SYN backlog and what will happen to the packet. If the internal state of the zombie indicates that there is a free entry in the SYN backlog, the fact that the SYN will be placed in the SYN backlog and the new status of the SYN backlog are returned by this function. Then `ProperReply` is called with this information as an argument, and this function will determine that the proper reply is a normal SYN/ACK, with the destination IP address as the source of the SYN, valid checksums, *etc.*

Another example is that a host (a victim or zombie) receives a SYN/ACK. `UpdateSynbacklog` returns the current state (*i.e.*, no changes will be made to the SYN backlog state) and then `ProperReply` will be called and will ignore `.synPIsThere` because the packet is not a SYN. The return value of `ProperReply` depends on the RST counter. If the RST counter is non-zero the return value of `ProperReply` will be a RST packet that the zombie will use for a reply and a reduced RST counter. If the RST counter is already zero, `ProperReply` will return a drop packet and zero still for the RST counter. All possible TCP, UDP, or ICMP packets and their corresponding replies are enumerated in `ProperReply` based on the reply that a typical network stack would send.

By forcing the zombies or the victims in both scenarios to reply at the same time step, the model stays in sequence. If a host in one scenario (*e.g.*, the victim in the closed port scenario) replies to a packet whereas the corresponding host in the other scenario (*e.g.*, the victim in the open port scenario) drops the packet, a "drop" entry is inserted in the destination host's queue as a filler and eventually ignored. This ensures that the packets that are received by the attacker vary only when non-interference is violated, *i.e.*, only when the sequence diverges.

SAL supports a suite of tools; the ones most relevant for the analysis discussed in this chapter include a deadlock checker, a symbolic model checker for finite state systems based on the CUDD BDD package, and a bounded model checker based on the Yices SAT solver. Properties of a transition system are specified in Linear Temporal Logic (LTL). Our analysis involved using properties of the form $G(\alpha)$, where α is a quantifier-free, modality free formula expressed using state variables, to mean that α holds in every state of the transition system. The non-interference property is specified as:

$$\vdash \mathrm{G}(\mathrm{PacketA1} = \mathrm{PacketA2})$$

This means that the sequence of packets the attacker receives in response to its probes from the zombie in the first scenario is always identical to the response from the zombie in the second scenario.

We have used SAL's bounded model checker for finding counterexamples as it is depth-first and explicitly enumerates states. SAL's symbolic model checker, which is exhaustive, is useful for finding smaller counterexamples as well as for proving properties of interest, which are often difficult to do by explicit state enumeration model checkers. A useful comparative study of exhaustive symbolic model checkers

and explicit state enumeration model checkers is in [22] for protocol analysis and controllers.

3.3.2 Assumptions to Reduce the Number of Model States

A number of assumptions were made to keep our model simple. Our strategy was to start with a simple model and introduce additional complexity into the model if no counterexamples are generated, and ensure that the abstractions we made caused no loss of generality that would exclude potential counterexamples.

A major abstraction in the model considers the proper reply to SYN/ACK packets to be "drop" for open ports and RST for closed ports. In reality, network stacks that respond differently to SYN/ACKs on open *vs.* closed/filtered ports typically respond with RSTs or ICMP and have different rate limits per port. Since the lower rate limit (typically ICMP) will cause drops before the higher rate limit, without loss of generality, we can consider open ports to simply drop SYN/ACK packets from the initial state. This is equivalent to assuming that the attacker immediately exhausts the lower rate limit.

We also exclude ICMP and UDP from the split SYN backlog version of our model. Since ICMP host error packets have the same effect on the SYN backlog as RSTs, and other ICMP and UDP packets make no relevant changes to the destination host's TCP state, ICMP and UDP do not affect the non-interference property for the SYN backlog structure. Invalid checksums in packet headers are also excluded, because they are dropped without affecting the state of the destination host in all cases.

Another major abstraction is that each of the two buffers in our split SYN backlog has only a single entry. There are three reasons why only a single entry in the SYN backlog is necessary in the model:

- Pending entries in the SYN backlog with source IP addresses and ports (possibly spoofed by the attacker) that correspond to invariant ports (that have the same status in both scenarios) cannot cause divergence in the internal state of any of the hosts in the two scenarios. Thus, no more than one such SYN backlog entry at a time can be useful for creating a counterexample.

- Even though RST rate limiting is performed separately for open and closed ports, the rate limit value stored by any host cannot be caused to diverge on invariant ports. Only the target port on the victim can cause divergence. Since only one such port exists, only one SYN backlog entry at a time can be useful for creating a counterexample. If we had not received a counterexample under this assumption, we would have incrementally allowed more entries in the SYN backlog.

- While the single entry is full, the SYN cookies generated in response to dropped SYN packets can only cause internal state differences if sent to the target port on the victim. If this is the case then the entry in the SYN backlog cannot also be a SYN packet with the victim IP address and target port, since duplicate SYNs are ignored by the SYN backlog. Thus, one SYN backlog entry per trust level (trusted and untrusted) is general enough to handle all of the cases of any number of SYN backlog entries.

Because of the above simplification of making the SYN backlog have a single entry for each trust level, we modeled only three ports without loss of generality. Port 1 is prohibited (*e.g.* by a firewall rule) to the attacker for the split SYN backlog implementation, meaning that the attacker cannot send packets to port 1. This was done so that we could include the RST rate limitation, which has important interactions with the SYN backlog, without receiving the RST rate limit counterexample. Port 1 is closed in both scenarios for the zombie; however, for the victim, it is open in one scenario and closed in the other. In other words, port 1 is the target port for

the attacker to get information. Port 2 is closed and port 3 is open on both hosts in both scenarios. Because closed ports are equivalent in terms of their responses, a single closed port per host is equivalent to any number of closed ports. Because the SYN backlog has a single entry and open ports only have different behaviors based on the status of the SYN backlog, a single open port per host is also equivalent to any number of open ports.

In real SYN backlog implementations, there is a timeout after which SYNs that have not become fully open TCP connections are dropped. Because our model allows the attacker to remove any entry from the SYN backlog at any time via a RST packet (which is also possible in reality for Linux SYN backlog implementations), our model need not incorporate this timeout. Also, RST rate limiting is done per a time period in reality. A fixed limit of RSTs for an unbounded amount of time is a generalization of this that does not exclude any counterexamples because for any violation of non-interference based on a rate limit a single time period is enough to create a counterexample.

3.4 Finding and Ameliorating Idle Scans

In this section, we describe the counterexamples that our modeling effort produced and give experimental results of an implementation of these counterexamples to demonstrate that they can indeed be used to do idle port scans.

3.4.1 Discovering Counterexamples

We now describe the two counterexamples that were discovered during the process of developing the model.

Port	Zombie status	Victim Status
1	Open	Open in scenario 1, closed in scenario 2
2	Closed	Closed
3	Open	Open

Table 3.1: Ports and their status in our model.

RST Rate Limiting Counterexample

When we applied SAL's bounded model checker to a simpler model in which the SYN backlog did not play any role, for the property "$\vdash$ G(PacketA1 = PacketA2)" SAL identified a counterexample with `RST_counter` set to 3 in the initial state. We simplified the model further by reducing the initial value of `RST_counter` to 1 and still received a counterexample. The counterexample in this case was found much more quickly, at depth 5 in the transition system.

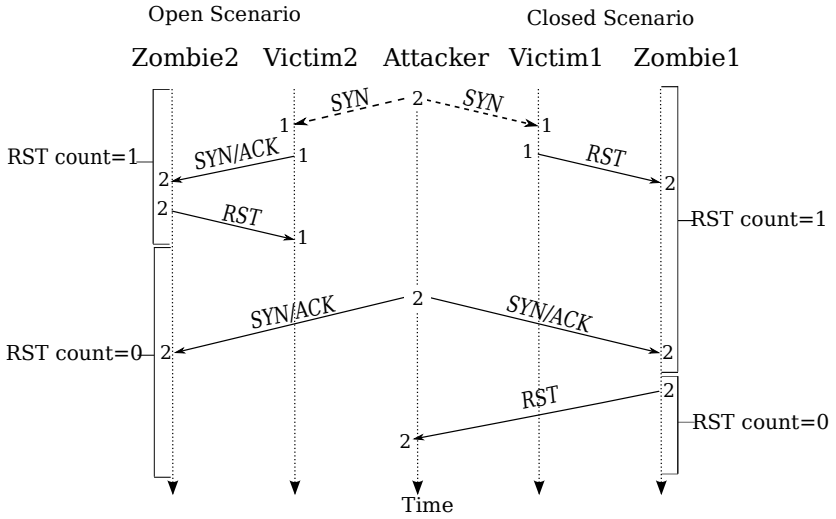


Figure 3.4: RST rate limiting counterexample.

This counterexample is illustrated in Figure 3.4. The figure shows the sequence of packets for the open *vs.* closed scenarios that that attacker can send to distinguish between the scenarios. Note that the attacker sends the same sequence of packets in

both cases. Dashed lines are spoofed packets (for Figure 3.4 the packets are spoofed so that they appear to come from the zombie). The numbers at the bases and heads of the arrows represent the source or destination port number, respectively. The RST count state for a period of time is the state of that variable for each scenario.

In this example the attacker wants to discern the port status (open or closed) of port 1 on the victim. Port 2 on the zombie is closed. The packets the attacker sends are identical in both scenarios. The attacker cannot see the packets that are sent between the victim and zombie or zombie and victim. The port status must be inferred by the difference in the expected packet sequence that the attacker will see between the two scenarios. First, the attacker forges a SYN to the victim on the target port that appears to be from the zombie with return port 2. If the target port on the victim is open, it will respond to the zombie with a SYN/ACK on the zombie's closed port 2, causing the zombie to send the victim a RST and decrement its RST count. If the port is closed, the victim will respond to the zombie with a RST which the zombie ignores. Next, the attacker sends a SYN/ACK packet, using its own return IP address, to the closed port on the zombie. If the attacker receives a RST in response, then it can infer that the victim target port status is closed since an open port would have caused the zombie to have already reached its RST rate limit.

SYN backlog Counterexample

For the second case, we tried a more complex model that included a SYN backlog. We started with a SYN backlog of size 2, then simplified it further to size 1, and SAL's bounded model checker still identified the counterexample to the non-interference property as illustrated in Figure 3.5.

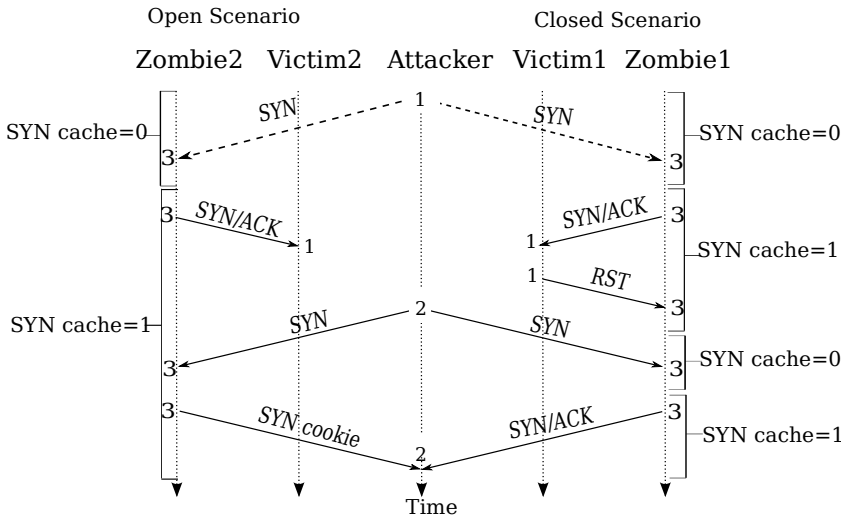


Figure 3.5: SYN backlog counterexample.

The relevant state in this case is the number of pending SYN/ACK entries in the SYN backlog, with a maximum value of 1 in our model. The notable thing about this form of idle scan is that the attacker never sends any packets to the victim, not even packets with spoofed return IP addresses. Instead, the attacker sends a SYN to the zombie on an open port, with the return IP address of the victim and the return port as the target port. The zombie places this SYN packet in the SYN backlog, which in our model has only a single entry, and sends a SYN/ACK response to the victim. If the victim target port is closed it will send a RST in response, which causes the zombie to remove the relevant SYN backlog entry so that there is now a free entry in the SYN backlog. An open target port on the victim will simply drop the SYN/ACK from the zombie, so that the SYN backlog of the zombie remains full since the zombie is still waiting for a response to the SYN/ACK. The attacker can then infer the status of the zombie's SYN backlog, and therefore the victim port status, by sending a SYN to the zombie with the attacker's own return IP address. A regular SYN/ACK means the SYN backlog entry was free, a SYN cookie indicates

that it was full.

Note that responses to SYN/ACKs on open, closed, or filtered ports vary for different operating systems, but all that matters is that for open *vs.* closed or open *vs.* filtered the response differs in some way under certain conditions. More discussion of the possibilities for this is in Section 3.4.2. The SYN backlog counterexample makes it possible to, *e.g.*, port scan a network on a port that is blocked for the entire network from outside the firewall. Imagine in Figure 3.5 that the zombie and victim are behind a firewall and the attacker is outside the firewall. Even if the firewall drops all incoming packets with destination port, *e.g.*, 22 for Secure Shell (SSH), the attacker can scan port 22 on the network by using other open ports. Also, there may be firewall rules that enforce that only trusted machines (*e.g.*, the zombie) can route packets to the victim. In this case the victim might be an internal database server and the zombie is the web server interface to the database, for example. Information about what ports the victim has open might give the attacker an idea of whether compromising the zombie to subsequently get access to the victim is worth the effort and risk. It might also be that the attacker can route packets to the victim, but not on the target port. For example, many machines leave certain ports open only for their backup servers that contact them nightly. Or, the system administrator might only allow incoming SSH sessions on their critical servers from their own office machines and not from other IP addresses. Knowing these kinds of trust relationships and exploiting them to find out more about the victim machines can be very valuable to an attacker.

For each host, both the SYN backlog and the reset rate limiting variables constitute shared, limited resources, which are the sources of violations of non-interference in our two counterexamples.

3.4.2 Experimental Confirmation of Counterexamples

We implemented both counterexamples to verify that these two new forms of idle scan that resulted from the modeling effort were possible for real hosts. Our results presented in this section demonstrate that the differences in the sequence of packets the attacker sees translate from the abstract notion of non-interference in our model to differences that can be seen in real network packet traces. Our implementations of the two idle scans are not optimized for speed or stealth, nor do they account for packet loss or packet delay, but in this section we discuss the practicality of these two forms of idle scan and conclude that they are both practical.

Experimental Setup

For our experiments, we set up VirtualBox [5] virtual machines connected using IPv4 on two different subnetworks with TUN/TAP interfaces. The attacker machine was the host, and one subnet contained a Linux kernel 2.4 host (Fedora Core 1) which served as the zombie for the SYN backlog idle scan implementation. The other subnet contained a Windows XP host with no service packs, a Linux kernel 2.6 host (CentOS 5.2), and a FreeBSD 7.1.1 host. The latter served as the zombie for RST rate limiting idle scan implementation. IP forwarding between these two subnetworks was performed by the host. Packets were generated and captured by separate threads using the Perl Net::RawIP and Net::Pcap libraries, respectively.

RST Rate Limiting Idle Scan Implementation

In our transition system model, RSTs are limited to a finite number for infinite time. For a real FreeBSD system, RSTs are limited to a default of 200 per second, with separate limitations for open and closed ports. Our implementation sends 2000 each

Port status	Mean	Std. dev.	Min.	Max
Open	1552.1	47.0	1429	1634
Closed	2000	0	2000	2000

Table 3.2: Results from RST rate limiting idle scan implementation.

of two different types of packets, each at a rate of 180 per second, to the victim and FreeBSD zombie, respectively. One type of packet is spoofed SYNs to the victim on the target port that appear to be from the zombie on a port that is closed on the zombie. The other is SYN/ACKs from the attacker to the zombie, which the zombie should reply to with RSTs. If the zombie is sending RSTs at a rate of 180 per second to the victim in response to the victim's SYN/ACKs (meaning the victim target port is open), this should interfere with the rate at which the zombie sends RSTs to the attacker. Thus the number of RSTs the attacker receives in our experiment can be used to infer the port status of the target port on the victim. We repeated the RST rate limiting idle scan experiment 700 times each for an open and closed port on the victim. The victim was a Linux kernel 2.6 virtual machine. The host-based firewalls on both machines were disabled, although for the victim the idle scan works whether the host-based firewall is enabled or not. For FreeBSD, RST rate limiting does not apply to filtered ports. The pf host-based firewall is disabled by default for FreeBSD installations.

The results from our RST rate limiting idle scan are shown in Table 3.2, where the results are based on the number of RSTs the attacker receives. When the victim port is closed, the attacker receives all 2000 RST responses from the zombie. When the victim port is open, the attacker receives at most 1634 RSTs. Thus, determining if the target port is open or closed is straightforward for idle scans based on RST rate limiting.

SYN Backlog Idle Scan Implementation

SYN backlog implementations vary for different operating systems[2]. While the SYN backlog idle scan is possible using virtually any host as a zombie, the simplest network stack to use as a zombie is Linux kernel 2.4. Linux kernel 2.4 uses a simple buffer for the SYN backlog, with between 128 and 1024 entries depending on the memory available on the system. Our Linux kernel 2.4 virtual machine zombie had a SYN backlog size of 128, but Linux enforces a rule that only three fourths of the SYN backlog can contain SYN packets from hosts that have not demonstrated their liveness in the recent past by completing a fully open TCP connection. This effectively reduces the SYN backlog size to 97. We did not enable SYN cookies, which are disabled by default in Linux. The attack works basically the same whether or not SYN cookies are enabled. We ran two separate sets of experiments for the SYN backlog idle scan implementation, one to demonstrate that it is possible for the attacker to detect the presence of live machines and perform a rudimentary form of operating system detection, and another to demonstrate that under certain circumstances the attacker can infer the port status of a target port on a particular victim IP address. For all experiments, 100 data points were generated for both open and closed port scenarios.

For checking for liveness, we scanned four different IP addresses. One is a default FreeBSD 7.1.1 machine (with the pf host-based firewall disabled, as is the default), another is a Windows XP machine with no service packs (with Windows firewall disabled, as is the default), and a third is a Linux kernel 2.6 machine (CentOS 5.2, with iptables enabled, as is the default). The fourth IP address has no live host so all packets are simply dropped. Forging packets from random return IP addresses on these victims is very likely to send SYN/ACKs to closed or filtered ports, so we

[2]In this chapter, our analysis was performed for Red Hat based distributions circa 2010, so may not generalize to entire Linux kernel branches.

choose random return ports for all spoofed SYNs where the attacker uses the victim as the return IP address. Varying this return port number is important because if the return port is not different then the spoofed SYNs will have the same IP addresses and ports for both the destination and source and the SYN backlog will drop such duplicates. Both RSTs and ICMP errors cause their corresponding entries to be removed from the SYN backlog when received by the zombie.

Because Linux responds to SYN/ACKs on filtered ports at a very low rate (about 10 per second) with ICMP host prohibited packets, FreeBSD responds to SYN/ACKs on closed ports at a rate of at most 200 per second, Windows responds on closed ports with RSTs at an unlimited rate—and IP addresses without live hosts simply cause SYN/ACKs to be dropped—it is possible for the attacker to idle scan a subnetwork and infer something about the operating systems that the live hosts discovered have installed. To scan a single IP address, our implementation sends 50 spoofed SYNs (that appear to be from the victim), then 50 each of spoofed SYNs and SYNs where the attacker uses their own return IP address, and then 200 more spoofed SYNs, all at a rate of 1000 per second. It then sends 200 each of spoofed SYNs and SYNs where the attacker uses their own return IP address at a rate of 400 per second. The number of SYNs where the attacker uses their own return IP address and receives a SYN/ACK response can then be used to infer the liveness and operating system of the IP address. The results from this experiment are shown in Table 3.3, where the results are based on the number of SYN/ACKs the attacker receives (note that for Linux kernel 2.4 network stacks SYN/ACKs are retransmitted five times until they time out after 190 seconds).

Under certain circumstances, it is also possible to port scan specific ports on a particular IP address using a SYN backlog-based idle scan. Specifically, if the response or rates differ for open *vs.* closed or filtered ports on the victim then scanning a target port is possible. Examples of this are FreeBSD with the pf host-

Host	Mean	Std. dev.	Min.	Max
Not live	109.1	7.4	96	123
Linux 2.6	126.9	6.3	111	138
FreeBSD	300	0	300	300
Windows XP	460.3	11.9	435	477

Table 3.3: Results from SYN backlog idle scan implementation for liveness and operating system.

based firewall disabled, where open ports and closed ports are rate-limited separately, or Linux hosts with the iptables host-based firewall enabled and an open port that does not use the stateful module of iptables.

To test the FreeBSD example, we developed a SYN backlog-based idle scan that simultaneously sends 20000 spoofed SYN packets (with random return ports that are closed on the zombie) as quickly as possible while sending, at half the rate, alternating spoofed SYNs with the target port on the victim as the source port and valid SYNs with the return address of the attacker. Because closed ports on the victim are rate limited due to the spoofed SYNs with random return ports coming from the zombie, the spoofed SYNs with the target port on the victim as their return port will quickly fill the SYN backlog if the target port is also closed and cause fewer entries to be free for non-spoofed attacker SYNs, therefore causing the attacker to see fewer SYN/ACKs in response. If the target port is open, the open port sends more RSTs before rate limiting begins meaning that more SYN backlog entries remain free and the attacker sees more SYN/ACKs. The results of this experiment are shown in Table 3.4, where the results are based on the number of SYN/ACKs the attacker receives. Some data points for both closed and open ports were thrown out due to failures of the Python pcap library at high packet rates. Packet loss due to the high rates could only make the distributions more similar, not less, because more packets are sent over the TUN/TAP interface for the open port scenario. Thus, the distributions for open and closed ports are clearly different. A two-sampled,

Port status	Mean	Std. dev.	Min.	Max
Open	262.1	41.8	120	447
Closed	218.0	39.3	68	318

Table 3.4: Results from SYN backlog idle scan implementation for port scanning FreeBSD.

Port status	Mean	Std. dev.	Min.	Max
Open	482.4	3.3	474	489
Closed	427.8	3.4	417	435

Table 3.5: Results from SYN backlog idle scan implementation for port scanning Linux.

unpooled t-test (which assumes neither known variances nor equal variances) for these two sets of data gives a t score of 7.71 with 197 degrees of freedom, which corresponds with a p-score of 0.999999999999696 meaning that a null hypothesis that the two distributions have an equal mean is rejected with very high confidence.

For port scanning Linux-based victims, the idle scan first sends 96 filler SYNs to fill all but one entry in the SYN backlog. SYN/ACK replies to filler SYNs are not counted in the results. Then it alternates, at an overall rate of 100 packets per second, spoofed SYNs with the return IP address of the victim and return port of the target port, filler SYNs, and probe SYNs. Table 3.5 shows the results of these experiments, where the results reflect the number of SYN/ACK responses to probe SYNs.

Stealth and Efficiency

Our idle scan implementations in this section are intended to show that the abstract counterexamples that resulted from our modeling effort were real divergences in real network stacks that could be exploited by the attacker for idle scans. Since the diver-

gences are based on rates in real network stacks we used hypothesis testing to show this. We only report a t-score and p-score for one set of experiments (the SYN backlog idle scan implementation for port scanning FreeBSD) because the distributions of the results for other experiments were so different that their high t-scores led to p-scores that were within floating point rounding error of 1.0. Our implementations of these idle scans were designed for this hypothesis testing and therefore are not optimized for attacker stealth or efficiency in carrying out the scan. For assessing the practicality of these idle scan techniques, we will now comment on stealth and efficiency.

For the RST rate limiting idle scan, the attacker cannot perform the idle scan without sending more than 200 SYN/ACKs to the zombie either directly or indirectly. However, the attacker need not send SYNs to the victim (spoofed from the zombie) at half this rate. It is possible to, *e.g.*, send SYNs to the victim at a rate of 20 per second and send SYN/ACKs (or any packet that will elicit a RST) to the zombie at 195 per second. Theoretically, the mutual information between the victim port status and the sequence of packets the attacker sees is non-zero even if the attacker sends only a single spoofed SYN to the victim, and even when packet loss is accounted for. Thus the attacker has a fair amount of flexibility in terms of trading off speed of the scan *vs.* stealth for packets it sends to the victim. Furthermore, sending SYNs simultaneously to multiple victims and multiple ports and measuring the zombie responses in the aggregate can increase the efficiency of the scan if the distribution of expected closed *vs.* open victim ports diverges from an equal distribution. To see this, consider the extreme case where a large subnetwork has only a single host with an open port, something similar to a binary search could greatly reduce the amount of time necessary for the scan in this case.

For the SYN backlog idle scans, which are more powerful in terms of the new capabilities they offer attackers beyond the currently known idle scan technique,

there is a wider range of efficiency and stealth tradeoffs that the attacker can make. Furthermore, unlike ICMP IPID- or RST rate limit-based idle scans, virtually any modern network stack that offers any type of protection against SYN flooding can be used as a zombie. We chose to use a low-memory Linux kernel 2.4-based zombie for our experiments due to its simplicity and small SYN backlog size, but larger SYN backlog sizes or more complex SYN backlog implementations are also easily exploited for SYN backlog idle scans. The SYN backlog only needs to be almost full for SYN backlog idle scans to work, and SYNs for half-open connections take 190 seconds to timeout in Linux by default. So even for high-memory Linux 2.4 machines with 1024 SYN backlog entries (of which 769 are used, compared to 97 for 128-entry SYN backlogs), the rate necessary to create the conditions for an idle scan only increases from 0.5 SYNs per second from the attacker to the zombie to about 4.1 SYNs per second (these rates keep the buffer almost full despite the timeouts). Once these conditions are created, the attacker effectively can do a SYN/ACK scan of the victim host or network at the cost of two packets sent per SYN/ACK query and three more generated as responses. It also does not matter whether or not the zombie implements SYN cookies, since SYN cookies are never retransmitted (compared to typically three to five retransmissions of regular SYN/ACKs for various zombie configurations) and also have easily identifiable statistical anomalies in their initial sequence numbers.

Some SYN backlog implementations that are not simple buffers like Linux 2.4 may make SYN-backlog idle scans slightly more difficult, but still possible and relatively efficient. For example, the FreeBSD SYN cache implementation [55] uses a SYN backlog with 512 buckets that each have 30 entries and are chosen uniformly at random using a hash of the IP address/port pairs and a host-generated secret. This mechanism is designed to stop denial-of-service, not idle scans. It creates some equivocations that can reduce the amount of information flow the attacker can exploit for idle scans but the attacker can still perform the scan relatively efficiently even with FreeBSD zombies. We did not explore the SYN backlog implementations of

Linux kernel 2.6 or Windows hosts as part of this work. All modern network stacks must have some form of SYN backlog for reliability purposes and a limit on this resource to prevent denial-of-service. Thus, only by making this resource non-shared is non-interference to prevent idle scans possible, and the current OSI network stack model with TCP/IP does not make the necessary trust distinctions to split the SYN backlog. Thus, virtually every end host machine that the attacker can route to at least one open port on is a potential zombie.

The rate at which the attacker must send packets to the zombie for a SYN backlog idle scan, and therefore the stealth of the scan, depends on the attacker's goals. If the zombie is a Linux kernel 2.4 machine and the attacker wants only to check the liveness of a range of IP addresses on the victim network, then between 0.5 and 4.1 packets per second plus the probes themselves is sufficient. Note that, in terms of stealth, it is also relevant that the attacker need not send any packets to the victim for this form of idle scan, not even packets with spoofed return addresses.

For detecting the operating system of a victim host or scanning individual ports on the victim, higher rates are necessary. Detecting a Linux machine on the victim network and port scanning it can easily be done at between 10 and 20 packets per second. We also discovered during our experiments that, at least for Linux kernel 2.4 hosts, it is easy for the attacker to not only remove their own packets from the SYN backlog manually, but any packet that they have spoofed, using spoofed RSTs. This is because only the IP address and port pairs are checked, the sequence and acknowledgment numbers for RSTs are ignored when deciding whether to drop an entry from the SYN backlog on the zombie. Thus, the attacker has a high degree of control over the SYN backlog status of the victim. Packet delay, packet loss, and interference from other machines that contact the zombie can easily be accounted for in this way, and the aggregate effect of scanning multiple victims at a time mentioned above also applies to SYN backlog idle scans. One possible avenue for future work

would be to model the capabilities of this attack as a Markov Decision Process and discern tight bounds on numbers of packets and rates needed for different attacker goals.

In terms of the practicality of our attacks, the ability to scan firewalled ports and discover machines on protected networks that the attacker cannot route packets to certainly underscores the need for good ingress filtering and DMZ management. Our attacks are applicable in all three of the following scenarios: when the victim and zombie are on the same subnet and communicate using ARP[1], when the victim and zombie are within the same network domain but on different subnets, and when the victim and zombie are geographically separated by some distance on the Internet. The attacker can be inside or outside the network domain of the zombie and victim. Thus, many possibilities for network inference arise. For example, the attacker can infer when a host opens ports only to other particular machines, such as a backup server or network administrator. While many network configurations prevent IP address spoofing, which is essential for both attacks described in this chapter, idle scans are a very general technique that can apply in a variety of scenarios.

3.4.3 Ensuring Non-interference Using SAL Model Checker

Based on our experimental results from implementing the two counterexamples as idle scan attacks, it is apparent that RST rate limiting and the SYN backlog interact in complex ways and cannot be considered separately. Thus we chose to leave RST rate limiting in the model for verifying non-interference of the split SYN backlog.

It is well-known that verifying properties using a model checker is much more difficult than finding a counterexample. We abstracted the model down to the simplest form that produces both counterexamples, and attempted to prove the non-interference property for cases where the shared, limited resources were split based on

trust relationships and therefore no longer shared. The zombie and victim consider each other trusted and the attacker untrusted. For the RST rate limiting counterexample, the hosts have separate `RST_count` RST counters for trusted *vs.* untrusted hosts, and the SYN backlog is removed. For the SYN backlog counterexample, we implemented a split SYN backlog structure with separate SYN backlog buffers for trusted *vs.* untrusted hosts.

In the first case, we removed the SYN backlog and focused only on the RST rate limitation counter example. Since symbolic model checkers are known to be better for verifying properties in contrast to explicit state enumeration based bounded-model checkers, we used SAL's symbolic model checker. It verified the property that:

$$\vdash \mathrm{G}(\mathrm{PacketA1} = \mathrm{PacketA2})$$

This verification completed in a little over 5 minutes.

Encouraged by this result, we introduced the SYN backlog back into the model. The symbolic model checker ran out of memory on a machine with 16GB of memory after three days. We then ran the bounded model checker up to depth 1000 (to mean that all sequences of transitions of length ≤ 1000 are checked for counterexamples), and the model checker did not report any counterexample, which is very encouraging. This means that the attacker cannot violate non-interference with any idle scan where less than 1000 transitions occur. The SYN backlog counterexample to our shared SYN backlog implementation required only 5 transitions. Informally, this result means that there exists no attack, even with only a single entry in the SYN backlog, where the attacker can violate non-interference with 1000 or fewer packets being generated by the attacker or by the zombie and victim's responses. One avenue of possible future work would be to explore alternatives to symbolic model checking for the split SYN backlog, including verifying the property through k-induction [26].

Attempting a proof by induction on an induction-based theorem prover such as ACL2 or RRL may also prove fruitful.

3.5 Concluding Remarks and Future Work

We modeled idle scans for modern network stacks using transition systems and analyzed them using model checking. This modeling effort led to the discovery of two new forms of idle scan, each of which was associated with a shared, limited resource. Our results demonstrate that non-interference for network protocol stacks warrants further study. We discovered two new forms of idle scan, one of which gives the attacker capabilities that no current attacker port scanning capabilities below layer 7 (the application layer) provide. We demonstrated in this chapter that it is possible for an attacker to port scan a network from outside the firewall on a port that the firewall blocks, for example. We also showed that this form of idle scan, based on SYN backlogs, can be used for a rudimentary form of operating system detection. In light of these results, a more formal treatment of information flow in networks is needed so that we can better understand advanced idle scans, both for existing networks and in future protocol designs.

We discussed the stealth and efficiency of the idle scans in Section 3.4.2. While it is clear both that the attacks are practical and that certain defenses exist in some situations, a more thorough treatment of possible scans and defenses to detect or eliminate them is needed.

Using SAL's model checkers, we were able to identify counterexamples to non-interference, in the form of idle scans, from our formal model of a network stack as a transition system. After fixing the model by splitting limited resources and separating them for trusted *vs.* untrusted hosts we are able to verify the non-interference property for the RST rate limit case. However, we were only able to show the non-

interference property with both RST rate limiting and a SYN backlog up to 1000 transitions. Verifying the non-interference property for this more general fix using a model checker remains a challenge. While non-interference and model checking proved useful for studying specific shared resources, we were not able to build a model complex enough to discover unexpected counterexamples.

Our model of network stacks was at the level of abstraction of sequences of packets. A richer model that includes memory usage, packet loss, and packet delay would likely produce more counterexamples to the non-interference property for idle scans. Thus we propose that trust relationships be made explicit all the way down to the IP layer in future protocol designs. Because all resources are inherently limited, giving protocol implementations a mechanism that can help divide these resources and remove sharedness is the only way to address the advanced network reconnaissance attacks of the future. Our results in Section 3.4.3 demonstrated that non-interference, which effectively eliminates idle scans, is achievable by statically dividing resources based on trust relationships.

```
Network_Protocol_Stack : CONTEXT =
BEGIN
   % Type Declarations
       Protocol : TYPE = {tcp, icmp, udp, invalidProtocol};
       IP: TYPE = [1..5]; % 1 is Attacker, 2 is Victim, 3 is Zombie
       PortStatus: TYPE = {open, closed, filtered};
       TCP_Flag : TYPE = {syn, fin, synack, rst, drop};
       Port: TYPE = [1..3]; % each host has 3 ports
       Type_ICMP: TYPE = [1..5];
       Packet: TYPE = [# sip_IP : IP, dip_IP : IP,  ihl_IP : ValidOrInvalid, ...
       Host : TYPE = [# ip : IP , portstatus : array Port of PortStatus, ...
...
   % Functions
       ProperReply(Packet, PortStatus, Rst_counter): Packet, Rst_counter;
         % This function returns the proper response to a given packet
         % and a reduced Rst_counter if a RST is sent.
...
       UpdateSynbacklog (Packet, Synbacklog): Synbacklog, SynPIsthereOrNot;
         % This function returns the new state of the SYN backlog, which can change
         % for SYNs or RSTs, and a value indicating if the SYN already exists, was
         % dropped, or was placed in the SYN backlog
...
     main : MODULE =
       BEGIN
         % Variable Declarations
           GLOBAL v1 : Host % victim 1
           GLOBAL v2 : Host % victim 2
           GLOBAL z1 : Host % zombie 1
           GLOBAL z2 : Host % zombie 2
...
           LOCAL PacketA1 : Packet % Packets sent to attacker in scenario 1
           LOCAL PacketA2 : Packet % Packets sent to attacker in scenario 2
...
         % Initialization Section
...
         % Transition Section
           TRANSITION
             [
               (v1.fullness /= QueueSize OR z1.fullness /= QueueSize )-->
               % Attacker creates a packet and puts it in queues of either
               % v1 and v2 or z1 and z2
...
             []
               (v1.fullness /= 0 AND v2.fullness /= 0 ) -->
               % v1 and v2 pop a packet and call ProperReply and UpdateSynbacklog to
               % choose a proper reply and update their internal state.
...
             []
               (z1.fullness /= 0 AND z2.fullness /= 0 ) -->
               % z1 and z2 reply and update their state
...
             ]
       END;
      theorem1: THEOREM main |- G( PacketA1 = PacketA2 );
END
```

Figure 3.1: Outline of SAL code for the network protocol stack model.

```
...
[]
                (z1.fullness /= 0 AND z2.fullness /= 0 ) -->
                % z1 and z2 reply and update their state
...
                collector_1'.packet = z1.queueOfHost[1] ;
                (FORALL (j: FullnessOfQueue ): z1'.queueOfHost[j-1] = z1.queueOfHost[j]);
                z1'.fullness = z1.fullness - 1;
...
                temp'= UpdateSynbacklog(...);
                z1'.synbacklog= temp'.host.synbacklog;
...
                response' = ProperReply(collector_1', z1.portsstatus[collector_1'.packet.dport],
                                temp' ) ;
...
                % Do the same for z2, which may have a different packet in its queue
                collector_2'.packet = z2.queueOfHost[1] ;
                (FORALL (j: FullnessOfQueue ): z2'.queueOfHost[j-1] = z2.queueOfHost[j]);
                z2'.fullness = z2.fullness - 1;
...
```

Figure 3.2: Overview of what happens when a transition fires.

```
...
   % Functions
       ProperReply(Packet, PortStatus, Rst_counter): Packet, Rst_counter;
         % This function returns the proper response to a given packet
         % and a reduced Rst_counter if a RST is sent.
IF ...
...
        ELSIF ( Packet.protocol_IP = tcp AND  (Packet.seqNum_TCP = known AND Packet.ack_TCP = known )
                AND Packet.flag_TCP = syn AND ps = open AND Packet.synPIsThere2 = put  )
        THEN
              packetOut with .packet.sip_IP := Packet.dip_IP
                        with .packet.sport := Packet.dport
                        with .packet.dip_IP := Packet.sip_IP
                        with .packet.dport := Packet.sport
                        with .packet.flag_TCP := synack
                        with ...
...
       UpdateSynbacklog (Packet, Synbacklog): Synbacklog, SynPIsthereOrNot;
         % This function returns the new state of the SYN backlog, which can change
         % for SYNs or RSTs, and a value indicating if the SYN already exists, was
         % dropped, or was placed in the SYN backlog
        IF ( ... AND Packet.flag_TCP = syn AND Host.synQeueEntries[1]=valid
                        ... AND Packet.sip_IP = host.synbacklog[1].sip_IP ...)
        THEN  # Ignore duplicate SYNs, i.e., that already exist in the SYN backlog
              synCollOut with .synPIsThere := exist
        ELSIF ( ... )
        THEN synCollOut
                with .synPIsThere := put
                with .host.synbacklog[1] := synCollIn.packet
...
```

Figure 3.3: Structure of the `UpdateSynbacklog` and `ProperReply` functions.

Chapter 4

Detecting Censorship via TCP/IP Side Channels

Tools for discovering intentional packet drops are important for a variety of applications, such as discovering the blocking of Tor by ISPs or nation states [13]. However, existing tools have a severe limitation: they can only measure when packets are dropped in between the measurement machine and an arbitrary remote host. The research question we address in this chapter is: can we detect packet drops between two hosts without controlling either of them and without sharing the path between them?

In this chapter, we describe a method for remotely detecting intentional packet drops on the Internet using idle scans. That is, given two arbitrary IP addresses on the Internet that meet some simple requirements, our proposed idle scan can discover packet drops (*e.g.*, due to censorship) between the two remote machines, as well as infer in which direction the packet drops are occurring. The only major requirements for our approach are a client with a global IP Identifier (IPID) and a target server with an open port. We require no special access to the client or server.

Our method is robust to noise because we apply intervention analysis based on an autoregressive-moving-average (ARMA) model. Also, our method uses a low packet rate that does not fill shared resource and therefore it is non-intrusive.

4.1 Introduction

As shown in Section 2.2, Antirez [12] proposed the first type of idle scan, which we call an IPID idle port scan. In this type of idle scan an "attacker" (which we will refer to as the *measurement machine* in our work) aims to determine if a specific port is open or closed on a "victim" machine (which we will refer to as the *server*) without using the attacker's own return IP address. The attacker finds a "zombie" (*client* in this chapter) that has a global IP identifier (IPID) and is completely idle. In this chapter, we say that a machine has a global IPID when it sends TCP RST packets with a globally incrementing IPID that is shared by all destination hosts. This is in contrast to machines that use randomized IPIDs or IPIDs that increment per-host. More details are in Section 2.2. Nmap [58] has built-in support for antirez's idle scan, but often fails for Internet hosts because of noise in the IPID that is due to the zombie sending packets to other hosts. Our method described in this chapter is resistant to noise, and can discover packet drops in either direction (and determine which direction). Nmap cannot detect the case of packets being dropped from client to server based on destination IP address, which our results demonstrate is a very important case.

Two other types of idle scans were presented in Chapter 3, including one that exploits the state of the SYN backlog as a side channel. Our method is based on a new idle scan technique that can be viewed as a hybrid of the IPID idle scan and the SYN backlog idle scan from Chapter 3. Whereas the SYN backlog idle scan required filling the SYN backlog and therefore causing denial-of-service, our technique uses

a low packet rate that does not fill the SYN backlog and is non-intrusive. The basic insight that makes this possible is that information about the server's SYN backlog state is entangled with information about the client's IPID field. Thus, we can perform both types of idle scans (IPID and SYN backlog), to detect drops in both directions, and our technique overcomes the limitations of both by exploiting the entanglement of information in the IPID and treating it as a linear intervention problem to handle noise characteristic of the real Internet.

Effectively, by using idle scans our method can turn approximately 1% of the total IP address space into measurement machines that can be used as vantage points to measure IP-address-based censorship, without actually gaining access to those machines. We can achieve this because of information flow in their network stacks.

In this chapter, we offer several major contributions:

- A non-intrusive method for detecting intentional packet drops between two IP addresses on the Internet where neither is a measurement machine.
- An Internet measurement study that shows the efficacy of the method.
- A model of IPID noise based on an autoregressive-moving-average (ARMA) model that is robust to autocorrelated noise.

Source code and data are available upon request, and a web demonstration version of the hybrid idle scan is at `http://spookyscan.cs.unm.edu`. The types of measurements we describe in this chapter raise ethical concerns because the measurements can cause the appearance of connection attempts between arbitrary clients and servers. In China there is no evidence of the owners of Internet hosts being persecuted for attempts to connect to the Tor network, thus our measurements in this chapter are safe. However, we caution against performing similar measurements in

other countries or contexts without first evaluating the risks and ethical issues. More discussion of ethical issues is in Chapter 7.

The rest of the chapter is structured as follows: After describing the implementation of our method in Section 4.2, we present our experimental methodology for the measurement study in Section 4.3. This is followed by Section 4.4, which describes how we analyze the time series data generated by a scan using an ARMA model. Results from the measurement study are in Section 4.5, followed by discussions and conclusions in Section 4.6. Related work is discussed in Chapter 6.

4.2 Implementation of Hybrid Idle Scan

In order to determine the direction in which packets are being blocked, our method is based on information flow through both the IPID of the client and the SYN backlog state of the server, as shown in Figure 4.1. Our implementation queries the IPID of the client (by sending SYN/ACKs from the measurement machine and receiving RST responses) to create a time series to compare a base case to a period of time when the server is sending SYN/ACKs to the client (because of our spoofed SYNs). We assume that the client has global IPIDs and the server has an open port.

Global IPIDs were explained in Section 4.1. The SYN backlog is a buffer that stores information about half-open connections where a SYN has been received and a SYN/ACK sent but no ACK reply to the SYN/ACK has been received. Half-open connections remain in the SYN backlog until the connection is completed with an ACK, aborted by a RST or ICMP error, or the half-open connection times out (typically between 30 and 180 seconds). The SYN/ACK is retransmitted some fixed number of times that varies by operating system and version, typically three to six SYN/ACKs in total. This SYN backlog behavior on the server, when combined with the global IPID behavior of the client, enables us to distinguish three different cases

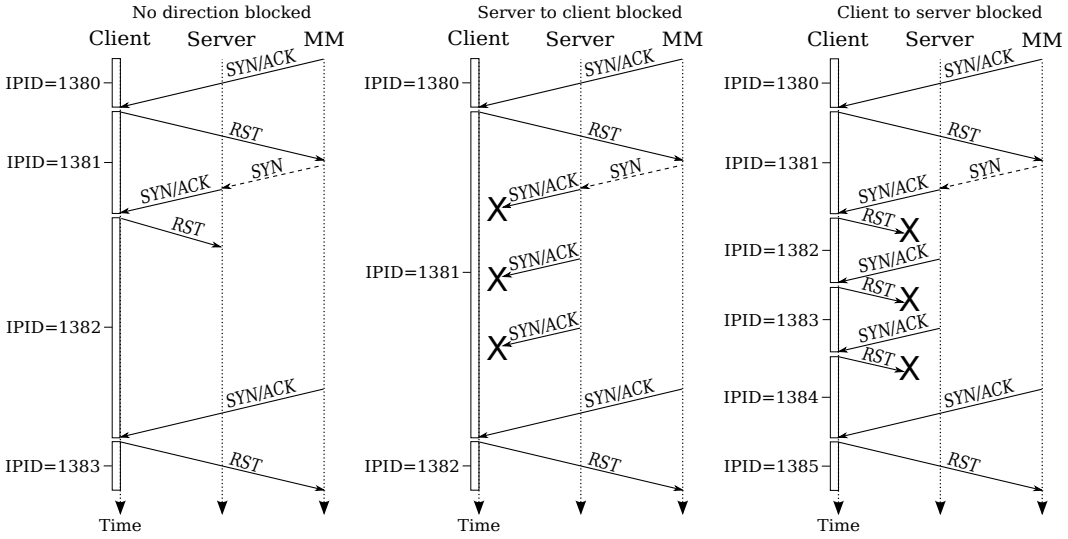


Figure 4.1: Three different cases that our method can detect. MM is the measurement machine.

(plus an error case):

- **Server-to-client-dropped:** In this case SYN/ACKs are dropped in transit from the server to the client based on the return IP address (and possibly other fields like source port), and the client's IPID will not increase at all (except for noise). See Figure 4.2.

- **No-packets-dropped:** In the case that no intentional dropping of packets is occurring, the client's IPID will go up by exactly one. See Figure 4.3. This happens because the first SYN/ACK from the server is responded to with a RST from the client, causing the server to remove the entry from its SYN backlog and not retransmit the SYN/ACK. Censorship that is stateful or not based solely on IP addresses and TCP port numbers may be detected as this case, including filtering aimed at SYN packets only. Also, if the packet is not dropped, but instead the censorship is based on injecting RSTs or ICMP errors, it will be detected as this case. Techniques for distinguishing these other

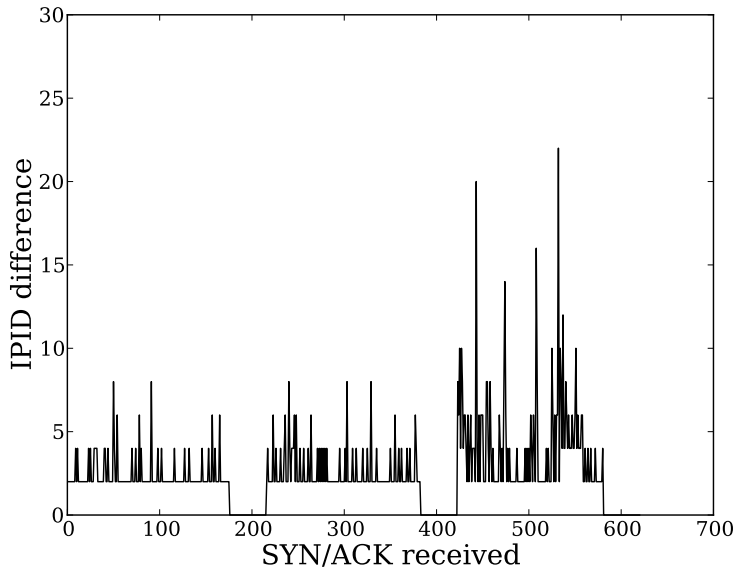


Figure 4.2: Example IPID difference time series' for three separate experiments that lead to detection of the **Server-to-client-dropped** case. Note the high amount of noise in the third experiment. Our ARMA modeling is able to detect this case correctly even in the presence of such high noise.

possibilities are left for future work.

- **Client-to-server-dropped:** In this case RST responses from the client to the server are dropped in transit because of their destination IP address (which is the server). When this happens the server will continue to retransmit SYN/ACKs and the client's IPID will go up by the total number of transmitted SYN/ACKs including retransmissions (typically three to six). See Figure 4.4. This may indicate the simplest method for blacklisting an IP address: null routing.

- **Error:** In this case networking errors occur during the experiment, the IPID is found to not be global throughout the experiment, a model is fit to the data but does not match any of the three non-error cases above, the data is too

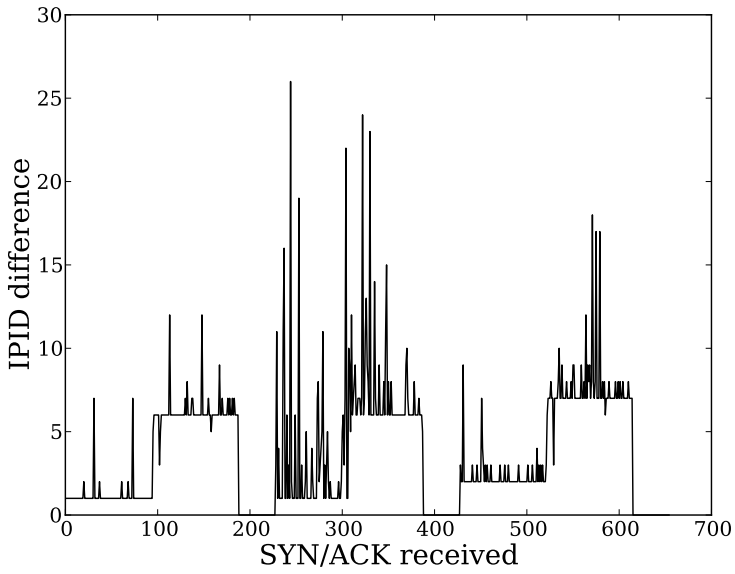


Figure 4.3: Example IPID difference time series' for three separate experiments that lead to detection of the **No-packets-dropped** case. Note the high amount of noise in the second experiment. Our ARMA modeling is able to detect this case correctly even in the presence of such high noise.

noisy and intervention analysis (see Section 4.4) fails because we are not able to fit a model to the data, and/or other errors.

Noise due to packet loss and delay or the client's communications with other machines may be autocorrelated. The autocorrelation comes from the fact that the sources of noise, which include traffic from a client that is not idle, packet loss, packet reordering, and packet delay, are not memoryless processes and often happen in spurts. The accepted method for performing linear intervention analysis on time series data with autocorrelated noise is ARMA modeling [18], which we describe in Section 4.4.

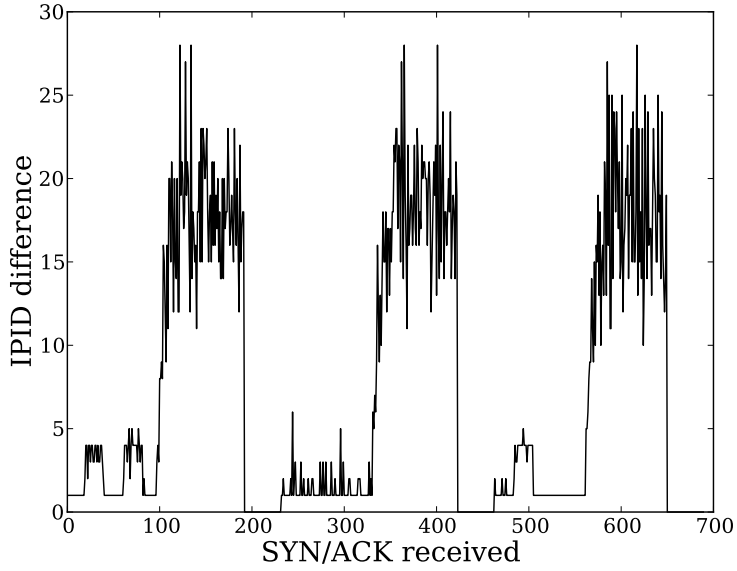


Figure 4.4: Example IPID difference time series' for three separate experiments that lead to detection of the **Client-to-server-dropped** case.

4.3 Experimental Setup

All measurement machines were Linux machines connected to a research network with no packet filtering. Specifically, this network has no stateful firewall or egress filtering for return IP addresses.

One measurement machine was dedicated to developing a pool of both client and server IP addresses that have the right properties for use in measurements. Clients were chosen by horizontally scanning China and other countries for machines with global IPIDs, then continually checking them for a 24-hour period to cull out IP addresses that frequently changed global IPID behavior (*e.g.*, because of DHCP), went down, or were too noisy. A machine is considered to have a global IPID if its IPID as we measure it by sending SYN/ACKs from alternating source IP addresses and receiving RSTs never incrementing outside the ranges $[-40, 0)$ or $(0, 1000]$ per

second when probed from two different IP addresses. This range allows for non-idle clients, packet loss, and packet reordering. It is possible to build the time series in different ways where negative IPID differences are never observed, but in this study our time series was the differences in the client's IPIDs in the order in which they arrived at the measurement machine. Our range of $[-40, 0)$ or $(0, 1000]$ is based on our observations of noise typical of the real Internet. The IPID going up by 0 is a degenerate case and means the IPID is not global.

Servers were chosen from three groups: Tor directory authorities, Tor bridges, and web servers. The ten Tor directory authorities were obtained from the Tor source code and the same ten IPs were tested for every day of data. Three Tor bridges were collected daily both through email and the web. Every day seven web servers were chosen randomly from the top 1000 websites on the Alexa Top 1,000,000 list [7]. All web server IPs were checked to make sure that they stood up for at least 24 hours before being selected for measurement. Furthermore, we checked that the client and server were both up and behaving as assumed between every experiment (*i.e.*, every five minutes).

A round of experiments was a 24-hour process in which measurements were carried out on the two measurement machines. Each 24-hour period had 22 hours of experiments and 2 hours of down time for data synchronization. For each measurement period on each of the two machines performing direct measurements, ten server machines and ten client machines from the above process were chosen for geographic diversity: 5 from China, 2 from countries in Asia that were not China, 1 from Europe, and 2 from North America. IP addresses were never reused except for the Tor directory authorities, so that every 24-hour period was testing a set of 20 new clients, 10 new servers, and the 10 directory authorities.

For each of the twenty clients and twenty servers geographical information provided by MaxMind was saved. MaxMind claims an accuracy of 99.8% for identifying

the country an IP address is in [64]. For each of the twenty server machines, a series of SYN packets was used to test and save its SYN/ACK retransmission behavior for the analysis in Section 4.4.

Every hour, each of our two main measurement machines created ten threads. Each thread corresponded to one client machine. Each thread tested each of the ten server IP addresses sequentially using our idle scan based on the client's IPID. No spoofed SYNs were sent to the server during the first 100 seconds of a test, and spoofed SYNs with the return IP address of the client were sent to the server at a rate of 5 per second for the second 100-second period. Then spoofed RST packets were sent to the server to clear the SYN backlog and prevent interference between sequential experiments. A timeout period of sixty seconds was observed before the next test in the sequence was started, to allow all other state to be cleared. Each experiment lasted for less than five minutes, so that ten could be completed in an hour. Every client and server was involved in only one experiment at a time. Each client/server pair was tested once per hour throughout the 24-hour period, for replication and also to minimize the effects of diurnal patterns. Source and destination ports for all packets were carefully chosen and matched to minimize assumptions about what destination ports the client responds on. Specifically, source ports for SYN packets sent to the server (both spoofed SYNs and SYNs with the measurement machine's IP address as the return IP address for testing) were chosen from the same range as the destination ports for SYN/ACKs sent to the client (always strictly less than 1024). We did not find it necessary to hold the source port for SYN/ACKs sent to the client to be always equal to the open port on the server, but this is possible.

4.4 Analysis

In this section, we set out our statistical model for our time series data. We then describe our process for outlier removal and for statistically testing if and in which direction packet drops are occurring.

We model each time series $y_1, \ldots, y_n$ as a *linear regression with ARMA errors*, a combination of an autoregressive-moving-average (ARMA) model with external linear regressors. ARMA models are used to analyze time series with autocorrelated data and are themselves a combination of two models, an autoregressive (AR) model and a moving-average (MA) model.

An AR model of order p specifies that every element of a time series can be written as a constant plus the linear combination of the previous p elements:

$$y_t = c + z_t + \phi_1 y_{t-1} + \cdots + \phi_{t-p} y_{t-p}$$

where z_t is a white noise series. An MA model of order q specifies that every element of a time series can be written as a constant plus the linear combination of the previous q white-noise terms:

$$y_t = c + z_t + \theta_1 z_{t-1} + \cdots + \theta_{t-q} z_{t-q}$$

Intuitively, each element is linearly related to the previous random "shocks" in the series. An ARMA(p, q) model combines an AR model of order p and an MA model of order q:

$$y_t = c + z_t + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i z_{t-i}$$

We use a linear regression with ARMA errors to model our time series data. This specifies that every element in a time series can be written as a constant plus the

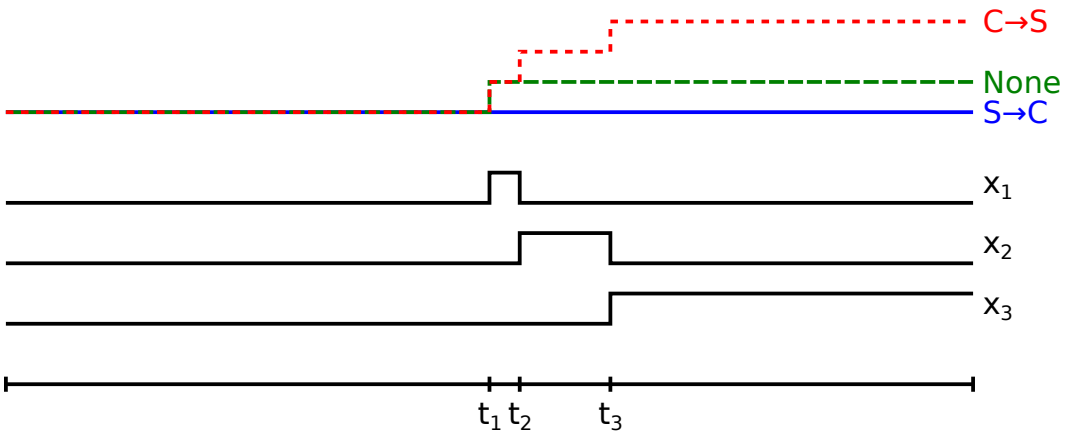


Figure 4.5: For a server that retransmits $r-1$ SYN/ACK's, each case can be expressed as the linear combination of regressors $x_1, \ldots, x_r$; shown is when $r = 3$ with SYN/ACK transmissions responding to the first spoofed SYN occurring at t_1, t_2, and t_3. C→S indicates client-to-server, and S→C indicates server-to-client.

linear combination of regressors $x_1, \ldots, x_r$ with an ARMA-modeled error term:

$$y_t = c + \sum_{i=1}^{r} \beta_i x_{it} + e_t,$$
$$e_t = z_t + \sum_{i=1}^{p} \phi_i e_{t-i} + \sum_{i=1}^{q} \theta_i z_{t-i}$$

We use the regressors x_i for *intervention analysis*, *i.e.*, for analyzing our experimental effect on the time series at a specific time.

For each experiment, we pick regressors according to which times the server (re)transmits SYN/ACK's in response to SYN's. For a server that (re)transmits r SYN/ACK's in response to each SYN, we have r regressors. We call time t_1 the time of the first transmission in response to the first of our spoofed SYN's, and we call t_{i+1} the time the server would send the ith retransmission in response to that

SYN. Then we define regressor x_i as the indicator variable

$$x_{ij} = \begin{cases} 1 & \text{if } t_i \leq j \text{ and either } j < t_{i+1} \text{ or } i = r \\ 0 & \text{otherwise} \end{cases}$$

In other words, x_1 is zeros until the time the server transmits the first SYN/ACK then ones until the server begins retransmitting SYN/ACK's. The remaining x_i are zeros until the time the server would begin retransmitting its ith SYN/ACK then ones until if/when the $(i+1)$th SYN/ACK's would begin being retransmitted. This definition allows us to model any of the possible level shifts in any case of packet drop as a linear combination of all x_i. See Figure 4.5 for an illustration.

We choose ARMA orders p and q by performing model selection over time series elements $y_1, \ldots, y_{t_1}$. We find the $p \leq 7$ and $q \leq 7$ for the ARMA(p, q) model that maximizes the corrected Akaike information criterion, a metric which rewards models that lose less information but penalizes models overfitted with too many parameters [44]. It is given by

$$AIC_C = -2 \ln L + 2k + \frac{2k(k+1)}{n-k-1},$$

where here the number of parameters k is $p + q + 2$ and where L is the estimated maximum likelihood over all ϕ_i and θ_i.

After p and q are chosen, we then simultaneously fit all ϕ_i, θ_i, and β_i of our linear regression model with ARMA errors over the entire time series $y_1, \ldots, y_n$ corresponding to the estimated maximum likelihood.

After fitting parameters, we remove outliers that might be caused by, *e.g.*, spikes in network traffic that may hamper our analysis. We use the $\hat{\lambda}_{2,T}$ test statistic proposed by Chang *et. al* [20] with significance $\alpha = 0.05$. After removing outliers, we iteratively refit the ϕ_i, θ_i, and β_i parameters and test for outliers until no additional outliers are removed.

For intervention analysis, we use hypothesis testing over the value of β_r to determine if packets are dropped and in which direction. If we send s spoofed SYN's, without noise, we would expect β_r to equal one of the following: 0 for the case where packets are dropped from the server to client, s for the case where no packets are dropped, or rs for the case where packets are dropped from the client to server. One might pick two thresholds, $k_1 = s/2$ in between the first two cases and threshold $k_2 = (1 + r)s/2$ between the last two cases; however, for the second threshold, we instead choose $k_2' = \min(2s, k_2)$ to be more robust to, *e.g.*, packet loss. Then we determine the case by a series of one-sided hypothesis tests performed with significance $\alpha = 0.01$ according to the following breakdown:

- **Server-to-client-dropped** if we reject the null hypothesis that $\beta_r \geq k_1$.
- **No-packets-dropped** if we reject the null hypotheses that $\beta_r \leq k_1$ and that $\beta_r \geq k_2'$.
- **Client-to-server-dropped** if we reject the null hypothesis that $\beta_r \leq k_2'$.
- **Error** if none of the above cases can be determined.

4.5 Results

Table 4.1 shows results from 5 days of data collection, where $S \to C$ is **Server-to-client-dropped**, *None* is **No-packets-dropped**, $C \to S$ is **Client-to-server-dropped**, and *Error* is **Error**. *CN* is China, *Asia-CN* is other Asian countries, *EU* is Europe, and *NA* is North America. For server types, *Tor-dir* is a Tor directory authority, *Tor-bri* is a Tor bridge, and *Web* is a web server.

Our expectation would be to observe **Server-to-client-dropped** for clients in China and Tor servers because of Winter and Lindskog's observation that SYN/ACKs

Client,Server	$S \to C$ (%)	None (%)	$C \to S$ (%)	Error (%)
CN,Tor-dir	2200 (73.04)	19 (0.63)	504 (16.73)	289 (9.59)
Asia-CN,Tor-dir	0 (0.00)	1171 (96.38)	1 (0.08)	43 (3.54)
NA,Tor-dir	1 (0.07)	1217 (90.69)	49 (3.65)	75 (5.59)
EU,Tor-dir	2 (0.28)	695 (97.89)	2 (0.28)	11 (1.55)
CN,Tor-bri	1012 (58.91)	565 (32.89)	31 (1.80)	110 (6.40)
Asia-CN,Tor-bri	0 (0.00)	626 (80.88)	9 (1.16)	139 (17.96)
NA,Tor-bri	0 (0.00)	657 (78.21)	30 (3.57)	153 (18.21)
EU,Tor-bri	0 (0.00)	313 (78.25)	9 (2.25)	78 (19.50)
CN,Web	28 (2.15)	995 (76.30)	36 (2.76)	245 (18.79)
Asia-CN,Web	1 (0.17)	569 (97.43)	1 (0.17)	13 (2.23)
NA,Web	0 (0.00)	606 (93.37)	0 (0.00)	43 (6.63)
EU,Web	0 (0.00)	305 (90.24)	0 (0.00)	33 (9.76)
All Web	29 (1.01)	2475 (86.09)	37 (1.29)	334 (11.62)
All Tor-bri	1012 (27.12)	2161 (57.90)	79 (2.12)	480 (12.86)
All Tor-dir	2203 (35.09)	3102 (49.40)	556 (8.85)	418 (6.66)

Table 4.1: Results from the measurement study.

are statelessly dropped by the "Great Firewall of China" (GFW) based on source IP address and port [107]. We would expect to see **No-packets-dropped** for most web servers from clients in China, unless they host popular websites that happen to be censored in China. Similarly, in the expected case we should observe **No-packets-dropped** for clients outside of China, regardless of server type. We expect a few exceptions, because censorship happens outside of China and because the GFW is not always 100% effective. In particular, Tor bridges are not blocked until the GFW operators learn about them, and some routes might not have filtering in place. Our results are congruent with all of these expectations.

In 5.9% of the client/server pairs we tested, multiple cases were observed in the same day. In some cases it appears that noise caused the wrong case to be detected, but other cases may be attributable to routes changing throughout the day [75]. That the data is largely congruent with our expectations demonstrates the efficacy of the approach, and some of the data points that lie outside our expectations have

patterns that suggest that a real effect is being measured, rather than an error. For example, of the 28 data points where web servers were blocked from the server to the client in China, 20 of those data points are the same client/server pair.

38% of the data we collected does not appear in Table 4.1 because it did not pass liveness tests. Every 5-minute data point has three associated liveness tests. If a server sends fewer than 2.5 SYN/ACKs in response to SYNs from the measurement machine, a client responds to less than $\frac{3}{5}$ of our SYN/ACKs, or a measurement machine sending thread becomes unresponsive, that 5-minute data point is discarded.

Two out of ten Tor directory authorities never retransmitted enough SYN/ACKs to be included in our data. Of the remaining eight, two more account for 98.8% of the data points showing blocking from client to server. These same two directory authorities also account for 72.7% of the **Error** cases for directory authorities tested from clients in China, and the case of packets being dropped from server to client (the expected case for China and the case of the majority of our results) was never observed for these two directory authorities.

When Winter and Lindskog [107] measured Tor reachability from a virtual private server in China, there were eight directory authorities at that time. One of the eight was completely accessible, and the other seven were completely blocked in the IP layer by destination IP (*i.e.*, **Client-to-server**). In our results, six out of ten are at least blocked **Server-to-client** and two out of ten are only blocked **Client-to-server** (two had all results discarded). Winter and Lindskog also observed that Tor relays were accessible 1.6% of the time, and we observed that directory authorities were accessible 0.63% of the time. Our results have geographic diversity and their results can serve as a ground truth because they tested from within China. In both studies the same special treatment of directory authorities compared to relays or bridges was observed, as well as a small percentage of cases where filtering that should have occurred did not.

To evaluate the assumption that clients with a global IPID are easy to find in a range of IP addresses that we desire to measure from, take China as an example. On average, 10% of the IP addresses in China responded to our probes so that we could observe their IPID, and of those 13% were global. So, roughly 1% of the IP address space of China can be used as clients for measurements with our method, enabling experiments with excellent geographic and topological diversity.

4.6 Conclusion

We have presented a method for detecting intentional packet drops (*e.g.*, due to censorship) between two almost arbitrary hosts on the Internet, assuming the client has a globally incrementing IPID and the server has an open port. Our method can determine which direction packets are being dropped in, and is resistant to noise due to our use of an ARMA model for intervention analysis.

In a measurement study using our method featuring clients from multiple continents, we observed that, of all measured client connections to Tor directory servers that were censored, 98% of those were from China, and only 0.63% of measured client connections from China to Tor directory servers were not censored. This is congruent with current understandings about global Internet censorship, leading us to conclude that our method is effective.

Chapter 5

Large-scale Spatiotemporal Characterization of Inconsistencies in GFW

A nation-scale firewall, colloquially referred to as the "Great Firewall of China," implements many different types of censorship and content filtering to control China's Internet traffic. Past work has shown that the firewall occasionally fails. In other words, sometimes clients in China are able to reach blacklisted servers outside of China. This phenomenon has not yet been characterized because it is infeasible to find a large and geographically diverse set of clients in China from which to test connectivity.

In this chapter, we overcome this challenge by using hybrid idle scan techniques that are able to measure connectivity between a remote client and an arbitrary server, neither of which are under the control of the researcher performing measurements.[1] In addition to hybrid idle scans, we present and employ a novel side channel in

[1]For more details on the hybrid idle scan, refer to Chapter 4.

the Linux kernel's SYN backlog. We demonstrate both techniques by measuring the reachability of the Tor network which is known to be blocked in China. Our measurements reveal that *1)* failures in the firewall occur throughout the entire country without any conspicuous geographical patterns, *2)* a network block in China appears to have unfiltered access to parts of the Tor network, and *3)* the filtering seems to be mostly centralized at the level of Internet exchange points. Our work also answers many other open questions about the Great Firewall's architecture and implementation.

5.1 Introduction

More than 600 million Internet users are located behind the world's most sophisticated and pervasive censorship system: the Great Firewall of China (GFW) [76]. Brought to life in 2003, the GFW has a tight grip on several layers of the TCP/IP model and is known to block or filter IP addresses [107], TCP ports [107], DNS requests [57, 90, 109], HTTP requests [23, 74, 24], circumvention tools, and even social networking sites [113].

This pervasive censorship gives rise to numerous circumvention tools seeking to evade the GFW by exploiting a number of opportunities [82]. Of particular interest is the Tor anonymity network [28] whose arms race with the operators of the GFW now counts several iterations. Once having had 30,000 users solely from China, the Tor network now is largely inaccessible from within China's borders as illustrated in Figure 5.1.

The amount of users trying to connect to the Tor network indicates that there is a strong need for practical and scalable circumvention tools. Censorship circumvention, however, builds on *censorship analysis*. A solid understanding of censorship systems is necessary in order to design sound and sustainable circumvention systems.

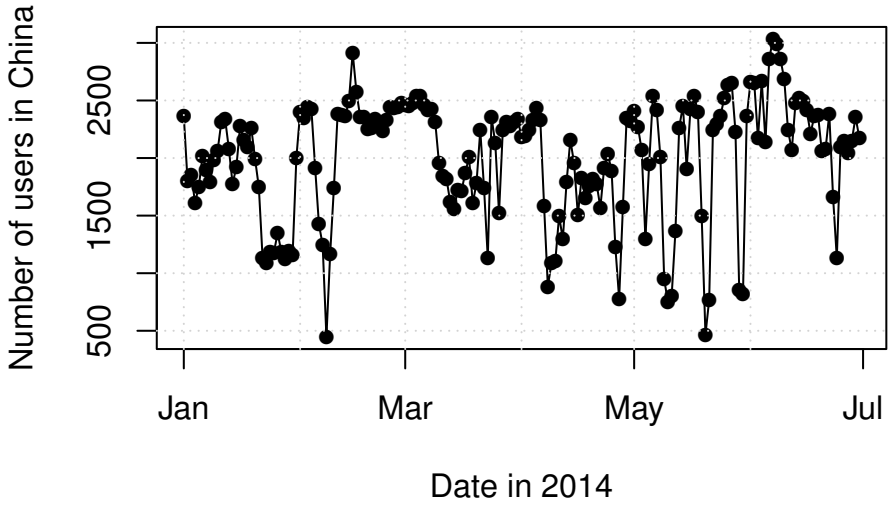


Figure 5.1: The approximate amount of directly connecting Tor users (as opposed to connecting over bridges) for the first months of 2014. While the number of users varies, it rarely exceeds 3,000.

However, it is difficult to analyze Internet censorship without controlling either the censored source machine or its—typically uncensored—communication destination. This problem is usually tackled by obtaining access to censored source machines, finding open proxies, renting virtual systems, or by cooperating with volunteers inside the censoring country. In the absence of these possibilities, censorship analysis has to resort to observing traffic on the server's side and inferring what the client is seeing.

Our work fills this gap by presenting and evaluating network measurement techniques which can be used to expose censorship while controlling *neither the source nor the destination machine*. This puts our study in stark contrast to previous work which had to rely on proxies or volunteers, both of which provide limited coverage of the censor's networks. By being mostly independent of source and destination machines, we are able to shed light on entirely unexplored areas of the Internet. We evaluate our techniques by applying them to the Tor anonymity network, thereby

handing the Tor Project practical tools to measure the reachability of their network. Such tools are needed because bridges[2] are frequently blocked in China without the bridge operators or the Tor Project noticing [56]. Our work makes it possible to test the reachability of these bridges without having a vantage point in China. As a result, the Tor Project is able to learn which subset of bridges is still reachable and hence undiscovered by the GFW. This knowledge facilitates the optimization of bridge distribution [101], *e.g.*, bridges blocked in China are only given out to users outside China.

Our techniques are currently limited to testing *basic IP connectivity*. Thus, we can only detect censorship on lower layers of the network stack, *i.e.*, before a TCP connection is even established. This kind of low-level censorship is very important to the censors, however. For example, while social media controls on domestic sites in China, such as Weibo, can be very sophisticated, users would simply use alternatives such as Facebook if the low-level IP address blocking were not in place to prevent this. Also, deep packet inspection (DPI) does not scale as well in terms of raw traffic as does lower-level filtering. Nevertheless, we acknowledge that our techniques are not applicable if censors only make use of DPI to block Tor as it was or is done by Ethiopia, Kazakhstan, and Syria [1].

We are interested not only in finding patterns in the GFW's *failures*, but also in gaining a better understanding of how the GFW is *architected* within China's backbone and provincial networks and whether previously observed details of its *implementation* are observed throughout the country. To this end, we focus our efforts on testing the following hypotheses that will illuminate the GFW's architecture and implementation. All hypotheses are with respect to the filtering of TCP/IP packets based on IP addresses and port numbers.

Hypothesis 1 *In general, from any client to any destination if a SYN packet is*

[2]Bridges are "hidden" Tor relays which are not listed in the public network consensus.

filtered by the GFW then a RST with the same source, destination, and port numbers will also be filtered. For brevity, we refer to this hypothesis as "RSTs are treated the same as SYNs."

Hypothesis 2 *There are no conspicuous geographic patterns in the GFW's failures. In other words, failures can occur in any part of the country. For brevity, we refer to this hypothesis as "No geographic patterns in failures."*

Hypothesis 3 *In general, the GFW blocks Tor relays by dropping SYN/ACK segments with IP address and port information that matches known Tor relays. Other types of filtering seen for Tor relays in China (*e.g.*, dropping SYN segments) are a negligible fraction of the censorship. For brevity, we refer to this hypothesis as "server-to-client blocking."*

Hypothesis 4 *At least some of the failures of the GFW are persistent, meaning that the client and server are able to communicate throughout the day. Note that this could also be due to intentionally whitelisted destinations, but in this chapter we refer to all cases where clients in China can access Tor relays as "failures." For brevity, we refer to this hypothesis as "some failures are persistent."*

Hypothesis 5 *At least some of the failures of the GFW exhibit diurnal patterns, where a client and blacklisted server can communicate at some times of the day but not others. For brevity, we refer to this hypothesis as "some failures have diurnal patterns."*

Hypothesis 6 *In general, packets that are subject to censorship traverse at least one or two hops, and sometimes more, into China before they are dropped by the GFW. For brevity, we refer to this hypothesis as "blocking is in the backbone."*

By testing the above hypotheses, we further increase the public's knowledge about the GFW and by presenting and evaluating our measurement techniques, we equip circumvention system developers with a set of tools to analyze and debug censorship incidents. In summary, this chapter makes the following contributions:

- We describe the first real-world application of the hybrid idle scan [33, 32], explained in more details in Chapter 4, to a large-scale Internet measurement problem, in which we measure the connectivity between the Tor anonymity network and clients in China over a period of four weeks.
- We present and evaluate a novel side channel based on the Linux kernel's SYN backlog which enables indirect detection of packet loss.
- We increase the community's understanding of how the GFW is architected and how its blocking of the Tor network looks from different clients all over China.

In the rest of this chapter, we discuss some background of the GFW in Section 5.2 and our measurement techniques in Section 5.3, which is then followed by our experimental methodology in Section 5.4. We analyze the data we gathered and present results in Section 5.5 and proceed with a discussion of our results in Section 5.6. The chapter is concluded in Section 5.7. Related work is covered in Chapter 6

5.2 Motivation and GFW Background

The hypotheses enumerated in Section 5.1 were chosen because we wanted to address the following open questions about the GFW:

- Are there geographic or other spatial patterns in the GFW's failures? This is important because such patterns could be exploited by evasion technologies

if the patterns exist, but if no such patterns exist then evasion efforts should focus on other aspects of the GFW.

- Are there temporal patterns in the GFW's failures? There are many different evasion efforts that periodically test their methods to see if they have been detected and blocked by the GFW. A solid understanding of temporal patterns (such as diurnal patterns) will help these projects to better understand the results of their tests.

- What kinds of packets are filtered in different parts of the country? This is important, because if an evasion technology is tested in, *e.g.*, Beijing but then fails to work in another part of the country, the developers of the evasion technology need to understand why.

- Where in China's Internet backbone does the filtering occur, and what is the role of routing? If an evasion technology is being tested from two different sources in China or two different destinations outside the country, the developers of the evasion technology may observe two different results for their tests and they need a good understanding of why this occurs.

Now we give more details about what was known before the work presented in this chapter. A more comprehensive overview of previous work is given in Chapter 6.

5.2.1 Spatial Patterns

In Chapter 4, we found that a small percentage of tests showed no signs of censorship. Our tests were taken between clients in China paired with Tor relays outside China.[3] However, our experimental methodology was designed to test if the failures in the

[3]Our work is published at [33, 32]

censorship observed by Winter and Lindskog [107] were also observed outside of Beijing or not. We made no attempt to choose clients or servers so that spatial patterns could be identified. In this chapter, our experiments were specifically designed to identify spatial and geographic patterns in the GFW's failures.

5.2.2 Temporal Patterns

Neither our previous work nor Winter and Lindskog attempted to characterize temporal patterns in the GFW's failures. This kind of characterization is difficult because, for a general understanding of temporal patterns, spatial patterns must be fully understood. Otherwise temporal patterns may be specific to one location. Also, temporal patterns are difficult to extract from idle scan measurements because of noise. This is why, in our experiments, we used traceroutes from a Tor relay to analyze temporal patterns.

5.2.3 Details of the Filtering

What kinds of packets are filtered? This is a key question, especially for evasion technologies that seek to evade the GFW *via* insertion and evasion in the IP and TCP layers. Winter and Lindskog described detailed results about what happens to SYN, SYN/ACK, ACK, and RST packets, but their results were specific to one location in China: Beijing. Also, any of their experiments that required observation on the server were only able to be carried out between Beijing and one Tor relay in Sweden. Our previous work in Chapter 4 had more spatial diversity in their experiments, but because of the nature of the hybrid idle scan the only packets that can be tested are SYN/ACKs from server to client and RSTs from client to server. SYN packets or any kind of stateful connection cannot be tested with the hybrid idle scan. All of these limitations in previous approaches is why our experiments include

another—previously unknown—idle scan that uses the SYN backlog to make more general inferences with a wider spatial variety.

5.2.4 Architecture of the GFW

There are generally regarded to be three theories about how the GFW is architected, posited in technical papers [24, 110, 23, 11] or other media [36, 100]. One theory posits that the filtering occurs at *choke points* where oversea cables carrying international Internet traffic enter the country. Another theory is that the majority of the filtering occurs in *three big Internet exchange points* in Beijing, Shanghai, and Guangzhou [106], near where international traffic enters the country but positioned more at central points in China's backbone network. A third theory that has been discussed is the possibility that the filtering occurs—or may increasingly occur as the GFW evolves—at the *provincial level* [110].

Our results about where the filtering of SYN/ACKs from Tor relays occurs are largely congruent with Xu *et al.*'s results about where RST injection based on deep packet inspection occurs. In their results, CNCGROUP performed most of its RST injection in the backbone, while CHINANET performed this type of censorship at the provincial level. Since their study, CNCGROUP has bought CHINANET, but the censorship at both the backbone and provincial levels, in about the same proportions as reported by Xu *et al.*, is also apparent in our results. This means that the routers that perform port mirroring for deep packet inspection are probably the same routers that enforce access controls such as blocking Tor by source IP address and TCP port. It also means that where the filtering occurs has not changed significantly since the study performed by Xu *et al.*.

In addition to providing more information about where the filtering occurs, our work presented in this chapter raises interesting questions about how the GFW is

architected, both in terms of implementations at routers and in terms of the big picture. Winter and Lindskog observed that for Tor relays only the SYN/ACK from the server is blocked, not the SYN from the client to the server. One of our key results in this chapter is that this observation also applies to China in general for a lot of different geographic locations. This raises a question: why block SYN/ACKs in the one direction, but not SYNs in the other?

One possible theory might be that the Border Gateway Protocol (BGP) plays a key role in the censorship by causing all international traffic to flow through the routers that implement the censorship. Because the GFW operators are presumably restricted to announcing BGP routes for autonomous systems (ASes) that are in China, they can only control routing in the direction of traffic that is entering China. Hence SYN/ACKs from Tor relays outside China to clients in China are blocked almost all the time, while SYNs from clients in China to Tor relays outside China are much less likely to be blocked.

Another theory is based on speculating about the way the GFW operators monitor traffic to decide what to block. In a description of the GFW written in Chinese by "Xylon Pan" [111], it is speculated that this is done because the server in an HTTP connection typically sends a lot more content to the client than the client sends to the server. Thus Netflow aggregation in the server-to-client direction works better, because there is more traffic to be sampled. One theory put forward by Xylon Pan is that since the GFW's operators think about network flows in the server-to-client direction, so they also write access controls (such as the blocking of Tor by IP address and TCP port) for the server-to-client direction.

The reason why this one-way blocking property (where SYN/ACKs entering China are much more likely to be blocked than SYNs leaving China) exists is left for future work. The major contribution of our present work in this regard is to confirm that this property is a general property that is observed all over China, not just in

the one or two locations where previous tests [107, 111] have been performed.

5.3 Networking Background

The research questions we seek to answer require high geographical diversity of clients in China. Typically, such a study would only be possible if we could find and control vantage points in all of China's provinces. Instead, we exploit side channels allowing us to detect intentional packet dropping—without controlling the two affected machines. In particular, we use *hybrid idle scans* (see Chapter 4 and Section 5.3.3) and *SYN backlog scans* (see Section 5.3.1). The idea behind these side channels as well as their prerequisites are discussed in this section.

5.3.1 Side Channels in Linux's SYN Backlog

A performance optimization in the Linux kernel's SYN backlog can be used to detect intentional packet dropping. Half-open TCP connections of network applications are queued in the kernel's *SYN backlog* whose size defaults to 256. These half-open connections then turn into fully established TCP connections once the server's SYN/ACK was acknowledged by the client. If a proper response is not received for an entry in the SYN backlog, it will retransmit the SYN/ACK several times. However, if the SYN/ACK and its respective retransmissions are never acknowledged by the client, the half-open connection is removed from the backlog. When under heavy load or under attack, a server's backlog might fill faster than it can be processed. This causes attempted TCP connections to not be fully handled while pending TCP connections time out. The Linux kernel mitigates this problem by *pruning* an application's SYN backlog. If the backlog becomes more than half full, the kernel begins to reduce the number of SYN/ACK retransmissions for all pending

connections [2]. As a result, half-open connections will time out earlier which should bring the SYN backlog back into uncritical state. We show that the Linux kernel's pruning mechanism—by design a *shared resource*—opens a side channel which can be used to measure intentional packet drops targeting a server. This is possible without controlling said server.

Our key insight is that we can remotely measure the approximate size of a server's SYN backlog by sending SYN segments and counting the number of corresponding SYN/ACK retransmissions. Starting with version number 2.2, the Linux kernel retransmits unacknowledged SYN/ACK segments five times [3]. As a result, we expect to receive the full number of five retransmissions when querying a service whose SYN backlog is less than half full. If, on the other hand, the backlog becomes more than half full, we will observe less than five retransmissions. When applied to the problem of intentional packet dropping, this allows us to infer whether a firewall blocks TCP connections by dropping the client's SYN or the server's SYN/ACK segment.

It is worth mentioning that a server's backlog state can also be inferred by coercing it into using *SYN cookies* [34]. A server using SYN cookies reveals that its SYN backlog is completely full. However, this measurement technique is effectively a SYN flood and TCP connections which were established using SYN cookies suffer from reduced throughput due to the lack of flow control window scaling. In contrast to triggering SYN cookies, our technique has no negative impact on servers or other clients' connections, when applied carefully.

5.3.2 The Global IP Identifier

As described in Section 4.1, IP identifiers (IPIDs) are unique numbers assigned to IP packets in case they are fragmented along a path. The receiving party is able

to reassemble the fragmented packets by looking at their IPID field. Most modern TCP/IP stacks increment the IPID field per connection or randomize it, as opposed to *globally incrementing* it. A machine with a globally incrementing IPID keeps a global counter that is incremented by 1 for every packet the machine sends, regardless of the destination IP address. Being a *shared resource*, the IPID can be used by a measurement machine talking to a remote machine to estimate how many packets the remote machine has sent to other machines. Throughout this chapter, we refer to machines with globally incrementing IPIDs as simply machines with "global IPIDs."

5.3.3 Hybrid Idle Scan

In Chapter 4, we described our method for remotely detecting intentional packet drops on the Internet via side channel inferences. This technique can discover packet drops (*e.g.*, caused by censorship) between two remote machines, as well as infer in which direction the packet drops are occurring. The only major requirements for this approach are a client with a global IPID and a target server with an open port. Access to the client or the server is not required. Conceptually, the hybrid idle scan technique can turn approximately 1% of the total IPv4 address space [33] into measurement machines that can be used as vantage points to measure IP address-based censorship—without having root access on those machines. This is why we employ the hybrid idle scan technique for our geographic study of how Tor is blocked in China.

As shown in Figure 4.1, the hybrid idle scan implementation queries the IPID of the client to create a time series. By sending SYN/ACKs from the measurement machine and receiving RST responses, the IPID of the client can be recorded. The time series is used to compare a base case (when no traffic is being generated other than noise) to a period of time when the server is sending SYN/ACKs to the client (because of our spoofed SYNs). By comparing two phases, one phase where no SYN

packets are sent to the server and one phase where SYN packets are sent to the server with the return IP address spoofed to appear to be from the client, the hybrid idle scan technique can detect *three different cases* (plus an error case), shown in Figure 4.1, with respect to IP packets being dropped by the network in between the client and the server:

1. **Server-to-client-dropped:** SYN/ACKs are dropped in transit from the server to the client causing the client's IPID to not increase at all (except for noise). See Figure 4.2.

2. **No-packets-dropped:** If no intentional packet dropping is happening, the client's IPID will go up by exactly one. See Figure 4.3.

3. **Client-to-server-dropped:** The RST responses sent by the client to the server are dropped in transit. In this case, the server will continue to retransmit SYN/ACKs and the client's IPID will get incremented by the total number of (re)transmitted SYN/ACKs, which is typically three to six. See Figure 4.4.

4. **Error:** A measurement error happens if networking errors occur during the experiment, the IPID is found to not be global throughout the experiment, a model is fit to the data but does not match any of the three non-error cases above, the data is too noisy and intervention analysis fails because we are not able to fit a model to the data, and/or other errors.

Note that these cases are the same cases as used for Chapter 4.

Auto-regressive moving average (ARMA) models are used to distinguish these cases. This overcomes autocorrelated noise in IPID values (*e.g.*, due to packet loss, packet delay, or other traffic that the client is receiving). More details about the ARMA modeling are described in Section 4.4

5.3.4 The Tor Network

The Tor network [28] is an overlay network which provides its users with anonymity on the Internet. Tor clients expose a local SOCKS interface which is used to anonymize TCP streams such as web traffic. As of April 2014, the network consists of approximately 4,500 volunteer-run *relays*, nine *directory authorities*, and one *bridge authority*. While the relays anonymize the network traffic of Tor clients, the authorities' task is to keep track of all relays and to vote on and publish the *network consensus* which Tor clients need in order to bootstrap. It is trivial for censors to download the hourly published network consensus and block all IP address/TCP port pairs found in it. Other circumvention systems suffer from the same problem [71].

All authorities are hard-coded in the Tor source code and their IP addresses remain static. As a result, they constitute attractive choke points for censors. In fact, blocking the IP addresses of all nine directory authorities is sufficient to prevent direct connections to the Tor network.[4] Our study focuses on the reachability of the authorities and relays, as it is known that the GFW is blocking them [107]. Our focus is on gathering more details about this blocking and characterizing it with a large-scale spatiotemporal study.

5.4 Experimental Methodology

In this section, we describe the challenges our experimental methodology was designed to address, the data sets we collected, how our measurements help us to test the hypotheses enumerated in Section 5.1, and other issues.

[4]Note that the Tor Project designed and implemented so-called bridges to tackle this very problem but the details are outside the scope of this work.

5.4.1 Encountered Challenges

Over the course of running our experiments and analyzing our data, we faced a number of challenges which we discuss here.

Churn in the Tor network: While the size of the Tor network does not vary considerably over a short period of time, the network's *churn rate* can render longitudinal studies difficult. For example, the median size of Tor's network consensus (*i.e.*, the number of Tor relays in the network) in March 2014 was 5,286. In total, however, March has seen 13,343 *unique relays*—many of which were online for only hours. To minimize the chance of selecting unstable Tor relays for longitudinal studies, only relays having earned the "Stable" flag should be considered [96]. Furthermore, the relay descriptor archives could be examined to calculate a relay's reachability over time [95]. We selected only Tor relays that had an uptime of at least five days, and filtered out all data points where a node appeared to have left the network. After having run our experiments, we removed one Tor relay in Argentina from our data because its Tor and web ports switched during our experiments.

Geolocation of routers: For geolocating routers, we used MaxMind's GeoIP2 City database [65]. As of April 2014, this database lacks accurate geolocation information for backbone routers in China. While provincial routers can typically be mapped to their province based on whois records, backbone routers are all mapped to the same bogus location at latitude 35 and longitude 105 which resides in an unpopulated area in central China. We also used MaxMind for geolocating clients, for which it is fairly accurate. For the location of routers, we used a combination of *whois information* and *round-trip delays* per hop. We discarded hops in our data that have whois records from China but are actually in Hong Kong or Pasadena, California (where ChinaNet has a Point of Presence).

Diurnal patterns: For most measurements in this chapter, we measured once

per hour throughout the day. This avoids bias and distortion. For example, if we measured one set of clients in the morning and one set at night, differences between the two sets of clients may be due to different traffic patterns at the different times of day and not a property of the different set of clients. Thus we always randomize the order of our experiments when possible and repeat all measurements every hour for at least one full day.

5.4.2 Experimental Design and Setup

Over the course of our experiments, we made use of three sets of Linux-based measurement machines in the U.S., China, and Europe. These three sets of machines correspond to the three main datasets that we collected.

Machines in the U.S.: The three machines used for our hybrid idle scans (see Section 5.3.3 and for more details Chapter 4) and SYN backlog scans (see Section 5.3.1) were located at our university campus (UNM) at the University of New Mexico. All machines had a direct link to a research network which is free from packet filtering and does not conduct egress filtering to block spoofed return IP addresses. Furthermore, the UNM measurement machines have IP addresses that are not bound to any interfaces in order to eliminate unsolicited network packets. For example, a measurement machine's kernel should never send a RST when it receives a SYN/ACK. The data set collected using the hybrid idle scan from these machines is a large-scale geographic pairing of many clients (in China and other countries) with many Tor relays and web servers around the world (mostly outside China). It complements the other data sets discussed below because it gives a complete cross-section of censorship between many clients and many servers. This data will be used to test Hypotheses 2 (*no geographical patterns in failures*) and 4 (*some failures are persistent*).

VPS in China: We rented a virtual private system (VPS) in China. The system was located in Beijing (AS 23028) and was used for our SYN backlog scans discussed in Section 5.3.1. Our VPS provider employed a transparent and stateful TCP proxy in front of our VPS which silently dropped unsolicited segments. We carefully implemented our SYN backlog scans so they first established state whenever necessary to be unaffected by the TCP proxy. These SYN backlog scans provide a dataset that speaks to our assumptions about how China blocks Tor. It complements the hybrid idle scan data set because, although the measurements are from a single client in China, it allows us see exactly how that client experiences the censorship. This data will be used to test Hypotheses 1 (*RSTs are treated the same as SYNs*) and 3 (*server to client blocking*).

Tor relay in Europe: We used a long-established Tor relay at Karlstad University in Sweden for our traceroute measurements discussed in Section 5.4.2. The relay has been part of the Tor network for several months, and using our VPS we manually verified it to be blocked in China. This data set shows blocking between one Tor relay and many clients in China. It complements the hybrid idle scan data set because access to the Tor relay allows us to collect more details about the blocking. This data will be used to test Hypotheses 4 (*some failures are persistent*), 5 (*some failures have diurnal patterns*), and 6 (*blocking is in the backbone*).

We now present our probing infrastructure as well as our measurement methodology used to investigate the theories posited in Section 5.3.

Hybrid Idle Scans

Recall that by using hybrid idle scans, we have more freedom in choosing clients in different regions to test their reachability to different servers. Our goal is to determine blocking of Tor relays (outside of China) from the perspective of a large

and geographically diverse set of clients (within China).

We are interested in knowing whether there exist different experiences of the censorship of Tor for different users in different regions. Past work showed that a small fraction of all Tor relays was accessible from a single vantage point in Beijing [107], but what about the rest of the country? Key questions are: how does the GFW's architecture and China's routing affect censorship in different regions?

IP address selection: We selected clients in China (CN), North America (NA), and Europe (EU). In order to be able to select random IP addresses in China without favoring specific locations—especially large cities featuring a vast number of allocated IP addresses—we divided the map of China into $33 * 65$ cells corresponding to one degree of latitude and longitude. We filled this grid with all IP addresses in MaxMind's database that were documented to be in China. Then, we collected IP addresses by randomly selecting a cell from our grid after checking that they employed global IPIDs. In an analogous manner, clients from the EU and NA were chosen by horizontally scanning these regions. After 24 hours, we gathered a pool of IP addresses that belonged to machines with a global IPID. Then, we continually checked the selected IP addresses for a 24-hour period to discard IP addresses that changed global IPID behavior, went down, or were too noisy. At the end we had 11 NA, 7 EU, and 161 CN clients to use for our measurements.

Servers were chosen from three groups: Tor relays, Tor directory authorities, and web servers. Tor relays were downloaded from a Tor relay status list [99]. We only selected relays with an uptime greater than five days. In order to select Tor relays in geographically diverse regions, we selected 10 Tor relays from Europe, 13 from the United States, 20 from Russia, and 101 from other countries. This way, our selected Tor relays were not biased toward Europe or the U.S., which exhibit more relays per capita than other regions. The 10 Tor authorities were obtained from the Tor source code. Web servers were chosen randomly from Alexa's top 50 websites in China [8].

Figure 5.2: The geographic distribution of all tested Tor relays (shown as onions) and of our global IPID clients in China (shown as red marks). Note that outside of Xinjiang the west of China has very little Internet penetration, which is why we have few data points in this region and the distribution is biased towards the eastern parts of China. (Map data © 2014 Google, INEGI)

All web server and Tor relay IP addresses were checked hourly to make sure that they stayed up for at least 24 hours before being selected for our measurement.

The geographic distribution of our Tor relays as well as all clients in China is illustrated in Figure 5.2.

Creating a complete bipartite graph: We used three machines at UNM (our university campus) to run the hybrid idle scan experiments. We started the experiments with 180 clients and 176 servers. Each day 20 clients and approximately 20 servers were selected for each of the machines. For 22 hours[5], every hour, we performed the hybrid idle scan for each possible pair of client and server. Every "scan round" performs: *1)* two minutes of hybrid idle scans, *2)* 30 seconds of sending RSTs to clear the server's backlog, and *3)* five seconds of testing the client to assure

[5]Two hours per day were reserved for server data synchronization.

that they remained online and kept their global IPID. Similar checks are performed to ensure that servers remain online throughout each experiment. At any given time, each IP address (client or server) was involved in only one test. After 27 days, each client's reachability was tested to all servers, *i.e*, *our clients and servers created a bipartite graph*. For more details about the experiment design refer to Chapter 4.

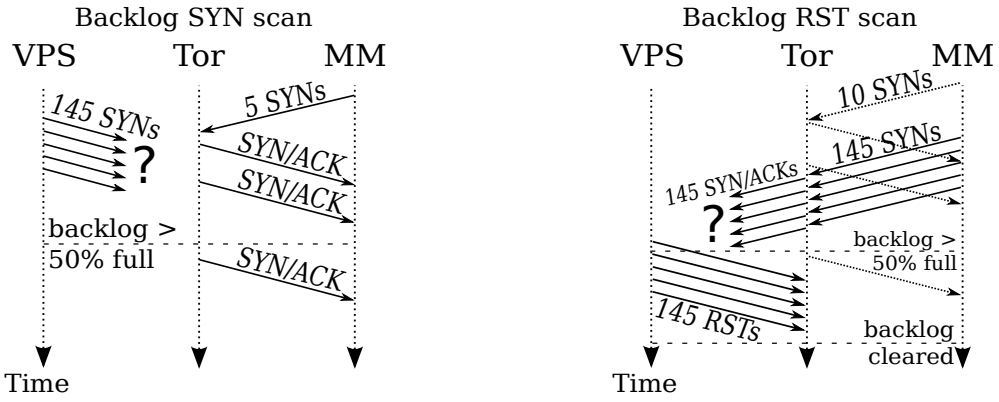
Pruning the data: We used the selected IP addresses throughout our experiments. Naturally, some of the hosts went down or were occasionally too noisy. Also, the host behind an IP address can change, *e.g.*, a client with a global IPID might lose its DHCP lease and get replaced with a client running a random IPID. To account for these issues, we perform tests throughout our experiments which cull out data points where basic assumptions are not met. For every server involved in the experiment, we had two checks: liveliness and the stable Tor flag test. After each scan, for five seconds we sent five SYN segments per second using UNM's unbound IP address. The data point passed the liveliness test only if it retransmits three or more SYN/ACKs. Also, if the server was a Tor relay, we verified that the relay was assigned the "Stable" flag (cf. Section 5.4.1).

For every client, for five seconds, we sent five SYN/ACKs per second using UNM's unbound IP address. We expect the client to respond with RST segments totaling in number to more than half the number of sent SYN/ACKs. If this is the case then the data point passes the client's liveliness test. The results of a scan were allowed into the data set only if both the client and server passed their checks. Note that each data point is one client and one server tested one time in a given hour. There was a several-hour network outage that caused a hole in a portion of one day of our data.

After culling out data that did not meet our basic assumptions, we were left with 36% of the total data collected. This 36% is the data described in Section 5.5 and used for our analysis.

Backlog Scans

After having presented the underlying side channel in Section 5.3.1, we now discuss the implementation of our two backlog scan types which can answer two questions, *1)* "Do SYN segments from China reach a Tor relay?" and *2)* "Do RST segments from China reach a Tor relay?". Basically, we answer both questions by first transmitting crafted TCP segments to a relay, thus manipulating its SYN backlog, and then querying its backlog size by counting the relay's SYN/ACK retransmissions. The conceptual implementation of both scan types is illustrated in Figure 5.3.



(a) SYN scan to infer whether SYN segments from VPS reach Tor. "MM" is our measurement machine.

(b) RST scan to infer whether RST segments from VPS reach Tor. "MM" is our measurement machine.

Figure 5.3: The two types of backlog scans we employ. The purpose of these scans is to verify if *1)* SYN segments from China reach a Tor relay and if *2)* RST segments from China reach a Tor relay.

SYN scan: The SYN scan—depicted in Figure 5.3(a)—is started by MM by sending five SYN segments to Tor in order to infer the relay's backlog size when under stress.[6] After a delay of approximately 500 ms, VPS proceeds by sending 145 SYN segments whose purpose is to fill the relay's backlog by more than half.

[6]We transmit five SYN segments rather than just one to account for packet loss.

Recall that the backlog size defaults to 256, so we only fill the backlog to 59%. That way, we can make the Tor relay's kernel prune MM's SYN segments, thus reducing their retransmissions. Finally, MM knows that VPS's SYNs reached the relay if the number of SYN/ACK retransmissions for its five SYNs is lower than five. Otherwise, VPS's SYNs did not reach the relay. This type of inference is necessary because, most of the time, China's GFW drops SYN/ACKs from known Tor relays.

RST scan: Our RST scan incorporates an additional step but is based on the same principle. As illustrated in Figure 5.3(b), MM starts by sending 10 SYN segments whose purpose is, analogous to the SYN scan, to monitor the relay's backlog size. Afterwards, MM proceeds by sending 145 spoofed SYN segments with VPS's source address. Note that we cannot send the SYN segments from VPS as they might be blocked. By sending spoofed SYN segments from an unfiltered network link, we can ensure that the segments reach the Tor relay. Upon receiving the SYN segment burst, the relay replies with SYN/ACK segments which we expect to be dropped by the GFW. In the final step, VPS sends a burst of RST segments to the Tor relay. The RST segments are crafted so that every RST segment corresponds to one of the relay's SYN/ACK segments. The purpose of the RST burst is to terminate all half-open connections, thus clearing the relay's backlog. Based on how many retransmissions we observe for the 10 "probing SYNs", we can infer whether the RST segments were dropped by the GFW or not. Receiving five retransmissions means that the backlog was not cleared and the RST segments were dropped. Receiving less than five retransmissions means that the backlog was successfully cleared and the RST segments were not dropped by the GFW. This kind of inference is necessary because machines outside China cannot measure directly what happens to RST packets sent from China, and machines inside China are very limited in their ability to infer what is happening on blocked IP address/TCP port pairs.

Implementation: We implemented our scans using a collection of bash scripts

and a patched version of the tool hping3 [87]. Accurate timing was crucial for our experiments. To keep the clock of our machines synchronized, we used the tool ntp which implements the network time protocol. Recall that the SYN backlog behavior we are exploiting is limited to Linux kernels (cf. Section 5.3.1). As a result, our scans targeted the subset of 94 out of our 144 Tor relays which are known to run Linux. Tor relays periodically publish their server descriptors—which includes their operating system—to all directory authorities so there is no need for us to guess the operating system of Tor relays.

Pruning the data: By pruning the backlog scan data, we aim to make sure that the relay runs an unmodified Linux TCP/IP stack. After scanning a relay, we send three "baseline SYNs" to it in order to query its original amount of SYN/ACK retransmissions. First, we discard scans in which the relay never sent five SYN/ACK retransmissions, Linux's default value since version 2.2. For example, we found embedded Linux relays which always retransmit SYN/ACK segments four times, regardless of their backlog size. Second, we also discard scans whose SYN/ACK retransmissions do not exhibit Linux's exponential backoff behavior. Third and finally, we discard scans where the relay was offline or other networking problems occurred. These three pruning steps discarded 774 out of all 2,094 scans (37%).

Traceroutes into China

We want to learn if there are *unfiltered routes* leading into China. To investigate this question, we used our Tor relay in Europe to run traceroutes to numerous destinations in China. After a country-wide scan, we obtained a list of 3,934 IP addresses in China that responded to SYN/ACKs and were distributed geographically in a diverse way, which served as our traceroute destinations. For every IP address, we ran two TCP traceroutes; one whose TCP source port was equal to the filtered Tor port 9001 and one whose TCP port was set to the unused and unfiltered port 9002. The

traceroutes had both their SYN and ACK bit set. We used a slightly modified version of the tool hping3 [87] to run the traceroutes as it allowed us to send TCP segments with a source port which is bound by the Tor process.[7] Starting on 4 May 2014, we ran the traceroutes on an hourly basis for two days, resulting in a total of $3,934 \cdot 24 \cdot 2 \cdot 2 = 377,664$ traceroutes. We determined where the traceroutes entered China using whois and round-trip time information. We culled out a small amount of data that did not enter China through a known backbone network, since all such data either appeared to enter China in Pasadena, California (a case we can handle but will require deeper analysis into whois records) or was destined for clients that we determined to actually be in Hong Kong.

5.4.3 Good Internet Citizenship

We took several steps to devise our scans to be minimally invasive. First, we set up a web server on our measurement machines whose index page informed visitors about our experiments. The page contained our contact information to provide alarmed network operators with an opportunity to contact us and opt out of our measurements. Furthermore, we carefully designed our measurements so that it is very unlikely that they harmed any computers or networks. Throughout the lifetime of our experiments, we did not receive any complaints. We discuss ethical aspects of our measurements in Section 5.6.3, and also in more details in Chapter 7.

5.5 Analysis and Results

We now analyze the three data sets we gathered; the hybrid idle scans, the backlog scans, as well as the traceroutes into China.

[7] We modified the tool to constantly increase the TTL of outgoing TCP segments. The default behavior is to wait for every hop to reply with a "TTL exceeded" ICMP message.

5.5.1 Hybrid Idle Scans

The hybrid idle scan data was collected from 15 March 2014 to 10 April 2014. One client was removed from the data because we determined that it was in Hong Kong and as a result not subject to the GFW's filtering.

Table 5.1 shows the results of our hybrid idle scans. The column $S \rightarrow C$ is short for **Server-to-client-dropped**, *None* means **No-packets-dropped**, $C \rightarrow S$ means **Client-to-server-dropped**, and *Error* simply means **Error**. In the table's rows, *CN* is short for China, *EU* means Europe, and *NA* means North America. As for the server types, *Tor−Dir* is a Tor directory authority, *Tor−Relay* is a Tor relay, and *Web* is a web server. Our results confirm that, in general, SYN/ACKs entering China from blacklisted IP address/TCP port pairs are blocked. Some web servers were censored, and some Tor nodes were censored outside China. This is to be expected because even in countries that do not perform nation-scale Internet censorship, organizations frequently take steps to filter material such as pornography or file sharing sites. Note that highly popular websites often contain material that is subject to censorship.

The most interesting result from the hybrid idle scans is that the **No-packets-dropped** case was measured all over the country without any noticeable geographic pattern. The geographic distribution of observed **No-packets-dropped** cases is shown in Figure 5.4. The case distribution closely matches the distribution of our clients which, in turn, matches the geographic Internet penetration patterns of China. This means that the failures in China's IP address/TCP port blacklisting mechanisms are not limited to one region or one network block. We provide a more thorough analysis in Section 5.5.2, which confirms Hypothesis 2.

We also observed that in many cases these filtering failures are persistent and *last throughout the day*. We witnessed four client/server pairs where all 22 measurements

Figure 5.4: The color temperature for clients corresponds to the number of observed **No-packets-dropped** cases over the entire experiment. No geographic or topological pattern is visible. Instead, the distribution matches the geographic Internet penetration patterns of China. (Map data © 2014 Basarsoft, Google, ORION-ME, SK planet, ZENRIN)

in a day returned **No-packets-dropped**. We redacted the clients' 16 least significant bits:

Client 58.193.0.0 (CN) → server 198.96.155.3 (CA)

Client 58.193.0.0 (CN) → server 161.53.116.37 (HR)

Client 58.193.0.0 (CN) → server 128.173.89.245 (US)

Client 121.194.0.0 (CN) → server 198.96.155.3 (CA)

This would give evidence towards Hypothesis 4, but our traceroute results reveal that CERNET does not perform the type of blocking we are measuring at

all so later in this section we will discuss similar failures in commercial networks. Clients 58.193.0.0 and 121.194.0.0 are part of the Chinese Educational and Research Network (CERNET). Server 198.96.155.3 is a long-established Tor exit relay at the University of Waterloo. 161.53.116.37 and 128.173.89.245 are Tor relays in Croatia and the U.S., respectively. There were also many instances where client/server pairs showed **Server-to-client-dropped** for most of the day but also showed **No-packets-dropped** once or a handful of times.

Table 5.1: Results from the hybrid idle scan measurement study.

Client	**Server**	$S \to C$ **(%)**	**None (%)**	$C \to S$ **(%)**	**Error (%)**
CN	Tor−Relay	116,460 (81.52)	555 (0.39)	786 (0.55)	25,061 (17.54)
CN	Tor−Dir	8,922 (64.91)	31 (0.23)	2,696 (19.61)	2,097 (15.25)
CN	Web	306 (1.23)	15,663 (62.95)	2,688 (10.80)	6,226 (25.02)
EU	Tor−Relay	18 (0.20)	8,589 (96.79)	22 (0.25)	245 (2.76)
EU	Tor−Dir	2 (0.25)	776 (96.76)	0 (0.00)	24 (2.99)
EU	Web	19 (1.23)	1,333 (86.28)	95 (6.15)	98 (6.34)
NA	Tor−Relay	45 (0.39)	11,022 (94.48)	33 (0.28)	566 (4.85)
NA	Tor−Dir	4 (0.37)	1,025 (94.73)	3 (0.28)	50 (4.62)
NA	Web	32 (1.52)	1,794 (85.06)	98 (4.65)	185 (8.77)

5.5.2 Temporal and Spatial Association

We now seek to answer the question of whether there are any temporal or spatial associations among the **No-packets-dropped** cases observed for Tor relays tested from within China.

Temporal association is shown in Figure 5.5. The probabilities are computed by a simple counting technique. We have the hourly count of the number of **No-packets-dropped** cases for each source. For each occurrence of **No-packets-dropped**, we check if there are other **No-packets-dropped** cases in the subsequent hours. We use 151 sources for this calculation, excluding the educational sources, which contained 353 **No-packets-dropped** cases in total. The final probabilities are averaged over all

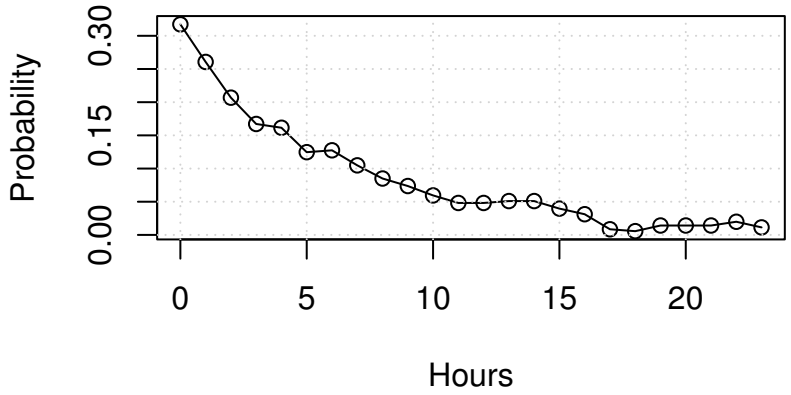


Figure 5.5: The temporal association between cases of **No-packets-dropped**. The x axis shows the amount of hours since the last **No-packets-dropped** case whereas the y axis shows the probability of observing another case of **No-packets-dropped**.

sources. With the increase in the lag amount in the x-axis, the probability decreases. This shows that **No-packets-dropped** cases generally happen in bursts of hours.

Spatial association is shown in Figure 5.6. We use the latitude and longitude of the sources as two-dimensional coordinates. The curvature of the earth is ignored while computing the distance between sources. For every source, we find the geographically K-nearest neighboring sources and average their count. We compute the Pearson's correlation coefficient between the count of **No-packets-dropped** cases for a source and the average of the same for the neighboring sources. Note that Pearson's correlation has a range of -1.0 to 1.0. Our maximum observed correlation value of 0.26 is, therefore, a very weak positive correlation and supports the fact that there is no significant geographical association between sources and their neighbors. With the increase of the neighborhood radius, the correlation decreases to below 0.1. Together with the fact that the cases of **No-packets-dropped** are distributed fairly evenly in all geographic regions (see Figure 5.4), this is strong support for Hypothesis 2.

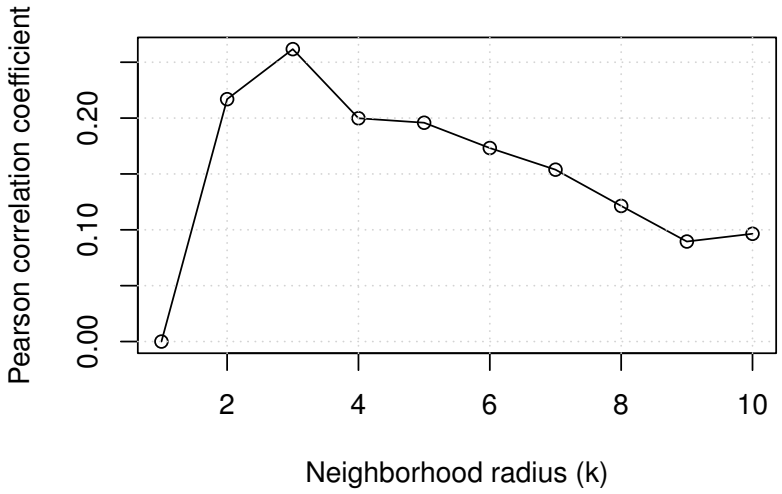


Figure 5.6: Spatial association between clients in China. The x axis shows the neighborhood radius (k) and the y axis shows the Pearson correlation coefficient.

5.5.3 SYN Backlog Scans

We began our backlog scans on 24 March 2014 and ran them twice a day with approximately 12 hours in between the scans until 10 April 2014. We gathered a total of 2,094 scans and after pruning, this effort yielded 1,320 scans (63%).

Reachable Tor Relays

Out of all 1,320 backlog scans, 33 scans (2.5%) to 12 unique IP addresses contained the respective Tor relay's SYN/ACK segments, indicating that no filtering was happening. Interestingly, 19 of these 33 scans targeted the directory authority 128.31.0.39 on port 9131. Only the RST scan and not the SYN scan yielded SYN/ACKs from the directory authority.

The results in Table 5.2 show that, in general, if a RST packet passes through the GFW then a SYN packet also will. This confirms one of the basic assumptions behind the hybrid idle scan, and confirms Hypothesis 1. Also, the fact that most

Table 5.2: Backlog scan results.

	RST passes	**RST dropped**
SYN passes	666 (80%)	39 (4.7%)
SYN dropped	68 (8.2%)	53 (6.4%)

Table 5.3: The results of our traceroute measurements.

	EDU Rand	**EDU Tor**	**COM Rand**	**COM Tor**
Stalled	1,061	1,045	111,133	163,095
Finished	428	433	53,479	429

SYNs were allowed to pass through the GFW confirms Hypothesis 3.

5.5.4 Traceroutes

Table 5.3 shows the results of our traceroute measurements. In the table, "EDU" indicates that the first hop in China in the traceroute is the educational and research network backbone, CERNET (210.250.0.0/16 or 101.4.112.0/24) or another scientific network called CSTNET (159.226.0.0/16). "COM" indicates that the first hop in China was a commercial backbone, one of: CNCGROUP (219.158.0.0/16), China Telecom/CHINANET (202.97.0.0/16), China Mobile Communications Corporation (211.136.1.0/24 or 221.176.23.0/24), or the China Telecom Next Carrying Network backbone (50.43.0.0/16). All other entry points were thrown out because they were actually in Hong Kong or Pasadena, and that usually indicated that the destination IP address was not in China or non-Chinese routing hops had not been properly culled. "Tor" means that the source port of the SYN/ACKs sent in the traceroute was the Tor port, and "rand" means that the source port was another port that the GFW does not filter. Thus, "Tor" traceroutes should always stop before the destination host if the filtering is effective on that route, and "rand" should reach the destination unless there are other types of filtering in play, such as ICMP filtering or firewalls not related to censorship. The elements in the table are the number of

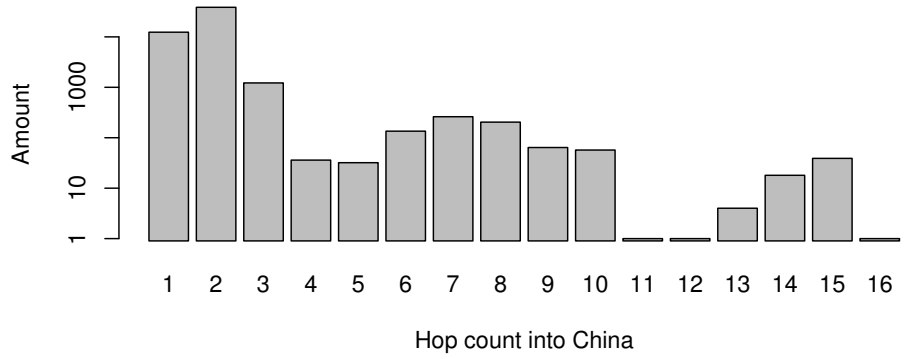


Figure 5.7: The amount of hops (log scale) in China, our filtered traceroutes could traverse. For example, a hop count of five means that a traceroute could successfully reach the fifth router inside China.

times that a traceroute reached all the way to the destination.

Surprisingly, the educational and research networks, in particular CERNET, do not seem to be implementing this type of filtering at all. The "Tor" and "rand" columns are nearly identical for the "EDU" traceroutes. The "COM" traceroutes, however, show that commercial networks are clearly censoring Tor by dropping SYN/ACKs. The "rand" traceroutes reached their destination 53,479 times, while the "Tor" traceroutes aimed at the same destinations only reached the destination end host 429 times. Similar to the hybrid idle scan results, these failures were all over the country and for any destination IP address where at least one failure was observed, the number of failures ranged from 1 to 48 (*i.e.*, all 48 hours of measurements). The number of failures in the most prominent destinations where the traceroute entered China on a commercial background included one instance where 48 failures were observed and two where 47 were observed. This means that sometimes the failures are relatively persistent, confirming Hypothesis 4.

Figure 5.7 shows the amount of hops into China, filtered "Tor" port traceroutes traversed before stalling. For each measurement of each hour of each day, we only add the data to Figure 5.7 if the "rand" traceroute reached the destination and the

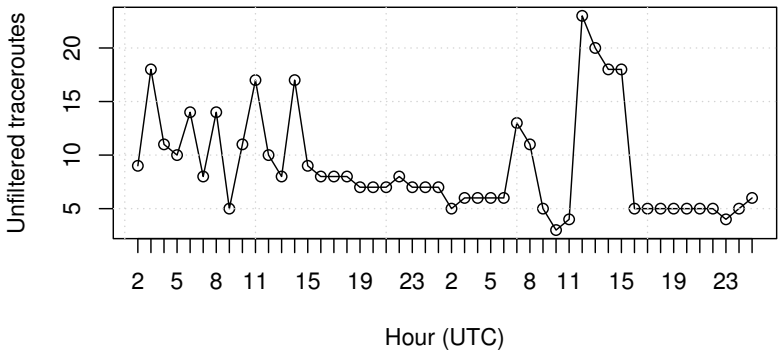


Figure 5.8: The amount of unfiltered traceroutes from our Tor relay to clients in China over time. A diurnal pattern is visible.

"Tor" traceroute did not. In most cases, the filtered packets make it two hops into China, confirming Hypothesis 6.

Figure 5.8 shows the number of failures for traceroutes that entered China on the commercial network backbone, per hour. The diurnal patterns apparent in the figure confirm Hypothesis 5. Note that 02:00 UTC is 10:00 (or, 10:00 am) in Beijing.

5.6 Discussion

We discuss three different aspects of our work in this section: what we learned about the filtering of Tor in China, what we learned about the architecture of the GFW, and ethical considerations.

5.6.1 Filtering of Tor in China

Our results suggest that the filtering of Tor in China has several interesting aspects, some of which may even be useful for circumvention efforts. We showed that the failures in the filtering occur in every part of the country, and they are sometimes

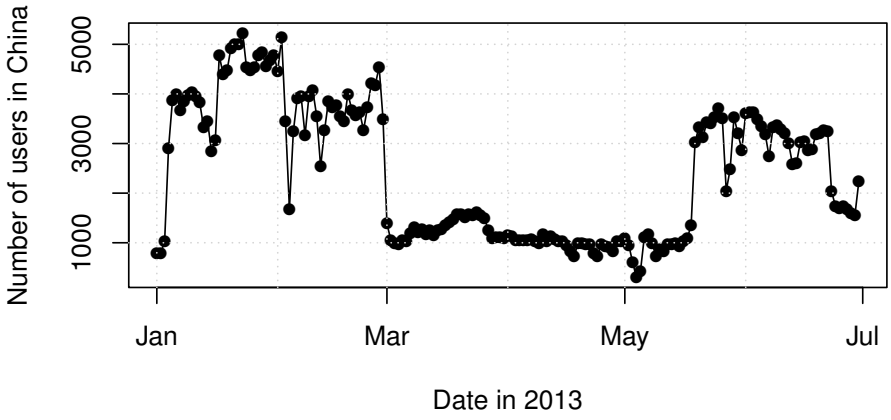


Figure 5.9: The amount of directly connecting Tor users over the first seven months of 2013. The diagram shows several spikes and a "valley" in between March and May.

intermittent and sometimes *persistent*. A historical example of intermittent failures is illustrated in Figure 5.9. The diagram shows the amount of directly connecting Tor users in China in the first seven months of 2013. A relatively stable "valley" in between March an May is clearly visible. This valley is surrounded by significantly higher usage numbers.

We also showed that this type of filtering does not occur on CERNET, the educational and research backbone of China's Internet. This might suggest that CERNET users can reach the Tor network, or it might suggest that CERNET employs a more sophisticated method for detecting and interfering with connections to the Tor network, perhaps something stateful and based on deep packet inspection.

Our results raise additional questions such as "is it possible to run a Tor relay in China?". In general, the Tor network represents a complete graph. As a result, every relay should be able to connect (and generally maintain connections) to all other relays in the network. Furthermore, relays must be able to connect to the directory authorities in order to upload their server descriptors. If CERNET is indeed whitelisted, a Tor relay inside CERNET might be able to successfully join

the Tor network. In addition, previous research suggested that domestic Tor traffic in China is not subject to blocking [107]. If filtering indeed happens at the Internet exchange point (IXP) level, as suggested by our data, it is not surprising that the GFW is generally unable to filter domestic network traffic as it typically does not reach IXP level[8] and is of significantly higher volume than international traffic. As a result, functioning Tor relays or bridges inside CERNET might be able to connect users in China to the rest of the Tor network.

5.6.2 The Architecture of the GFW

Our results also shed light on the architecture of the GFW, at least with respect to the mechanism that blacklists IP address/TCP port pairs. As discussed in Section 5.3, the three theories about how the GFW is architected are that *1)* the filtering occurs at choke points where undersea cables enter the country, *2)* the filtering occurs in the backbone in large IXPs, and *3)* the filtering occurs at a regional level. While our results show some filtering occurring many hops into China and some filtering occurring before packets can even enter China, the majority of the filtering happens about two hops into China (presumably at the large IXP in Beijing). Thus, Hypothesis 6 is most consistent with the theory that the filtering occurs in the backbone. Note that this observation is in accordance with other recent research efforts which focused on the GFW's DNS injection [11]. The small amount of routes that are filtered at the provincial level, which were also observed by Xu *et al.* [110], can be explained by the strategy employed by China's formerly second-largest ISP, CNCGROUP, which was recently bought by the largest (CHINANET).

While whitelisting would appear as persistent failures in the filtering and the filtering apparatus getting overloaded with traffic would appear as intermittent fail-

[8] We ignore routing phenomena such as "boomerang routing".

ures, the mix of intermittent failures and diurnal patterns with persistent failures suggests that routing is a major reason why the filtering fails. Hypotheses 4 and 5 are most consistent with the theory that the filtering occurs in the backbone, because provincial networks in China are very hierarchical [97] and undersea cables are few in number [6]. Hypothesis 2 is also most consistent with backbone-level filtering for this reason.

5.6.3 Ethical Considerations

Our work has two ethical considerations that need to be discussed. First, our SYN backlog scans briefly fill a Tor relay's backlog in order to be able to observe packet drops. In general, the rate at which we are sending SYN packets, without intention of completing a connection, is not enough to create a denial-of-service condition on any modern network stack.

Second, our idle scans create unsolicited traffic between a client and a server. This traffic—which can be observed by the censor—is only SYN/ACKs from the server to the client and RSTs from the client to the server. As a result, we are not causing any meaningful communication other than background noise as it is also caused by port scanning activity. In terms of the traffic that the censor sees, the hybrid idle scan technique is no different from if Tor relay operators performed simple connectivity measurements by directly sending SYN/ACKs.

For more detailed discussion on ethics, refer to Chapter 7.

5.7 Conclusion

In this chapter, we have characterized the mechanism that the Great Firewall of China uses to block the Tor network using a hybrid idle scan that can measure connectivity from the perspective of many clients all over China. We have also presented a novel SYN backlog idle scan that can infer packets received by a server without causing denial of service. These novel Internet measurement techniques open up whole new possibilities in terms of being able to measure the Internet from the perspective of arbitrary clients and servers. This is extremely important when it comes to characterizing and documenting Internet censorship around the world, because of the difficulty in finding volunteers geographically dispersed throughout a country.

We also evaluated our techniques which led to several new insights about the inner workings of the Great Firewall. Our data shows that *1)* at least several machines inside CERNET (China Education and Research Network) are able to connect to Tor relays, *2)* filtering seems to be centralized at the IXP level, and *3)* filtering is quite reliable with the Tor network being either almost completely reachable or almost completely blocked in different parts of the country.

Our code is available at: `http://cs.unm.edu/~royaen/gfw/`.

Chapter 6

Related Work

This chapter will give an overview of related work in idle and port scanning techniques (see Section 6.1) and censorship measurement (see Section 6.2).

6.1 Idle and Port Scanning

Port scanning is an important first step in most network attacks and in network analysis [73]. There has been a fair amount of techniques focused on port scanning a target host or a network. Port scans can be initiated from a single source and can be detected by using techniques such as anomaly detection [51]. They can also be distributed, meaning that the attacker uses multiple hosts to coordinate a stealth scans [38, 39, 47]. Javed *et al.* proposed a general approach for detecting distributed scans in which individual attack sources each operate in a stealthy, low-profile manner. Bhuyan *et al.* [17] created a survey on port scans and their detection methodologies.

In general, there have been many efforts to detect port scans. Staniford *et al.* [93] use simulated annealing to detect stealthy scans. Leckie and Kotagiri [54] present a

probabilistic approach for detecting port scans, and Muelder *et al.* [70] proposes a visualization approach. The scan behavior of Internet worms has been studied [88, 105, 42, 104], as has the scan detection problem at the backbone level [92, 91] and measurements of port scans and their side effects at Internet telescopes [72]. Jung *et al.* [46] describe an approach based on sequential hypothesis testing to detect port scans. To our knowledge, our techniques in Chapter 3 represent the first study to model idle scans that can be used for stealth scans but also for inferring IP-based trust relationships. Passively identifying hosts that have no routable IP address and are hidden by network address translation [14, 53] is a related problem to idle scans, but assumes a very different threat model where some amount of traffic can be viewed passively by the attacker. As discussed earlier, Antirez [12] proposed the first type of idle scan which we call an IPID idle port scan. Based on Antirez' work, we proposed a RST, SYN backlog, and a hybrid idle scans which were all explained in detail in previous chapters. The main benefit of using our hybrid idle scan is that it can be used to detect intentional packet dropping based on IP addresses and it requires no commonalities between the measurement machine's routes to the server or client and the routes between the server and client. Similar to our work is iPlane [59]. The iPlane project sends packets from PlanetLab nodes to carefully chosen hosts, and then compounds loss on specific routes to estimate the packet loss between arbitrary endpoints without access to those endpoints. This, however, does not detect IP-address-specific packet drops.

There exist other advanced methods for inferring information about remote networks and hosts. Qian *et al.* [79] demonstrate that firewall behavior with respect to sequence numbers can be used to infer sequence numbers and perform off-path TCP/IP connection hijacking. Chen *et al.* [21] use the IPID field to perform advanced inferences about the amount of internal traffic generated by a server, the number of servers in a load-balanced setting, and one-way delays. Morbitzer [68] explores idle scans in IPv6. Knockel *et al.* use a side-channel technique to infer

whether two machines are exchanging packets on the Internet [52]. Queen [102] utilizes recursive DNS queries to measure the packet loss between a pair of DNS servers, and extrapolates from this to estimate the packet loss rate between arbitrary hosts. Reverse traceroutes can be performed by spoofing return IP addresses and using the IP options for recording routes and timestamps [48]. De A. Rocha *et al.* [25] present a method for estimating the average variance and delay based on spoofed return IP addresses and the IPID field. Qian *et al.* make use of the IPID field to infer in which direction port blocking is happening in the context of email spamming [80].

6.2 Censorship Measurement

I consider censorship to be one of the most important issues facing the Internet today. Accordingly, there is a growing body of work in this area. With respect to deployed platforms that have been used for censorship, Sfakianakis et al. [89] designed CensMon that is a web censorship monitor which is run on top of PlanetLab [77]. Filastò and Appelbaum presented OONI [37] that can be used to measure traffic manipulation and content blocking. Herdict [45] and OpenNet [69] have been monitoring and reporting on Internet filtering and surveillance practices. The Chokepoint project [4] is focused on the collection, analysis, and reporting of information related to Internet blockages and network neutrality issues around the world. More recently, Anderson *et al.* [10] proposed to use the RIPE Atlas network [60] to measure censorship. I believe our work, idle scan inference techniques, is an effective method for measuring Internet censorship around the world. The most notable difference to previous work is that our measurement techniques do not require control over either machine which is part of the censored communication. While this enables large-scale distributed studies, it comes at the cost of reduced flexibility.

To the best of my knowledge, my work is the first to employ idle scan infer-

ence techniques for a large-scale Internet measurement study where the collected data represents the "view" of geographically distributed clients. Platforms such as DIMES [62], M-Lab [63], PlanetLab [61], and RIPE Atlas [60] have traditionally been the only way to measure from the perspective of a large number of clients, but they can be very limited, especially in non-Western regions of the Internet such as China. My work overcomes a fundamental limitation of Internet measurement: that measurements traditionally have only been possible from the perspective of the measurement machines under the control of the researchers.

Next we focus on related work specific to the Great Firewall of China(GFW). The GFW was first described in an article in 2600 magazine [98]. In 2006, Clayton, Murdoch, and Watson investigated the firewall's keyword filtering mechanism and demonstrated that it can be circumvented by simply ignoring the firewall's injected RST segments [23]. Clayton *et al.*'s study was limited to how the filtering works. *What* it filters was covered by Crandall *et al.* in 2007 [24], along with more details about routing. Using latent semantic analysis, the authors bootstrapped a set of 122 keywords which were used to probe the firewall over time. The study also shows that filtering is probably not happening at the border of China's Internet. Xu, Mao, and Halderman made an effort to pinpoint where exactly the filtering is h happening [110]. The authors came to the conclusion that most filtering is happening in border ASes but some filtering is also happening in provincial networks. Park and Crandall revisited the GFW's keyword filtering mechanism and discussed why the filtering of HTML responses was discontinued in late 2008 [74].

In addition to topology and HTTP filtering, another direction of research focused on how the GFW operates on the TCP/IP layer. In 2006, Clayton *et al.* already showed that the GFW is terminating suspicious HTTP requests using injected RST segments. Weaver, Sommer, and Paxson showed that it is possible to not only distinguish genuine from injected RST segments but also to fingerprint networking

devices injecting the segments [103]. More recently in 2013, Khattak *et al.* probed the GFW in order to find evasion opportunities on the TCP/IP layer [50]. Resorting to techniques first discussed by Ptacek and Newsham in 1998 [78], the authors showed that there are numerous evasion opportunities when crafting TCP and IP packets.

In addition to the design and topology of the GFW, some work focused on how the GFW blocks application protocols other than HTTP. In 2007, Lowe, Winters, and Marcus showed that the GFW is also conducting DNS poisoning [57]. A more comprehensive study was conducted by anonymous authors in 2012 [90]. The authors sent DNS queries to several million IP addresses in China, thereby demonstrating that the GFW's DNS poisoning causes collateral damage, *i.e.*, interferes with communication outside China. A similar DNS-related study, also by anonymous authors, was done in 2014 [11]. The authors attempted to localize the DNS injectors' location, extracted the GFW's DNS blacklist, and used side channel's in the GFW's design to estimate its design. Most work discussed so far treated the firewall as a monolithic entity. Wright showed in 2012 that there are regional variations in DNS poisoning, thus suggesting that censorship should be investigated on a more fine-grained level with attention to geographical diversity in measurements [109]. In addition to DNS and HTTP, the GFW is known to block the Tor anonymity network. Using a VPS in China, Winter and Lindskog [107] investigated how the firewall's active probing infrastructure is used to dynamically block Tor bridges.

6.3 Summary

To summarize, my dissertation work is the first work to enable measuring connectivity between two hosts on the Internet without having access to either of those hosts or being in the path between them. Idle scans, and, in general, TCP/IP side channels, offer many opportunities for Internet measurement. The aim of my dissertation

work is to serve as a foundation for this nascent research area.

Chapter 7

Ethical Considerations

This thesis proposes several experiments which involve sending unsolicited traffic to computers on the Internet. In particular, we proposed two types of experiments which require further discussion of potential ethical issues.

The first experiment is the *SYN backlog scan* which is discussed in Section 5.4.2. The backlog scan briefly fills the SYN backlog of a Tor relay in order to be able to determine if packets are dropped on their way to the relay. As long as the backlog does not become completely full, no harm is done. If however, the backlog is filled entirely, a computer would begin using SYN cookies [16] which reduces the throughput of TCP connections. To minimize the chance of harming a computer or network while scanning it, we carefully crafted the parameters of our scans to make them as uninvasive as possible. As a result, it is very unlikely that our experiments ever fill a computer's SYN backlog.

The second experiment which should be discussed are our *idle scans* which create unsolicited traffic between two computers. A passive adversary observing an idle scan could believe that the two scanned machines are deliberately communicating with each other. This could have negative consequences if a censor believes that a user is

communicating with a sensitive or forbidden IP address. However, it is unlikely that a censor would come to such a conclusion as the client-to-server traffic consists only of RST segments and the server-to-client traffic only consists of SYN/ACK segments. An adversary would not witness a full TCP handshake, let alone any actual data transfer. As a result, we believe that it is unlikely that a censor would consider our idle scans to be malicious. Nevertheless, we stress that these experiments should only be carried out after assessing the political situation in the respective country and verifying that something as simple as TCP connections are unlikely to lead to political consequences.

Durumeric *et al.* [30] provide a more comprehensive ethics discussions of Internet scans in general. The work in this thesis follows these best practices. The Menlo report [29] proposes a framework for ethical guidelines for computer and information security research based on the 1979 Belmont report [85]. Wright *et al.* [109] attempt to start a discussion about the potentially significant ethical and legal concerns which are often faced in censorship measurement.

More related to networking research, Allman and Paxson [9] discuss issues around sharing network measurement data. The authors suggest a set of guidelines which should be followed when providing data as well as when using provided data. When it comes to raw network traces produced by TCP/IP side channel scans such as in this thesis, there are a number of tradeoffs to consider when releasing the data. By obfuscating the IP address, but not the network that the IP is on and other important topological information, various risks could be reduced without reducing the value of the data. For example, the risk that listing the IPs of the clients makes them targets for future studies could be mitigated through obfuscation. On the other hand, if there is a risk that network administrators will mistakenly attribute communication attempts to the client, having the raw data available through the IP address of the measurement machine along with an explanation of data collection can help the

owners of client machines to make the case that they did not initiate communication. We leave a release of our raw data and the associated ethical concerns for future work.

Chapter 8

Conclusion and Future Work

The thesis statement that this dissertation began with is: *because modern network stacks have shared resources, there is a wealth of information that can be inferred off-path by both attackers and Internet measurement researchers.*

We showed that current designs of network protocols are susceptible to side channels that leak information. We built practical methods—in the form of idle scans—that employ shared resources to learn about the communication status between remote hosts. In particular, we presented and evaluated a novel side channel based on the Linux kernel's SYN backlog which enables indirect detection of packet loss. We then explained our hybrid idle scan which is a non-intrusive method for detecting intentional packet drops between two IP addresses on the Internet where neither is a measurement machine. We applied intervention analysis based on an autoregressive-moving-average model to increase the practicality of our techniques even in the face of traffic noise. We then described the real-world application of the SYN backlog and hybrid and idle scans to a large-scale Internet measurement problem in which we measured the connectivity between the Tor anonymity network and clients in China. These techniques enabled us to increase the community's understanding of

how the GFW is architected and how its blocking of the Tor network looks from different clients all over China. Thus the above thesis statement was demonstrated to be true.

8.1 Future Work

The research presented in this thesis can be continued in many directions. With respect to ethical concerns, we plan to explore the idea of using routers with a global IPID as a client. That way, no individual end user can possibly experience negative consequences for unsolicited traffic, which would make our measurements even safer for users. We conducted an initial and promising study to investigate the possibility of finding global IPID routers. A more detailed experiment needs to be performed, however.

Another research direction can be to analyze censorship for popular domains using side channels. The goal could be to create a comprehensive, real-time, and global view of the censorship of popular domains which also makes it possible to *compare* censorship incidents of countries. Only by using side channels, especially idle scans, are we able to use geographically distributed machines, and we are no longer limited to controlling limited vantage points. Such a system could be a significant contribution to longitudinal censorship measurement systems. Such longitudinal studies are essential to understand global trends in Internet censorship practices.

With respect to the Great Firewall of China, we were able to prove and disprove several hypotheses about its architecture. However, numerous questions still remain unanswered, *e.g.*: 1) what role does BGP and routing in general play in the GFW?, 2) why does the GFW (mostly) only filter SYN/ACKs coming into the country from Tor relays, but not SYNs or RSTs going out?, 3) are the routers that drop a Tor relay's SYN/ACKs the same routers that have port mirroring and do deep packet

inspection?, 4) how and when is the GFW obtaining the Tor network consensus?, 5) is active probing also happening for Tor relays?, and 6) has the blocking changed in the past two years?

Finally, recall that our techniques are currently limited to testing basic IP connectivity. Thus, we can only detect censorship on lower layers of the network stack, *i.e.*, before a TCP connection is even established. To overcome this limitation, future work could focus on finding more side channels that can be used in higher layers of the network stack. Such side channels would significantly improve measurement flexibility and enable an entirely new array of measurements.

References

[1] Censorship Wiki. https://censorshipwiki.torproject.org.

[2] Linux kernel source tree. http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/net/ipv4/inet_connection_sock.c?h=4d0fa8a0f01272d4de33704f20303dcecdb55df1#n562.

[3] tcp(7) - Linux man page. http://linux.die.net/man/7/tcp.

[4] The Chokepoint project. https://chokepointproject.net/.

[5] VirtualBox. https://www.virtualbox.org.

[6] Submarine Cable Map 2014. Available at http://submarine-cable-map-2014.telegeography.com/.

[7] Alexa. Alexa top 1,000,000 sites. http://www.alexa.com/topsites.

[8] Alexa. Alexa top sites in China. http://www.alexa.com/topsites/countries/CN.

[9] M. Allman and V. Paxson. Issues and Etiquette Concerning Use of Shared Measurement Data. In *Internet Measurement Conference*. ACM, 2007.

[10] C. Anderson, P. Winter, and Roya. Global Censorship Detection Over the RIPE Atlas Network. In *Free and Open Communications on the Internet*. USENIX, 2014.

[11] Anonymous. Towards a Comprehensive Picture of the Great Firewall's DNS Censorship. In *Free and Open Communications on the Internet*. USENIX, 2014.

[12] Antirez. new TCP scan method, 1998.

[13] arma. Research problem: Five ways to test bridge reachability. Tor Blog, 1 December 2011, available at https://blog.torproject.org/blog/research-problem-five-ways-test-bridge-reachability.

[14] S. M. Bellovin. A Technique for Counting NATted Hosts. In *Internet Measurement Workshop*. ACM, 2002.

[15] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Munoz, S. Owre, H. Ruess, J. Rushby, V. Rusu, H. Saidi, N. Shankar, E. Singerman, and A. Tiwari. An Overview of SAL. In C. M. Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, 2000. NASA Langley Research Center.

[16] D. J. Bernstein. SYN Cookies. http://cr.yp.to/syncookies.html.

[17] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita. Surveying port scans and their detection methodologies. *The Computer Journal*, page bxr035, 2011.

[18] S. Bisgaard and M. Kulahci. *Time Series Analysis and Forecasting by Example (Wiley Series in Probability and Statistics)*. Wiley, 2011.

[19] S. Bratus, M. E. Locasto, M. L. Patterson, L. Sassaman, and A. Shubina. Exploit Programming: from Buffer Overflows to Weird Machines and Theory of Computation. *USENIX ;login:*, 36(6), 2011.

[20] I. Chang, G. C. Tiao, and C. Chen. Estimation of time series parameters in the presence of outliers. *Technometrics*, 30(2):193–204, 1988.

[21] W. Chen, Y. Huang, B. F. Ribeiro, K. Suh, H. Zhang, E. de Souza e Silva, J. Kurose, and D. Towsley. Exploiting the IPID Field to Infer Network Path and End-System Characteristics. In *Passive and Active Network Measurement*. Springer, 2005.

[22] Y. Choi. From NuSMV to SPIN: Experiences with model checking flight guidance systems. *Form. Methods Syst. Des.*, 30(3):199–216, 2007.

[23] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies*. Springer, 2006.

[24] J. R. Crandall, D. Zinn, M. Byrd, E. Barr, and R. East. ConceptDoppler: A Weather Tracker for Internet Censorship. In *Computer and Communications Security*. ACM, 2007.

[25] A. A. De A. Rocha, R. M. M. Leão, and E. De Souza e Silva. A Non-cooperative Active Measurement Technique for Estimating the Average and Variance of the One-Way Delay. In *Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*. Springer, 2007.

[26] L. de Moura. SAL Tutorial. CSL technical note, Computer Science Laboratory, SRI International, 2004. Available at http://sal.csl.sri.com/doc/salenv_tutorial.pdf.

[27] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, 1975.

[28] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*. USENIX Association, 2004.

[29] D. Dittrich and E. Kenneally. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical report, U.S. Department of Homeland Security, Aug 2012.

[30] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. In *USENIX Security Symposium*. USENIX Association, 2013.

[31] S. C. Edmund, E. M. Clarke, N. Sharygina, and N. Sinha. State/Event-based Software Model Checking. In *Integrated Formal Methods*, pages 128–147. Springer-Verlag, 2004.

[32] 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. Available at http://arxiv.org/abs/1312.5739.

[33] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall. Detecting Intentional Packet Drops on the Internet via TCP/IP Side Channels. In *Passive and Active Measurement Conference*. Springer, 2014.

[34] 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 *USENIX Security Symposium*. USENIX Association, 2010.

[35] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall. Large-scale Spatiotemporal Characterization of Inconsistencies in the World's Largest Firewall. *CoRR*, abs/1410.0735, 2014. Available at http://arxiv.org/pdf/1410.0735.

[36] J. Fallows. The Connection Has Been Reset. Atlantic Monthly, March 2008.

[37] A. Filastò and J. Appelbaum. OONI: Open Observatory of Network Interference. In *FOCI*. USENIX, 2012.

[38] C. Gates. *Co-ordinated port scans: a model, a detector and an evaluation methodology*. PhD thesis, 2006.

[39] C. Gates. Coordinated Scan Detection. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 09)*, Feb. 2009.

[40] Y. Gilad and A. Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In *Privacy Enhancing Technologies Symposium*. Springer, 2012.

[41] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[42] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley. Worm Detection, Early Warning and Response Based on Local Victim Information. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference*, pages 136–145, Washington, DC, USA, 2004. IEEE Computer Society.

[43] H. Hamed, E. Al-Shaer, and W. Marrero. Modeling and Verification of IPSec and VPN Security Policies. In *ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols*, pages 259–278, Washington, DC, USA, 2005. IEEE Computer Society.

[44] C. M. Hurvich and C.-L. Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989.

[45] T. Hwang. Herdict: A distributed model for threats online. *Network Security*, 2007(8):15–18, 2007.

[46] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[47] M. G. Kang, J. Caballero, and D. Song. Distributed Evasive Scan Techniques and Countermeasures. In *DIMVA '07: Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 157–174, Berlin, Heidelberg, 2007. Springer-Verlag.

[48] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. Anderson, and A. Krishnamurthy. Reverse Traceroute. In *Networked Systems Design & Implementation*. USENIX Association, 2010.

[49] R. A. Kemmerer. Shared resource matrix methodology: an approach to identifying storage and timing channels. *ACM Trans. Comput. Syst.*, 1(3):256–277, 1983.

[50] S. Khattak, M. Javed, P. D. Anderson, and V. Paxson. Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion. In *Free and Open Communications on the Internet*. USENIX Association, 2013.

[51] H. Kim, S. Kim, M. A. Kouritzin, and W. Sun. Detecting network portscans through anomoly detection. In *Defense and Security*, pages 254–263. International Society for Optics and Photonics, 2004.

[52] 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)*. USENIX Association.

[53] T. Kohno, A. Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Symposium on Security and Privacy*, May 2005.

[54] C. Leckie and R. Kotagiri. A probabilistic approach to detecting network scans. In *IEEE/IFIP Network Operations and Management Symposium. (NOMS).*, pages 359–372, 2002.

[55] J. Lemon. Resisting SYN flood DoS attacks with a SYN cache. http://people.freebsd.org/~jlemon/papers/syncache.pdf.

[56] I. A. Lovecruft. Automating Bridge Reachability Testing, 2012.

[57] G. Lowe, P. Winters, and M. L. Marcus. The Great DNS Wall of China. Technical report, New York University, 2007.

[58] G. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.Org LLC, Sunnyvale, CA, USA, 2009.

[59] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *Operating Systems Design and Implementation*. USENIX Association, 2006.

[60] Global RIPE Atlas Network Coverage. Available at https://atlas.ripe.net/results/maps/network-coverage/.

[61] World map of PlanetLab nodes. Available at https://www.planet-lab.org/generated/World50.png.

[62] The DIMES project: Active Agents by Countries in Last 7 Days. Available at http://www.netdimes.org/new/?q=node/52.

[63] M-Lab Platform: Server Map. Available at http://www.measurementlab.net/infrastructure.

[64] MaxMind. How accurate are your GeoIP databases? Available at http://www.maxmind.com/en/faqaccurate.

[65] MaxMind. GeoIP2 City, 2014.

[66] S. McCamant and M. D. Ernst. A Simulation-based Proof Technique for Dynamic Information Flow. In *PLAS 2007: ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, San Diego, California, USA, 2007.

[67] S. McCamant and M. D. Ernst. Quantitative Information Flow as Network Flow Capacity. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation*, Tucson, AZ, USA, 2008.

[68] M. Morbitzer. TCP Idle Scans in IPv6. Master's thesis, Radboud University Nijmegen, The Netherlands, 2013.

[69] M. Morley, R. Atkinson, D. Savic, and G. Walters. OpenNet: An application independent framework for hydraulic network representation, manipulation and dissemination. In *Hydroinformatics 2000 Conference, University of Iowa, Iowa City, USA*, pages 23–27, 2000.

[70] C. Muelder, K. Ma, and T. Bartoletti. Interactive visualization for network and port scan detection. In *Proceedings of 2005 Recent Advances in Intrusion Detection*, page 05, 2005.

[71] D. Nobori and Y. Shinjo. VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls. In *Networked Systems Design and Implementation*. USENIX, 2014.

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

[73] S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier. An experimental evaluation to determine if port scans are precursors to an attack. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 602–611. IEEE, 2005.

[74] J. C. Park and J. R. Crandall. Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China. In *Distributed Computing Systems*. IEEE, 2010.

[75] V. Paxson. End-to-end Internet packet dynamics. *SIGCOMM Comput. Commun. Rev.*, 27(4):139–152, 1997.

[76] A. Peterson. China has almost twice as many Internet users as the U.S. has people, 2014.

[77] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman. PlanetLab Architecture: An Overview. Technical Report PDN–06–031, PlanetLab Consortium, 2006.

[78] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[79] Z. Qian and Z. M. Mao. Off-path TCP Sequence Number Inference Attack. In *Security & Privacy*. IEEE, 2012.

[80] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. Investigation of Triangular Spamming: a Stealthy and Efficient Spamming Technique. In *Symposium on Security and Privacy*. IEEE, 2010.

[81] R. W. Ritchey and P. Ammann. Using Model Checking to Analyze Network Vulnerabilities. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 156, Washington, DC, USA, 2000. IEEE Computer Society.

[82] D. Robinson, H. Yu, and A. An. Collateral Freedom, 2013.

[83] J. Rushby. SAL Tutorial: The Needham-Schroeder Protocol in SAL. CSL technical note, Computer Science Laboratory, SRI International, Menlo Park, CA, 2003. Available at http://www.csl.sri.com/users/rushby/abstracts/needham03.

[84] J. Rushby. SAL Tutorial: Analyzing the Fault-Tolerant Algorithm OM(1). CSL technical note, Computer Science Laboratory, SRI International, Menlo Park, CA, 2004. Available at http://www.csl.sri.com/users/rushby/abstracts/om1.

[85] K. J. Ryan, J. V. Brady, R. E. Cooke, D. I. Height, A. R. Jonsen, P. King, K. Lebacqz, D. W. Louisell, D. W. Seldin, E. Stellar, and R. H. Turtle. The Belmont Report: Ethical Principles and Guidelines for the Protection of Human Subjects of Research, 1979.

[86] SAL project. Examples. http://sal.csl.sri.com/examples.shtml.

[87] S. Sanfilippo. hping, 2006.

[88] S. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, French Riviera, France, September 2004.

[89] A. Sfakianakis, E. Athanasopoulos, and S. Ioannidis. CensMon: A Web Censorship Monitor. In *FOCI*. USENIX, 2011.

[90] Sparks, Neo, Tank, Smith, and Dozer. The Collateral Damage of Internet Censorship by DNS Injection. *SIGCOMM Computer Communication Review*, 42(3):21–27, 2012.

[91] A. Sridharan and T. Ye. Tracking port scanners on the IP backbone. In *LSAD '07: Proceedings of the 2007 workshop on Large Scale Attack Defense*, pages 137–144, New York, NY, USA, 2007. ACM.

[92] A. Sridharan, T. Ye, and S. Bhattacharyya. Connectionless port scan detection on the backbone. In *Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International*, pages 10 pp.–576, April 2006.

[93] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *J. Comput. Secur.*, 10(1-2):105–136, 2002.

[94] D. Sutherland. A Model of Information. In *Proceedings of the 9th National Computer Security Conference*, 1986.

[95] The Tor Project. Relay descriptor archives. https://metrics.torproject.org/data.html#relaydesc.

[96] The Tor Project. Tor directory protocol, version 2. https://gitweb.torproject.org/torspec.git/blob/HEAD:/dir-spec.txt.

[97] Y. Tian, R. Dey, Y. Liu, and K. W. Ross. Topology Mapping and Geolocating for China's Internet. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1908–1917, 2013.

[98] Tokachu. The Not-So-Great Firewall of China. 2600 Magazine, Winter 2006–2007.

[99] TorStatus. Tor Network Status. http://torstatus.blutmagie.de.

[100] K. Wagstaff. Web mystery: China Internet traffic winds up in Wyoming. NBC News, available at http://web.archive.org/web/20140123101929/http://www.nbcnews.com/technology/welcome-wyoming-how-chinas-great-firewall-could-have-sent-web-2D11970733.

[101] Q. Wang, Z. Lin, N. Borisov, and N. J. Hopper. rBridge: User Reputation based Tor Bridge Distribution with Privacy Preservation. In *Network and Distributed System Security*. The Internet Society, 2013.

[102] Y. A. Wang, C. Huang, J. Li, and K. W. Ross. Queen: Estimating Packet Loss Rate between Arbitrary Internet Hosts. In *Passive and Active Network Measurement*. Springer, 2009.

[103] N. Weaver, R. Sommer, and V. Paxson. Detecting Forged TCP Reset Packets. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA*. The Internet Society, 2009.

[104] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 3–3, Berkeley, CA, USA, 2004. USENIX Association.

[105] D. Whyte, E. Kranakis, and P. C. van Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network. In *Proc. of the 12th annual Network and Distributed System Security symposium*, pages 181–195, 2005.

[106] P. Winter and J. R. Crandall. The Great Firewall of China: How it Blocks Tor and Why it is Hard to Pinpoint. USENIX ;login: 37 (6), 2012, pp. 42-50.

[107] P. Winter and S. Lindskog. How the Great Firewall of China is Blocking Tor. In *Free and Open Communications on the Internet*. USENIX Association, 2012.

[108] J. C. Wray. An Analysis of Covert Timing Channels. In *Security & Privacy*. IEEE, 1991.

[109] J. Wright. Regional Variation in Chinese Internet Filtering. Technical report, University of Oxford, 2012.

[110] X. Xu, Z. M. Mao, and J. A. Halderman. Internet Censorship in China: Where Does the Filtering Occur? In *Passive and Active Measurement Conference*. Springer, 2011.

[111] Xylon Pan. Over the wall and implementation of the principles of the router (in Chinese). Available at http://xylonpan.com/2013/12/14/, accessed 1 August 2014.

[112] A. Yumerefendi, B. Mickle, and L. P. Cox. TightLip: Keeping Applications from Spilling the Beans. In *Networked Systems Design and Implementation (NSDI)*, 2007.

[113] T. Zhu, D. Phipps, A. Pridgen, J. R. Crandall, and D. S. Wallach. The Velocity of Censorship: High-Fidelity Detection of Microblog Post Deletions. In *USENIX Security Symposium*. USENIX Association, 2013.