ON UNDERSTANDING THE INTERNET VIA EDGE MEASUREMENT

by

MATTHEW SARGENT

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

August 2015

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis of

MATTHEW SARGENT

candidate for the **DOCTOR OF PHILOSOPHY** degree*

Committee Chair: Michael Rabinovich

Committee Member: Mark Allman

Committee Member: Vincenzo Liberatore

Committee Member: Gultekin Ozsoyoglu

Committee Member: Francis Merat

Committee Member: Marvin Schwartz

Date: 05/14/2015

*We also certify that written approval has been obtained for any propriety material contained therein.

Contents

List of Tables

List of Figures

Acknowledgments

It brings me great joy knowing that this document only exists due to contributions from countless individuals I have met and worked with over my lifetime. Thank you to everyone who had a hand in helping me get to where I am today. You all have my deepest gratitude.

MATTHEW SARGENT

Case Western Reserve University

August 2015

On Understanding the Internet Via Edge Measurement

Abstract

by

MATTHEW SARGENT

The design philosophy of the Internet enables the transmission of packets between "smart" network edges via a "dumb" middle, or core portion of the network. Core networks are ultimately responsible for the single task of routing packets between hosts that are not physically connected. They do so by operating at the network layer of the Open System Interconnection model using the Internet Protocol. Routing requires no understanding of what type of traffic is being transmitted, but only of where a packet is ultimately destined. Whereas the core network's primary job is to correctly route packets, edge networks have additional "smarts" as they contain myriad end point devices each responsible for implementing network protocols and applications. Common tasks like transmitting email, streaming video, or visiting web pages must all be implemented by end point devices. Thus, edge networks and the devices they contain are largely responsible for the evolution of overall network characteristics.

In this dissertation we examine several changes edge networks have undergone recently and leverage empirical measurements to understand how edge network evolution has affected various network characteristics. We begin by studying traffic characterization and

connection performance on a residential Fiber-To-The-Home network. We then shift our attention to the Transmission Control Protocol (TCP) and how its performance has been affected by a mismatch between the protocol specification and packet round trip times on the modern network. Next, we examine ways network applications drive sending patterns in TCP connections and how these patterns affect overall TCP performance. We conclude with two studies focusing on security related topics for edge networks. First, we develop and test a methodology that aims to broadly understand the port filtering policies in place throughout the network through passive traffic observation. Finally, we undertake a study of Internet Group Management Protocol (IGMP) traffic characteristics and demonstrate how IGMP traffic can be used to launch several attacks from edge networks. The list of topics we tackle is by no means exhaustive, but each topic does represent important work that allows us to keep our understanding of edge network behaviors up-to-date.

Chapter 1

Introduction

The design philosophy of the Internet enables the transmission of packets between "smart" network edges via a "dumb" middle, or core portion of the network [Cla88]. Core networks are "dumb" in that they are ultimately responsible for the single task of routing packets between hosts that are not physically connected. They do so by operating at the network layer of the Open System Interconnection (OSI) model [Zim80] using the Internet Protocol (IP) [Pos81a]. Routing requires no understanding of what type of traffic is being transmitted, but only of where a packet is ultimately destined. Whereas the core network's primary job is to correctly route packets, edge networks have additional "smarts" as they contain myriad end point devices that are responsible for implementing network protocols and applications. Common types of tasks like transmitting email, streaming video, or visiting a web page must all be implemented by end point devices. Regardless of how much complexity is introduced by devices in edge networks, routers must still only understand the IP layer to function properly.

Modern networks have come a long way since the "smart" versus "dumb" paradigm was introduced. Certainly one can argue that the core network has gained "smarts" and is no longer strictly operating on the IP layer. Routers are capable of inspecting packets and enacting policies based on application behavior, data rates, allowed types of traffic, or

other traffic characteristics. Additionally, middleboxes will sometimes change a packet as it is transmitted along its path [DHB+13]. For example, network address translation (NAT) [SH99] allows multiple hosts on a private network to share a single public IP address. Each time a local host sends a packet to the wide area network, the NAT is responsible for rewriting the source IP address of the packet to the reflect the public IP address rather than the local, private address placed in the packet by the end host. Meanwhile, when packets come to the NAT from the wide area network, the NAT is reponsible for rewriting the packets so they are destined to the correct local host. The NAT maintains a table of ongoing connections for all local hosts—based on port numbers—and looks up the correct mapping in this table when rewriting a packet.

While these tasks move beyond "dumb" routing behavior, the core network is still "dumb" relative to the edges and hence the "smart" versus "dumb" paradigm still shapes how each portion of the network is able to evolve over time. The types of innovations possible in the core network are limited by the need to maintain the underlying routing functionality, which means that even as changes are adopted they are often rolled out slowly.[1]

As networks change, network measurements remain an important tool to maintain our understanding of network characteristics. Sometimes, behaviors on the network can be measured in ways that highlight how the community had previously misunderstood some aspect of the network (e.g., due to network behavior evolving over time). For example:

- Packet reordering, at one point, was thought to be rare and typically caused by faulty equipment in the core network. However, in [BPS99] the authors show that packet reordering is not only more common than previously thought, but also usually caused by parallelism between routing components in the core. This example illustrates that while the old model was not necessarily incorrect when initially developed, the network evolved which invalidated the model. Only through empirical observation

[1] For example, IPv6 [DH98] adoption by many different metrics is still low globally [CAZ+14] despite the IPv6 specification being over 20 years old.

was the original, invalid model corrected.

- Network session arrivals were once all modeled as Poisson processes. This modeling has roots in telephony where call arrival processes initiated by humans are well modeled as Poisson processes. Work by Paxson and Floyd [PF95] shows that while user initiated TCP sessions can be modeled by Poisson processes, other types of connection arrivals, which are quite common, deviate from a Poisson model and hence treating the overall rate of connection arrivals as Poisson processes produces an invalid traffic model.

- In the absence of empirical measurement, anecdotal observations sometimes lead to midguided conclusions about general network properties. In [GN12] the author demonstrates that filling oversized buffers in a home router by initiating a large file transfer adds a burdensome amount of delay to other network traffic. From this anecdote, a general model of Internet router queues as highly bloated is formed (the so-called "bufferbloat" phenomenon). However, recent empirical measurements [All13, CR13, HPC$^+$14] suggest that bufferbloat is a more modest problem than the home router anecdote suggests. These works show that while bufferbloat can be demonstrated using artificially generated traffic, naturally occurring bufferbloat does not happen as often or add as much delay as conjectured. Empirical measurement allowed researchers to better understand a modest network problem was not as great a threat as initally thought to be.

While not an exhaustive list, these examples highlight cases where empirical measurements have been used to keep network understanding up-to-date and based on the realities of a changing network. Often changes are driven by the edge network since, in contrast to the core, devices on edge networks are capable of rapid innovation and evolution. End hosts operate knowing that they can send arbitrary data without requiring changes in the core. New devices, protocols, or applications can all be added at the edge of the network

and expect to operate correctly as long as the other end of a connection also implements the proper protocols. This freedom to send arbitrary traffic from the edge means that overall network characteristics are largely shaped by the types of traffic most in demand by edge network devices. Only by taking *continued* network measurements can we hope to keep pace with these rapidly changing networks and keep our understanding of network characteristics up-to-date.

In this dissertation, we leverage measurements from edge networks to better understand their evolution and how these changes have shaped network characteristics. We focus on separate, but inter-connected areas where edge networks have been rapidly evolving in recent years.

Available bandwidth: The bandwidth residential customers have at their disposal has been increasing in recent years as companies begin installing fiber optic networks and other high-bandwidth technologies in more areas. Current residential fiber connections can offer bandwidth up to 1 Gbps, which is up to 1,000 times as much as previous access technologies. An increase this large effectively leaps past the bandwidth requirements for modern applications. Two natural questions arise when considering this large jump forward: (i) How will end hosts make use of the available bandwidth? and (ii) will the implementation of underlying protocols enable users to fully use the bandwidth?

Chapter 2 studies user behavior and protocol performance on the Case Connection Zone (CCZ) [Cas], a fiber optic network connecting about 90 homes with bi-directional 1 Gbps connectivity. This experimental network allows us to build a preliminary understanding of protocol and user behavior on a fiber network. Even though the number of homes is modest, the duration of our dataset allows us to develop insight about the operation of a Fiber-To-The-Home network.

Transport protocols: When studying transport layer characteristics, one must be mindful that the transport layer is directly impacted by both a changing network and changing applications. In other words, the layers above and below the transport layer have an impact

on protocol performance.

For example, TCP has parameters which are tuned based on some notion of the delay a connection can expect while traversing the core network infrastructure. TCP may use improper parameters if the underlying network inflates or reduces the amount of time a typical packet needs to traverse the network. Incorrectly tuned parameters may have a negative effect on TCP's performance. In the other direction, transport protocols rely on applications to provide the data to be sent across the network. Previous work [PF01] suggests that a reasonable way to simulate application behavior is to treat TCP connections as bulk transfers sampled from a log-normal distribution. However, applications have changed their behavior over time and have begun to incorporate pauses between bursts of data. As TCP has been designed and tuned with bulk transfers in mind, we must understand whether application patterns are having an effect on TCP performance. We study TCP performance and how it is affected from each of these directions.

In Chapter 3, we examine whether decreasing round trip times (RTTs) on the network have caused TCP's initial retransmission timeout (RTO) to have a positive or negative impact on connection performance. The original TCP specification sets the initial RTO recommendation based on decades-old data [Jac88, PA00]. We gather datasets spanning seven years to investigate a new value for the initial RTO and study the effect a possible new initial RTO would have on modern network traffic.

In Chapter 4 we study whether the underlying mental model of TCP connections being "bulk transfer" holds given a modern network load. We study edge network traffic as it crosses into the wide area network from two different vantage points to understand how applications drive TCP's sending patterns. We observe that while bulk transfer connections do exist, the general model for a TCP connection should not be assumed to be bulk based on the variety of applications that use TCP. We also demonstrate how long pauses in TCP sending often has a negative effect on TCP's performance.

Policy and security threats: Edge network operators have incentives to remove unwanted

traffic from their network. This could be to to reduce the operational load, to protect end users from attacks, and to prevent their network resources from being misused to launch an attack on other networks. Since edge networks are free to generate arbitrary types of traffic, operators are mostly reactionary to security threats. They are not typically able to prevent an attack unless they first know how an attack operates. Even when an attack is well understood, sometimes operators are unable to filter out all packets that could be linked to an attack without also harming other end users.[2]

This dissertation contains two separate efforts to understand security related to edge networks. In Chapter 5 we aim to broadly understand fine-grained network policy by leveraging passive data collection at a large darknet. We first develop a methodology to detect port filtering policies that exists on the network based on the expectation of malware indiscriminately scanning unused address space. Using our methodology, we study filtering related to a large malware outbreak, the Conficker worm [CAI13]. After presenting the methodology and assessing the state of Conficker filtering, we examine the efficacy of our methodology for detecting filtering in the general case.

Chapter 6 studies the behavior of Internet Group Management Protocol (IGMP) [Dee89, Fen97, CDK+02] traffic on the network. IGMP traffic is used between routers to communicate multicast routing information. However, routers implementing IGMP have also been found to respond to arbitrary requests from end hosts [MDP+11, MVdSD+09]. Responding in this manner creates a vector that can be exploited to launch a distributed denial of service attack [Naz08] against an unsuspecting victim. An attacker A leverages two key properties associated with IGMP responses when launching an attack: reflection and amplification. First, A can reflect IGMP packets to a victim IP address V by spoofing IP address V as the source address of a probe it sends to an IGMP enabled router. The

[2]For example, DNS amplification attacks [AAC+06] could be prevented by filtering out all DNS traffic on an operator's network. However, this would also prevent all legitimate DNS traffic from being transmitted and would prevent most network connections.

IGMP router will send its response packet to V, and V will never receive packets from an IP address associated with the attacker A. Reflection in this case allows A to remain hidden from V while still successfully sending unwanted packets to V. Additionally, IGMP responses are often larger than the original requests sent by A. In this way, the amount of traffic A sends is amplified by the IGMP enabled routers before the traffic ultimately reaches V.

To understand IGMP behavior and its potential as an attack vector, we scan the entire IPv4 address space for hosts that will openly respond to IGMP request packets. We combine the responses we collect with a series of follow up probes to understand the characteristics of the potential vulnerability. We use response information to craft a theoretical attack that leverages IGMP as an amplification vector to highlight IGMP as a potential security threat.

The topics in this dissertation leverage a variety of edge network measurements to understand the behavior, characteristics and operation of modern edge networks. While we choose topics deliberately with the goal of studying the edge network through a broad lens, we note that the topics we choose are not meant to be exhaustive. Certainly other topics must continue to be studied in order to keep our understanding of the network up-to-date. Regardless, the results presented in this dissertation represent important work that improves our understanding of modern edge network behavior.

Chapter 2

Fiber-To-The-Home Traffic: Characterization and Performance

Fiber-To-The-Home (FTTH) networks are on the cusp of bringing significantly higher capacities to residential users compared to traditional commercial broadband offerings. Commercial ISPs have been offering on the order of ten megabits/second fiber service for several years (e.g., Verizon's FiOS, AT&T's Uverse). However, several research projects have started to connect residential users with significantly more capacity. For instance: Google has connected 850 homes near Stanford university [Goob] and 50K–500K more homes in Kansas City [gooa] with 1 Gbps fiber. Also, Chatanooga's power utility has connected 100K homes via fiber and started to offer network services [Cha10] and Case Western Reserve University has an operational testbed of roughly 90 homes near campus connected via fiber [Cas]. Finally, the United States' "National Broadband Plan" calls for 100 million residential connections to have at least 100 Mbps downlink and 50 Mbps uplink networks—or 1–2 orders of magnitude more than current commodity networks—by 2020 [Nat].[1]

As we have thought about these networks of the (near) future, we find ourselves coming back to two basic questions:

[1] The work in this chapter resulted in the following published papers: [SSDA12, SA14].

- *What will users do with significantly higher capacity?* Traditionally, residential network capacity has lagged behind content. In other words content providers have been producing ever increasing quality of content (e.g., HD and 3D video) in advance of commodity residential networks ability to transmit the content. The envisioned FTTH networks effectively leap-frog our current content and hence a natural question is whether and how users will leverage the increased capacity.
- *Are our protocols up to the task of utilizing significantly higher bandwidth in edge networks?* In other words, do the processes we impose in our general protocols (e.g., TCP, HTTP) impose a bound on performance that makes it difficult to use significantly higher capacity?

In this chapter we tackle both of these big picture questions in the context of monitoring a neighborhood connected via FTTH, the Case Connection Zone (CCZ) [Cas]. This experiment connects each of roughly 90 residences adjacent to the Case Western Reserve University campus via a bi-directional 1 Gbps fiber link. After discussing the CCZ network, our data collection and the calibration of our data in the next section we then address the above questions from a number of angles in the remaining sections in the chapter.

We stress that this chapter is an initial study leveraging a network that is modest in size and collecting data from a larger set of users would clearly be better. However, our goal is to gain an *initial empirical understanding* of the workings of a high capacity FTTH network *in the wild*. This study is (as far as we know) the first word on the topic and by no means anywhere near the last word.

2.1 Data

We use a packet-level monitor to record the traffic at the border between homes on the CCZ and the broader Internet. Each of the CCZ homes is connected via a 1 Gbps fiber. These fibers ultimately come together in a switch which is served by a 1 Gbps link to the broader

network. Traffic is mirrored to a switch port where our monitor can observe and record the traffic. The mirrored traffic stream is our only visibility into the CCZ network. We do not have direct access to hosts, routers or NATs within the CCZ network. The switch can represent a choke point given the mismatch between the aggregate capacity to the residences and the capacity to the Internet. However, as detailed below, aggregate traffic is generally around 10 Mbps when averaged across each day, which is well within the capabilities of our measurement apparatus. We collect two general kinds of data from January 25, 2011 through August 31, 2012, as detailed in the next two subsections. Additionally, data from September 1, 2012 to December 31, 2012 became available before conducting our performance analyses and we include these additional 4 months of data in all analyses in § 2.4 onward.

We stress that we are not claiming our data is in some way "*representative*" or "*typical*". We present our analysis as an initial look at a single FTTH edge network. As we know from years of Internet measurement studies, no single vantage point will give a conclusive picture. Therefore, this study provides only an initial look at these emerging networks. Indeed we encourage the community to replicate our analyses in additional FTTH environments to develop a broader sense of the area. Additionally, we note that this study also provides new data about residential networks, as others have studied, e.g., [MFPA09].

2.1.1 Protocol Behavior Logs

First, we use Bro 1.5.2 [Pax99] to continuously monitor the CCZ network and record various traffic logs using its collection of protocol analyzers. We denote this dataset $\mathcal{L}_x$ where x identifies the particular protocol log. We use the following logs in our study:

Connection Logs: $\mathcal{L}_c$ includes information about each transport-layer connection. For each connection these logs include the start time, involved IP addresses and port numbers, number of bytes transmitted in each direction, duration of the connection and ancillary

information (e.g., whether TCP's three-way handshake was successfully completed).

HTTP Logs: $\mathcal{L}_h$ includes records for each HTTP [FGM+99] transaction. These records include the IP addresses and port numbers of the underlying TCP connection, the type of request (GET, POST, etc.), the URL being requested, the response code from the server and the size of each request and response.

BitTorrent Logs: $\mathcal{L}_b$ includes records for BitTorrent [Coh08] transactions. These records include IP addresses and port numbers for all observed peers as well as myriad information about the protocol, such as *have*, *choke* and *unchoke* messages and handshaking information.

We additionally recorded SMTP logs, but found only scant amounts of SMTP traffic and hence do not use these logs in our analysis. Finally, we log all DNS transactions, but we have not included any analysis of these in this chapter.

The log data—especially $\mathcal{L}_c$—requires calibration before use such that measurement errors and ambiguities[2] do not skew our analysis. This turned into an arduous multi-step process as outlined below.

Bro will report in $\mathcal{L}_c$ information for all attempted connections. For 16 days during our collection period we hit a bug in Bro whereby each arriving packet was reported as its own "connection". While often this manifests after some particular time within the day we remove all such logs from further analysis as obviously Bro was not working properly. Since we cannot readily re-create the situation that triggers this bug we were not able to determine its cause. It may be caused by some network effect and hence bias our analysis by preventing us from observing that phenomenon. However, given the small number of days removed (2.8%) we do not believe this has a large biasing effect on our overall insights. Further, even if we understood the cause, the data is left in such a state such that accurately analyzing it would be impossible.

[2]We are not the first to notice such in the Bro connection logs. E.g., see the discussion in [APT07] about the ambiguities of determining a connection's final "state"—e.g., "properly terminated"—from these logs.

For the remaining days in the $\mathcal{L}_c$ dataset we find a large number of "connections" that are essentially bogus for the purposes of further analysis. For instance, scanning traffic that sends a SYN to a non-active service port will show up in the connection logs as a "connection", but accomplishes no useful work. We filtered connections from our analysis for five basic reasons:

Filter F.1: As noted above, $\mathcal{L}_c$ includes both successful and unsuccessful connections (e.g., port scanning, or contacting an unresponsive host). This latter connection type typically manifests as either (i) having a duration that is either zero or unknown or (ii) reporting unknown values for the number of bytes sent in one or both directions. Since these entries are incomplete, we remove them from further analysis.

Filter F.2: While monitoring the network, Bro logs unexpected events in its "notice" and "weird" logs. We can then use these unexpected events to further remove connections from $\mathcal{L}_c$ that may skew our analysis. An example unexpected event that we have found particularly useful is an entry that indicates Bro has observed a TCP ACK for sequence number x, but has only observed data sequence numbers through y (for $x > y$). In constructing the $\mathcal{L}_c$ logs Bro will assume all bytes through x have been transmitted. However, given the missing data this may be a wrong assumption. We assume that in the cases where Bro reports a gap the difference is caused by an errant ACK and not by measurement loss (since measurement-based loss is rare, see § 2.1.2). We verified our heuristic by spot checking potentially problematic entries in $\mathcal{L}_c$ with packet-level traces (see § 2.1.2) and we indeed find connections with relatively few packets and a large errant ACK, which corrupts the byte counts.

Filter F.3: Given that one errant packet can lead to a gross mis-calculation of a connection's size we started an augmented logging scheme on January 11, 2012 using Bro's "conn-stats" policy. Using this the $\mathcal{L}_c$ logs include packet counts (in each direction). This makes it straightforward to filter out cases where we observe a disproportionate amount of data volume for the number of packets observed.

Filter F.4: Additionally, the logs in $\mathcal{L}_c$ contain Bro's version of each connection's "history". The history records (for each direction) whether Bro has observed: SYNs (and SYN+ACKs), data-carrying packets, ACKs, FINs and resets. We can then use this to ensure connections progress as expected. We adopt the following criteria (and order) for removing connections from further analysis: (1) Connections with no SYN. (2) Connections with no SYN+ACK. (3) Connections reporting multiple SYN+ACKs with intervening packets (i.e., not simply retransmitted SYN+ACKs). (4) Connections reporting non-zero data bytes sent for a given direction, but for which no data-carrying packets are observed. (5) Connections that have no history of acknowledging transmitted data.

Filter F.5: The above checks winnow $\mathcal{L}_c$ to what looks to be a reasonable set of connections in the logs taken after January 11, 2012—i.e., when we started recording packet counts and hence could leverage those in our filtering process. However, we are left with a small number of connections that if accurate would reflect an extremely fast sending rate (e.g., just under 1 Gbps). We suspect these connections are not accurate since after January 11, 2012 we no longer find such high transmission rates. Therefore, we adopted the following strategy. We observe the fastest connection post-January 11, 2012 is 142 Mbps. Based on this, we set a per-connection threshold of 178 Mbps—25% higher than the maximum observed rate—and remove from further analysis any connection that occurs before January 11, 2012 and exceeds the threshold. This is a fairly crude filter. However, given the lack of such high transmission rates after January 11, 2012 we are confident that Bro erred in some way when logging these connections.

In Table 2.1 we present a high-level summary of the connections filtered from further analysis by our calibration strategy. Since we used a slightly different strategy for the logs before and after January 11, 2012 we list the results separately. We find that a significant fraction of the connections are removed for various reasons. This is not surprising given previous analysis of similar connection logs (e.g., [APT07] shows more than 90% of logged connections are caused by scanners and hence are junk). We find the bulk of the

Filter Type	Pre-Jan/11	Post-Jan/11
All Conns.	849M	445M
F.1	365M	126M
F.2	11.6K	5.5K
F.3	—	15.5K
F.4	30M	17.2M
F.5	3.8K	—
Remaining	454M	302M
	53.5%	67.8%

Table 2.1: Connection filtering breakdown.

filtered connections are removed by filter F.1 due to incomplete information or by filter F.4 for showing an unexpected history. Since the logs before January 11, 2012 do not include packet counts filter F.3 is not employed. Likewise, filter F.5 was added as a heuristic in the absence of packet counts and hence is not used after January 11, 2012. In total we find nearly 756M legitimate connections over our 19 month data collection.

Additionally, recall that we find single connections with rates of at most 142 Mbps and hence set a filtering threshold of 178 Mbps for a single connection in the pre-January 11, 2012 logs. Our final dataset contains 68 connections that show performance between 142–178 Mbps. We consider these connections to be somewhat suspect. However, since the number is small they do not overly skew our results in general.

Finally, as with any long-term measurement effort we find glitches that cause some of our Bro logs to be "short"—meaning that either the log completes before the day is over or there is a gap in recording traffic in the middle of the log. In some cases this can be because there was no traffic (e.g., a general power or network outage). In other cases this is because of mundane logistical issues such as the logging disk becoming temporarily full. For short days we have no reason to believe the data that was recorded is bogus. Further, we find no evidence of a network effect causing short logs and therefore do not believe these drop outs reflect a biasing of the data. Therefore, in most instances in this paper we use these logs in our analysis. However, we do exclude logs that are short by more than 10 minutes from longitudinal per-day analyses as comparing a full day's worth of activity

to a partial day's activity can clearly be problematic and lead to wrong conclusions. We find 34 short days in our 19 month dataset.

2.1.2 Packet Traces

Our second source of data is packet-level traces. The torrent of traffic precludes the capture of all packets for the entire measurement period. We therefore collect packet traces from the 11^{th} through the 18^{th} of each month, as follows. We divided each day in the collection period into six hour blocks and collect a one-hour trace starting at a random time within each block. While we captured full packet payloads, saving all such traces quickly became logistically burdensome. We therefore randomly chose one trace to retain in full for each day and stripped the payload from the remaining three traces. This leaves us with 7 hours of full payload traces and an additional 21 hours of header-only traces for every month in our collection. In addition, for the last five months of our data collection we have captured TCP SYN, FIN and RST packets for the entire period from the 11^{th} to the 18^{th} of each month to better understand the device population in the CCZ (see § 2.2.1).

Failing to record all packets that cross our monitoring point during our observation period can lead to biased or wrong conclusions. Therefore, before using our packet traces we must assess measurement-based packet loss. While *tcpdump* reports a number of losses this is often not telling as it is difficult for the *tcpdump* application to understand (and hence count) what was not observed. Rather, we analyze the traces themselves for signs of missing packets. In particular, we analyze TCP traffic for cases where we observe an acknowledgment for data that never appears in the trace. These "gaps" represent cases where the ultimate recipient clearly received the data, but the data was not recorded in our traces.

Bro's "gap analysis" only works for packet traces that contain full packet payloads as the analysis is part of the payload reassembly procedure. Therefore, we analyze the 133 traces in our corpus that contain payload. Further, Bro's analysis only covers TCP

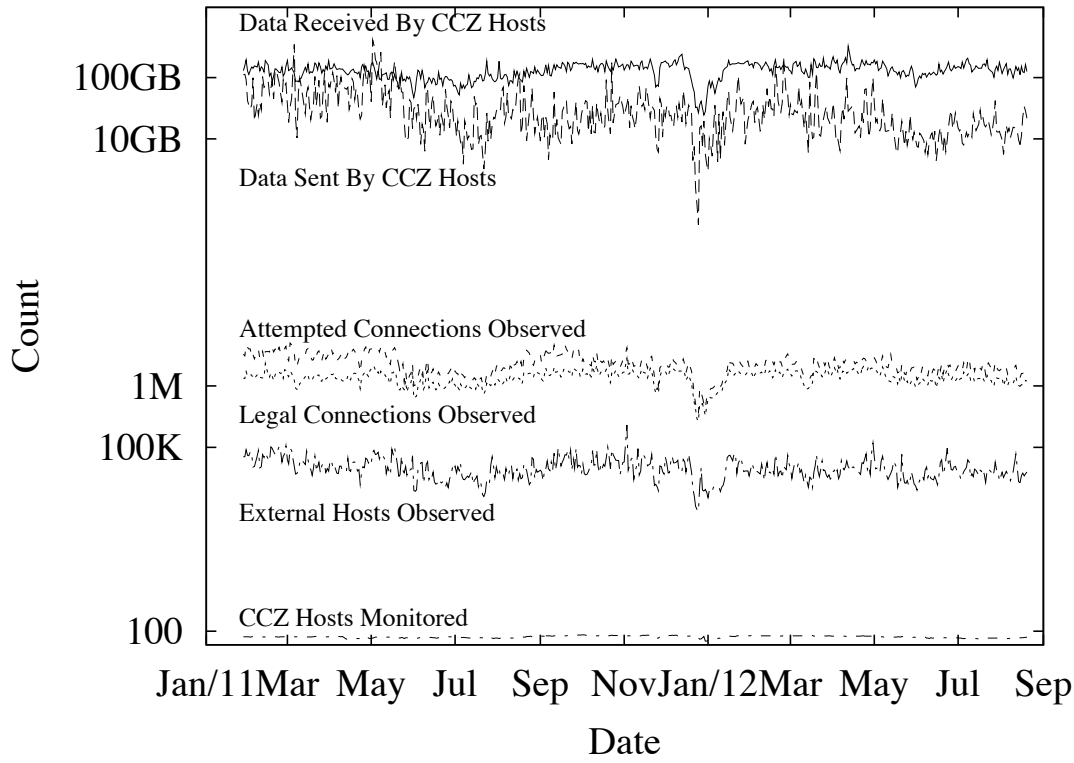

Figure 2.1: Overview of CCZ traffic.

traffic as it has sequence numbers that make it readily amenable to such analysis. Therefore, we stress that our analysis of measurement-based loss is not comprehensive. Rather, it is suggestive of the loss rates we expect from the collection apparatus. In 46 of the 133 traces we find no measurement-based loss. In the remaining 87 traces we find loss rates of (i) less than 0.001% in 62 traces, (ii) $[0.001\%, 0.01\%)$ in 21 traces and (iii) $[0.01\%, 0.05\%)$ in 4 traces. Therefore, while individual measurement-based loss events may impact individual analyses we undertake, we conclude that in a general and statistical sense the measurement-based loss rate is low enough to not impact the insights we derive from our dataset.

2.1.3 A First Look

Figure 2.1 gives a sense of the overall CCZ network and traffic. The remainder of the chapter will further analyze these aspects of the dataset, but here we aim to provide an

overview to build the reader's intuition. First, we observe 90 local IP addresses active on each day of our dataset. This is as expected based on the neighborhood size and because each house is provided with a standard router that does NATing for all internal devices. On median we observe traffic to 43K remote peers per day with the 5^{th} percentile at 22K and the 95^{th} percentile at 84K. Additionally, we observe a median of 2.3 million connections in $\mathcal{L}_c$ each day. Of these, a median of 1.4 million—or 63%—are valid (per the discussion in § 2.1.1). Finally, the top two lines on the figure show the data volume from/to CCZ hosts on a daily basis. The amount of data sent by CCZ hosts is roughly 23 GB per day on median, with a maximum of 411 GB. Meanwhile the amount of data received by CCZ hosts is 134 GB per day on median, with a maximum of 324 GB.

2.2 Origins and Destinations

Our first set of analyses aims to understand traffic sources and destinations CCZ users employ.

2.2.1 Devices

We know that there are roughly 90 homes connected to the CCZ network, but we do not know the number of devices connected within these homes—even though this is key to understanding how to interpret the observed traffic. The CCZ project provides each residence with a router which connects in-home devices to the fiber link via WiFi or wired Ethernet.[3] This router acts as a NAT such that each house uses only a single public IP address. We use a technique similar to that described in [MSF11] to count hosts behind these NATs. For each of our 530 hour long packet traces we run *p0f* [Zal06] to obtain SYN-based operating system determinations for each connection. Further, if payload is present—as it is for

[3]Note, we have no direct access to these in-home devices and therefore cannot collect data the device may log.

133 of our traces—*p0f* reports the HTTP user agent string associated with HTTP requests. Finally, for the last four months of our dataset we captured full 24 hour traces that contain all TCP SYN, FIN and RST packets in addition to the four sample hours of all packets. These give us a more comprehensive view for the SYN-based fingerprinting.

To determine the number of devices behind a particular NAT we start with the number of operating systems produced by the SYN-based signatures, denoted D_i for each CCZ IP address i. This is a lower bound on the number of devices. For instance, if we find a particular CCZ IP address x has traffic from both Windows 7 and OSX we can determine that at least two devices are present behind the NAT, but we cannot determine how many of each device is present. Therefore, we set $D_x = 2$.

For the connections on which we also have HTTP user agents we seek opportunities to augment our count by noticing that one OS has traffic from multiple browsers and assuming each device would have only one primary browser this indicates multiple devices. Continuing the example from above if we note OSX traffic from Chrome, Firefox and Safari then we would change OSX's device count contribution in D_x from 1 to 3 devices. Therefore, in total for this example we have $D_x = 4$. This use of the user agent string can in one sense be viewed as a lower bound. Similar to not being able to tease apart two Windows 7 machines based only on SYN signatures we cannot tease apart two Windows 7 machines both running Chrome, even with the extra user agent information. On the other hand, if our assumption that each device runs only a single browser in general is wrong we are overestimating the number of devices by attributing different browser traffic to different devices. We stress, however, that we aim to only roughly determine the number of devices behind each NAT, hence small errors are not problematic.

Finally, we note that our methodology is conservative for most of our observation period because our traces cover only 4 hours each day in which we can used SYN-based fingerprints and we have only a single random hour per day of user agent information. Therefore, we are no doubt undercounting the number of devices behind each NAT. This is

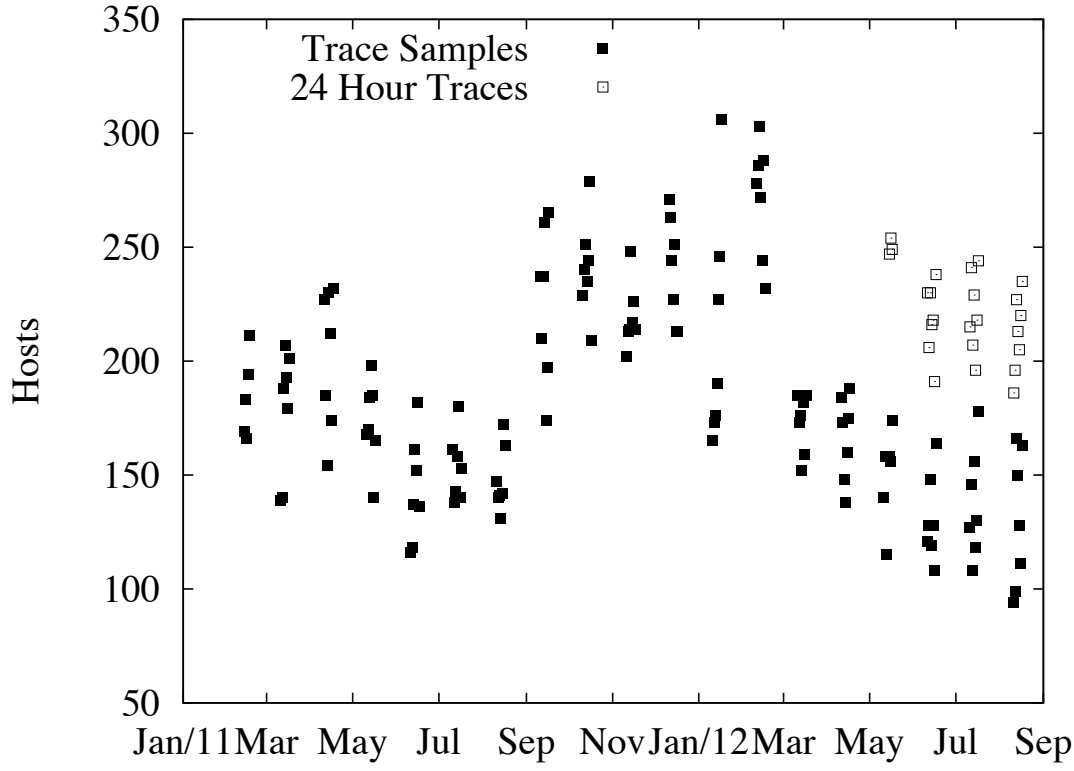

Figure 2.2: Number of unique devices observed.

likely especially true for mobile devices that may not be around when our traces are being taken. This is mitigated in the latter portion of our dataset by augmenting our count with 24 hour SYN/FIN/RST traces.

Figure 2.2 shows the number of devices we find to be active on each day we collect packet traces. The filled points are derived from our four hour per day trace samples. We find that like many of our analyses, there are dips in the plot that are dictated by the academic calendar. For instance, before summer 2011 we generally observe 160–220 devices per day. In the summer of 2011 and 2012 we find the user population drops off to around 140–180 devices per day. In the fall of 2011 we observe 200–300 devices per day, a healthy increase over the previous academic year. We find the January and March 2012 device counts to be low. This is explained by the January data being collected before spring semester classes started and the March data being collected over spring break.

The open points on the plot for the last four months of our dataset are derived from

OS	Min	Mean	Max
Windows 7/8/XP	30	40	55
OSX	10	20	28
iOS	9	16	25
Linux	3	11	23
Other	0	12	20

Table 2.2: Minimum, mean and maximum percentage of OS observed per day.

the 24 hour SYN/FIN/RST traces. These show the impact of using only a four hour sample of packet traces on the overall count. Using the full 24 hour traces (coupled with the one sample hour of user agent information) the count increases by 37–98%. In other words, in the best case the four hour traces are finding roughly three-quarters of the hosts and in the worst case these only capture half the devices.[4] On average we find that residences each have at least 2–3 active devices.

We next turn to understanding the device population in terms of operating systems. This gives us some idea about the general makeup of the user population (e.g., we find some technical people using Linux), as well as the prevalence of mobile devices versus more traditional computers. Table 2.2 shows the breakdown of the minimum, maximum and average percentage of various operating systems per day in our dataset. We find variability across time—which we largely ascribe to our data collection methodology (i.e., using a one-hour trace sample per day). As discussed above, with a more comprehensive view we believe the variability would be reduced. In general each OS's maximum population is 2–3 times its minimum population and the mean is the middle of the two ends of the spectrum. Also, we note that the median, while not shown, is $\pm 2\%$ of the mean in all cases.

2.2.2 Peer Location

We next assess the location of the remote peers in our $\mathcal{L}_c$ dataset. In particular, we assess the prevalence of remote peers that correspond to end-user devices and not servers. We

[4]Note, obtaining additional user agent information may well increase the estimate further.

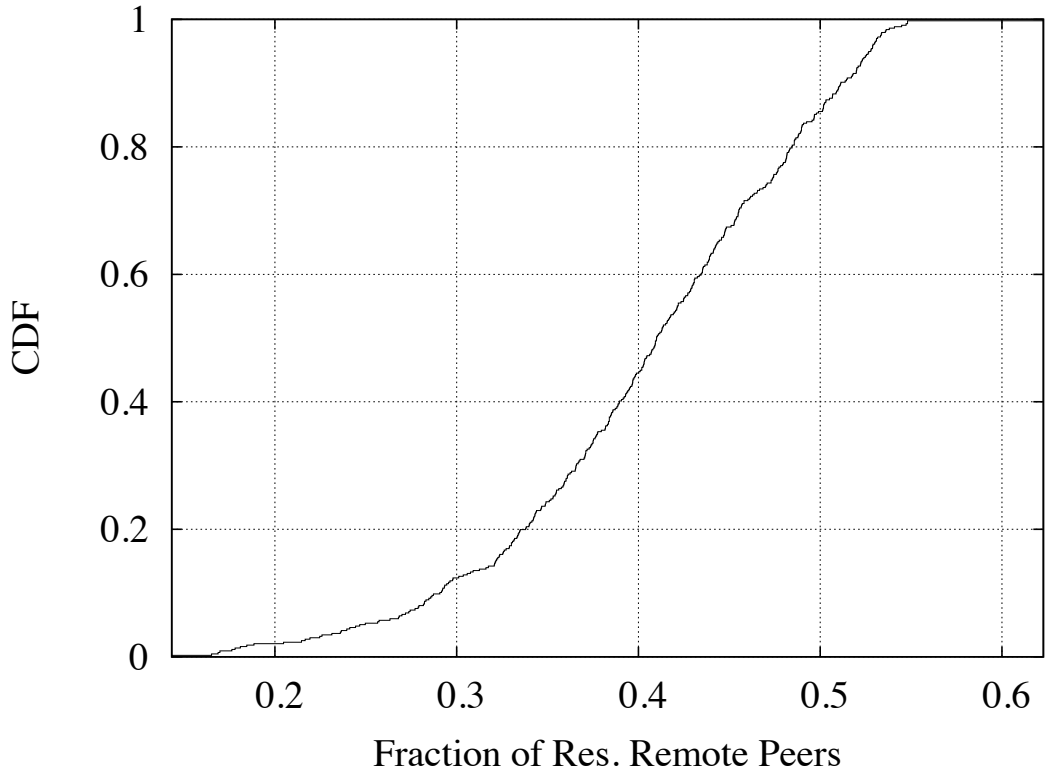

Figure 2.3: Distribution of residential remote peers.

use the SpamHaus PBL [Spa] to determine if a remote IP address is an end-user device.[5] While the PBL is maintained with help from the owners of some address blocks, the list still represents an approximation. However, we note that the list is generally viewed as "good enough" for operational tasks such as email filtering as evidenced by the widespread use of the PBL. Colloquially we will refer to end-user devices as "residential"—even though strictly speaking they are not all in homes—and all other hosts as "non-residential".

CCZ hosts make valid connections to a median of 43K remote peers per day over the course of our $\mathcal{L}_c$ dataset. Figure 2.3 shows the distribution of the fraction of peers that are residential for each day of the dataset. We find that as little as 10% and as much as 55% of the peers are residential across the dataset. The median of the distribution indicates

[5]Note, we have historical snapshots of the PBL and used the first snapshot for a given day to analyze our connection logs. We are missing a PBL snapshot from April 22, 2012 and for this day we use the previous day's snapshot.

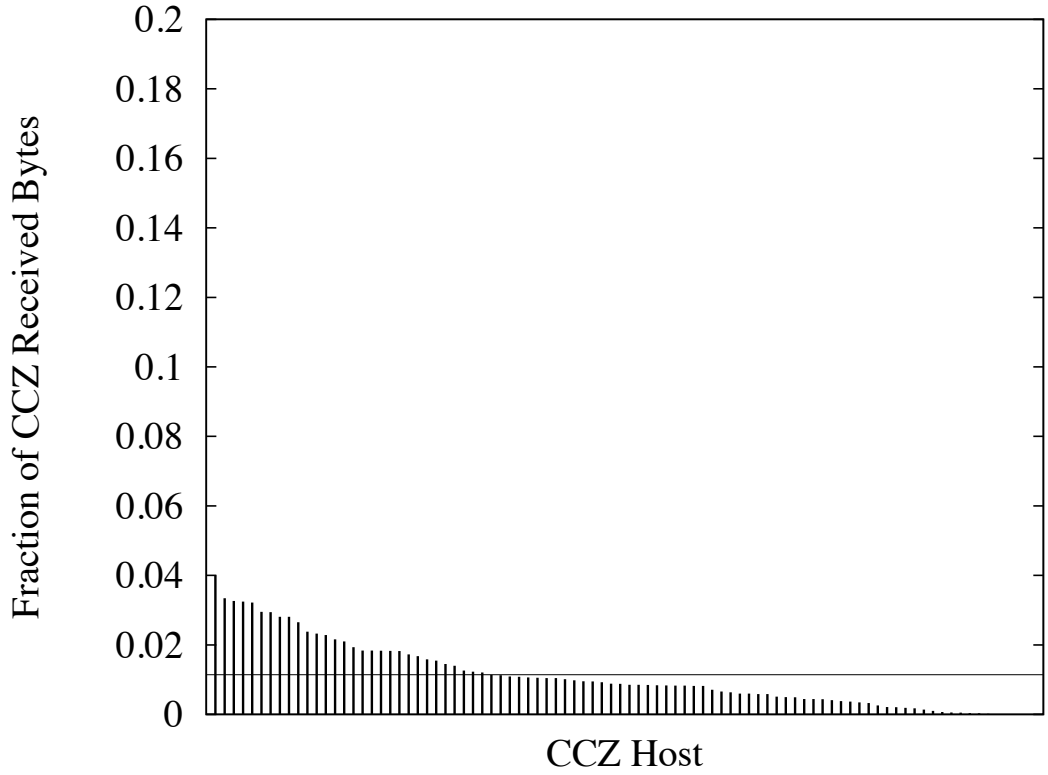

Figure 2.4: Relative contribution of each CCZ host for arriving bytes.

that approximately 40% of the remote peers are residential. In § 2.3.3 we revisit peer location with an eye towards correlating it with traffic characteristics. However, from our analysis in this section it is clear that CCZ hosts are not simply communicating with fixed infrastructure, but are also engaged in distributed peer-to-peer communication.

2.3 Traffic Mix

We next turn our attention to providing an overview of the traffic patterns we observe.

2.3.1 Heavy Hitters

We first assess each CCZ host's contribution to the aggregate traffic observed. Figure 2.4 and Figure 2.5 show the fraction of bytes sent to each CCZ host and from each CCZ host, respectively. We sorted the hosts by their relative contribution (the sort is independent

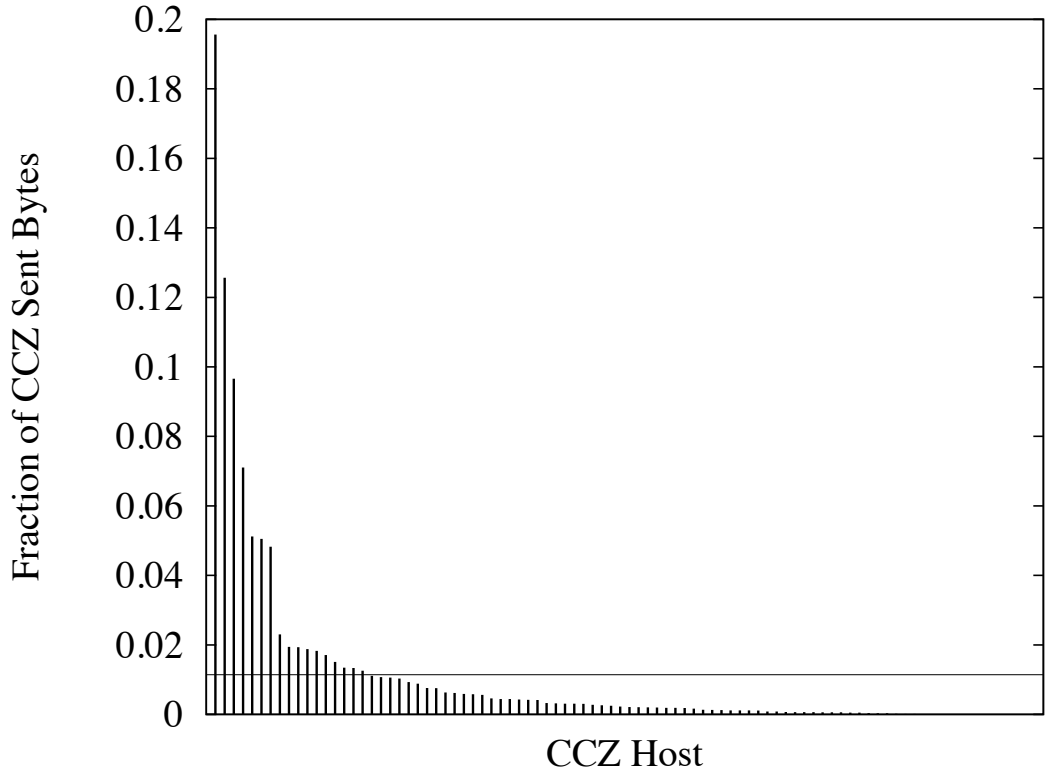

Figure 2.5: Relative contribution of each CCZ host for transmitted bytes.

for each plot). The horizontal line shows the average per-host contribution. In terms of arriving data (Figure 2.4) we find that 35 hosts receive at least an average share of the traffic. Further, we find that the top host receives over three times as much data as an average host —receiving over 3% of the bytes across our dataset. Meanwhile, the smallest share is 11,000x less than the average.

Figure 2.5 shows the relative data volume transmitted by CCZ hosts and shows a more skewed result than when considering data received. We find that the top CCZ sender transmits nearly 19% of the data volume across our dataset! Meanwhile in relative terms we find some hosts contribute nearly nothing to the overall data volume sent. Only 20 hosts contribute at least the average share of data transmission volume.

We note that there is some overlap between the top senders and top receivers. Considering the top ten hosts in each direction we find five hosts that appear on both lists. The host sending nearly 19% of the data departing from the CCZ network is not within the top

ten in terms of data receivers. However, the top host in terms of data reception ranks second in terms of data transmission.

2.3.2 Applications

We now analyze our dataset to determine the top applications. We identify applications first by Bro's "service" determination in the $\mathcal{L}_c$ logs. This leaves a significant fraction of the traffic unclassified. In particular, while Bro logs fine-grained details about BitTorrent traffic in $\mathcal{L}_b$ it classifies such traffic as "other" or "http" in the $\mathcal{L}_c$ logs. We therefore analyze the $\mathcal{L}_b$ logs to determine which "other" and "http" connections to classify as BitTorrent and change their designation in $\mathcal{L}_c$. Additionally, Bro does not recognize some traffic using TCP port 51314 as BitTorrent. We have found this to be the default port used by the Transmission BitTorrent client and the traffic we find on this port is consistent with BitTorrent activities and therefore we roll this traffic into the BitTorrent count. Finally, we use the service port number—i.e., the destination port of the SYN that starts a connection—to identify the remaining applications.

Using this process we find thousands of applications. To focus our attention on the most prevalent applications we winnow our analysis to applications that constitute at least 1% of either (i) the aggregate number of connections, (ii) the total traffic volume sent by CCZ hosts or (iii) the total traffic volume received by CCZ hosts. Using this approach identifies the ten most popular application protocols.

Table 2.3 shows the traffic breakdown for the top applications. There are three service port numbers that are ambiguous in our data. Port 1111 traffic seems to correspond to either Flash or Daodan malware. We analyzed the host fan-in and fan-out of the port 39457 and 31690 and these are consistent with peer-to-peer traffic involving thousands of peers. We cannot tell that this is BitTorrent traffic and therefore do not label it as such, but we are confident it is some form of peer-to-peer file sharing service. Given only a rough understanding of these services, we do not settle on specific designations but leave them nebulous

Service	Hosts	Conns.	Sent	Rcvd.
HTTP	90	321 M	1.1 TB	62 TB
Flash	89	444 K	6.0 GB	4.5 TB
BitTorrent	72	28 M	9.7 TB	3.4 TB
HTTPS	90	52 M	776 GB	1.9 TB
Steam	65	442 K	176 MB	819 GB
DNS	90	255 M	11.2 GB	63.7 GB
Other-39457	25	956 K	290 GB	45.3 GB
Other-1111	30	1.4 M	776 GB	40.1 GB
Other-31690	33	166 K	293 GB	23.6 GB
Minecraft	27	6.2 M	353 GB	7.7 GB
Unclassified	88	92.8 M	8.1 TB	5.0 TB
	98%	12%	38%	6%

Table 2.3: Aggregate traffic volume for popular application protocols.

in the table.

The second column of the table shows the number of CCZ IP addresses we observe using the particular application over the course of our 19 month dataset. As expected we find DNS, HTTP, HTTPS and Flash are used by (nearly) all users. We find BitTorrent is used by over three-quarters of the users. The Steam gaming application is used by nearly two-thirds of the users. The remaining top applications are used by roughly one-quarter of the user population each. The last two columns of the table show the traffic volume in each direction. We find that even among the top applications there is a difference of four orders of magnitude in data volume across the list (for both directions). Further we find that the top application—HTTP for data reception and BitTorrent for data transmission—comprises 9- and 12-fold increases over the second ranked application for sending and receiving respectively.

Table 2.3 also gives the amount of traffic we do not attribute to one of the top ten applications in the "Unclassified" row. As shown in the table, in terms of connections and bytes received the fraction of traffic involving other applications is modest and expected in that we know from previous studies and intuition that the top applications will not be responsible for all traffic. However, the percentage of bytes sent that are unclassified is

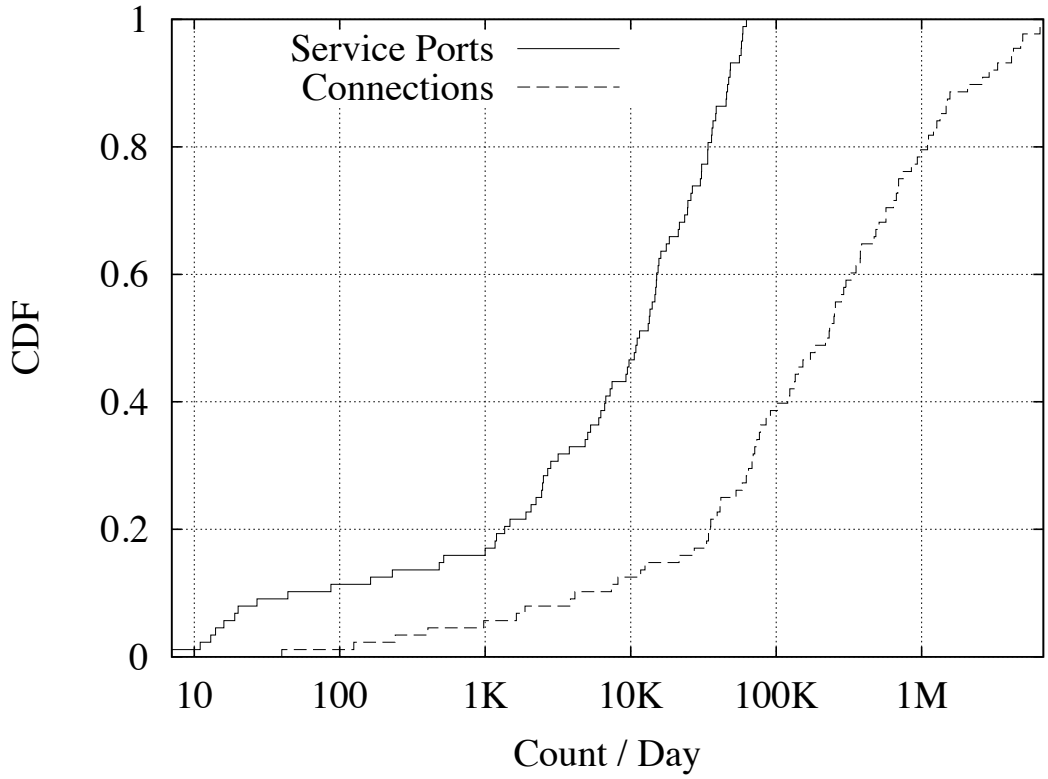

Figure 2.6: Distribution of the number of connections and service ports for each day.

38%—which is striking. We know from our analysis to determine the top applications that no one port number is responsible for more than 1% of the bytes sent across any day in our dataset. This suggests that this uncategorized traffic volume may well be using a large variety of ports as a policy evasion technique.[6]

In Figure 2.6 we plot the distribution of both the number of unclassified connections per day and the number of distinct service port numbers found in the unclassified traffic. The difference between the number of service ports and the number of connections is roughly a factor of 20 across the distribution—with the difference growing towards the tail. In other words, we find each service port to be used in only tens of connections. We also find the number of service ports has a median of over 17K and a maximum of 64K, which is the number of possible ports. This illustrates the heterogeneity of the traffic

[6]The CCZ does not impose policy-based restrictions based on port number, but the remote peer could well be under such constraints.

Service	Host	Conn.	Sent	Rcvd.
HTTP	2.07	1.02	2.55	1.01
Flash	4.00	6.02	6.17	2.32
BitTorrent	5.04	3.95	1.32	3.46
HTTPS	2.94	3.23	3.12	3.67
Steam	5.96	7.32	7.39	5.15
DNS	1.01	1.99	5.33	5.93
Other-31690	8.80	8.13	6.70	8.27
Other-39457	8.28	7.60	7.36	8.28
Other-1111	8.31	8.42	8.33	8.45
Minecraft	8.59	7.31	6.74	8.46

Table 2.4: Average rank of each of the top application protocols per day.

and that it belies aggregation given the information present in the $\mathcal{L}_c$ logs. However, the port spread and the traffic volume suggests that much of the unclassified traffic is likely peer-to-peer traffic trying to avoid detection.

While Table 2.3 gives an overview of the entire 19 month dataset it does not give any indication of the volatility across time. To gain a sense of the relative contribution of each of the ten top applications we rank them by (i) number of hosts using the application, (ii) number of connections, (iii) bytes sent and (iv) bytes received for each day in our dataset. Table 2.4 gives the average rank of each of the top ten applications in each category across the entire dataset. In terms of the number of hosts using an application and the amount of traffic received the average daily rank is nearly identical to the aggregate rank (with only the last few applications changing positions in both cases). This suggests that these rankings are fairly stable across our dataset. However, in terms of the data volume sent by CCZ hosts and the number of connections we find that only the top three or four, respectively, applications are identical across the aggregate and average daily ranking. After that the ranking diverges and hence shows that non-trivial daily variations in application usage do in fact occur.

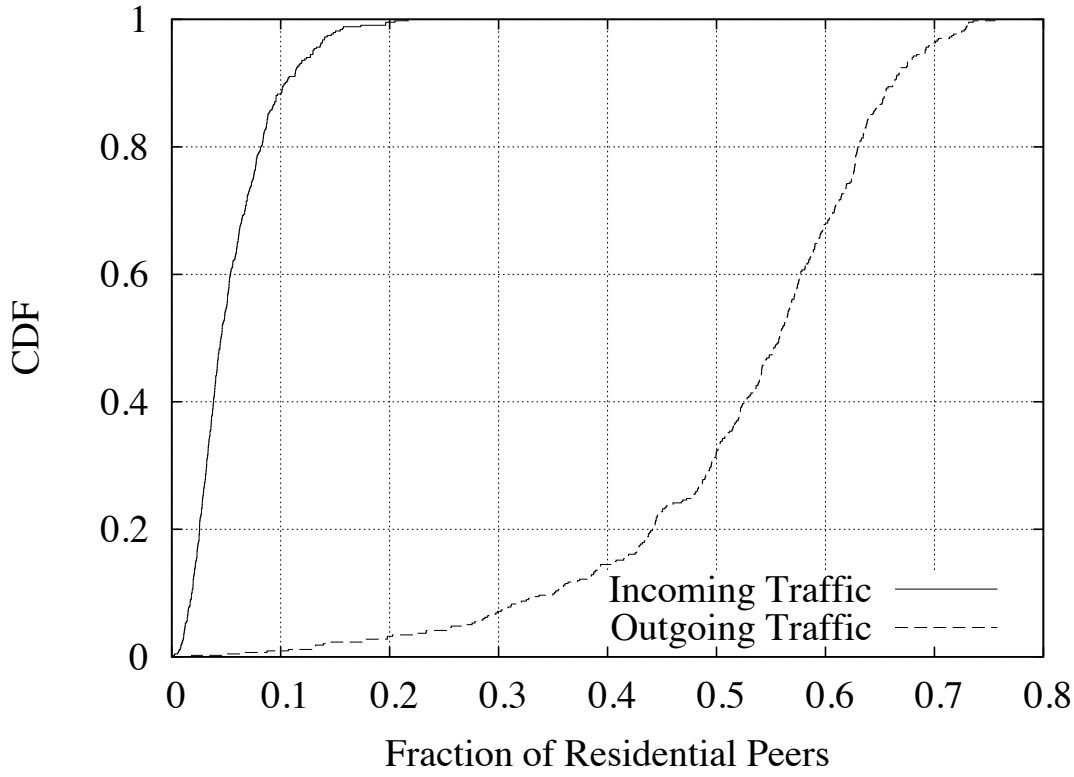

Figure 2.7: Distribution of the fraction of traffic exchanged with residential hosts.

2.3.3 Volume vs. Peer Classification

In § 2.2.2 we consider the prevalence of residential hosts in the set of remote peers with which CCZ hosts communicate. We now turn our attention to understanding the contribution of these remote peers to the overall traffic volume. Figure 2.7 shows distributions of the fraction of incoming and outgoing bytes exchanged with residential peers for each day in our dataset. The plot shows a dramatic difference between incoming traffic and outgoing traffic. We find that residential peers contribute around 4% of the incoming traffic on median and at most 20% of the incoming volume. However, we find that in terms of outgoing traffic the median is around 53% and the maximum is approximately 75%. This makes sense when taken with the results presented in § 2.3.2 that highlight popular applications. The top applications in terms of received traffic volume are traditional client/server applications (e.g., HTTP) where we would expect to be receiving data from a centralized server

Site	Vol (%)	Days	Min,Med,Max		
youtube.com	19	561	1	2	5
nflximg.com	16	561	1	3	88
llnwd.net	14	561	1	3	247
edgesuite.net	9.7	561	1	4	145
apple.com	2.6	561	1	7	67
xvideos.com	1.5	558	4	9	60
hulu.com	1.5	561	1	17	317
akamaihd.net	1.2	442	1	14	5,250
fbcdn.net	1.2	561	5	10	26
tumblr.com	0.84	560	5	15	223
pandora.com	0.75	560	5	15	1,291
megaupload.com	0.71	393	1	22	7,137
filesonic.com	0.68	376	1	87	5,012
espn.com	0.68	502	2	4,248	6,877
—.com	0.64	556	5	18	2,531

Table 2.5: Top 15 sites in terms of incoming traffic volume.

rather than a end-user host. Meanwhile, in terms of outgoing data volume we find that the top application is BitTorrent, which is by its very nature a distributed system that relies on end hosts and not on infrastructure level servers. Therefore, the disparity in Figure 2.7 is not surprising, even if striking.

2.3.4 Web Servers

Table 2.3 above shows that HTTP is by far the largest source of traffic flowing to CCZ hosts and is the second ranking application in terms of traffic sent by CCZ hosts. Given HTTP's prominence in the traffic mix we briefly assess the popular servers being accessed as an indication of how users are employing their large capacity networks. For this analysis we consider the "Host" header in HTTP traffic, which is recorded in our $\mathcal{L}_h$ logs. This is then correlated with our $\mathcal{L}_c$ dataset to produce traffic volumes for each site. We aggregate sites based on the second level domain.

Table 2.5 shows the top 15 web sites[7] in terms of traffic downloaded to CCZ hosts.

[7]We have elided the actual name of the last site due to its explicitness.

Site	Vol (%)	Days	Min,Med,Max		
google.com	19.6	561	1	3	10
youtube.com	6.8	561	1	3	13
facebook.com	6.8	561	1	1	11
yieldmanager.com	3.6	561	1	4	13
datei.to	3.5	4	1	2	5,321
4shared.com	3.3	413	1	1,139	6,721
fbcdn.net	2.5	561	2	5	12
doubleclick.net	2.3	561	2	6	11
revsci.net	1.1	561	4	11	34
adnxs.com	1.0	561	2	12	67
google-analytics.com	0.9	561	7	11	20
yahoo.com	0.9	561	1	14	45
nflximg.com	0.7	561	5	17	180
turn.com	0.6	561	2	19	47
twitter.com	0.6	561	2	20	39

Table 2.6: Top 15 sites in terms of outgoing traffic volume.

The table gives the site, the percentage of traffic volume the site contributes, the number of days the site is ranked in the top 15 sites across the dataset and then the minimum, median and maximum daily ranking of the given site. The table largely shows popular sites (e.g., as listed by Alexa [Ale]) and CDNs are the most common. Further, we find some volatility in the top 15 sites. For instance, two-thirds of the sites are within the top sites nearly every day in our dataset with over half the top sites are ranked first at least once. We also find that some sites are bursty. An example is "filesonic.com" which ranks as high as first, is only ranked in the top 15 on two-thirds of the days, is ranked lower than 87^{th} on half the days and ranks as low as $5{,}012^{th}$ on one day. We also find that nearly 40% of the aggregate data volume is from video streaming sites. Another roughly 25% of the volume is from content distribution networks.

Finally, Table 2.6 shows the top 15 web sites in terms of data transmitted by CCZ users to web sites. In this case we see more consistency in the sites across the dataset than for incoming traffic, as all but two of the top sites are in the top 15 list every day. However, we find volatility here, as well. For instance, nearly half the top 15 sites reach the highest rank on at least one day. In addition, we again find bursty behavior. E.g., "datei.to" is

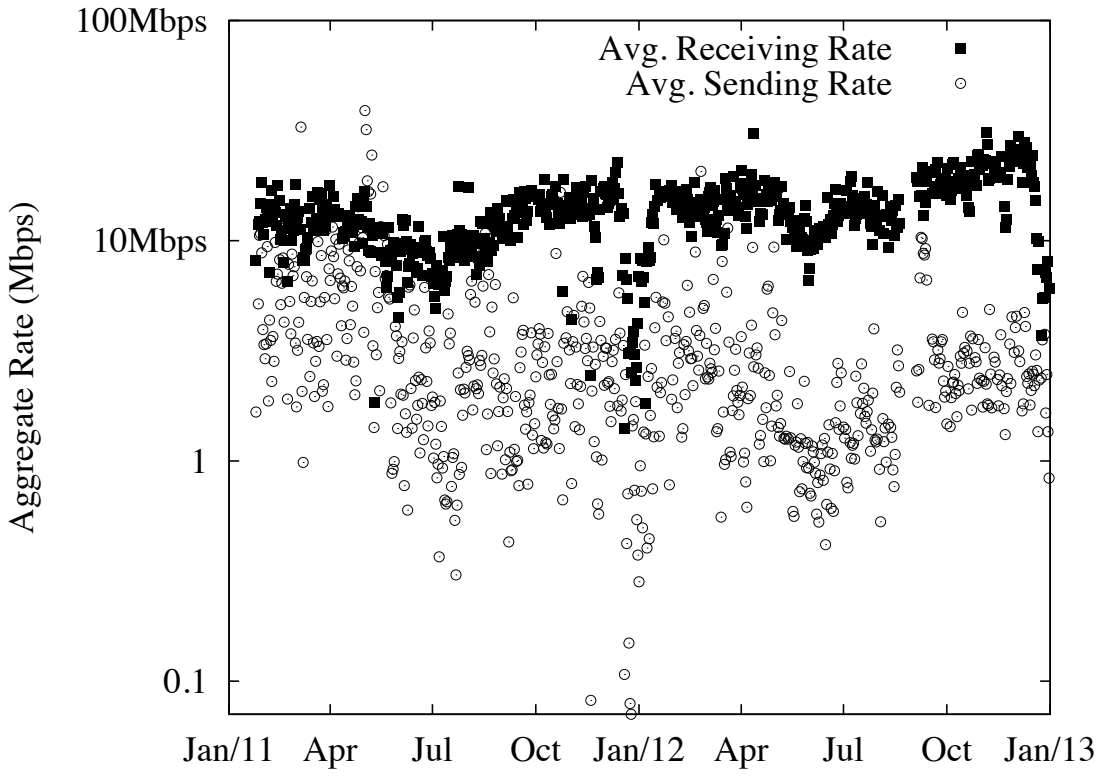

Figure 2.8: Aggregate transmission rates per day.

overall in the top 15 sites in terms of volume, but appears in the daily top 15 list only four times. Given the median rank is 2, it is clear that this site was only used briefly, even if energetically.

2.4 Observed Transmission Speed

We now turn our attention to the salient feature of FTTH networks: speed. Figure 2.8 shows the aggregate sending and receiving rates for all TCP traffic across all hosts in the CCZ for each day in our dataset. The average daily aggregate incoming traffic rate is roughly 13.4 Mbps. We find that patterns in the data follow the academic calendar. As some students leave the CCZ during academic breaks, the overall population of CCZ is

reduced.[8] This reduction in population naturally causes the overall sending and receiving rates to decline with the largest reductions taking place during the winter breaks. In terms of local hosts transmitting data we find that the average aggregate rate is 3.4 Mbps with similar modest dips during academic breaks.

2.4.1 Per-Host Speed

We will now focus on the capacity individual CCZ hosts consume.[9] For each connection in the $\mathcal{L}_c$ logs we evenly distribute the number of bytes transmitted over the duration of the connection. We then construct 1 second bins (86,400 bins per day) and assign the byte count to the appropriate bins. We track each direction independently. This even spreading of data across a connection does not reflect reality for two basic reasons: (i) as we show in Chapter 4 applications do not send and receive data uniformly across the duration of a connection [SBA14] and (ii) TCP's congestion control algorithms [APB09] constantly adjust the sending rate based on the perception of the network conditions. However, both of these dynamics happen outside our view and therefore for this initial analysis using uniform spreading suffices.

As we sketch above, we break our data into 10.7 billion one-second bins—i.e., 86,400 bins for each direction, day and host in our dataset. To concentrate on periods when hosts are transmitting relatively rapidly we winnow our dataset to the top 1% (53.8M) bins in each direction. Figure 2.9 shows rate distributions for the top bins in each direction. The first point on each line shows the 99^{th} percentile of our entire distribution (since we focus this on the top 1%), which is a per-host sending rate of approximately 0.5 Mbps and a receiving rate of roughly 3.2 Mbps.

The figure shows that more than 90% of the top 1% receiving bins represents a

[8]The CCZ user population is roughly 60% students and 40% full-time residents of the neighborhood [Gon12].

[9]We consider each CCZ IP address to be a "host". This is not necessarily correct due to NATs, but for our purposes only a rough approximation is necessary.

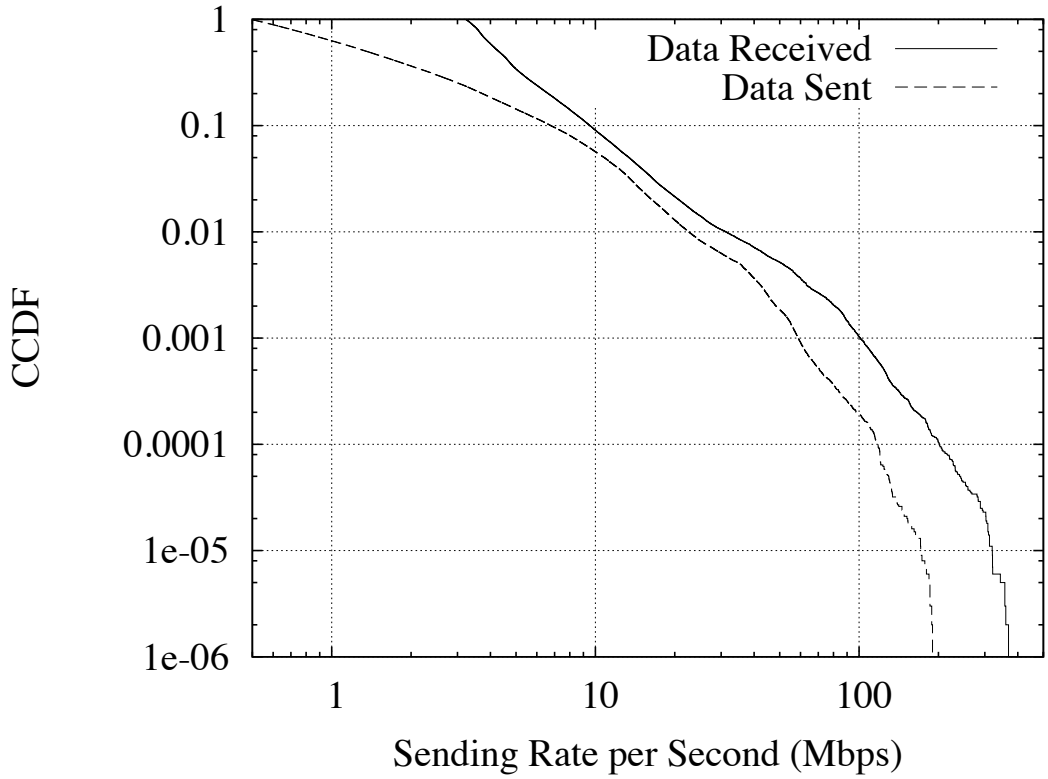

Figure 2.9: Throughput for top 1% bins.

rate under 10 Mbps. In other words, over 99.9% of the overall bins do not exceed a rate available from common commodity residential networks.[10] Or, on average each user spends approximately 1.3 minutes per day employing higher-than-commodity network capacity. We also find that 0.1% of the top 1% receiving bins—or less than one second per day per host—shows an aggregate receiving rate of more than 100 Mbps.

Due to commercial networks often being asymmetric, the CCZ network provides a larger relative improvement in uplink capacity than in downlink capacity. While users only exceed a nominal commodity receiving rate (10 Mbps) 0.1% of the time, they exceed a nominal commodity uplink of 0.5 Mbps 1% of the time.[11] Further, we note that CCZ user transmission rates exceed 10 Mbps approximately 0.06% of the time and 100 Mbps roughly

[10] We are aware of faster commodity networks, but 10 Mbps is the right order.

[11] Again, our aim is not to quibble about commodity rates, but to illustrate the difference between the uplink use and downlink use by CCZ users.

Service	Recvd (%)	Bins (%)
HTTP	82.5	96
Likely BitTorrent	5.4	19
BitTorrent	3.8	9.3
HTTPS	0.8	49.3
Unclassified	7.5	49.2

Table 2.7: Breakdown for top receiving applications.

0.0002% of the time. The data suggests that residential users' current usage patterns and applications are generally well-served by commodity downlinks, but when provided more outbound capacity users will take advantage of these resources to some degree.

Finally, we note that in 3 million instances (5.6%) whereby a given host's corresponding sending and receiving bins are both in the top 1% lists. This illustrates that in a non-trivial number of cases a particular host is engaged in high-speed data transfers in both directions, e.g., as part of a peer-to-peer network.

2.4.2 High-Rate Applications

We now briefly analyze which applications are active during periods of high capacity use. The following analysis takes into account only the top 1% of the bins as discussed in § 2.4.1. Note that in oder to remove ambiguities about the types of connections in the top 1% of bins we add an additional classification for "Likely BitTorrent" during these fast sending periods. "Likely BitTorrent" denotes otherwise unclassified traffic that involves a CCZ host that is simultaneously known to be using BitTorrent. Table 2.7 shows the percentage of the incoming data volume for each application that receives at least approximately 1% of the total incoming data volume in the top bins. Additionally, the table shows the percentage of the top bins in which we find the service. The table shows that HTTP/HTTPS is responsible for over 82% of the data volume during high utilization periods. Further, 96% of the top bins contain HTTP/HTTPS traffic. Finally, we also find BitTorrent to be a mild contributor during periods of high rate data reception.

Service	Sent (%)	Bins (%)
Likely BitTorrent	41	78
BitTorrent	35.4	55.3
Other-1111	8.9	1.4
Minecraft	6	10.6
HTTP	2.9	71
HTTPS	1.9	53
Unclassified	3.9	46.5

Table 2.8: Breakdown for top sending applications.

Table 2.8 shows our findings for top applications in terms of data transmitted by CCZ users during the top 1% utilization periods. We find BitTorrent to be the largest contributor—both in volume and active bins. We additionally find web traffic, Minecraft and port 1111 traffic to each modestly contribute to high rate data transmission.

2.5 Transmission Speed Causes

In § 2.4 we study transmission rates on a host-level basis. We find the hosts do not often use anywhere close to the available capacity. A natural question is: *why?* In this section we strive to analyze our packet traces to gain an initial understanding of what is limiting performance. TCP's performance is dictated by a set of congestion control algorithms [APB09] and has a number of dependencies, including (i) the TCP receiver's advertised window, (ii) the size of the TCP sender's retransmission buffer, (iii) the RTT of the network path, (iv) the loss rate along the network path, (v) the application's sending pattern, and (vi) the available capacity along the network path. Of these, (ii), (v) and (vi) are not readily visible in packet traces, while the others are either exposed directly by the protocol or can be estimated from traces.[12] In this section we use these pieces of information to study connection-level transmission speed.

[12] Although capacity is not limited on the local portion of the connection, the remote end of a connection may have a limiting amount of capacity.

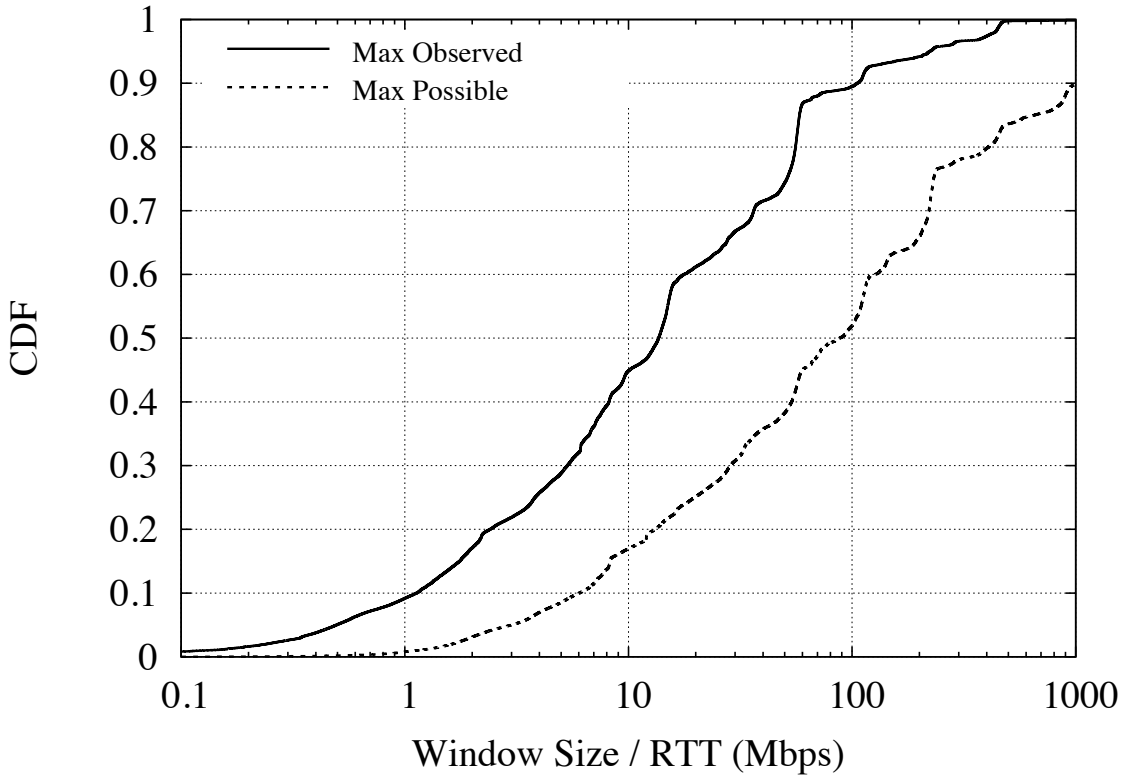

Figure 2.10: Maximum sending rate based on the advertised window for incoming traffic.

2.5.1 Potential Speed

TCP's performance is ultimately constrained by the RTT of the network path and the receiver's advertised window. In particular, the upper-bound on performance is $\frac{advwin}{RTT}$. This upper-bound requires (i) the sender's retransmission buffer to be at least as big as the advertised window, (ii) the application to keep the TCP buffer full and (iii) no loss along the network path such that TCP's congestion window dynamically reduces the sending rate. For the purposes of assessing how fast FTTH-connected hosts *can* send and receive data we assume these requirements hold in this subsection. Following analyses in this subsection, we examine the case where these assumptions do not hold.

Figure 2.10 and Figure 2.11 show two distributions for incoming and outgoing traffic, respectively. The solid line on each plot shows the performance if the connection were to use the maximum advertised window we observe over the course of the connection. We

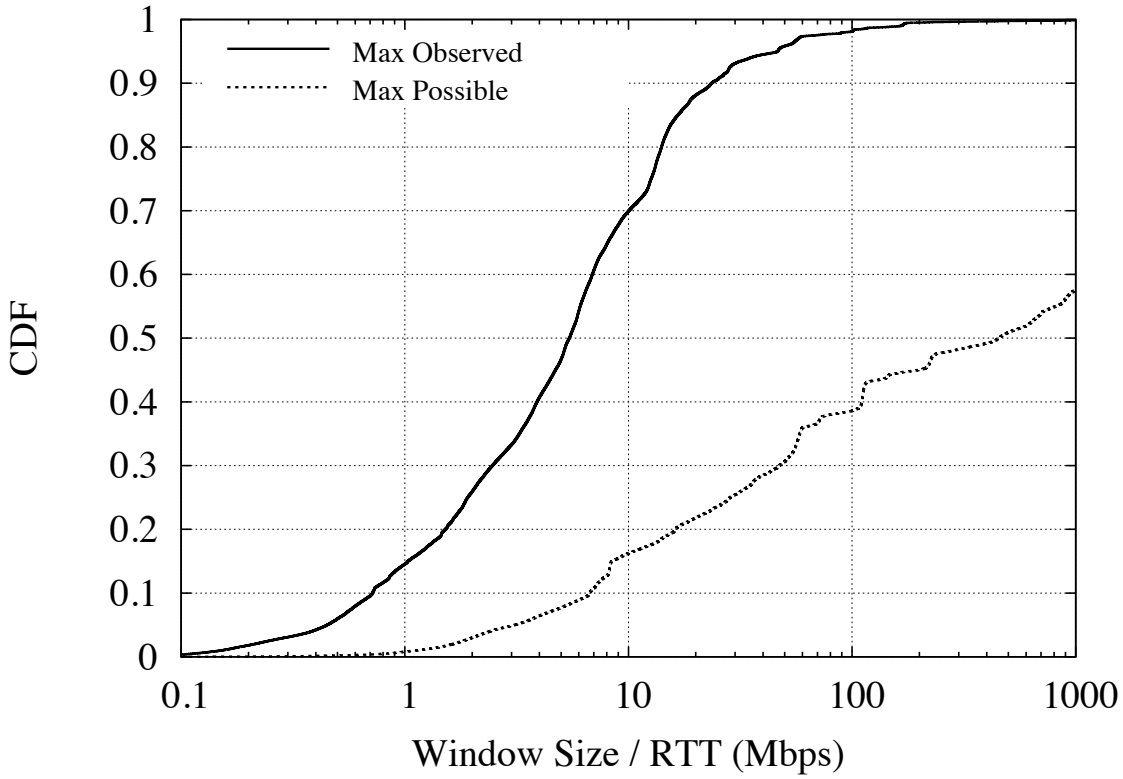

Figure 2.11: Maximum sending rate based on the advertised window for outgoing traffic.

find that less than 0.2% of the connections can possibly utilize the full 1 Gbps at the disposal of these hosts based on their advertised window sizes (for both incoming and outgoing traffic). Meanwhile, as we illustrate above the median transmission rate is 13.6 Mbps for incoming traffic and 5.4 Mbps for outgoing traffic. Further, we find the entire distribution to show generally higher incoming rates than outgoing rates.

Some TCP implementations use "autotuned" socket buffers, whereby the size of the socket buffers—and therefore the advertised window—dynamically adjusts with the connection's rate [SMM98]. In other words, if a host detects that the advertised window is hampering performance, additional buffer space is allocated to the connection and subsequently the size of the advertised window is increased. In this case, the analysis above does not correctly portray performance limits because advertised window is not necessarily hindering a connection's performance, but rather is dictated by the connection's performance.

Unfortunately, there is no direct way to understand whether a host is autotuning its

socket buffers. We therefore sketch a bound on how well hosts can perform, as shown by the second (dotted) line on the plots in Figure 2.10 and Figure 2.11. This line shows the distribution of the transmission rate based on the maximum possible advertised window size that can be expressed within the TCP headers. Nominally, the maximum advertised window is 64 KB, but the window scaling option [JBB92] can increase this to up to 1 GB. We find the theoretical window size is far greater than the maximum observed advertised window across directions, showing that hosts are not using the maximum socket buffers that can be encoded. We find that roughly 10% of the incoming connections and over 40% of the outgoing connections could encode windows that would yield a rate of at least 1 Gbps.

Above we establish that hosts *actually* advertise quite modest windows—which hamper performance—even though they *could* advertise larger windows. A final question is how often hosts increase their advertised window during the course of a connection. This speaks to the popularity of autotuned socket buffers. As a quick check we analyze each connection in our dataset for each hosts' initial and maximum advertised windows. We use the initial advertised window as a "base" of sorts since this is reflective of the buffering allotment at the connection's inception. We find that in roughly 80% of the connections the local host's maximum advertised window equals its initial advertised window. For remote hosts this equivalence holds in approximately 59% of the connections. This shows that autotuned socket buffers are in fact in use, but not on a majority of the connections.

2.5.2 Connections Without Loss

We now turn from the potential best-case rates that TCP can attain to examining the rates TCP does attain in our dataset and the reasons for those rates. As an initial investigation we examine the largest 10 connections in terms of data transmitted *from* a CCZ host in each of our 642 one-hour trace files. We winnow to only these large connections for several reasons. First, the performance of short connections is less dependent on capacity

than on delay. Therefore, these are not useful to understanding whether TCP—or implementations thereof—can in fact use the capacity FTTH affords. Second, the process of analyzing connections for loss, RTT and flight size is burdensome in terms of both memory and computation—and especially so given that most connections that require memory and processing are short and irrelevant for our study.

Figure 2.12 shows the distribution of the connection sizes for our corpus of the top 10 connections per trace. We analyze the data flowing from CCZ hosts to remote hosts as our vantage point is then close to the sender and hence makes estimation of various sender properties straightforward (e.g., RTT, congestion window size, loss rate). A vantage point close to the receiver makes these sender properties difficult to estimate [Pax97]. We find that 90% of the connections are at least 1 MB in size. The remaining 10% are legitimate, but come from low usage periods of our corpus (e.g., overnight on weekends or holidays). Our corpus of 6,420 connections involves 84 CCZ hosts (nearly the entire population) and 5,074 remote hosts. We first only consider connections that do not experience loss—or 926 of the 6,420 connections in our corpus. (We will consider the balance in the next subsection.)

Raw Performance: Figure 2.13 shows the throughput for each connection without loss as a function of the data volume. The clear trend is that performance increases with data volume. This is expected for two reasons. First, connections start transmission using slow start [Jac88, APB09], which by definition aims to begin transmission at a low rate and ramp up the speed based on the capabilities of both the network path and the remote peer. Slow start ends when congestion is detected (usually via lost data) or the sender fills the receiver's advertised window. During the process TCP will generally be underutilizing the available network capacity while searching for an appropriate sending rate at the beginning of each connection. The impact of slow start on overall transmission speed is greater for shorter transfers than for longer transfers whereby the impact is ameliorated over more time. Second, we conjecture that as data volume increases the applications better manage

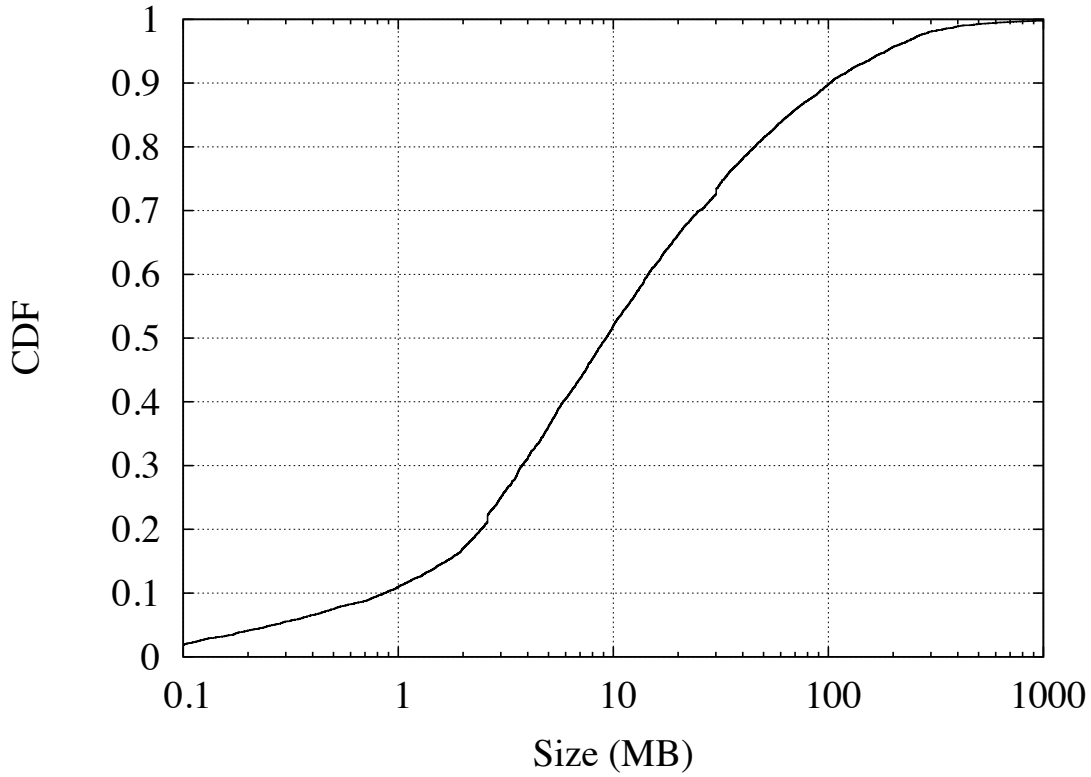

Figure 2.12: Sizes of the top 10 connections per trace.

the transmission process (adjust buffer sizes, etc.) in an attempt to well utilize the available capacity. Such efforts are less useful for small transfers that are more dependent on the raw delay between endpoints than the actual capacity. We find that sending rates generally top out at around 10 Mbps, but on occasion do reach nearly 100 Mbps. We now turn to investigating why the TCP performance is much lower than the available capacity.

Advertised Window Limits: We use Bro to determine the advertised window and the maximum flight size for each connection. The flight size is the amount of data transmitted but not acknowledged at any given time and approximates TCP's congestion window. We then compare the maximum flight size with the maximum advertised window to assess whether the TCP sender is limited by the receiver's advertised window. We find that in 11.6% of the connections without loss the sending TCP is in fact constrained by receiver's advertised window. In the remainder of the connections there is some other phenomenon that is constraining the sending rate.

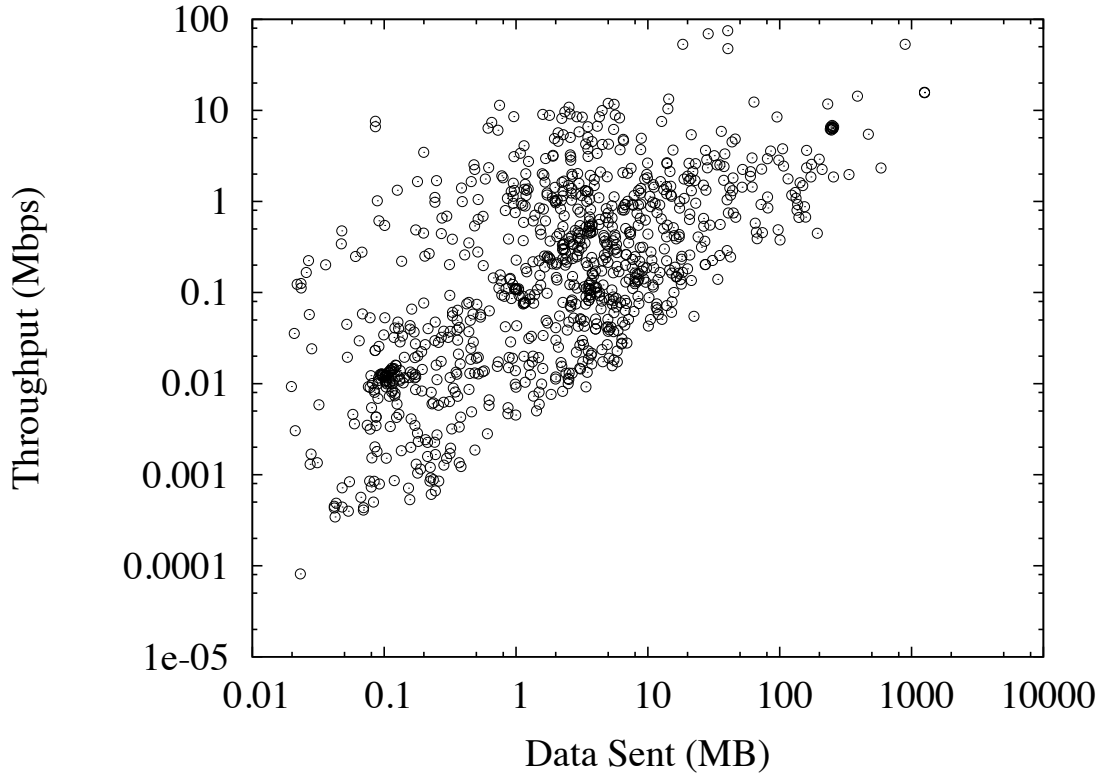

Figure 2.13: Connection sizes vs. throughput for connections without loss.

Sender Buffer Limits: In Figure 2.14 we plot the distribution of maximum flight size over all connections with no loss and for which we do not find to be advertised window limited. We find modes of varying size in this plot at 16KB, 32KB, 64KB, 96KB and 128KB. These are suggestive of some sender-side buffering issue that is limiting the flight size. The natural candidate would be the sender's TCP retransmission buffer—which limits the amount of data that can be transmitted before receiving an ACK in case the data is lost and needs resent. The limit could also come from an application—e.g., in an attempt to limit the overall sending rate. While the 1 Gbps fiber link is unlikely to become overloaded it is possible that some applications are trying to protect infrastructure within a house (e.g., a wireless network). Alternatively, some applications may have a "natural rate" and exceeding this rate is pointless (e.g., vastly exceeding a video playout rate just means the end host will have to buffer content until its appointed playout time). When summing the various modes we find they account for roughly 45% of the connections with no loss.

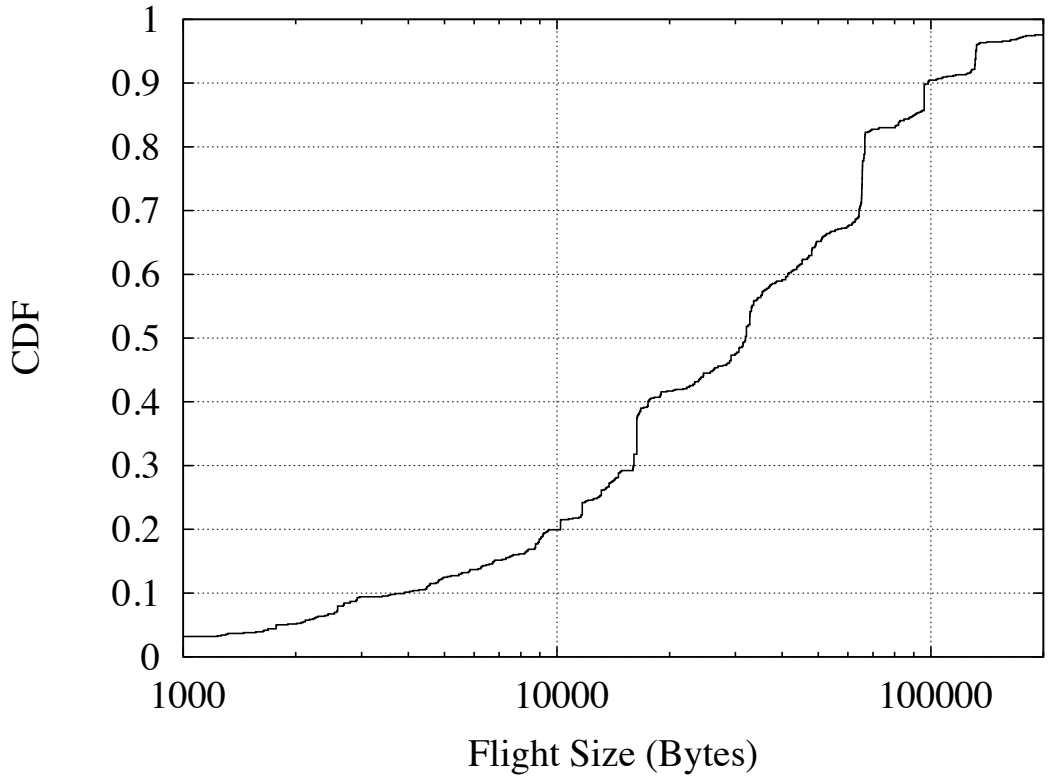

Figure 2.14: Distribution of flight size in connections without loss.

Remote Peer Limits: We next compare the remote peers' IP address to the SpamHaus PBL [Spa] to determine whether the remote is a likely residential host. While this heuristic is not perfect, our results here and previous work in the literature [All13] show it is a reasonable approximation. Using this definition we find that 26% of the remote peers are in residential settings, indicating that there is likely a fairly low capacity limit imposed on the remote side of the connection that could explain some of the low performance we observe.[13] The fastest connection involving a remote peer in a residential setting is 4.5 Mbps. Additionally, we observe 30% of the connections to non-residential remote peers attain throughput in excess of 4.5 Mbps—and topping out at 75 Mbps.

Summary: We find 11.6% of the connections without loss are constrained by the advertised window and the data suggests that another 45% are hampered by some sender-side

[13] We are noting that residential networks likely place a capacity limit on their hosts, not that non-residential networks do not.

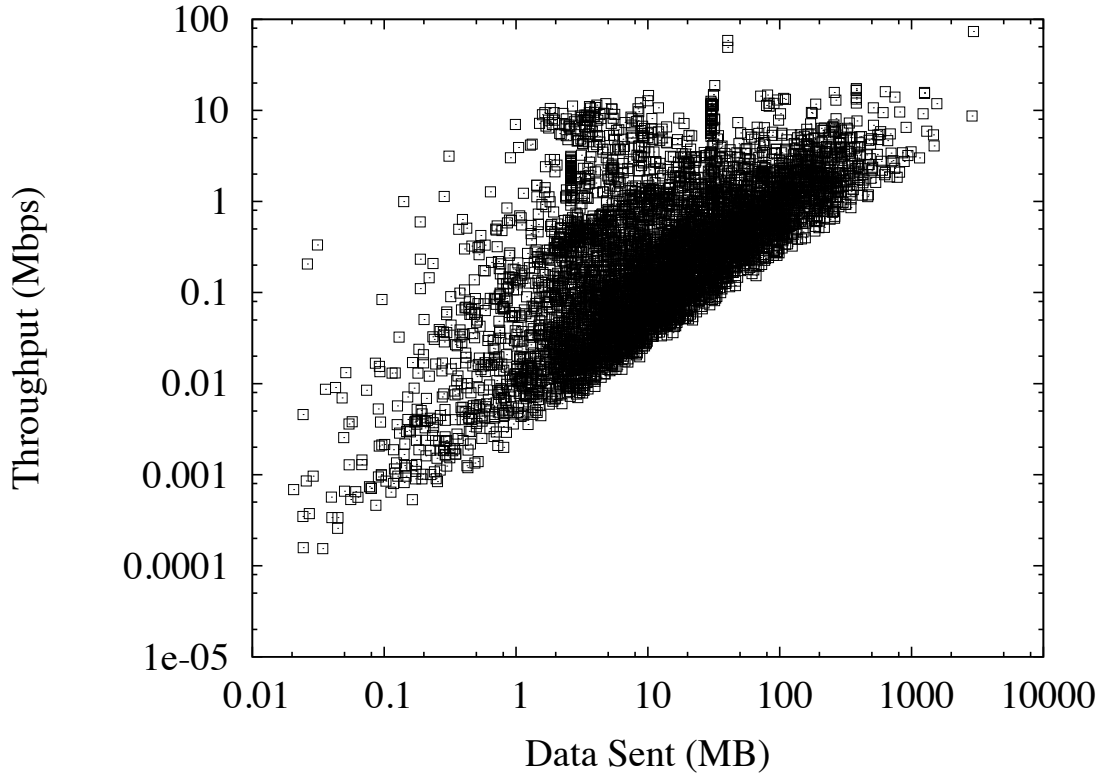

Figure 2.15: Connection sizes and throughput for connections with loss.

buffer. Further, approximately one-quarter of the connections involve residential remote peers that likely have an anemic capacity limit (relative to the 1 Gbps available to CCZ users). Without additional insight from the end hosts themselves it is hard to reason about the performance of the remaining connections. While we cannot pinpoint the cause, we can say that buffering issues (on both hosts) do not appear to be the first order constraints.

2.5.3 Connections With Loss

Finally, we turn to the 5,494 connections in our large connection corpus that experience loss. In these cases theory suggests the loss rate and RTT combine to dictate performance [MSMO97, PFTK98]. However, the advertised window, retransmission buffers and application behavior can still limit performance. As such, we repeat the step-wise analysis we conduct for the no loss case above.

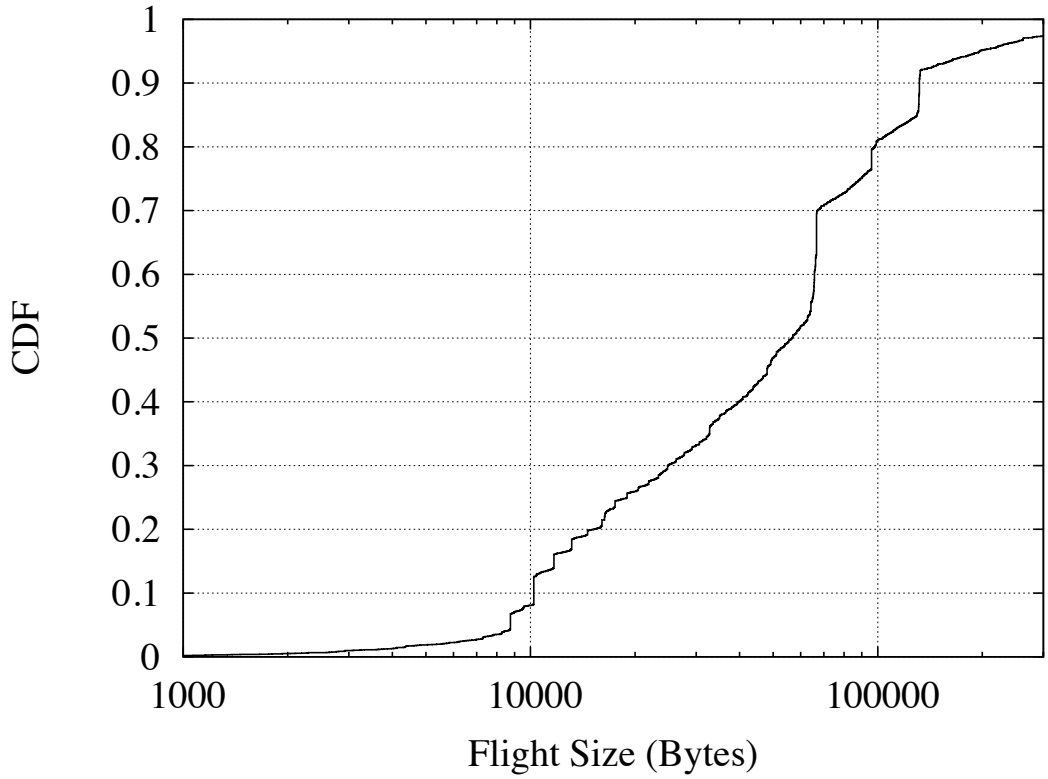

Figure 2.16: Distribution of flight size in connections with loss.

Raw Performance: As with the no loss case we describe above we first seek to understand the relationship between transfer size and performance. Figure 2.15 shows the results for the connections in our corpus with loss. We again find that the minimum performance increases with the transfer size. Further, we find that 77% of the connections in the corpus do not attain even 1 Mbps. Finally, 1% of connections exceed 10 Mbps.

Advertised Window Limits: Next, we find that the maximum flight size reaches the maximum advertised window size in 15% of the connections with loss, which is a slightly higher proportion than in the no loss connections.

Sender Buffer Limits: As above we next plot the maximum flight size distribution in Figure 2.16. As with the no loss case we find several modes that suggest a sender-side buffer limit that ultimately constrains performance. In this case we find this happens in over 35% of the connections that experience loss.

Remote Peer Limits: As in the no loss case above, we classify the remote peers as res-

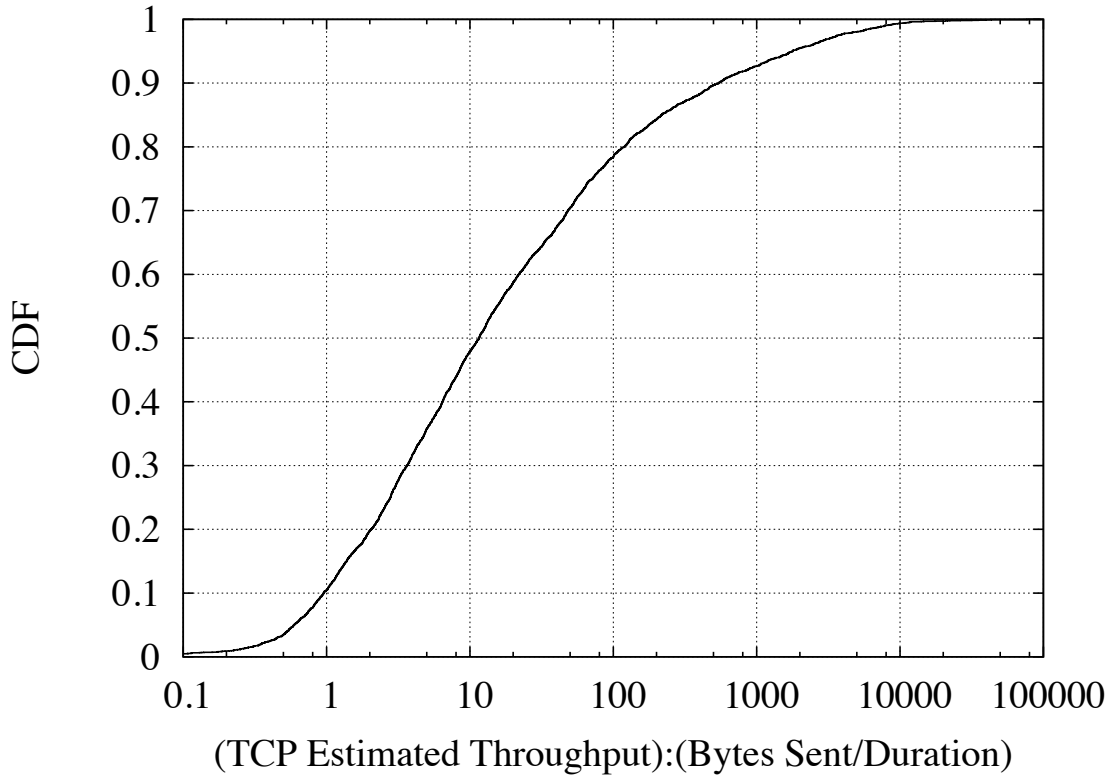

Figure 2.17: Ratio of theoretical throughput to actual throughput for each connection.

idential or non-residential using the SpamHaus PBL. In the set of connections with loss we find roughly half involve a residential peer. This is roughly twice as many as in the no loss case, which shows that these connections are prone to congestion—as one would expect. We find that all of the connections with loss and a remote residential peer in our dataset have throughput under 14 Mbps. Meanwhile the performance exceeds 14 Mbps to non-residential peers in just 1% of the connections. This shows the speeds are more homogeneous across type of remote endpoint than in the case of no loss. We do find a maximum throughput of 73 Mbps for non-residential peers.

Path Characteristics: As a final test we assess how the observed performance relative to theoretical throughput predictions derived from the given network path characteristics. As there are 50% of connections with loss and 43% of connections without loss being constrained by an unknown phenomenon, we attempt to determine if TCP itself is a limiting factor given network characteristics. We used the TCP model that was developed in

[MSMO97, PFTK98] and applied in [HFPW03] to calculate the rate each connection could potentially use given the connection's RTT and loss rate.[14] Figure 2.17 shows the distribution of the rate suggested by the TCP model versus the rate we observe for each connection with loss in our dataset. We find that in less than 10.4% of the cases the observed throughput actually outperforms the model. In the remaining roughly 89% of the cases the observed throughput is less—often by orders of magnitude—than the performance predicted by the model. This plot re-enforces our finding that the network path and TCP's congestion control algorithms should allow connections to transmit more rapidly than they in fact do. This means that host limits and application behavior—such as playing data out at a constant rate rather than as fast as possible—are causing lower performance than TCP could theoretically attain across the given path. As a final check, we examine the performance relative to the model when the data is partitioned by remote endpoint type (residential vs. non-residential). We find the results based on this split are similar to the distribution for all of the connections. This suggests that the type of remote host we connect to does not have an effect on this metric.

2.6 Related Work

We are not aware of any study directly related to characterization of FTTH network traffic. Our work does however relate to various previous efforts.

The first set of work includes passive observations of residential network traffic. In [MFPA09], the authors offer a characterization of residential network usage based largely on packet traces covering 20K DSL users of a given European ISP. Our study is similar in that we monitor residential users and take similarly fine-grained (packet level) measurements. While one aspect of our study is in some sense a re-appraisal of the results found in [MFPA09], we also contribute a look at traffic from a fundamentally different

[14] We used the median RTT for the connection and $b = 1$ for the model given in [HFPW03].

network where—as we show—capacity is essentially infinite. Another study from the literature monitors traffic involving residential DSL and fiber customers in Japan [CFEK06]. As with the German ISP, this is another investigation of residential network traffic "in the wild". This study, however, mainly focuses on high-level characteristics as the data is largely packet counts from ISP-level routers. The work also considers sampled NetFlow logs from one particular ISP and with these is able to conduct some analysis that overlaps with our work (e.g., using the port numbers to find application breakdowns). While these datasets allow for the study of broad and high-level aspects of residential traffic that we cannot study with our data, we are able to analyze user traffic in much greater detail (e.g., to understand where performance bottlenecks may be located). We note that while our work in some cases provides a similar data point to these previous efforts, we do not directly compare the results because of the large number of variables that are different between the monitored networks. The literature is rich with evidence of heterogeneity in network traffic and therefore direct comparisons to traffic studies from years ago on different continents are not likely useful.

A second set of related work uses active measurements to better understand residential and/or end host networks. This work takes various approaches from (i) installing a custom gateway in the home network that can measure network characteristics from a user's perspective (e.g., BISmark [SdDF$^+$11]) to (ii) using cooperative probing at the behest of a user (e.g., Netalyzr [KWNP10] to (iii) unassisted probing of remote residential networks (e.g., [DHGS07]). These efforts broadly shed light on the capabilities and characteristics of residential networks. However, these studies differ from our study in that they do not observe in-situ user traffic.

2.7 Summary

This chapter aims to present an initial broad characterization of traffic from an operational 1 Gbps FTTH network. We make several contributions, as follows.

- We are the first (to our knowledge) to characterize myriad aspects—from structural aspects to traffic patterns to capacity utilization issues—of an operational FTTH network over a 23 month time period.
- Our study provides another data point on the use of residential networks (e.g., a reappraisal of some aspects of [MFPA09]).
- We find that even when given virtually unlimited bandwidth the majority of the time users do not retrieve information from the Internet in excess of commercially available data rates. However, in terms of transmitting data we find the FTTH users in our study use modestly more capacity than available via commodity broadband.
- Additionally, we find that the applications used by FTTH users roughly mirror those used by non-FTTH residential users [MFPA09]. In other words, no innovative new applications that utilize the unique capabilities of FTTH networks have yet gained a widespread foothold.
- Similar to the last point we find 1 Gbps links to a single household are not well utilized, which presents an opportunity for new applications and services to capitalize on such resources.
- We find that TCP connections do not attain anywhere near 1 Gbps in performance even though plenty of unused capacity exists and TCP theory suggests the network paths are amenable to (much) higher rates than realized.
- We find that the likely reasons for TCP's low performance are end host buffering issues in many cases. In some cases this manifests in TCP's advertised window, but in others we find evidence of a sender-side buffer limitation.

Finally, we stress that our goal in this chapter was for broad characterization. Every analysis we present begs many additional questions. Our future work will involve digging more deeply into many of these questions. Additionally, we encourage others with access to additional FTTH networks to begin investigating these networks to provide a broader understanding than our data alone can.

Chapter 3

Revisiting TCP's Initial Retransmission Timeout

The Transmission Control Protocol (TCP) [Pos81b, M. 15] enables the reliable transfer of data across the network between a sender and receiver. The sender transmits a sequence of bytes in order on the network. Each TCP packet includes a sequence number, which when combined with a packet's length can be used by the receiver to reassemble the byte stream in order. Receivers respond with acknowledgment packets (ACKs), which enables the sender to keep track of the in-order bytes that have been received by the recipient. Each ACK packet contains the sequence number of the largest in-order byte the receiver has successfully received.[1]

TCP packets may be dropped as they are being transmitted across the network. When a loss happens, the sender recovers in one of two ways.[2] When additional data

[1]Receivers may also respond with selective acknowledgements (SACK) [MMFR96], which allow the acknowledgement of packets received even when an earlier packet has been lost. This prevents packets that have been received from needing to be retransmitted by the sender. Reconsidering the initial retransmission timeout is a relevant problem regardless of whether cumulative ACKs or SACKs are used later in a connection.

[2]Three if you also include SACK-based loss recovery [BAW+12]. Focusing on two methods is meant to contrast cases where a sender receives packets back from the receiver with cases where a sender must

packets are transmitted after a packet is lost, the sender can recover by observing triple duplicate ACKs from the receiver. Recall that a receiver that has a gap in its sequence of data but continues to receive packets will send ACKs that correspond to the beginning of the gap containing the missing data. If a sender receives three duplicate ACKs, it infers that a loss has happened and can retransmit data starting with the original missing packet.

Triple duplicate ACKs only reveal loss when it happens in the middle of a flight of packets. Consider instead the case where a single packet of data is sent and subsequently dropped along the network path between sender and receiver.[3] Since no additional packets are sent after the lost packet, the receiver will never have a chance to send out duplicate acknowledgments to signal that a loss has happened to the sender. In fact, the receiver will not know that a packet was sent at all. Since only the sender knows that a packet has been sent, it must timeout and retransmit in a reasonable amount of time if it does not receive an ACK from the receiver. The retransmission timeout (RTO) for a connection is calculated based on sampled round trip times (RTTs) as a connection progresses. The RTO is meant to be short enough that a sender can retransmit in a timely manner, but long enough that a sender will not retransmit too early, even in the precense of variance in the RTT. If an amount of time equal to the RTO passes without receiving an ACK from the receiver, a sender will infer a loss has happened and will begin retransmitting the last flight of data it sent. The specification for the RTO algorithm first appeared in [Jac88] and was formally specified in [PA00].[4]

A TCP connection begins with the sender transmitting a single SYN packet to a receiver. As noted above, the only way to detect if this packet is lost is via the RTO. While the RTO will approach an appropriate value as the connection progresses, at the start of a

timeout.

[3]Note that having a single, outstanding packet is not the only time a connection relies on the RTO. This example is picked to illustrate the need for the RTO.

[4]The current specification for the RTO algorithm appears in [PACS11], which the results in this chapter helped bring to fruition.

connection there have not yet been any RTT estimates taken to be used when computing an RTO. Therefore connections must rely on a initial RTO value until RTT estimates can be taken during the connection. Choosing a proper initial RTO has performance implications for TCP connections. If the intial RTO is set too low for most connections, then these connections will always timeout after sending a SYN and will send spurious retransmisisons for every connection. TCP will infer that the timeout and retransmission happened because of network congestion, and the subsequent transmission rate for the connection will be reduced [Jac88] [APB09]. If the initial RTO is too large, then connections will not recover from actual loss of the initial SYN in a timely manner. TCP's original specification set the initial RTO to a value of 3 seconds [PA00].

In this chapter we examine whether dropping TCP's intital RTO from 3 seconds to 1 second would have an adverse effect on TCP connections. The rationale for this change in TCP's specification stems from work showing that a majority of connections have round trip times under 1 second [Chu09], and that modern networks are simply faster than networks were when TCP's specification was originally written. We study how often connections would see their performance improved or harmed if a lower initial RTO were employed by examining data from multiple vantage points over a six year period on both wired and wireless networks.

3.1 Data

We obtain data from four different vantage points on both wired and wireless networks. Table 3.1 describes the data used in this study. The "LBL" data was taken at the Lawrence Berkeley National Laboratory and the "ICSI" data from the International Computer Science Institute. Both of these traces were collected at the border between the respective institution and the wide area network. The "SIGCOMM" data is from the wireless network that served the attendees of SIGCOMM 2008 and the "Dartmouth" data was collected from Dartmouth

Name	Dates	Packets	Connections	Clients	Servers
LBL-1	Oct/05–Mar/06	292M	242K	228	74K
LBL-2	Nov/09–Feb/10	1.1B	1.2M	1,047	38K
ICSI-1	Sep/11–18/07	137M	2.1M	193	486K
ICSI-2	Sep/11–18/08	163M	1.9M	177	277K
ICSI-3	Sep/14–21/09	334M	3.1M	170	253K
ICSI-4	Sep/11–18/10	298M	5M	183	189K
Dartmouth	Jan/4–21/04	1B	4M	3,782	132K
SIGCOMM	Aug/17–21/08	11.6M	133K	152	29K
Total	Jan/2004–Sep/2010	3.3B	17.7M	5.9K	1.4M

Table 3.1: Overview of packet trace statistics.

College's wireless network. These latter two datasets are available from the CRAWDAD data repository [Cra, SLS09, HKA04]. The datasets we use span from 2004 to 2010 and the table lists the dates of the data collection, the number of packets collected, the number of TCP connections observed, the number of local clients monitored, and the number of remote servers contacted. We consider only connections initiated near the tracing vantage point for our analyses since we are exploring a TCP sender behavior.

3.2 Data Analysis

We begin by establishing how often SYN packets are retransmitted in the datasets we analyze. We initially focus only on connections with retransmitted SYNs to establish the benefits of reducing the initial RTO. SYNs are retransmitted in 0.03% to 2% of connections across our datasets. The ICSI-4 trace contains the lowest percentage of retransmitted SYNs while the LBL-1 and Dartmouth traces contain the largest percentage. Observing up to 2% of connections with retransmitted SYNs represents a non-negligible portion of the overall connections, implying that reducing the initial RTO would have a real, measurable impact for network traffic.

We next shift our attention to connections which could be harmed by a lower initial RTO. Reducing the RTO from 3 seconds to 1 second should not be done without first

analyzing the number of connections which have RTTs longer than 1 second as these connections would always timeout after sending a SYN. For almost every dataset we examine, an initial RTO of 1 second would result in fewer than 0.1% of connections spuriously retransmitting the SYN. The exception to this observation is the Dartmouth dataset, where approximately 1.1% of connections experience an initial RTT of more than 1 second and hence would spuriously resend the SYN with a 1 second initial RTO. We believe this dataset has longer RTTs due to RF effects.

We note that a 1 second initial RTO will be too aggressive for any connection with an RTT greater than 1 second. In order to mitigate spurious retransmits that would be caused by an overly aggressive initial RTO, connections that do timeout after an initial RTO should reset their RTO to 3 seconds after the initial timeout. This prevents the case where the RTT of the connection falls between 1 second and 3 seconds and continually forces the connection to timeout. After the connection is established, the RTO can be adjusted based on measured RTTs, per the standard algorithm. When the initial RTO is too short, there are two penalties for the connection. First, the connection will send a spurious SYN packet. Second, the initial congestion window for the connection will be limited to a single segment [APB09]. While the spurious SYN is a negligible penalty to pay when it comes to connection performance, having a limited congestion window does put connections with RTTs longer than 1 second at a disadvantage. The effect of the second penalty should be modest on overall network performance as these penalties are incurred rarely in the datasets we observe.

While a small percentage of connections would be penalized by having to send a spurious SYN, there are obvious performance benefits that come from retransmitting lost SYNs with a reduced initial RTO. Across our datasets, the percentage of connections that retransmitted a SYN and would realize at least a 10% performance improvement by using the smaller initial RTO specified in this document ranges from 43% (LBL-1) to 87% (ICSI-4). The percentage of connections that would realize at least a 50% performance

improvement ranges from 17% (ICSI-1 and SIGCOMM) to 73% (ICSI-4). Generally, the smaller the amount of data to be transfered, the greater the benefit there is to reducing the connection length by 2 seconds. In other words, while an hour long streaming video would not have its throughput increased by much with a 2 second reduction in connection length, transfering a small web object in 2 seconds rather than 4 seconds represents a doubling of throughput for the connection.

3.3 Conclusion

This chapter examines the performance benefits and penalties of reducing the intital RTO for a TCP connection from 3 seconds to 1 second. Up to 2% of connections in our datasets retransmit their SYN packet and could benefit from a reduction in the RTO. We find that the benefits of reducing the RTO far outweigh any penalties that would be incurred by connections as RTTs are generally short enough that only around 0.1% of connections would be penalized by having to send one extra SYN and limit their initial congestion window due to a premature timeout. This is in contrast with 2% of connections that must retransmit their initial SYN due to loss which would benefit from a shortened RTO. Up to 73% of the connections with initial loss of a SYN would see a performance increase of at least 50% if the RTO were reduced to 1 second.

These results were used as the basis for updating the specification for TCP's initial retransmission timeout in RFC 2988 [PA00]. The updated RTO specification in RFC 6298 [PACS11] allows TCPs to use an initial RTO as low as 1 second.

Chapter 4

Deriving Application Sending Patterns From the Transport Layer

In this chapter we seek to broadly understand the ways that modern applications use the underlying protocols and networks. In particular, we are interested in the transmission patterns of applications as viewed at the transport layer. While previous studies have documented these issues to some degree, we are motivated by the following two points.[1]

- We aim to ensure that our mental models of application-imposed behavior are up-to-date. For instance, [PF01] suggests that while application behavior varies, when simulating Internet traffic a reasonable rule of thumb is to use connection sizes described by the log-normal distribution. In other words, a TCP connection is established, a given number of bytes sent, and then the connection is torn down. This behavior approximates traditional applications like HTTP/1.0 and FTP. However, some in the community have stated their belief that applications' use of TCP has evolved to a more transaction-oriented nature wherein an application re-uses connections for a number of small transactions (e.g., as part of a web application) [Che12].
- Second, good network engineering crucially depends on an empirical understand-

[1] The work in this chapter resulted in publishing of [SBA14].

ing of the system. For instance, intrusion detection systems must understand the difference between an abandoned connection and a quiescent application. Another example is understanding the importance of the so-called "last window" problems in TCP (e.g., [DCCM12]). The amount of justifiable additional complexity in TCP to deal with such problems depends on whether there is one "last window" in a connection (e.g., the bulk transfer case) or there are numerous "last windows" (e.g., at the end of every transaction in a connection with many transactions).

As an initial check on these two points we examine packet traces from the Lawrence Berkeley National Laboratory (LBNL) and the International Computer Science Institute (ICSI). For each connection we compute the maximum duration between data segments. Bulk transfers would tend to show sub-second gaps, while multiple distinct transactions would likely show a larger maximum gap driven by application behavior. We find that in both datasets, the *proportion* of connections with maximum gaps of more than one second and the *duration* of the gaps increases over time. In the LBNL dataset roughly 55% of the connections have a maximum silent period of at most 275 msec in both 2003 and 2013. The distributions then diverge with 4% more connections containing a gap of at least 1 second in 2013 than in 2003 and 12% more connections having a gap of at least 10 seconds. Similarly, in the ICSI data, the distribution of the maximum gap per connection is similar for 2007 and 2013 data up to 1 second—covering about two-thirds of the connections. However, 13% more connections have a maximum gap of at least 10 seconds in 2013 than in 2007. While this analysis is simple and anecdotal it suggests an in-depth exploration of modern application behavior is warranted.

We use packet-level traces from two vantage points—a small research laboratory and a small residential network—as the basis of an initial study into application patterns from TCP's perspective. We contribute both an application agnostic methodology and an initial understanding of modern TCP-based applications.

	CCZ	ICSI
Time	2/11–3/12	9/12–3/13
Length (hrs)	98	1,176
Total Conns.	6.5M	56.9M
Conns. w/o Data	2.6M	27.9M
Port Filtered	-	1.4M
Remaining	3.9M	27.6M

Table 4.1: Data overview.

4.1 Related Work

There are two general classes of related work. First, there is a vast and long-standing vein of work that characterizes and models specific application protocols. These studies span much time and many protocols, from the largely outdated (e.g., [Pax94]) to a rich understanding of early web traffic (e.g., [AW97, BC98]) to modern applications (e.g., [XYLL12]). A second class of previous work attempts to identify applications based on the behavior they exhibit on the network (e.g., [KPF05, KkcF+08]). We do neither of these things, preferring to understand the traffic patterns applications impose on the transport protocol.

4.2 Data

We analyze the two sets of packet traces summarized in Table 4.1.[2] The first dataset is gathered from the border of a residential fiber-to-the-home network, the Case Connection Zone (CCZ) [Cas]. The CCZ connects roughly 90 residences with bi-directional 1 Gbps fiber. While the connection is abnormal for US residential users, we establish in Chapter 2 that actual use of the bandwidth is modest—topping out at roughly 10 Mbps in the typical case—and the application mix is in line with previous studies of residential network users. Our second dataset is gathered from the border of the International Computer Sci-

[2]Note, the LBNL data we present earlier is anecdotal in that each trace covers only a single hour. We believe it is useful for motivating the problem, it is not sufficient for deeper analysis and therefore not used in the remainder of the paper.

ence Institute, and covers roughly 100 users. In both cases we gather data between the 11^{th}–17^{th} of each month. We capture all packets from our ICSI vantage point. Our measurement capabilities within the CCZ network are more modest and we collect a one-hour trace from a random time for each day. As we develop in more detail in Chaper 2, the CCZ measurement apparatus does not often drop packets during the collection process, with no detectable measurement-based loss in the majority of the traces and the loss rate reaching 0.013% in the worst case. The tracing apparatus at ICSI experiences more measurement-based loss than the CCZ monitor, with an average loss rate of roughly 2.1%. We account for measurement-based loss in our analysis by either not considering missing packets or inferring their existence (by noting progression of TCP's sequence space for missing packets), as appropriate.

We prune the datasets before use for two reasons. First, we do not consider connections that do not have at least one byte of data flowing from the monitored network to the remote network. This rule largely removes scanning and backscatter. Further, in the ICSI dataset we noticed two large traffic anomalies that turned out to be part of an independent experiment: (i) a large crawl of the *whois* databases and (ii) a large backhauling of data to Amazon's EC2. These activities are sufficiently voluminous to affect our results. Therefore, since this traffic is also abnormal, we filter it from further analysis. Table 4.1 shows the number of connections we remove from further analysis.

4.3 Dividing Connections

Our general strategy for analyzing application behavior is to take stock of the amount and temporal location of silence in TCP connections. Under this model, traditional bulk data transmission would show few instances where a connection was not actively transmitting data in one or both directions except at the beginning and end of a connection. Of course, our approach is not fool-proof. For instance, streaming may look like bulk transfer in that

there are few silent periods, but may be pushing only as fast as required for the given media and not as hard as a bulk transfer. While this is also an important aspect of application behavior to understand, we leave it for future work.

Given our data, we do not have details of the precise application operations. Additionally, our lack of application payload precludes a study based on application protocol semantics.[3] We approximate application behavior with the following process:

ON/OFF Periods: As a first cut we divide connections into ON and OFF periods with respect to the transmission behavior of the local host (the host close to our monitor) in the connection. Each connection begins in an OFF period and transitions to an ON period when we observe the local host sending a data segment. Transitioning from an ON period to an OFF period happens when two conditions are met: (i) all outstanding data sent by the local host is acknowledged (ACKed),[4] and (ii) either the local host sends an ACK containing no data or at least 5 msec passes without the local host sending another data segment. Note that once we are in an ON period we are able to deal with loss from the local host by advancing the TCP sequence number based on local packets being sent after the loss or by noticing a gap in the sequence space once rule (i) is met and all of the outstanding data has been ACKed. Lost packets during an ON period will not change the length of the ON period that we detect. Rule (ii) ensures that the local TCP does not have application data waiting to be sent. A bare ACK indicates directly that the TCP buffer is empty. The 5 msec rule is otherwise necessary to account for TCP's slow start behavior [Jac88, APB09]. Consider a local host that sends a single segment; when that segment is ACKed, criteria (i) is met. However, in slow start, we expect the local host to use the ACK to open the congestion window and transmit additional data. Therefore, data coming within a short amount of time should be considered part of TCP's dynamics and not part of the application's dynamics. We studied the length of the OFF periods without criteria (ii) to find a reasonable threshold, and thresholds of 1–10 msec show similar results. The 5 msec threshold is a somewhat

[3] Additionally, encrypted traffic is not amenable to such analysis.

[4] Note, this criteria naturally keeps original transmissions and their retransmissions in the same period.

arbitrary choice within that range.

Refinement: Two-Way Traffic: The ON/OFF analysis only accounts for traffic in one direction (from the local to the remote). This approach does not reveal the applications' full complexities, but reconstructing the TCP state of hosts distant from a monitor is known to be difficult [Pax97]. Therefore, we use the following heuristics to glean enough information about returning data to conduct our analysis without reconstructing the entire state of the remote host. We couple the ON/OFF classification above with information about the data flow from the remote host to the local host to refine our classification into four types: Local-only periods are ON periods where we do not observe data sent by the remote host, Remote-only periods are OFF periods where we observe data sent by the remote host, Both periods are ON periods where we also find data sent by the remote host, and None periods are OFF periods where we find no data sent from the remote host. N periods are a first approximation of the silent periods we describe at the beginning of this section. We find that R periods hide silence at times. Consider the case where a single data segment is sent from the remote just after the start of an OFF period and then the connection goes silent for a long period of time. In this case, we classify the entire period as R, when most of the period is in fact silent. We remedy this by terminating an R period—at the point of the last data segment arrival—if twice the minimum observed RTT for the connection elapses without another data segment from the remote host. Twice the minimum RTT provides some robustness to network and TCP behaviors while ensuring that the model transitions in a timely fashion. An N period is inserted for the remaining duration of the shortened R period. R periods that do not trigger this rule may still contain some silence, but the duration of this error is bounded by twice the minimum RTT. Together, these heuristics provide a conservative estimate of the silent periods. Any N period in the analysis is a true silent period, but there may be short application silences hidden in L, R, or B periods.

As a next step, we build a map for each connection that consists of a string corresponding to the order of the various periods in the connection. For instance, a map of

Location	CCZ	ICSI
No N	31%	51.2%
Internal-only	14.4%	18.3%
Trailing-only	32.3%	20.7%
Internal & Trailing	22.3%	9.8%

Table 4.2: Prevalence of N periods at various positions.

NLR indicates an initial OFF period, then a period of local data transmission and the connection ending with a period of data transmission from only the remote host. We find over 155 K and 579 K unique maps within our CCZ and ICSI datasets, respectively. This shows that the applications display significant variety in their behavior. Over millions of connections, we find an average of 25 and 50 connections share each map in the CCZ and ICSI datasets, respectively. Further, we find that there are 12 "popular" maps, or maps that make up at least 1% of the connections, in the CCZ dataset and 10 popular maps in the ICSI dataset. Three maps—NBN, NLR and $NLRN$—are popular in both datasets. Popular maps account for a total of 63% of the connections in both datasets. These results underscore the vast heterogeneity in application behavior observed.

Next, we analyze where N periods fall within connections. Since many connections start with an N period following the three-way handshake due to TCP dynamics, we ignore initial N periods for this analysis. Table 4.2 shows the prevalence of N periods in various locations within the connection. First, we find that about one half to two thirds of the connections in both datasets contain periods where the application is silent. We believe this illustrates that the majority of the connections are not simple bulk transfers. Further, we find that of the connections with silent periods a plurality have only "trailing" silent periods (e.g., persistent HTTP keeping a connection open in case further requests are forthcoming, but ultimately closing with no such requests). Finally, we find that between a quarter and a third of the connections have an internal silent period, indicating an application pause. We present in-depth analysis in the next two sections.

4.4 Trailing Silent Periods

We first study trailing silent periods, or connections that transfer data and then go silent before terminating. Persistent HTTP follows this model, as connections may speculatively persist after the "final" response in case the browser subsequently needs more objects. This mechanism aids performance by allowing subsequent transactions to avoid the overhead of starting a new connection [NGBS+97]. As we note above, 54.6% and 30.5% of connections from CCZ and ICSI, respectively, end with a silent period. Note that these connections may not violate the bulk transfer model of TCP behavior, as they may behave as bulk transfers that simply do not close immediately when activity completes.

Figure 4.1 shows the distribution of the duration of trailing silent periods. Trailing silence of less than 1 second happens in about 30% and 20% of the connections for CCZ and ICSI, respectively. These likely represent applications finishing processing tasks before closing the connection. On the other hand, we find that just under half of the trailing silent periods last longer than 10 seconds in both datasets. This likely indicates the application speculatively leaving a connection open in case further work materializes—which never happens in these cases. These trailing silent periods can be lengthy, with nearly 20–25% of the periods extending beyond 2 minutes. Further, 10% of the trailing silent periods exceed 4 minutes in each dataset.

We next study the behavior of specific applications[5] with respect to trailing silent periods. Figure 4.2 shows the characteristics of each port that contributes at least 1% of the connections with trailing silent periods. The labels on the x-axis indicate the dataset—"C" for CCZ, "I" for ICSI—and port number for the applications, with "other" being a combination of all ports not shown independently. The number just above the x-axis shows the percentage of connections with trailing silent periods that the given port is responsible for in the given dataset. For each port, the box shows the quartiles of the distribution of the

[5] Our traces include only packet headers and therefore we rely on port numbers to identify applications—as crude as that can sometimes be.

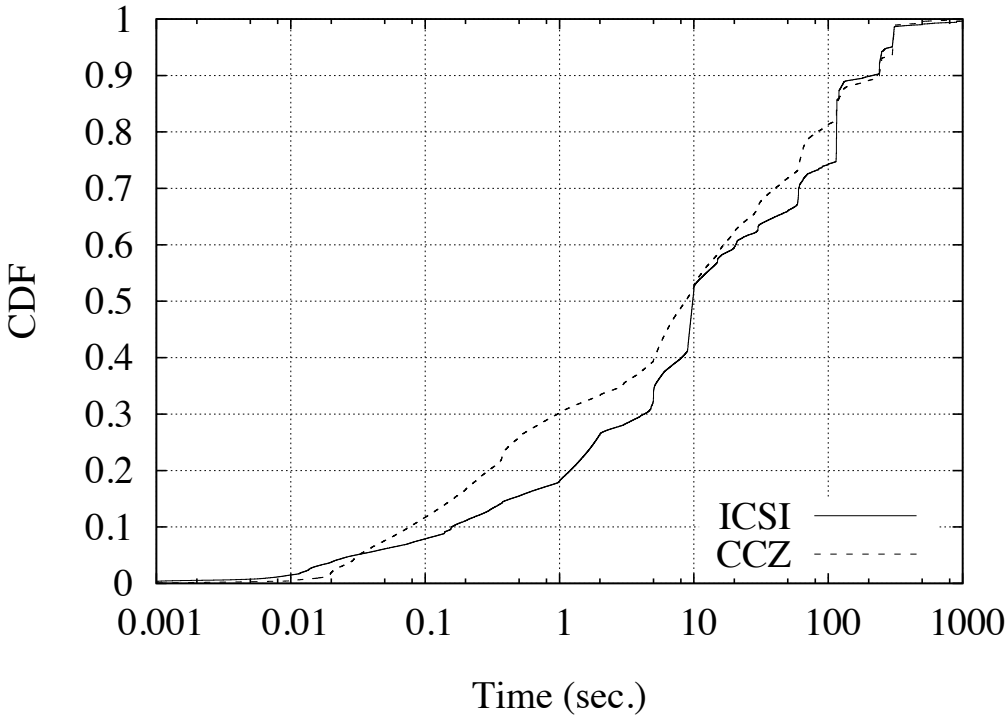

Figure 4.1: Duration of trailing N periods.

duration of the trailing silent periods and the whiskers show the 1st and 99th percentiles.

The figure shows that at least three-quarters of the connections with trailing silent periods across datasets are web traffic (ports 80 and 443) and web traffic generally shows the longest trailing silent periods. Additionally, we find three times as much "other" traffic in the CCZ data as in the ICSI data. This is natural in that CCZ traffic contains more peer-to-peer traffic that is widely distributed across the port range and therefore confounds such simple port-based classification (see Chapter 2 for details). We find that CCZ traffic using port 8332 has short and highly uniform trailing silent periods.[6] The "other" traffic generally has the largest spread of trailing silent periods, as one might expect, given that it is an amalgamation of different applications. The ICSI dataset includes many SMTP connections with trailing silent periods; while half of these are at least 10 seconds, the

[6] As discussed in Chapter 2, we have not been able to fully disambiguate this traffic between Bitcoin and an experimental security camera application known to be in use within the CCZ.

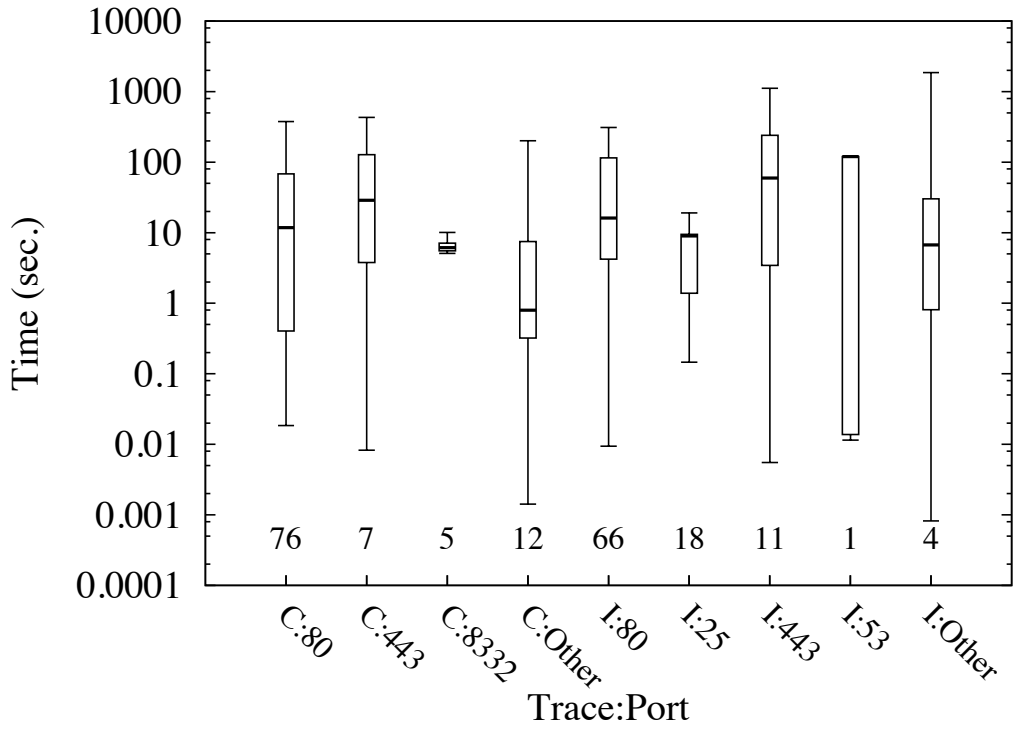

Figure 4.2: Duration of trailing N periods for common ports.

99$^{\text{th}}$ percentile is only 19 seconds, which suggests that a fairly tight timeout is in play. Finally, we find that TCP-based DNS traffic in the ICSI dataset is responsible for roughly 1% of the trailing silent periods. Two ICSI hosts are responsible for most of this DNS traffic, and the general pattern of their connections is consistent with a single, short DNS lookup followed by a 2 minute timeout—which is consistent with the behavior specified in RFC 1035 [Moc87].

4.5 Internal Silent Periods

Our next analysis is of silent periods that happen between periods of activity within connections. These periods indicate an application imposing a non-bulk transfer structure on their activity. There could still be periods in which the application—and therefore TCP—tries to move data as fast as possible in bulk transfer fashion, but these silent periods indicate that

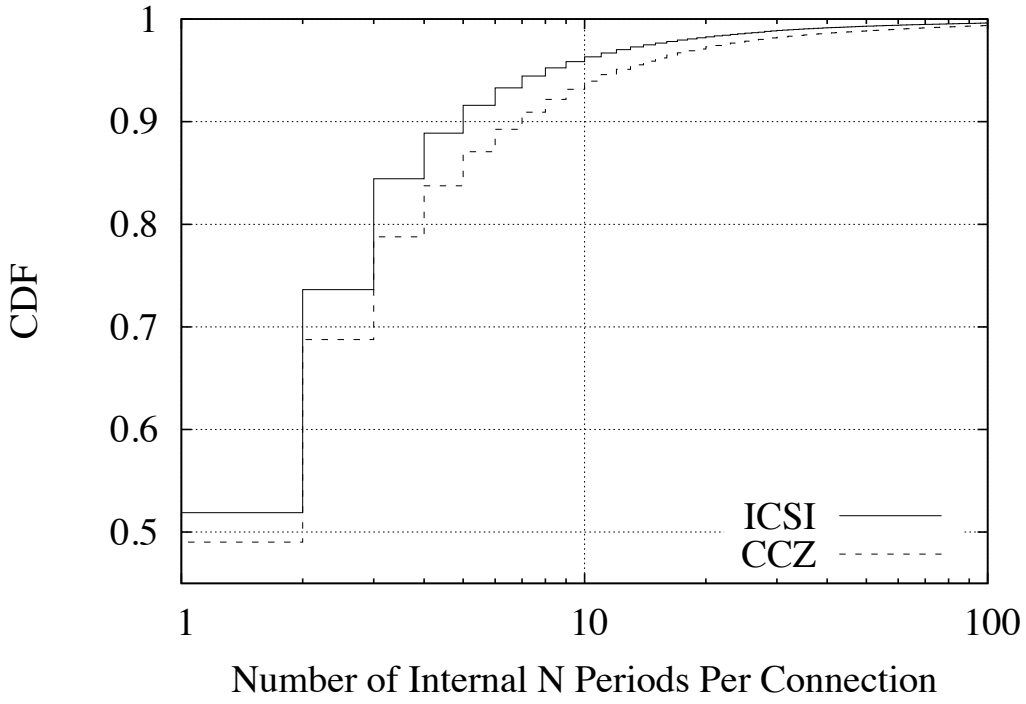

Figure 4.3: Number of internal N periods per connection.

is not the applications' exclusive goal.

Silent Periods Per Connection: Recall from Table 4.2 that 36.7% and 28.1% of the connections in the CCZ and ICSI datasets, respectively, contain at least one internal silent period. From this we understand that a non-trivial fraction of the connections are not solely concerned with bulk transfer. Figure 4.3 shows the distribution of the number of internal silent periods per connection in our two datasets. We find general agreement between the datasets with roughly half the connections having only one internal silent period, and over 90% of the connections having no more than ten internal silent periods. Therefore, while we find that internal silent periods are not rare, we also find that they are in general not numerous on a per-connection basis.

Figure 4.4 breaks down the number of silent periods per connection by port for ports that contribute at least 1% of the connections with internal silent periods. Again, the overall fraction of connections is given just above the x-axis, the bars represent quartiles and the

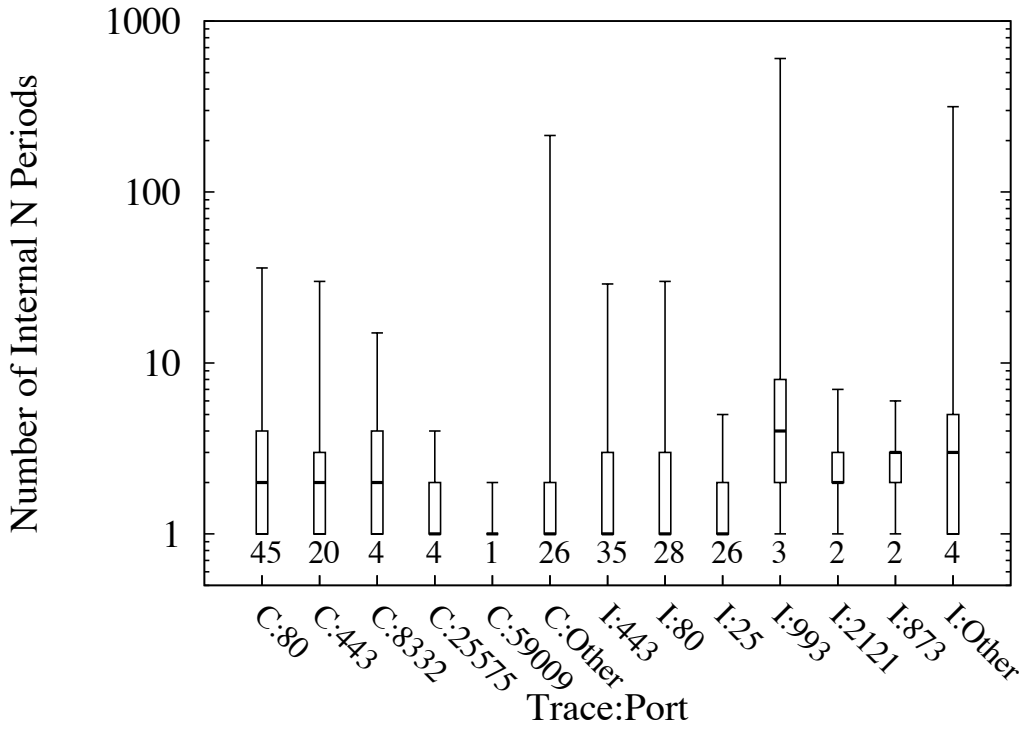

Figure 4.4: Number of internal N periods per connection for common ports.

whiskers show the 1st and 99th percentiles. We find that over 60% of the connections with internal silent periods in both datasets are web traffic (ports 80 and 443). Further, most of the popular ports have a median of one internal silent period per connection and the 75th percentile is under 10 periods across ports. This is consistent with the overall distribution given in the left figure and shows that popular ports do not drastically depart from the overall distribution. We do find that IMAP connections at ICSI (port 993) show a large 99th percentile—604 silent periods. This is expected for email clients that leave connections open for pushed email.

Silent Period Duration: We next assess the duration of internal silent periods, as we show in Figure 4.5. This plot shows that most such periods are short—with at least 30% lasting at most 100 msec and two thirds lasting at most 1 second. These durations are consistent with the "active off" periods previously identified in web traffic [BC98]. However, more than 10% of the internal silent periods across connections last at least 10 seconds. These

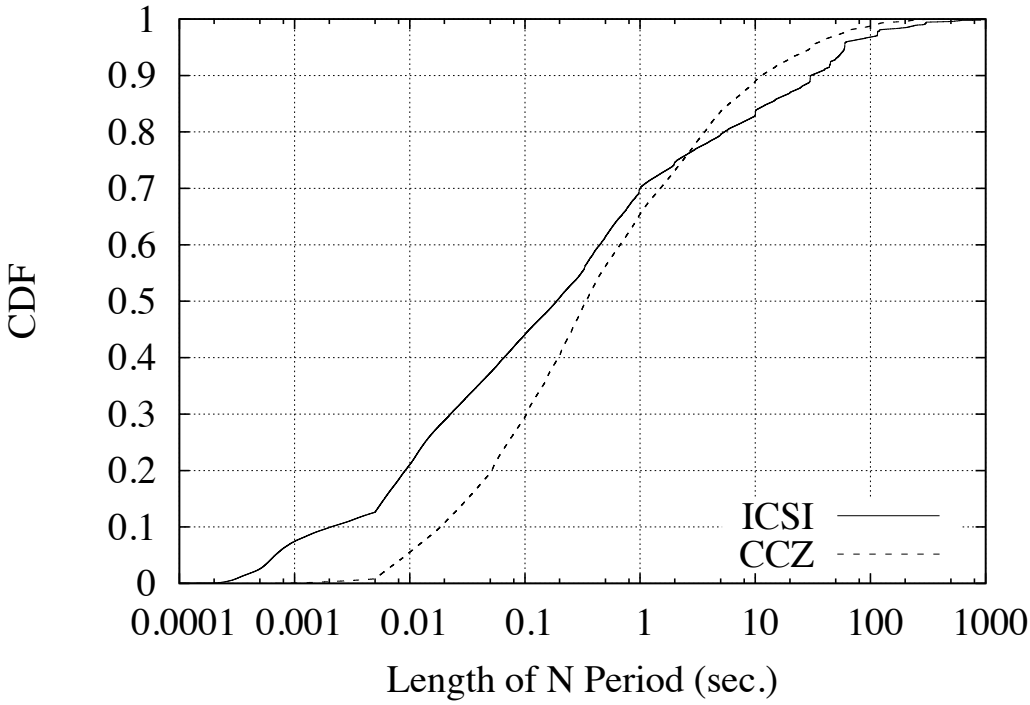

Figure 4.5: Duration of internal N periods.

periods likely represent applications that have run out of networking tasks.

The duration of internal silent periods is not as uniform across applications as their number, as shown in Figure 4.6. For example, SMTP (port 25) is largely rapid exchanges, with 75% of silent periods lasting less than about 100 msec and no silent period lasting more than a few seconds. On the other hand, web traffic (ports 80 and 443) shows significantly longer internal silent periods in both the ICSI and CCZ traces. Interestingly, we note that port 443 has longer internal silent periods than port 80 in both datasets—but more exaggerated in the ICSI dataset. We speculate that this may be due to more aggressive caching of HTTPS connections to avoid the higher setup cost of SSL/TLS.

We now turn from focusing on individual internal silent periods to the amount of aggregate silence we find across an entire connection. We calculate the total fraction of each connection with least one internal period that is spent in silence. Figure 4.7 shows the distribution of the total fraction of each connection that is spent in silence. We find that

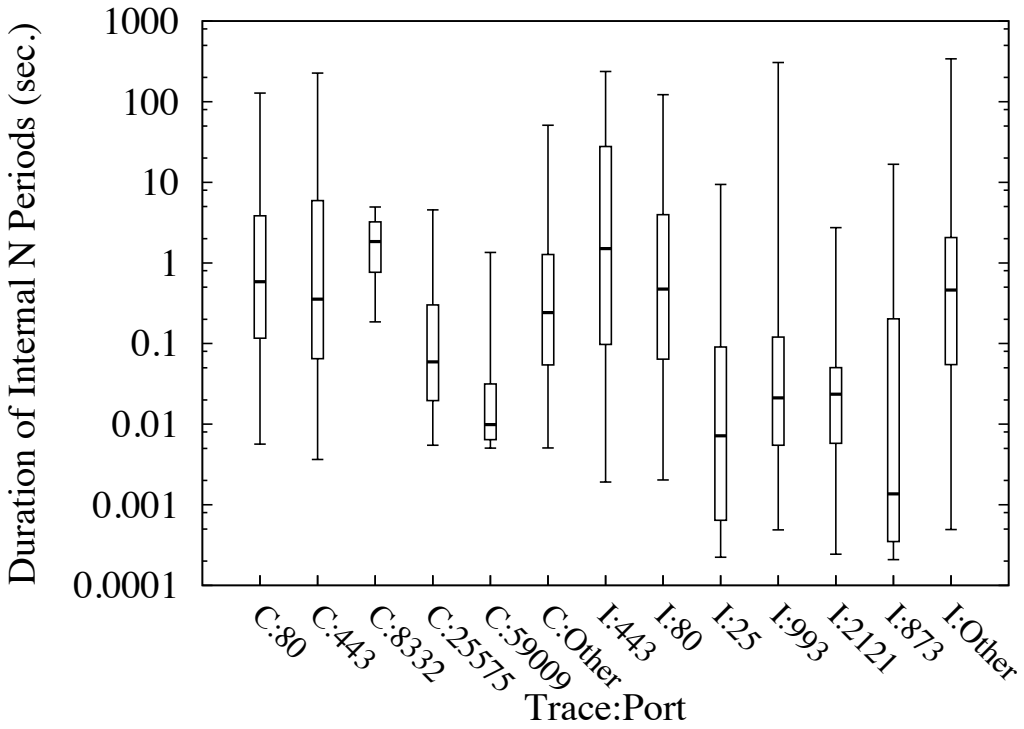

Figure 4.6: Duration of internal N periods for common ports.

two thirds of the connections are fairly uniformly distributed between nearly no silence and roughly 90% silence across the connection. However, in the other one-third of the connections across datasets over 90% of the connection is silent—with roughly 20% of the connections in both datasets showing near total silence. The distribution of the number of silent periods for connections that are at least 90% silent shows that these connections have more silent periods than the overall distribution (which is shown in Figure 4.3)—indicating that a single silent period is not driving the overall behavior.

The Last Window Problem: TCP's loss recovery depends on the acknowledgment of packets received. The information in returning ACKs is used to drive retransmission decisions, by assuming that multiple incoming ACKs that do not acknowledge outstanding data indicate that the data was lost. However, ACKs are sent only when data is received, and there is no data after the last window to generate new ACKs. Hence, it is comparatively more difficult for TCP to determine that the final packets of a window have been

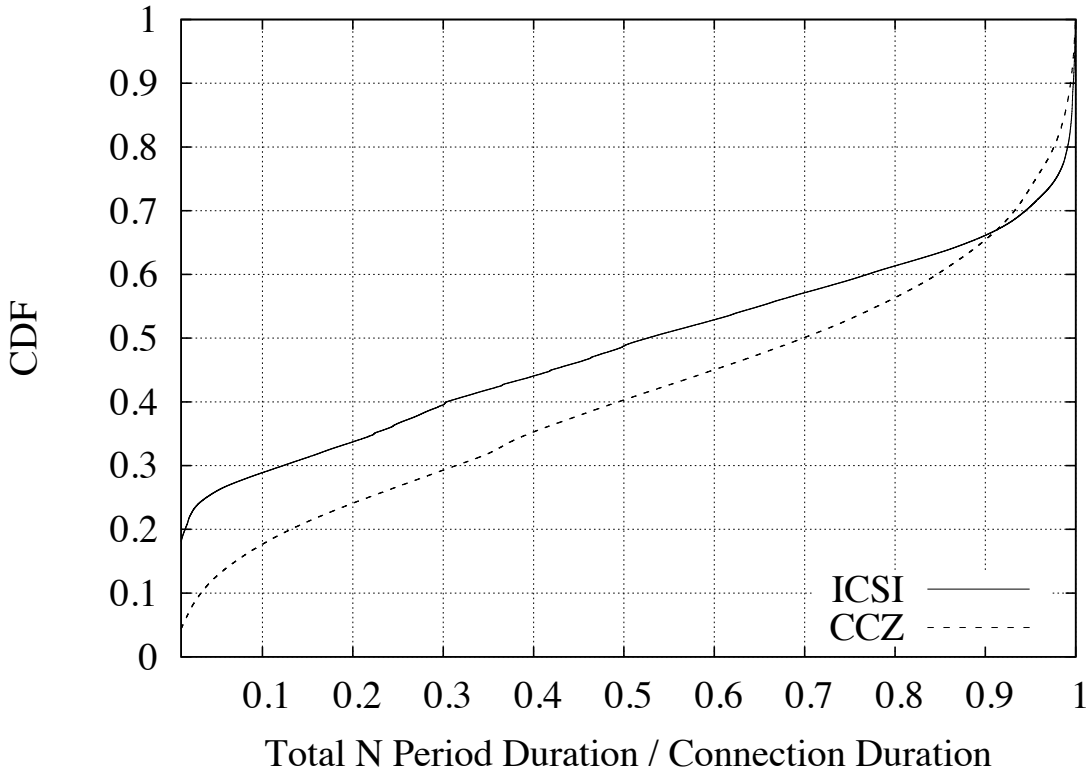

Figure 4.7: Relative connection duration spent in an N period.

lost; in many algorithms, this situation is detected only by a relatively long retransmission timeout (RTO). TCP also uses ACKs to trigger the transmission of new data. However, after a period of silence there are no incoming ACKs, and thus this "ACK clock" cannot be used to immediately pace out new data. This can lead to either a large burst of segments [Jac88, VH97] or the need to wait a full RTT for ACKs for the new data to return [VH97]. In other words, events that happen in a routine and timely fashion most of the time can be problematic at the "end" of a connection. A silent period within a connection can manifest the same behaviors.

Various proposals exist to deal with TCP's "last window" (e.g., [DCCM12]). However, understanding the frequency of this phenomenon is crucial to determining how much complexity should be added to TCP to deal with the issue. Our approach to assess this is to treat the window before a silent period as a "last window" as long as the silent period is relatively long, which we define as roughly the length of an RTO. We use this approxima-

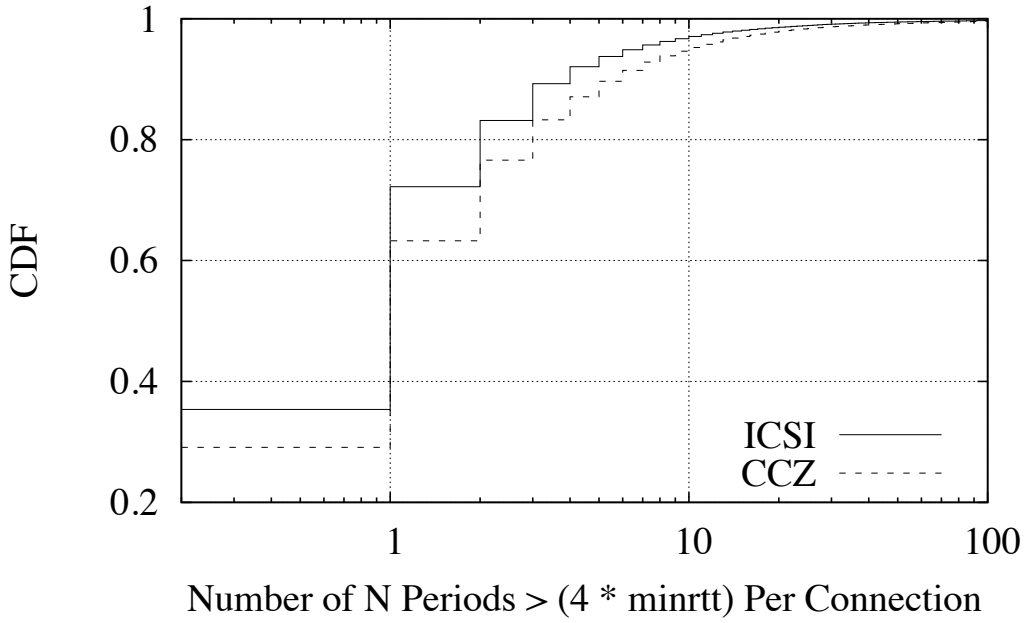

Figure 4.8: Number of N periods $>$ RTO.

tion because of the recommendation that TCP collapse its congestion window after an RTO worth of idle time [APB09]. Since the specifics of the RTO vary across implementations we use $4 \times minRTT$ as an approximation.

We find that 65–71% of the connections have internal silent periods that last at least $4 \times minRTT$—which represents at least a doubling of last windows (i.e., one internal and one actual last window). Figure 4.8 shows the distribution of the number of silent periods that exceed $4 \times minRTT$ per connection. We find that 32% and 24% of the connections that have internal silent periods for CCZ and ICSI, respectively, have 2–10 silent periods of at least $4 \times minRTT$. These results show that a non-trivial number of connections would benefit from techniques that mitigate last window issues.

Class	Med.	Mean	StdDev	# Cnns
CCZ Active	2	2.80	1.13	139k
CCZ Simple	3	3.45	1.34	2.5M
CCZ Complex	8	20.0	199	1.4M
ICSI Active	2	2.66	5.15	4.3M
ICSI Simple	4	4.79	3.88	19.8M
ICSI Complex	8	27.2	714	7.8M

Table 4.3: Length and diversity of connection maps.

4.6 Application Complexity

We next assess the diversity of patterns of activity within connections. For this analysis, we classify connections into three types: (i) "active" connections consist only of L, R, and B periods, with no N period, (ii) "simple" connections may have initial and/or trailing N periods, but all other periods must be L, R, or B (note that active connections are a subset of simple connections) and (iii) "complex" connections which may have any combination of periods. Table 4.3 shows a summary of our analysis. The data suggests that active and simple connections are much more likely to consist of a small number of exchanges followed by termination, whereas complex connections—those with at least one internal N period—display a large diversity of internal structure, involving a comparatively larger number of exchanges and period transitions.

The tendency of simple connections to be classic bulk transfers is strong. Out of the CCZ simple connections, 90% of the maps (2.2M connections) consist of no more than two periods containing data—with 60% being LR, with or without initial and trailing N periods—suggesting a simple request-response bulk transfer. The ICSI data is somewhat more diverse, with the corresponding maps accounting for 47% of the simple connections. Further, 40% of the connections are either LR or RL with or without initial and trailing N periods. This suggests that the simple connections in the ICSI dataset are somewhat more complicated than in the CCZ dataset, but the overall diversity remains markedly lower than for complex connections.

4.7 Conclusions

This chapter makes several contributions: (i) we provide an application agnostic methodology for studying application patterns from the transport's perspective, (ii) we confirm that TCP is non-trivially used for non-bulk transfer applications, which breaks our often-employed mental model, (iii) while silent periods within connections exist, they are mostly short, (iv) we find that TCP's "last window" problem is exacerbated by the transactional nature of some connections and (v) we find that connections with internal silent periods have more complicated interactions than those without such periods.

Chapter 5

Inferring Filtering via Passive Observation

In this chapter we develop a methodology for broadly understanding policy-based network filtering across the Internet.[1] We begin with three observations from previous work:

Policy-based Filtering Happens: We understand from experience and anecdote that network operators apply policy-based filters to traffic leaving their networks. These filters are used for myriad reasons, including (i) because particular traffic types are not meant to traverse wide-area networks (e.g., internal file sharing), (ii) to prevent services from being leveraged by external devices (e.g., using an internal mail server as an open relay), (iii) to funnel all user traffic through some proxy (e.g., to implement capacity-saving caching or content-based filtering) and (iv) to prevent propagation of malware. The community has previously taken modest steps to empirically understand such filtering. For instance, the Netalyzr [KWNP10] tool determines whether 25 popular services are blocked or not via active probing from within the network under study.

Missing Traffic Illuminates Network Behavior: Previous research shows that we can detect broad network outages by monitoring dark address space for the *curious absence*

[1] The work in this chapter resulted in the publishing of [SCAB15]

of traffic. In other words, when a large darknet suddenly receives no background radiation from a previously active network, we can conclude there is a change in policy. This has been studied in the context of both political events [DSA+11] which cause authorities to sever ties with the Internet, as well as natural disasters [BDcA13] which have the same impact on network traffic, even if these do not share the goal of policies that thwart communication of political adversaries.

Malware is Ubiquitous: A wealth of compromised devices on edge networks try to indiscriminately propagate using a set of vulnerabilities that span services [APT07, WKB+10].

We believe the above suggests we can leverage the ubiquity of background radiation to form an expectation that specific *marker traffic* should arrive from a given origin network. When the expectation fails to hold, we are left with the strong suggestion of a policy-based filter hindering the specific kind of traffic in a given origin network. As a concrete exemplar, we study this technique in the context of over 96 billion Conficker packets that arrive at our darknet to form a broad understanding of TCP port 445 filtering in origin networks across the Internet.

By studying one week of traffic arriving at five /8 darknets—roughly 2.25% of the IPv4 address space—we find evidence that both supports and refutes our hypothesis. We find that in the case of Conficker—a large malware outbreak—detecting silence from a given origin network for a given kind of traffic does in fact allow us to understand the policy filters in place across the Internet. On the other hand, while we observe much malware in our datasets, we find each specific kind of traffic rarely spans enough of the origin networks to broadly develop an expectation that the given traffic should be present and thus develop conclusions based on the absence of such traffic. Therefore, we also learn that searching for silence in darknet traffic is limited to only significant events—i.e., full outages or large malware outbreaks. However, even with the limitations, we will show that the general approach does increase our broad understanding of policy-based traffic filtering.

5.1 Related Work

We leverage a number of technologies and techniques that have been developed by the community, including observing background radiation (e.g., [PYB+04, WKB+10]), and using darknets as an observatory (e.g., [BCJ+05]). None of this previous work addresses the topic of inferring service-level network policy via passive observation, which we tackle in this chapter.

Meanwhile, studying policy-based network filtering of various kinds has previously been conducted via active measurements from the edge network under study (e.g., [CBG10], [KWNP10], [BBHc09], [BHM+07]). The policies the previous work addresses are myriad—from the impact of bogon filtering to the ability to spoof packets to service-level policies. The wealth of work illustrates the interest in this topic. Our goals are similar to some of this previous work; however, our approach is to leverage passive measurements to understand the Internet broadly without the need to instrument every edge network, which is at best a large logistical undertaking.

The closest work to ours is in using the lack of background radiation from a given network to detect large scale outages that stem from natural disasters [BDcA13] or political events [DSA+11]. Our work shares their general notion that a lack of background radiation destined to a darknet can illuminate events within the network. We take this notion a step further and detect service-level policies applied to network traffic.

5.2 Data Collection

We use two primary sources of data. The first dataset is a list of known Conficker infected hosts obtained via the Conficker domain sinkhole [Kri09]. The Conficker worm [PSY09] has been plaguing the Internet since 2008 and, six years later, continues to be the top globally-detected worm in the first half of 2014 [FS14]. It propagates via several vulnerabilities in Microsoft Windows, as well as via dictionary attacks on passwords. Propagation

via the network vector involves scanning random IPs on TCP port 445 [CAI13]. A flaw in the random number generator results in Conficker only targeting IP addresses with both second and fourth octets less than 128, which effectively excludes more than three-quarters of addresses from ever being scanned [RL09]. One of the main ways that Conficker has been disabled by researchers is to pre-emptively determine and register botnet-related domain names—which are generated algorithmically—that the malware uses for command and control. Subsequently, by observing communication to these domains, we are able to discover IP addresses of Conficker-infected hosts [Kri09]. The list of infected IP addresses we use in this study was collected at the same time as our darknet data (described below) and contains 17.5M Conficker infected hosts from 1.6M /24 networks.

The second dataset is a set of packet traces of traffic arriving at five unallocated IPv4 darknets: 23.0.0.0/8, 37.0.0.0/8, 45.0.0.0/8, 100.0.0.0/8, and 105.0.0.0/8. We obtained permission from the Regional Internet Registrars (RIRs) to simultaneously announce these network blocks for one week, January 14–20, 2011. We validated that our routes for these prefixes were globally visible to the majority of Route Views' [Uni] 121 peers during the week of our data collection. In aggregate, our darknet observes traffic to nearly 84M IPv4 addresses or roughly 2.25% of the usable IPv4 address space. While using darknets is a well-known technique (e.g., [WKB+10]), to our knowledge, this is the largest simultaneous IPv4 darknet collection to date.

In total, our darknet data comprises roughly 96.1B packets from 4.1M /24 address blocks in the Internet. Table 5.1 gives a broad characterization of our darknet data. Due to the lack of two-way traffic, we are unable to directly estimate how much measurement-based packet loss impacts our dataset. However, we have previously used the monitor to capture traffic at 1 Gbps without significant loss and the average rate of the darknet data is less than 98 Mbps. Therefore, we do not believe the amount of traffic our monitor failed to collect rises to the point of impacting our high-order conclusions.

Next, we classify the darknet data into five categories: (i) *Conficker* traffic repre-

Address Block	Packets (billions)	Bytes (trillions)	Rate (Mbps)	Rate (Kpps)	Source /24s (millions)
100/8	22.1	1.7	22.5	36.7	3.1
105/8	17.1	1.1	15.0	28.2	2.1
23/8	16.9	1.8	23.4	28.0	2.6
37/8	21.7	1.5	20.3	35.9	2.4
45/8	18.2	1.3	16.6	30.1	2.3
All	96.1	7.4	97.8	159	4.1

Table 5.1: Darknet data characterization.

sents TCP SYNs to port 445 from a known Conficker-infected host; (ii) *Likely Conficker* traffic includes TCP SYNs to port 445 from hosts not on the Conficker-infected host list but to an IP address that Conficker is known to target; (iii) *Scanning* traffic represents TCP SYNs that could not be produced by Conficker processes; (iv) *Backscatter* traffic represents SYN+ACK packets that are likely the result of SYNs spoofed to be from our darknet; and (v) *Other* traffic, which includes all traffic not falling into one of the other categories. Figure 5.1 shows the breakdown of the traffic captured to each /8 we monitor. We note that the amount of Conficker traffic is relatively uniform across the /8 blocks we monitor.

A final caveat is that we cannot verify the source addresses in packets arriving at our monitor. We know spoofing is both possible and likely present—e.g., see the amount of backscatter in Figure 5.1 as an indication of the prevalence of spoofing. Therefore, in the remainder of the paper we take care to include this ambiguousness in our interpretation of the results.

5.3 Preliminaries

Our hypothesis is that we can use the background radiation from malware to infer filtering policies across the Internet. In this section we offer several comments on the efficacy of this approach in general and also for specifically detecting policy-based TCP port 445 filtering.

General Coverage: A natural first question is whether we in fact observe traffic in our

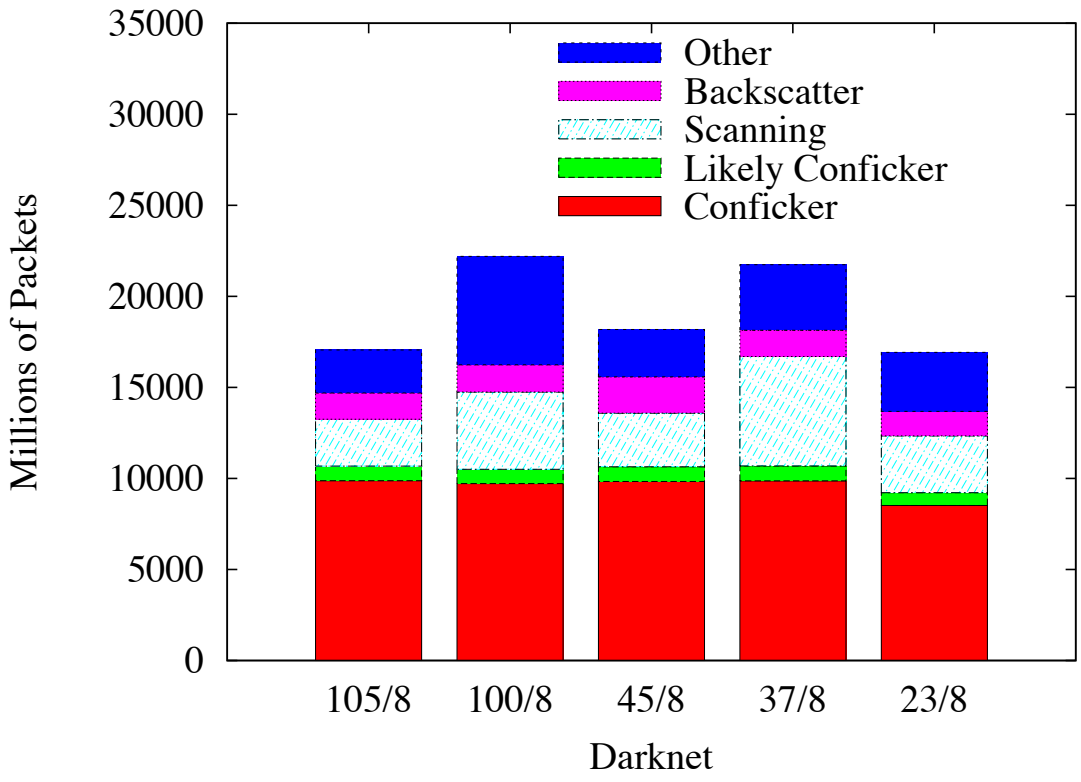

Figure 5.1: Traffic volume by category for each darknet.

darknet from a broad spectrum of Internet endpoints. To quantify the fraction of the Internet that transmits traffic to our darknet we use routing tables from Route Views at the beginning of our darknet collection (January 13, 2011) to determine that 2.43B addresses are routed. The set of /24 networks we receive traffic from corresponds to 2.40B IP addresses when taking into account routed prefix size—or, 98.8% of the routed IP addresses. Some of this traffic is no doubt spoofed, so we compute the number of addresses belonging to /24s that send at least five scanning or backscatter packets.[2] We find 1.85B such addresses—or, 76.1% of the routed IP addresses. This analysis leads us to conclude that background radiation—and the lack thereof—arrives at our darknet from a broad spectrum

[2]Five is a somewhat arbitrary choice that weeds out /24 address blocks that send exceedingly little traffic for illustrative purposes.

of the Internet and therefore is a potential source of information about policy-based filtering in the Internet.

Conficker Coverage: While the amount and breadth of background radiation offers hope that we can broadly detect filtering policy, Conficker is an imperfect marker. As we note above, Conficker-infected endpoints are known to inhabit 1.6M of 4.1M /24 address blocks we observe sending traffic to our darknet. This partially stems from the error in Conficker that prevents it from scanning three-quarters of the network. While the footprint of the marker scopes the amount of the network we can assess, we are unaware of any other technique that achieves this level of coverage. While not ideal, we believe even an imperfect marker can provide a better understanding than we have today.

Conficker Behavior: Another preliminary question we must tackle pertains to the behavior of Conficker. Before we can infer that we are missing some marker traffic, we must have an expectation about how much such traffic we should observe. In order to remain undetected, Conficker infectees only scan after five minutes of keyboard inactivity on a given host [Chi09]. Further, Conficker has four scanning modes—a number of them localized in scope. Finally, an infected host obviously cannot scan when the host is powered off or disconnected from the network. Given these constraints, we cannot simply compute an expectation based on a model of each host scanning continually and uniformly.

We can develop a rough idea of whether we should expect to observe traffic from each infectee, as follows. We know that, when scanning, each infected machine pauses between 100 msec and 2 sec between probes [Chi09]. Given that we observe nearly 84M IP addresses, we would expect to observe one out of every 52 probes—or, one probe every 104 seconds if we assume the slowest scanning rate. Or, if we are to observe 10 probes from a given infected machine on each /8 we monitor, the host would have to scan for 86 minutes over the course of the week—or less than 1% of the week. Therefore, our first order assumption—which we revisit in § 5.4—is that we should observe Conficker activity from all infected hosts.

5.4 Validation

While the cursory analysis in § 5.3 suggests inferring policy-based filtering of TCP port 445 should be possible given both the proliferation of Conficker and our broad vantage point, this section tests our assumptions and frames the confidence we can gain from the results. We note that given the breadth with which we aim to develop understanding, we have no ground truth. Therefore, we cannot absolutely prove our inferences correct, but aim to illustrate that they are likely to be so.

An Anecdote: Comcast provides a list of ports that are subject to policy filtering for its residential customers—including TCP/445 [Com]. In our darknet data we find nearly 3M packets from Comcast's 76.102.0.0/15 address block. As expected, we find no TCP/445 traffic even though our list indicates 81 Conficker-infected hosts within the given address block. While this is an obviously anecdotal case, it is illustrative of our goal to detect policy from the absence of specific traffic from given address blocks.

Conficker Sending Behavior: The preliminary analysis in § 5.3 suggests our darknet is big enough to observe all Conficker-infected hosts scanning with high probability based on what we know about Conficker's behavior. To check this we consider all Conficker infectees from /24 address blocks where we observe some traffic to TCP port 445. In this case, we do not believe there is a general policy against TCP/445 traffic at the /24 level. However, we find TCP/445 traffic from only 51% of the infected hosts across these cases. Our data does not shed light on why we do not observe 49% of the Conficker hosts. The reasons could be many, including policy at finer granularity than a /24 (even to the host granularity), reactive filtering in response to scanning and removal of Conficker from the machine. We combat this situation by requiring multiple Conficker infectees per address block to overcome the seeming failure of some Conficker hosts to send scanning traffic.

Active Measurement: As part of its suite of active measurements, Netalyzr [KWNP10] attempts to establish a TCP/445 connection to a known server. We have obtained the Netalyzr test results starting one month before and ending one month after our darknet data

collection. We find 1,555 hosts in the Netalyzr data that are also infected with Conficker. We therefore can evaluate our technique using the Netalyzr results as ground truth. First, we find 176 hosts (11%) where Netalyzr is run multiple times and shows inconsistent results. This shows that filtering policy and end-host behavior are not consistent across two months and therefore that the Netalyzr data is at best an approximation of ground truth with respect to the darknet data. For another 647 hosts, Netalyzr concludes a port-based filter is in place. The darknet data agrees with this assessment in 97% of the cases. In the 3% of the cases where Netalyzr concludes port filtering, we find a minimum of 17 TCP/445 packets from each host, with a median of 1,369 TCP/445 packets—and therefore we conclude that no filter is in place. We believe the likely cause for this is a policy change. Finally, Netalyzr finds 732 hosts to be unfiltered. However, we only observe 279 (38%) send traffic to our darknet, seemingly leaving our method with a large error. However, we note that the analysis in the last paragraph shows that we can only expect traffic from roughly half the infected Conficker hosts. Applying that expectation, the accuracy of the inference from the darknet data increases to 76%. As we note previously, the error can come from myriad places. Further, we show below that using multiple infected hosts can increase our confidence in our inferences.

Broad Comparison: Finally, we again compare our darknet observations with Netalyzr's results, but instead of using single IP addresses we will now aggregate results across /24 address block, routed block (determined from Route Views) and autonomous system. This allows us to bring multiple infected hosts to bear on our inference, but at the expense of possibly observing multiple policy domains.

Figure 5.2 shows the accuracy of our inference with respect to the Netalyzr results as a function of the number of Conficker infected hosts for the given aggregate block.[3] This plot first illustrates that regardless of level of aggregation the accuracy roughly levels off

[3]There are more Conficker infected hosts in some of the routed blocks and ASes, however, we truncate the plot at 255 for comparison with /24 blocks.

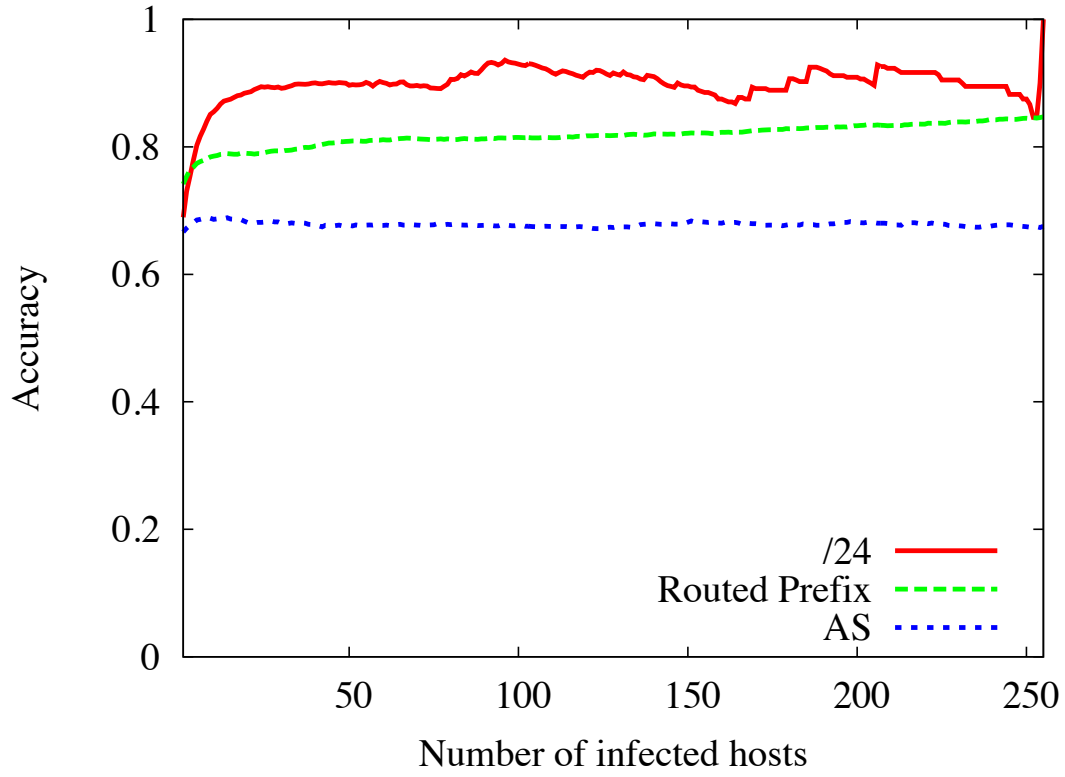

Figure 5.2: Accuracy of methods when when making comparisons with Netalyzr.

once a handful of Conficker infectees are present within the block. Second, the tighter we scope the block the better the accuracy, with /24 blocks showing the best accuracy, followed by routed blocks and then autonomous systems. We believe this is because as we increase the aggregation the instances of multiple policy domains also increases. Therefore, trying to treat the entire block the same leads to incorrect inferences.

We find that approximately half the hosts that contact the Conficker command and control structure ultimately show up in our darknet data. We see this manifests in the accuracy rate in Figure 5.2. Requiring five infected hosts per /24 should mean one of the Conficker infectees sends traffic with a 96% likelihood. When applying this threshold and comparing with the Netalyzr results we find an accuracy of 80%. In approximately 6% of the cases Netalyzr determines the network is filtering traffic while we observe Conficker from the given /24 in our darknet data. Finally, in 14% of the cases Netalyzr is able to

establish a TCP/445 connection while we find no Conficker in our darknet collection and hence infer the given /24 is filtering TCP/445. While the reason for this discrepancy is not clear, we note that it will cause an over-estimate of the amount of filtering in the network.

Summary: As we show in this section, looking for the curious absence of traffic to understand fine-grain network filtering policy is not a clean process. We clearly need to understand the signal we expect to find. However, our conclusion is that, while this process is not perfect, we can use it to gain an approximate understanding of policy filtering in the network. Finally, while active measurements may be more precise, they are much more difficult to obtain on a large scale basis and therefore we are trading absolute precision for breadth of understanding.

5.5 Data Analysis

After establishing the promise of our methodology in § 5.3 and § 5.4, we now return to our high-level goal of understanding network filtering of TCP port 445 traffic using Conficker as a marker.

5.5.1 /24-Based Policy

As we develop above, we believe Conficker is a marker that will illuminate network filtering policy for the broad regions of the network where it is known to exist—even if the marker is less than ideal in some situations. As a starting point, we aggregate and label traffic based on the source /24 address block, our expectations of Conficker, and the traffic that arrives in our darknet.

First, as we sketch in § 5.3, we do not expect Conficker from roughly 60% of the /24 blocks observed at our darknet monitors. For roughly 0.2% of the /24 blocks from which we do not expect Conficker traffic we do in fact observe Likely Conficker at our darknet. This shows that the list of Conficker-infected hosts is comprehensive and not missing a

significant portion of hosts infected with the malware. We do not further consider address blocks where we do not expect Conficker as we can infer nothing from its absence in these cases.

This leaves us with Conficker infectees in roughly 40% of the /24 address blocks in our darknet data. We now need a process to label each /24 address block by its filtering policy. Given our validation work in § 5.4, we proceed in two steps. First, when we observe Conficker traffic from a /24 block we determine there is no general TCP/445 filtering. Second, we know we cannot expect Conficker from all infectees, and so the absence of the marker does not necessarily indicate a network filter. Rather, we determine a /24 block is filtering TCP/445 when (i) we find no TCP/445 traffic in our darknet data and (ii) the /24 block has at least five infectees. As we develop in § 5.4 the second criteria gives us at least 96% confidence that Conficker should arrive and therefore when it does not we infer a policy-based filter.

We find 434K (27%) of the 1.6M /24 blocks with Conficker infectees are not imposing TCP/445 filtering on their traffic. Meanwhile, we infer that 448K /24 blocks (28%) filter TCP/445 traffic. That is, we are able to confidently characterize the filtering policy of 882K /24 networks—or 9.3% of all the routed address space. This is, by far, a larger portion than previous methodologies can claim—e.g., Netalyzr runs from the month surrounding our data collection cover 23K /24 networks. Our analysis leaves 747K /24 blocks (45%) from which we do not observe TCP/445 traffic but which do not contain five infectees. These are cases where we have an indication of possible filtering, but cannot develop high confidence in this determination.

5.5.2 Routed Prefix-Based Policy

We next turn to a larger aggregation of address blocks to better understand filtering policy at a coarser granularity. We leverage routed prefixes as found in Route Views at the time of our darknet data collection for this analysis. Our general method to infer whether filtering

Classification	Amount	Percentage
No Filtering	10,084	13%
Filtering	27,351	35%
Multiple Policies	14,536	18%
Low Signal	22,075	28%
Muddled/No Filtering	5,178	7%

Table 5.2: Labels assigned to routed prefixes /23 or larger.

happens for an entire prefix is to look for consistent behavior from the /24 blocks within the prefix. Since we tackle /24 address blocks above, in this section we only study the 140K routed prefixes that are at least a /23 (out of 254K total routed prefixes).

Of the 140K prefixes we consider, we find no Conficker infectees and no TCP/445 traffic for 61K of the prefixes. We cannot further study these prefixes as we have no expectation of TCP/445 traffic and therefore the absence of such traffic does not inform our assessment of filtering. This leaves roughly 79K prefixes on which we have some expectation of observing TCP/445 traffic. We summarize our results in Table 5.2.

First, when each /24 block containing at least one Conficker infectee within the routed prefix produces TCP/445 traffic we conclude the network applies no general TCP/445 filtering. Table 5.2 shows 13% of the prefixes do not filter TCP/445. Similarly, when we observe no TCP/445 traffic for each /24 block containing at least one infectee across a prefix with at least five total infectees we conclude filtering is in place for the entire prefix. We find prefix-wide filtering in 35% of the prefixes. We also find cases where no TCP/445 traffic arrives at our darknet, but the routed prefix contains fewer than five infectees. We cannot confidently determine that these prefixes filter TCP/445—even if the data suggests this may be the case. We denote these cases "low signal" in the table and find 28% of the prefixes fall into this category.

Finally, we are left with prefixes that have indications of both no filtering—i.e., we observe TCP/445 traffic—and filtering—i.e., the infectee list suggests we should observe more TCP/445 traffic than we do. For cases where we observe traffic from at least five

infectees we conclude that the prefix has multiple policies. In other words, we are confident in our determination that filtering is occurring within the prefix and yet we still observe TCP/445 traffic from the prefix. We find this happens in 18% of the cases. As the size of the address blocks we consider increases this is a natural finding that follows our intuition—i.e., that the block would be split up into multiple policy domains. Finally, we have cases where we observe TCP/445 traffic and there are also indications we should see additional traffic, but from less than five infectees. In this case, we know filtering is not in use across the entire prefix and, even though we have some indication that filtering may be happening, we cannot conclude it is with confidence. We find 7% of the prefixes in this "muddled" state.

We next consider the fraction of each prefix we use to determine its filtering policy. For each routed prefix, we calculate the fraction of the constituent /24 blocks (i) with a known Conficker infectee and (ii) where we conclusively determine that filtering is or is not present. Figure 5.3 shows the distribution of prefixes according to these fractions. The "all" distribution in the plot shows the expected prefix coverage based on the Conficker infectee list, whereas the "classified" distribution shows the fraction of /24 blocks we actually use in concrete prefix classifications. Comparing the distributions shows that, when making a classification, we generally use more of the prefix (i.e., more /24s) than the expectation predicts, which adds to our confidence in the classifications.

Next, we examine the size of the routed prefixes we are able to concretely classify. The distribution of the size of all routed prefixes we consider, as well as the distributions of the routed prefix sizes for each concrete classification we make are given in Figure 5.4. The figure shows that the distribution of network size for networks we can concretely detect filtering policy is similar to the distribution of the size of all origin networks. In other words, neither our detection nor results are biased by prefix size. Further, we find that networks that filter TCP/445 are slightly larger than networks that do not filter TCP/445. This perhaps indicates that operators of larger networks are more diligent about security

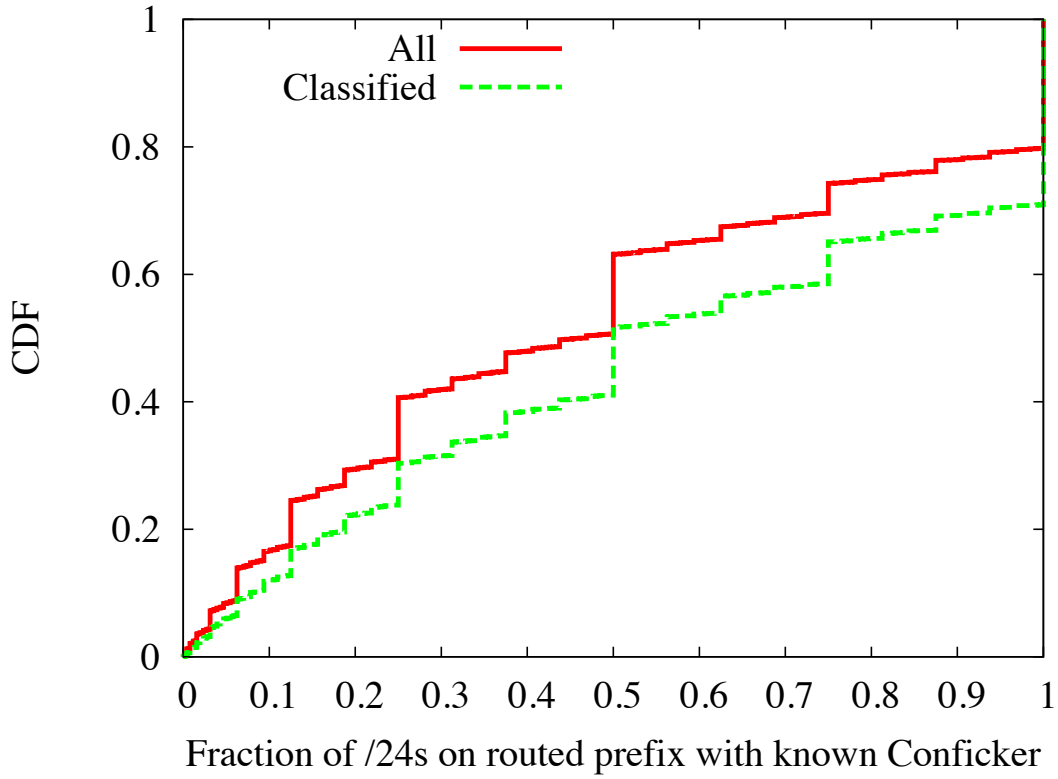

Figure 5.3: CDF of the fraction of /24s on a routed prefix with known Conficker.

policy than those of smaller networks. Finally, we find that networks with multiple policies are larger than networks with a single policy. As we note above, this is natural because as network sizes increase the tendency to have multiple administrative and policy domains to cope with a variety of situations arises.

Finally, we note that we are able to confidently determine a single filtering policy in roughly half of the /23 and larger routed prefixes. This corresponds to 699M IP addresses or 28% of the routable addresses during the week of our darknet data collection.

5.6 Limitations

From previous research we understand that full network outages—whether caused by policy decisions or natural disasters—can be detected by the absence of traffic arriving at darknets. Further, in the previous sections we illustrate that we can use similar strategies

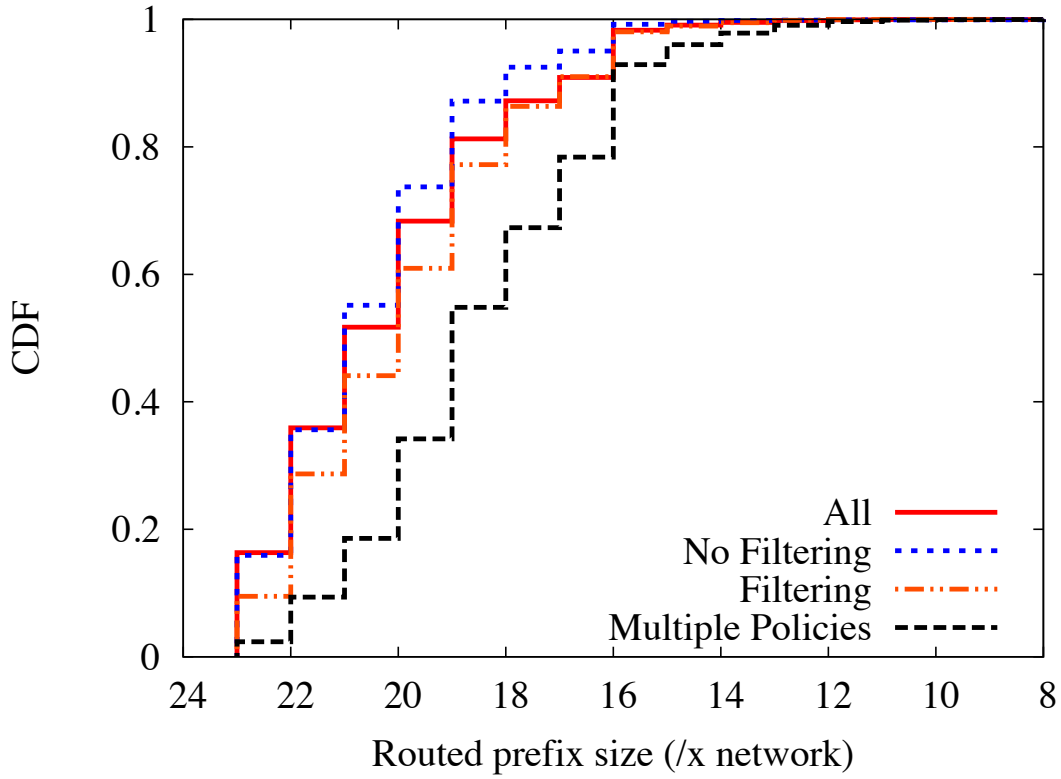

Figure 5.4: CDF of the routed prefix sizes on which we make judgements.

to infer finer-grained policy such as port blocking. As developed thus far, both the course- and fine-grained policy discovery requires big events—i.e., a broad swatch of the Internet becoming unreachable or malware that is both prevalent and energetically propagating.

A natural next question is whether the aggregate background radiation that appears at darknet monitors provides enough information to form further general understanding of policies across the Internet. To address this question we first determine the top TCP ports arriving at our darknet. We then calculate the number of origin /24 networks that source each kind of traffic and compare this to the total number of origin /24s we observe. Table 5.3 shows the results. In the best case—port 80—we find SYNs from only 18% of origin /24s we observe. This either means 82% of the /24s either (i) are subject to policy blocking or (ii) do not source radiation to port 80. We believe the latter is far more likely than the former. That is, background radiation does not in general energetically target our darknet enough to develop a solid expectation that the traffic should be there and hence

Darknet	# /24s Receiving SYNs	% /24s w/SYN for TCP/80	TCP/139	TCP/1433	TCP/22
100/8	2.0M	14.2%	1.5%	<1%	<1%
105/8	1.5M	4.0%	1.1%	<1%	<1%
23/8	1.7M	6.2%	1.0%	<1%	<1%
37/8	1.6M	21.6%	1.0%	<1%	<1%
45/8	1.6M	5.6%	1.1%	<1%	<1%
All	3.1M	18.2%	1.3%	<1%	<1%

Table 5.3: Percentage of /24s observed sending SYNs to prevalent destination ports.

draw conclusions about its absence. Further, for the other top ports the prevalence is even smaller than for port 80 and, hence, makes any conclusions about policy even more tenuous. Therefore, our conclusion is that while the general technique of searching for the absence of traffic can be useful, it has its limits.

5.7 Conclusions

This chapter makes several high-order contributions:

Methodology: We develop a novel methodology for detecting service-level network filtering based on passive observation of traffic markers. While this aspect of the Internet has been previously studied, our passive observation-based technique allows for developing an understanding at a breadth previously unattainable. Using Conficker as our exemplar, we are able to conclusively determine the network filtering policy of 699M IP addresses or roughly 28% of the routed IPv4 address space. Although this is a modest fraction of the Internet, it is much larger than previous attempts. For instance, the original Netalyzr study [KWNP10] reports results from 100K test runs. Even if each Netalyzr run represents a /24 network our results cover 27 times as much of the Internet.

State of TCP/445: Of the address space we can conclusively assess, we find filtering of outgoing TCP/445 traffic occurs in two-thirds of the cases. We also note that as the size of the routed prefix under study increases the chance of finding multiple service-level filtering

policies within the prefix also increases. While we believe it is a natural and expected result that larger networks would encompass more than one administrative and policy domain, we believe this offers a cautionary note in that aggregating too much of the network can dilute any understanding we derive.

Methodological Limitations: Finally, we illustrate that there are limits to the methodology of using the absence of background radiation to infer policy. In particular, we can leverage large events to infer policy, but more run-of-the-mill instances of background radiation are not energetic and wide-spread enough to allow us to form the expectation of traffic and hence draw conclusions when the expectation fails.

Chapter 6

Understanding IGMP *Neighbors2* Response Behavior

The Internet Group Management Protocol (IGMP) [Dee89, Fen97, CDK+02] is a transport layer protocol that allows neighboring routers to exchange and manage multicast group and routing information. IGMP packets are either (i) flooded to neighboring multicast routers via broadcast, or (ii) sent via unicast between specific routers. In both cases IGMP operates as a connectionless protocol.

Broadcasting packets (case i) is limited to the local network where a router resides. Meanwhile, previous work shows that some routers will respond to unicast IGMP requests (case ii) from an arbitrary end host and that these responses can be leveraged to study network topology [MDP+11, MVdSD+09]. These previous efforts leverage the Distance Vector Multicast Routing Protocol (DVMRP) [WPD88] which operates on top of unicast IGMP messages to explicitly request information about a router's multicast neighbors. A DVMRP *AskNeighbors2* request is first sent to a series of routers. DVMRP enabled routers will respond back to the source of the *AskNeighbors2* request with a unicast *Neighbors2* response. Each response contains a list of the router's multicast enabled interfaces. Each interface in turn contains its own list of the interface's multicast neighbors as well as some

ancillary information about the interface such as whether it is down/disabled, whether its neighbors are reached via a tunnel, or whether the interface represents a leaf node on the multicast tree [WPD88].

Our goal in this chapter is to gain an understanding of *Neighbors2* response characteristics via a scan of the IPv4 address space and then to use this understanding to comment on security risks associated with having openly responding DVMRP enabled routers. We focus our study on two key properties of DVMRP enabled routers:

Reflection: Recall that IGMP is a connectionless protocol and that *Neighbors2* responses are sent directly to the source IP address in an *AskNeighbors2* request. If the source IP address in the request is spoofed, a DVMRP enabled router will still respond to the spoofed IP address even though the spoofed IP is not expecting the response. An attacker can leverage this reflection to direct packets to a victim via a DVMRP enabled router. Reflection makes detecting the ultimate source of the unwanted packets difficult for the victim to discern, as the victim will receive packets addressed from the router, rather than the attacker.

Amplification: A *Neighbors2* response will—in theory—always be at least as large as the original *AskNeighbors2* response to which it corresponds. Response sizes are not limited and may be so large that they are either split into multiple, distinct *Neighbors2* packets or fragmented at the IP layer. Any response larger than the initial request provides an attacker with byte amplification, and responses spanning multiple packets allows for packet amplification. Attackers issue a small request, which will then be turned into a larger response by the DVMRP enabled router.

When these properties are combined, an attacker will not only be able to mask their true network location via reflection, but the traffic the victim receives will be amplified compared to the traffic the attacker must send.

In the remainder of this chapter, we seek to understand the reflection and amplification properties of *Neighbors2* responses and then use our understanding of responses to illustrate the potential harm from leveraging DVMRP enabled routers.

6.1 Related Work

Previous work uses *AskNeighbors2* probes while studying network characteristics [MDP+11, MVdSD+09]. Tools developed in these studies, *mrinfo* [MVdSD+09] and *MERLIN* [MDP+11], enable researchers to learn about network topology by studying the routing information contained in the *Neighbors2* responses. The focus of our work differs from these studies. First, our goal is to understand *Neighbors2* response characteristics globally by scanning the entire IPv4 address space for routers that will respond to our probes. Both *MERLIN* and *mrinfo* scan from a seeded list of routers that grows as responses are received which contain additional router IP addresses. As such, their view of the network will be limited to routers that have a path to their starting seeded list. Second, our goal is to study response characteristics in order to understand the security implications of having routers respond to requests from arbitrary hosts. Previous work has a strong focus on the topology information contained in responses and trying to understand when multiple IP addresses correspond to multiple interfaces on a single router.

Routers that will respond directly to packets from arbitrary hosts create a potential security risk via an an amplification attack. Various attack vectors for amplification attacks exist [Ros14] and are well documented (e.g., DNS [AAC+06], NTP [Sys14], SSDP [Sch02], CharGen [Tec13]). While amplification attacks themselves have been studied, to the best of our knowlege no research exists on understanding *AskNeighbors2* requests as an attack vector. Understanding an amplification attack that targets routers—which are typically connected via high-bandwidth links—is particularly interesting as the routers would be capable of receiving and handling large floods of packets which would each be amplified. This is in contrast to other amplification attacks that leverage open network services, like DNS, where the target resolvers used for amplification may be located on low-bandwidth residential links [SCRA13] that would rate limit large floods of packets used in an attack.

6.2 Initial Scan

We begin our study with the goal of scanning the entire IPv4 address space for routers that will respond to *AskNeighbors2* queries. While tools like *mrinfo* [MVdSD+09] and *MERLIN* [MDP+11] exist, they are not well suited for scanning the entire network in a timely manner. Both of these tools scan at low rates in an effort to match responses with the exact probe causing the response in order to accurately map out network topology. Both tools also process responses in real-time as they expand their list of known routers to scan in the future. We operate under a different set of constraints as we do not process responses in real time. Instead, we choose a tool specifically designed to scan the entire network in a timely manner, *ZMap* [DWH13, ZMa]. *ZMap* can either scan at a specified rate or operate with the goal of scanning as quickly as possible based on the available bandwidth on the network. *ZMap* is also extensible through writing custom probe modules that allow arbitrary types of packets to be sent out during a scan. We wrote a custom module for *ZMap* that allows us to send *AskNeighbors2* requests over IGMP.

After working carefully with network administrators at our scanning site we chose a modest scanning rate of 9K packets per second.[1] This moderate rate combined with *ZMap's* random scanning behavior means that we are unlikely to overwhelm any single remote network with traffic. We also realize that observing even a single IGMP packet may come across as potentially alarming on networks that closely monitor traffic and do not expect to observe IGMP. We implement a blacklist for any complaints we received during our scan. We split the overall scan into 10 slices and update our blacklist between slices.[2]

While running our scans with *ZMap*, we simultaneously capture all IGMP packets

[1]This rate was chosen based on the tradeoff of scanning speed versus the need to not overwhelm our edge network with scanning traffic and is based on our particular scanning setup. Scanning at this rate allows our scan to finish in under one week while using only a small portion of the available bandwidth at our edge network.

[2]We received 5 such complaints. The blacklisted prefixes correspond to 135K IP addresses, or 0.003% of IPv4 address space. Hence, we do not believe the blacklist biases our data collection. Additionally, we

Start Date	End Date	Outgoing Pkts	Incoming Pkts.	Responding IPs
2015/01/12	2015/01/18	4.2B	263M	305K

Table 6.1: Overview of data collected.

related to the experiment using *tcpdump* [JLM89] at the border of our network and the wide area network. These packet traces are then analyzed to build our initial understanding of the *Neighbors2* response and responder characteristics.

6.2.1 Scan Analysis

Table 6.1 gives an overview of the data collected during our initial scan with *ZMap*. Out of the 4.2B IP addresses we send probes to, we receive responses from 305K hosts.[3] Out of these responding hosts, 8K (2.2%) respond multiple times throughout our scan. This likely indicates that a router is responding for multiple interfaces through a single outgoing interface.

Given the responses we collect and the addresses we blacklist, we have an IP-based hit rate of 0.007%. While the hit rate is a small percentage of the IPv4 address space, 305K IP addresses represent a non-trivial number of hosts. We next turn our attention to understanding the characteristics of responses we observe from the responding hosts.

We define a single response as a series of packets from a single source IP address I that arrive at our monitor with a maximum of 1 second between packets. A 1 second threshold is consistent with [MDP+11] and our own data analysis which finds that most response packets to a probe arrive within 1 second of sending the probe. Our relaxed definition,

blacklist reserved address space 10.0.0.0/8. As we would not expect to observe responses from this address space, we again do not believe this biases our data collection and analysis.

[3] That is, we receive responses that use 305K unique IP source addresses. Each IP address does not necessarily represent a unique router, as routers can have multiple interfaces each with its own IP address. Therefore, we are likely overestimating unique hosts on the network. For the current work and for ease of exposition, we equate an IP address and a host

which allows responses to keep growing as long as packets arrive within 1 second of the previous packet, ensures we capture all the responses to a given probe. Note that while it is possible that multiple probes may trigger a series of unique responses from a single host, the chances of multiple responses being in consecutive seconds is probabilistically unlikely unless they are triggered by a single request. Assume that a single host responds on behalf of every IP addresses on a single /24, and that the host always responds from the same IP address. Given our scanning rate of 9Kpps and the random scanning behavior of *ZMap*, we expect to send a probe to a /24 network during a specific 1-second period approximately 0.054% of the time. That is, if we have just scanned a /24 in the past second, the chance of hitting that same /24 in the next second is approximately 0.054%. However, rather than send a single response back, certain routers will send a stream of packets much larger and longer than any reasonable response would be. Sometimes these responses contain thousands of packets and last dozens of minutes. Based on our reasoning above, we believe that the chance these packet streams are in response to a multiple probes is statistically unlikely. In addition to the above reasoning, we can trigger streams of packets by manually sending a single probe to certain hosts—i.e., eliminating the chance the stream is caused by multiple requests. We revisit these large, anomalous responses in § 6.4.

Figure 6.1 shows the byte amplification factor for the responses we observe. About 15% of responding hosts offer no amplification. We observe a median byte amplification factor of 2.4. The largest 6% of responses yield an amplification factor of at least 50 and 1% of responding hosts yielding an amplification factor of at least 100. As we are sending packets that are 28 bytes in length, the largest 1% (3K) hosts send responses that are at least 2,800 bytes.

While probes can trigger responses that are larger relative to the requests in terms of bytes, responses can also be split across multiple packets. Whereas one potential attack would rely on byte amplification to exhaust a victim's bandwidth, packet amplification could also exhaust the packet processing capacity of the network. Routers have some

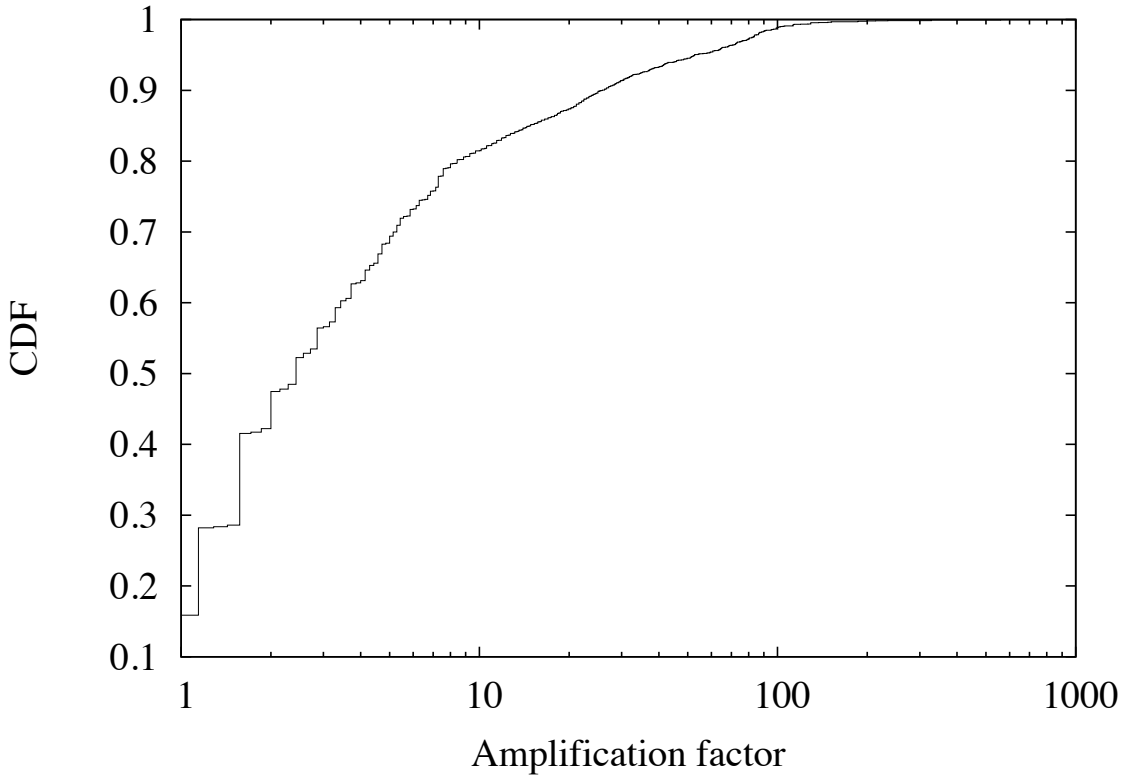

Figure 6.1: Distribution of byte amplification factors during full IPv4 scan.

amount of fixed overhead associated with handling a packet and forwarding it along the correct path and have a limited number of packet buffering slots available. Inflating the number of extra packets a router must process prevents the router from using its resources to handle legitimate packets. In addition to processing time, extra packets taking up too many slots in a router's buffer could cause legitimate, ongoing connections to have some of their packets dropped due to inferred congestion at the router.

Figure 6.2 shows the distribution of packets returned in response to a single *AskNeighbors2* probe. For 87% of the responses, we observe no packet amplification. The final 5% of responding hosts send at least 5 packets in response to a single probe. While a specific set of hosts will yield a moderate amount of packet amplification, *AskNeighbors2* packet amplification is not as great as the byte amplification we observe.

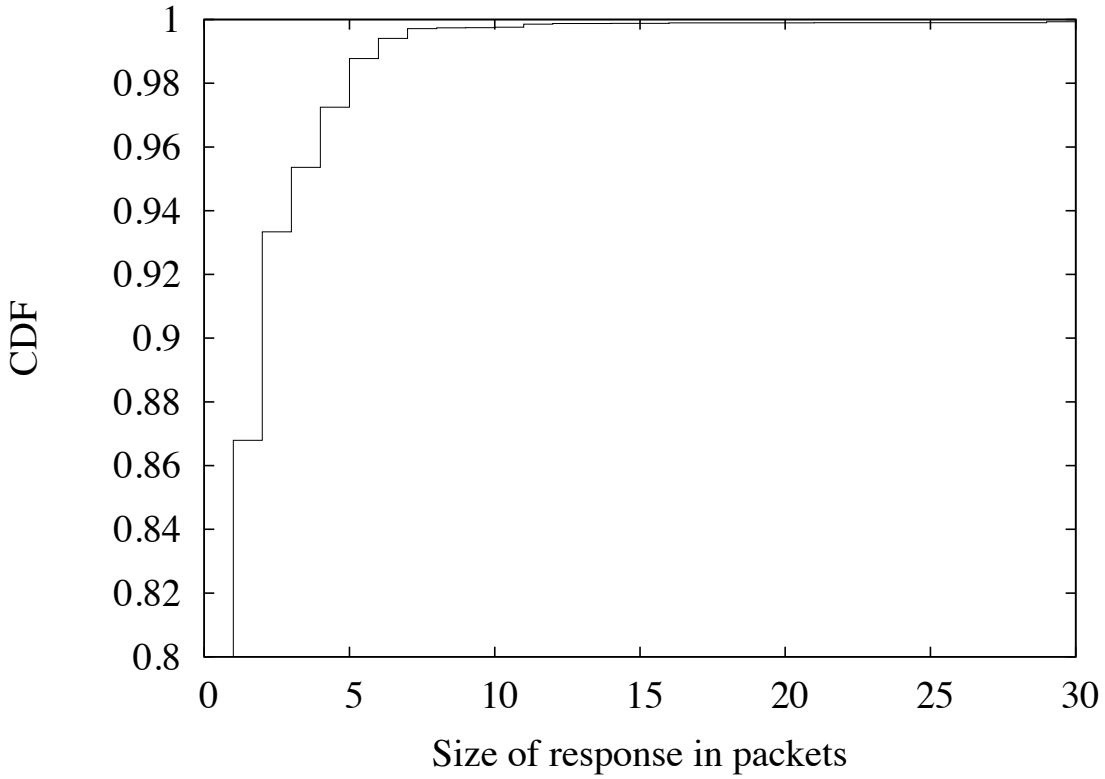

Figure 6.2: Distribution of packet amplification during full IPv4 scan.

6.3 Scanning Over Time

We next focus our attention on studying the stability of responses from each host over time. A natural question is whether the 305K hosts that respond initially will continue to do so in a consistent manner over time. To understand how routers behave over time, we organize three additional rounds of probing. For each response from a host H we record in our original trace, we send an additional probe to H 10, 20, and 30 days after the original response from H was recorded. We collect packet traces in the same network location as the original scan.

6.3.1 Scan Analysis

Out of the 305K hosts that respond to our initial scan, we observe 262K (86%) of them respond to at least one round of our re-probing. We find stability among some routers, as

	Original Scan	**+10 days**	**+20 days**	**+30 days**
Total hosts observed in trace	305K	227K	202K	228K
Hosts observed exclusively during a specific scan	43K	14K	3K	11K

Table 6.2: Summary of hosts observed during scans.

161K (52.8%) respond to all three rounds of re-probing. For the routers responding to some but not all of our re-probes, 73K (24%) respond to two queries and 28K (9.2%) respond to only one out of three rounds of re-probing. Note that there are 43K (14%) hosts that do not respond to any of our re-probes. This could happen for several reasons such as (i) the IP address being reassigned to a new router that is not DVMRP enabled or openly responding to *AskNeighbors2* requests, (ii) subsequent filtering of traffic related to our experiment being implemented after our initial scan along the path to the router, (iii) the IP address could be an outgoing interface on a router that responds on behalf of its other interfaces but does not respond to probes directly, or (iv) response packets from a router could be dropped.[4]

We next consider whether the number of responding hosts we observe decreases over time during our re-probes. Table 6.2 shows the number of hosts responding to the original scan and each re-probe, as well as the number of hosts that appear exclusively in a re-probe. We observe 202K-228K out of 305K hosts that respond to our probes respond during each re-probe. We also note that all three re-probes contain hosts that appear exclusively during that re-probe. This shows that there is churn in which hosts will respond and when they respond. Just because a host is unresponsive on one day does not mean it will remain unresponsive in the future. Likewise, a host responding on a given day does not mean it will continue to respond in the future.

[4]While loss is possible, general loss rates on the Internet are low and would not explain the broad trend of hosts not responding.

6.3.2 Stable Responders

We now turn our attention to those hosts which respond in each of our re-probes. We refer to these hosts as *stable responders*. Studying stable responders allows us to better assess the security risks involved with having DVMRP enabled routers that will always respond to requests from arbitrary hosts. Such behavior would interest a potential attacker, who could add stable responders to a “hit list”, or a list of known targets to leverage during an attack. Once in possession of a hit list, an attacker no longer has to scan for targets to use during an attack as they can simply target hosts on the premade hit list.

Assuming that stable responders represent entries on an IGMP hit list, potential attackers would likely be interested in "guaranteed" amplification from each host. As such, we focus our attention on the minimum amplification factor we observe for each stable responder across our original scan and three re-probes. The dotted line in Figure 6.3 shows the distribution of the minimum amplification factor we observe for each stable responder. About 16% of responders yield no amplification in at least one re-probe. We find a median amplification factor of 2.4 (as we did in the overal scan in § 6.2.1) and that 1.5% of stable responders offer an amplification factor of at least 50.

We also examine the packet amplification for stable responders and find that packet amplification is not as great as byte amplification. Just under 88% of stable responders transmit their response in a single packet and 0.5% of responsers would give a potential attacker a packet amplification factor of 10. In general terms, launching attacks based on packet amplification would be limited to a small set of stable responders even if an attacker obtained a hit list of targets to use during an attack.

One final characteristic of stable responders we consider is how much their response sizes change over time. We examine this by calculating the minimum and maximum byte amplification a stable responder with IP address I responds with, min_I and max_I, respectively. We then calculate the ratio $max_I : min_I$ and plot the distribution of the resulting values. The dotted line in Figure 6.4 shows that over 84% of stable responders send the

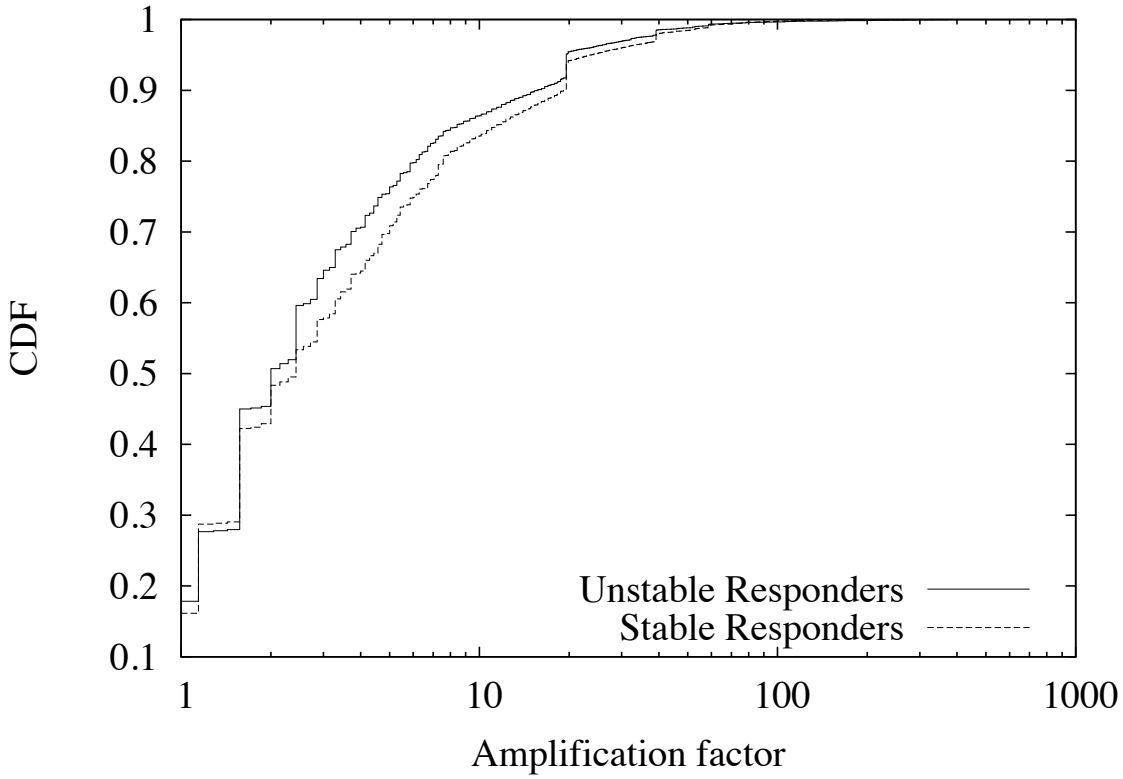

Figure 6.3: Distribution of byte amplification factors for stable responders.

same size response over a 30-day period. Around 7% of stable responders have at least two responses differing by at least a factor of two, and under 1% of stable responders have responses differing by a factor of 10 or more. This suggests that while an attacker would have to take possible fluctuations into account, they would still be able to expect reasonably consistent amplification from a hit list of stable responders.

6.3.3 Unstable Responders

Finally, we turn our attention to hosts that respond during the original scan, but later are missing a response from at least one of the re-probes. We call these hosts *unstable responders*. We seek to understand how unstable responders differ from their stable counterparts. Obviously, unstable responders do not respond as consistently as stable responders, but a

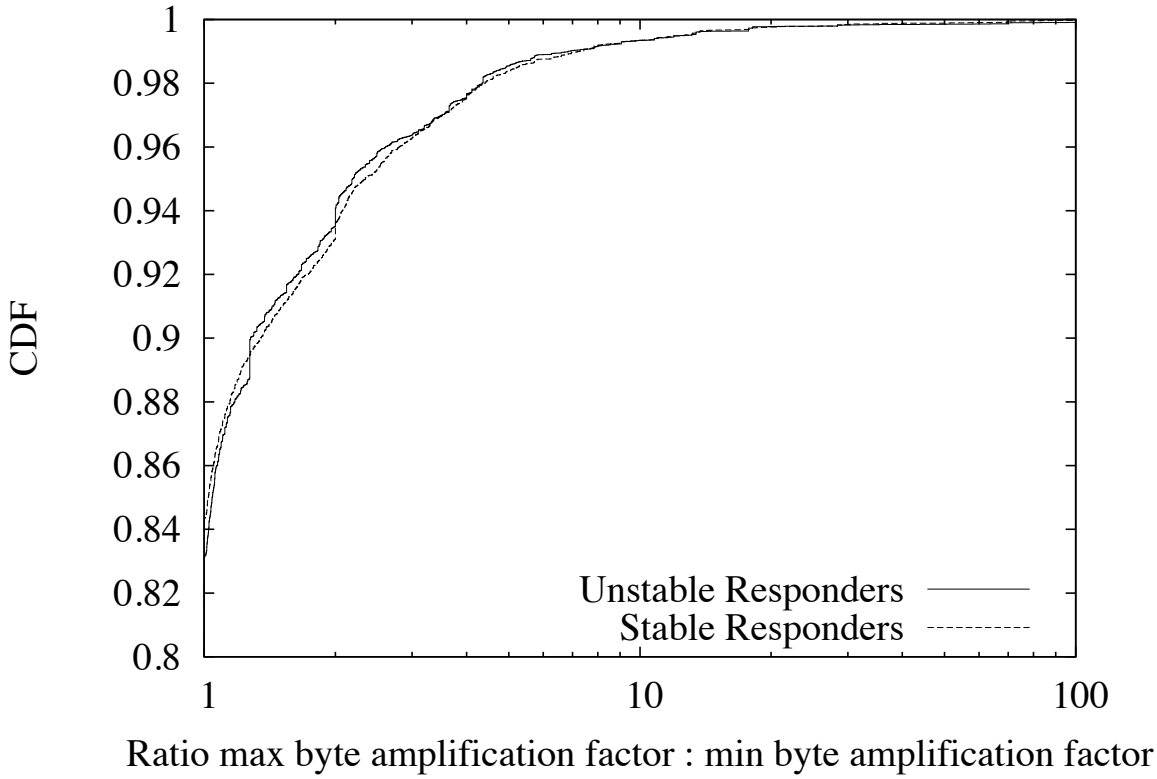

Figure 6.4: Distribution of ratios of $max_I : min_I$ for each IP address I.

natural question to consider is whether unstable responders also differ from stable responders in terms of byte and packet amplification.

Refer again to Figure 6.3. The solid line on the plot shows the byte amplification for unstable responders. Unstable responders exhibit no amplification 18% of the time. The median amplification factor for unstable responders is 2 and 1.2% of unstable responders have a byte amplification of at least 50. While there is slighlty less byte amplification for unstable responders when compared to stable responders, the distributions differ by less than an order of magnitude.

We next consider packet amplification for unstable responders. As we find with stable responders, unstable responders do not exhibit packet amplification as great as the original scan. Almost 96% of unstable responders send a single packet when answering an *AskNeighbors2* query and under 0.01% of unstable responders send at least 10 packets.

Finally, we examine how much byte amplification for unstable responders changes

over time for the set of unstable responders that respond during the original probe and at least one re-probing round. The solid line in Figure 6.4 shows the distribution for unstable responders is similar to the stable distribution plotted, with only slight variation.

6.4 Anomalous Responses

During both our original scan and subsequent re-probes we observe single hosts respond with a stream of *Neighbors2* packets over a period of time lasting up to an hour. Each packet in these streams arrives at our monitor within 1 second of the previous packet, and the streams last for an unpredictable amount of time. The packets in the stream contain no routing information, but they are valid *Neighbors2* responses. Individually each packet would yield no byte or packet amplification, but together they represent amplification factors that can grow to be millions.

More curious, we cannot always replicate this behavior at will and therefore this behavior remains puzzling. However, we make several comments about these responses:

- Anecdotal evidence [Kri14] exists that documents sustained streams of packets in response to an *AskNeighbors2* request.[5] While evidence suggests that some routers will send hundreds of thousands of packets or more, probing these routers manually does not yield streams of packets at the time of this writing.

- We observe a host that responds with byte amplification factors of 817K, 1.3M, and 120K for our original probe, re-probe 1, and re-probe 2, respectively. The large response disappears in the third re-probe and manual probes sent to this host's IP address yield single packet responses at the time of this writing. While we cannot

[5]The observations in [Kri14] were made by John Kristoff, a collaborator of ours in this current effort to understand *Neighbors2* responses. While his observations are a key part of this project, they were made while sending *AskNeighbors2* requests independently of the probing efforts described in this paper.

trigger this response currently, the host did display this anomalous behavior over a period of time.

- Through manual testing of anomalous responders, we have been able to identify a host that responds with a stream of packets when we send it a single probe. To better understand this behavior, we began sending this host 1 probe per hour over the course of 1 week (168 probes total). We observe streams of responses for each of the first 135 probes we send to the host. The responses vary in time from 20 seconds to 101 seconds long with a median value of 68 seconds. We receive a minimum of 274 KB in response to a single 28 byte probe and a median of 696 KB. This corresponds to byte amplification factors of 9.8K in the minimum case and 24.8K in the median. The maximum amplification factor this host yields is 40.4K, which corresponds to 1.13 MB of data. For the final 33 probes we sent to the host we observe no response packets and the host remains unresponsive at the time of this writing. This same host has byte amplification factors of 6K, 20K, and 18K during the original scan, re-probe 1, and re-probe 3, respectively. It was unresponsive during re-probe 2.

The observations above leave us perplexed when it comes to understanding these large responses. We have evidence of large amplification happening in response to single packets across time coming from various hosts. Observing hosts exhibit this behavior across time leads us to believe that the responses are not caused by some sort of measurement artifact. Likewise, being able to trigger a stream of responses with a single packet adds confidence that previously observed responses were also triggered by single packets and not measurement errors. On the other hand, we do not currently understand why a host "fixes" itself and stops sending streams of packets in response to a probe, although likely candidates may be patching the router or changing its configuration to prevent sending to arbitrary hosts. Another possibility is that a router only exhibits anomalous behavior when it is in a specific, but rare, state and that a bug in a router's software will occasionally be

triggered when receiving an *AskNeighbors2* request in this state. We observe 16 hosts that respond with a minimum of 500 packets and a maximum of 8.1M packets across our scans.

In addition to anomalous behavior that appears to be largely unpredictable, we also observe a set of 12 hosts belonging to a single /16 located at the University of Texas at San Antonio that send large, predictable streams of packets. In response to a single packet, the hosts will respond with 288 identical 464-byte packets. These hosts behave identically in each of our three re-probes and when sending probes manually after the original set of scans. These responders each have a byte amplification of 4.7K, representing the largest, predictable byte amplification factor we observe.

6.5 Responder Locality

Next, we examine how widespread the hosts are that respond to our probes. We once again turn our attention to the list of 305K responders from our original scan. Their IP addresses are located on 188, 9.5K, and 99K unique /8, /16, and /24 netblocks, respectively. If we instead focus on the 161K stable responders across our original scan and re-probes, we observe IP addresses from 185, 6.5K, and 62K unique /8, /16 and /24 netblocks, respectively. This translates into having at least one stable responder on almost 10% of all /16 netblocks and 72% of all /8 netblocks. This shows the breadth of the problem and that fixing the issue is not likely a quick fix for only a few network administrators.

6.6 Attacks

Up until this point our focus has largely been on understanding *Neighbors2* responses themselves. While we have been examining *Neighbors2* response characteristics with security implications in mind, we will now explicitly consider potential security risks by assessing the broad vulnerabllity to IGMP-based attacks.

6.6.1 Sustained Denial of Service Attack

Consider the situation where an attacker controls a moderately-sized botnet containing 2K bots and has compiled the list of 48K stable *Neighbors2* responders that offer a minimum of 5 times byte amplification (i.e., the most vulnerable of the stable responders). Assume that the goal of the attacker is to exhaust a victim's 10 Gbps line via an attack that leverages both the amplification and reflection provided by *AskNeighbors2* requests and *Neighbors2* responses. Recall that reflection allows an attacker to order bots to send *AskNeighbors2* requests to stable responders while spoofing the source IP address of the requests as the victim's IP address. Stable responders will send the *Neighbors2* responses to the victim, thus shielding the bots from being directly identifiable by the victim. As the responses will be at least 5 times as large as the requests being sent by the bots, the attacker requires its bots to possess only a fraction of the bandwidth that they wish to exhaust at the victim. Consider an attack using the following strategy:

First, assign each target *Neighbors2* responder from the attacker's hit list to one of the 2K bots we leverage in this attack so that each bot has a list of targets approximately the same in length. In this case, each bot is assigned a list of about 24 IP addresses to target.[6] Next, suppose that each bot sends at a rate of 1,272 packets per second and sends to each IP address using a round robin strategy. Each target host will receive 53 packets per second from the infected bots.

Using this strategy we can now calculate how much bandwidth would be exhausted at our victim. Hitting every host with an amplification factor of at least 5 with a single probe would yield 24 MB of data being sent to our victim while requiring our bots to send only 4.6 MB of data. Scaling this up by 53 responses per second yields 1.27 GB of data per

[6]This particular attack formation is for ease of exposition and additional configurations are possible (e.g. all bots could randomly probe all 48K targets). Our goal is to illustrate what is possible and not to optimize our attack in a particular fashion.

second, or 10.2 Gbps. In total, the bots send at a rate of just under 570 Mbps, or an average of 0.285 Mbps each. This strategy provides an overall amplification factor of 17.8.

6.6.2 Pulse Attack

The denial of service attack in § 6.6.1 is meant to be a sustained attack. However, an attacker could also launch a pulse attack leveraging the same list of *Neighbors2* responders. Whereas the goal of the sustained attack is to exhaust all bandwidth at a victim for a period of time, pulse attacks attempt to disrupt congestion control for ongoing connections at the victim [TH04]. An attacker accomplishes this by alternating periods of sending and not sending packets to a victim. During each sending period, an attacker sends a large burst, or pulse, of packets to the victim within a sort time period. A pulse of packets arriving at a victim temporarily congests its network. Following a pulse of packets, an attacker pauses for a brief moment before sending the next pulse of packets. Pulses and pauses are alternated in this way over a period of time.

Pulse attacks aim to disrupt congestion control for ongoing connections at a victim's network. Each pulse of data temporarily congests the victim's network and will cause ongoing connections at the network to reduce their sending rate as they detect the congestion through loss. Once the pulse is over, connections will slowly ramp back up to their fair share of bandwidth, but soon the next pulse will arrive and congest the network. Connections will once again reduce their sending rate and then attempt to recover during the next pause, only to be disrupted again by the following pulse. Through pulses an attacker can leverage congestion control to effectively hold down the victim's bandwidth utilization without sustaining the effort to use that capacity constantly.

Again consider the 48K stable responders that have an amplification factor of at least 5 and assume that an attacker is in control of a botnet with 2K bots. If the attacker were to have its bots send a single packet to each stable responder at the same time while spoofing the victim's IP address, it would generate 24 MB worth of responses at the victim.

Recall from § 6.2.1 that most responses arrive within 1 second of a query being sent. Thus, the victim will receive at least 192 Mbps worth of traffic arrive during each pulse.[7] If the attacker sends a pulse every few seconds, ongoing connections on the victim's network will be rate limited by their congestion control due to constantly fluctuating network conditions.

6.6.3 Infinite Loop Attack

The expected response, if any, to an *AskNeighbors2* request is a *Neighbors2* packet. However, we observe 79 hosts that instead respond to an *AskNeighbors2* request by sending back the same request they received. Such behavior enables an attack on each router exhibiting this behavior. The attacker identifies two misbehaving routers R_1 and R_2 that will respond to an *AskNeighbors2* packet with the same *AskNeighbors2* packet. The attacker sends an *AskNeighbors2* packet to R_1 and spoofs the source IP of the packet as R_2. When R_1 receives the packet with the spoofed address of R_2, it responds by sending an *AskNeighbors2* packet to R_2. Likewise, when R_2 receives the packet from R_1, it will respond to R_1 with an *AskNeighbors2* packet. The routers will continue to circulate the *AskNeighbors2* request back and forth to each other in an infinite loop. Given the connectionless nautre of this interaction, a packet that is dropped while being transmitted between the routers will stop the infinitely circulating behavior. However, an attacker can simply introduce additional packets into the infinite loop to combat any packet loss that would occur. In addition to combatting the effects of possible packet loss, each additional packet introduced into the infinite loop also increases the severity of the attack. A crafty attacker could start thousands of multi-packet infinite loops by sending packets to each misbehaving router addressed from each of the other misbehaving routers.

[7]The victim receiving 192 Mbps worth of traffic assumes packets are spread evenly across a 1-second period of time. Packets will likely arrive at the victim faster than one second. Thus, the actual spike of traffic will be greater than 192 Mbps. A spike of at least 192 Mbps worth of traffic will be disruptive to a victim regardless and illustrates the severity of a pulse attack even though the actual spike would be more severe.

6.7 Future Work

In this chapter, we have begun to understand *Neighbors2* responses and behaviors associated with the hosts that send the responses. After studying response characteristics, we establish that these hosts represent a security risk by enabling to distinct types of attacks. However, there are still additional aspects of *Neighbors2* responses and responding hosts we wish to understand in future work:

Response Rate Limiting: In our sustained attack, we assume that each router will respond to 53 requests per second. While we do have anecdotal evidence suggesting that routers will not rate limit their responses, we would like to organize a broad rate-limit test for all reponders. Such a test requires extra care to organize in order to not attack the routers we are measuring.

Consistency Among Interfaces: If we understand when multiple IP addresses correspond to a single router, we can analyze whether multiple router interfaces behave consistently in terms of amplification or anomalous behavior. As we currently overestimate the number of responding hosts, understanding when multiple IP addressess are used by a single router will give us a more accurate understanding.

Scanning across time: The temporal analysis in this chapter allows us to understand response behavior across a 30-day period. However, one could argue that we are missing information by waiting 10 days between our probes to each host. Future scans could be organized to happen more rapidly to better understand how router behavior and response characteristics change at a finer granularity.

Anomalous Responses: Large, anomalous responses are still mostly a mystery. Future work will aim to untangle what causes large responses to manifest on the network.

Each of these analyses would refine our understanding of *Neighbors2* responses presented in this chapter.

6.8 Conclusion

In this chapter we describe a scan of the Internet for hosts that will openly respond to *AskNeighbors2* requests over IGMP. While these types of requests have been leveraged to study network topology, to the best of our knowledge we are the first to examine response characteristics as they relate to network security. We study byte and packet amplification for responders over time and find 161K hosts that respond consistenly over a 30-day period. We find byte amplification is 2.4 times in the median case, but also show that leveraging the top 48K hosts in terms of byte amplification provide an attacker enough amplification to launch two separate types of attacks. Additionally, we describe how an attacker can launch a third attack by creating infinite streams of packets between a small set of misbehaving routers that respond to *AskNeighbors2* packets with *AskNeighbors2* packets rather than *Neighbors2* packets.

Chapter 7

Conclusion

In this dissertation we leverage empirical measurements to keep our understanding of network characteristics up-to-date. Each chapter presents a disctinct project with its own set of insights, and we refer the reader to the end of an individual chapter for a summary of its contributions. In addition to presenting each chapter as a distinct piece of work, we make several broad observations based on combining insight from multiple projects.

Applications: When studying application sending patterns in the Case Connection Zone we find that a non-trivial number of applications exhibit multiple distinct data transactions with pauses between them, rather than a single bulk transfer of data. However, TCP parameters are often tuned to optimize performance in steady state during bulk transfer of data. The prevalence of transactional sending patterns suggests that TCP could benefit from mechanisms to improve the performance of connections with multiple transactions. For example, TCP may wish to begin tracking an estimate of the available capacity between transactions and allow transactions to start sending at a rate closer to this capacity estimate rather than re-entering slow start at the beginning of each transaction.

CCZ users are in the unique position of having nearly infinite last mile capacity when compared to an average residential network. However, the types of applications in use on the CCZ are largely the same as applications being used by residential users with

more modest last mile bandwidth. This suggests that non-bulk transfer demand is likely pervasive. Additionally, there is room in future networks for high-speed applications to be deployed, as currently the applications use only a modest amount of bandwidth available for high-speed networks.

TCP Performance: The behavior of a TCP connection is ultimately defined by both the underlying protocol specification as well as the specifics of TCP implementations in end hosts. If TCP's specification or implementation fails to keep pace with changing network characteristics, then TCP's performance will be negatively impacted. For example, we observe undersized buffers in TCP implementations—on both the sender and receiver—limit throughput for bulk transfer connections. While the connections we study are on a high-bandwidth residential network, the maximum throughput achievable by these connections is only a small fraction of the available bandwidth. Note that performance would be limited by undersized buffers in many cases even if the last mile bandwidth were 100 Mbps, or 10% of the capacity of the last mile links we study. This highlights how one network shift—increased bandwidth—has outpaced TCP's implementation in end hosts. Likewise, we show how an ascpect of the TCP specification—the initial retransmission timeout—outgrew the current state of network delay. We take a modest step in bringing the TCP specification up-to-date by revisiting TCP's initial retransmission timeout and showing that it can safely be lowered without negatively impacting most TCP connections.

In addition to tuning TCP parameters to better reflect current network properties, we may need to evolve TCP's underlying mechanisms to better account for how applications use TCP. As we note above, additional mechanisms may take into account application sending patterns and optimize for distinct types of sending (i.e., bulk transfers versus transactional connections).

Security: A common way to attack the network is to exploit an unwitting network device by having it send traffic on an attacker's behalf. End hosts may become infected with malware, giving control of the machine to an attacker who then uses the infected machine

to send traffic during an attack. Likewise, open network services on servers or routers that respond to requests from arbitrary hosts will provide an attacker with reflection and often amplification to use during an attack. We study how one previously uninvestigated type of traffic—IGMP *AskNeighbors2* requests—can be used to launch several network attacks by leveraging routers that openly respond to requests from any end host. IGMP is an otherwise benign protocol that is used to transfer multicast routing information between routers on an operational network. Regardless of the protocol's original intent, routers that respond to requests from arbitrary hosts are widespread enough to create an attack vector that has, to the best of our knowledge, not yet been exploited. A simple way to prevent an IGMP-based attack, or other attacks leveraging open services, is to have stricter policies in place that filter out traffic from most hosts. While we anecdotally know that filtering happens on the network, we do not have a grasp of the breadth of this mechanism across traffic types. To this end, we develop a methodology that allows us to quantify the extent of port filtering in edge networks. We use our methodology to study Conficker port filtering and find that while filtering does happen in various edge networks, filtering Conficker is not done in all networks. Therefore, while it is likely beneficial to preemptively filter out certain types of traffic in order to protect network resources, network operators may take a more reactionary stance and only filter out types of traffic that become problematic on their network.

Each preceding topic contains observations that go beyond the scope of a single study we perform. When combined with the insight from each individual study, we bring our understanding of many network characteristics up-to-date, but we also identify topics where continued study is needed. These topics for future work include a practical reassessment of the TCP specification for better performance on faster networks and in the face of changes in how applications use TCP, creating high-bandwidth applications or restructuring the network in ways that take advantage of high-bandwidth residential networks, and continued study of how edge networks react to known attacks through policy.

Bibliography

[AAC+06] A. Aina, J. Akkerhuis, K. Claffy, S. Crocker, D. Karrenberg, J. Ihrn, R. Joffe, M. Kosters, A. Mankin, R. Mohan, et al. SSAC Advisory SAC008 DNS Distributed Denial of Service (DDoS) Attacks, 2006.

[Ale] Alexa: The Web Information Company. `http://www.alexa.com`.

[All13] Mark Allman. Comments on Bufferbloat. *ACM SIGCOMM Computer Communication Review*, 43(1), January 2013.

[APB09] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control, September 2009. RFC 5681.

[APT07] Mark Allman, Vern Paxson, and Jeff Terrell. A Brief History of Scanning. In *ACM SIGCOMM/USENIX Internet Measurement Conference*, October 2007.

[AW97] Martin Arlitt and Carey Williamson. Web Server Workload Characterization: The Search for Invariants (Extended Version). *IEEE/ACM Transactions on Networking*, 5(5), October 1997.

[BAW+12] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, and Y. Nishida. A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP, August 2012. RFC 6675.

[BBHc09] Robert Beverly, Arthur Berger, Young Hyun, and kc claffy. Understanding the Efficacy of Deployed Internet Source Address Validation Filtering. In *Proceedings of the ACM SIGCOMM conference on Internet Measurement*, IMC'09, 2009.

[BC98] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS*, July 1998.

[BCJ+05] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. In *Proceedings of Network and Distributed System Security Symposium*, NDSS'05, pages 167–179, 2005.

[BDcA13] K. Benson, A. Dainotti, k. claffy, and E. Aben. Gaining Insight into AS-level Outages through Analysis of Internet Background Radiation. In *Traffic Monitoring and Analysis Workshop*, TMA'13, Apr 2013.

[BHM+07] Randy Bush, James Hiebert, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Testing the Reachability of (New) Address Space. In *Proceedings of the SIGCOMM workshop on Internet Network Management*, INM'07, pages 236–241, New York, NY, USA, 2007. ACM.

[BPS99] Jon CR Bennett, Craig Partridge, and Nicholas Shectman. Packet Reordering is Not Pathological Network Behavior. *Networking, IEEE/ACM Transactions on*, 7(6):789–798, 1999.

[CAI13] CAIDA. Conficker/Conflicker/Downadup as seen from the UCSD Network Telescope. `http://www.caida.org/research/security/ms08-067/conficker.xml`, 2013.

[Cas] Case Connection Zone. `http://caseconnectionzone.org/`.

[CAZ+14] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. Measuring IPv6 Adoption. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 87–98. ACM, 2014.

[CBG10] David R. Choffnes, Fabián E. Bustamante, and Zihui Ge. Crowdsourcing Service-Level Network Event Monitoring. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM'10, 2010.

[CDK+02] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3, October 2002. RFC 3376.

[CFEK06] K. Cho, K. Fukuda, H. Esaki, and A. Kato. The Impact and Implications of the Growth in Residential User-To-User Traffic. In *ACM SIGCOMM Computer Communication Review*, volume 36.4, pages 207–218. ACM, 2006.

[Cha10] Chattanooga, Tenn Announces only 1 Gigabit Broadband Service in U.S. for Both Residential and Business Customer. `http://chattanoogagig.com/pdf/Chattanooga_GPON_EPB.pdf`, 2010.

[Che12] Yuchung Cheng. Re: [tcpm] Adopting draft-fairhurst-tcpm-newcwv. IETF TCPM Mailing List, December 2012.

[Chi09] Eric Chien. Downadup: Attempts at Smart Network Scanning. `http://www.symantec.com/connect/blogs/downadup-attempts-smart-network-scanning`, January 2009.

[Chu09] J. Chu. Tuning TCP Parameters for the 21st Century. `http://www.ietf.org/proceedings/75/slides/tcpm-1.pdf`, 2009.

[Cla88] David D. Clark. The design philosophy of the DARPA internet protocols. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM'88, pages 106–114, 1988.

[Coh08] Cohen, B. The BitTorrent Protocol Specification. `http://bittorrent.org/beps/bep_0003.html`, 2008.

[Com] Comcast. Blocked Ports List. `https://customer.comcast.com/help-and-support/internet/list-of-blocked-ports/`.

[CR13] Chiara Chirichella and Dario Rossi. To the moon and back: Are internet bufferbloat delays really that large? In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 417–422. IEEE, 2013.

[Cra] Crawdad: A Community Resource for Archiving Wireless Data at Dartmouth. `http://crawdad.cs.dartmouth.edu`.

[DCCM12] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis. TCP Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses. Internet-Draft draft-dukkipati-tcpm-tcp-loss-probe-00.txt, July 2012. Work in progress.

[Dee89] S. Deering. Host Extensions for IP Multicasting, August 1989. RFC 1112.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification, December 1998. RFC 2460.

[DHB+13] Gregory Detal, Benjamin Hesmans, Olivier Bonaventure, Yves Vanaubel, and Benoit Donnet. Revealing Middlebox Interference With Tracebox. In *Proceedings of the 2013 Internet measurement conference*, pages 1–8. ACM, 2013.

[DHGS07] Marcel Dischinger, Andreas Haeberlen, Krishna Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *ACM Internet Measurement Conference*, October 2007.

[DSA+11] Alberto Dainotti, Claudio Squarcella, Emile Aben, Kimberly C. Claffy, Marco Chiesa, Michele Russo, and Antonio Pescapé. Analysis of country-wide internet outages caused by censorship. In *ACM Internet Measurement Conference*, IMC '11, 2011.

[DWH13] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *USENIX Security*, pages 605–620. Citeseer, 2013.

[Fen97] W. Fenner. Internet Group Management Protocol, Version 2, November 1997. RFC 2236.

[FGM+99] R. Fielding, Jim Gettys, Jeffrey C. Mogul, H. Frystyk, L. Masinter, P. Leach, , and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. RFC 2616.

[FS14] F-Secure. Threat Report H1 2014. `http://www.f-secure.com/documents/996508/1030743/Threat_Report_H1_2014.pdf`, 2014.

[GN12] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark Buffers in The Internet. *Communications of the ACM*, 55(1):57–65, 2012.

[Gon12] Lev Gonick. Personal Communication, April 2012.

[gooa] Ultra high-speed broadband is coming to Kansas City, Kansas. `http://googleblog.blogspot.com/2011/03/ultra-high-speed-broadband-is-coming-to.html`.

[Goob] Bringing ultra high-speed broadband to Stanford homes. http://googleblog.blogspot.com/2010/10/bringing-ultra-high-speed-broadband-to.html.

[HFPW03] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification, January 2003. RFC 3448.

[HKA04] T. Henderson, D. Kotz, and I. Abyzov. "crawdad trace dartmouth/campus/tcpdump/fall03 (v. 2004-11-09)". http://crawdad.cs.dartmouth.edu/dartmouth/campus/tcpdump/fall03, November 2004.

[HPC+14] Oliver Hohlfeld, Enric Pujol, Florin Ciucu, Anja Feldmann, and Paul Barford. A QoE perspective on sizing network buffers. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 333–346. ACM, 2014.

[Jac88] Van Jacobson. Congestion Avoidance and Control. In *ACM SIGCOMM*, 1988.

[JBB92] V. Jacobson, R. Braden, and D. Borman. RFC1323 - TCP Extensions for High Performance, 1992.

[JLM89] Van Jacobson, Craig Leres, and S McCanne. The tcpdump Manual Page. *Lawrence Berkeley Laboratory, Berkeley, CA*, 1989.

[KkcF+08] H. Kim, k. claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In *ACM SIGCOMM CoNEXT*, December 2008.

[KPF05] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. In *ACM SIGCOMM*, 2005.

[Kri09] John Kristoff. Experiences with Conficker C Sinkhole Operation and Analysis. In *Proceedings of Australian Computer Emergency Response Team Conference*, 2009.

[Kri14] John Kristoff. DVMRP Ask Neighbors2: an IGMP-based DDoS/leak threat. `https://www.cymru.com/jtk/talks/nanog62-an2.pdf`, October 2014.

[KWNP10] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the Edge Network. In *ACM Internet Measurement Conference*, November 2010.

[M. 15] M. Duke and R. Braden and W. Eddyand and E. Blanton and A. Zimmermann. RFC7414 - A Roadmap for Transmission Control Protocol (TCP) Specification Documents, February 2015. RFC 7414.

[MDP+11] Pascal Mérindol, Benoit Donnet, J-J Pansiot, Matthew Luckie, and Young Hyun. MERLIN: MEasure the router level of the INternet. In *Next Generation Internet (NGI), 2011 7th EURO-NGI Conference on*, pages 1–8. IEEE, 2011.

[MFPA09] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On Dominant Characteristics of Residential Broadband Internet Traffic. In *ACM Internet Measurement Conference*, November 2009.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options, October 1996. RFC 2018.

[Moc87] Paul Mockapetris. Domain Names - Implementation and Specification, November 1987. RFC 1035.

[MSF11] G. Maier, F. Schneider, and A. Feldmann. NAT Usage in Residential Broadband Networks. In *Passive and Active Measurement*, pages 32–41. Springer, 2011.

[MSMO97] Matt Mathis, Jeff Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, 27, July 1997.

[MVdSD+09] Pascal Mérindol, Virginie Van den Schrieck, Benoit Donnet, Olivier Bonaventure, and Jean-Jacques Pansiot. Quantifying ASes multiconnectivity using multicast information. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 370–376. ACM, 2009.

[Nat] The National Broadband Plan. `http://www.broadband.gov/plan/`.

[Naz08] Jose Nazario. DDoS Attack Evolution. *Network Security*, 2008(7):7–10, 2008.

[NGBS+97] Hanrik Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Lie, and Chris Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *ACM SIGCOMM*, September 1997.

[PA00] V. Paxson and M. Allman. Computing TCP's Retransmission Timer, November 2000. RFC 2988.

[PACS11] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer, June 2011. RFC 6298.

[Pax94] Vern Paxson. Empirically-Derived Analytic Models of Wide-Area TCP Connections. *IEEE/ACM Transactions on Networking*, 2(4), August 1994.

[Pax97] Vern Paxson. Automated Packet Trace Analysis of TCP Implementations. In *ACM SIGCOMM*, September 1997.

[Pax99] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, December 1999.

[PF95] Vern Paxson and Sally Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking (ToN)*, 3(3):226–244, 1995.

[PF01] Vern Paxson and Sally Floyd. Difficulties in Simulating the Internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, 2001.

[PFTK98] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *ACM SIGCOMM*, September 1998.

[Pos81a] J. Postel. Internet Protocol, September 1981. RFC 791.

[Pos81b] J. Postel. Transmission Control Protocol, September 1981. RFC 793.

[PSY09] P. Porras, H. Saidi, and V. Yegneswaran. An Analysis of Conficker's Logic and Rendezvous Points. Technical report, SRI International, March 2009.

[PYB+04] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson. Characteristics of Internet Background Radiation. In *Proceedings of the ACM SIGCOMM conference on Internet Measurement*, IMC'04, 2004.

[RL09] Matt Richard and Michael Ligh. Making Fun of Your Malware. Defcon 17, 2009.

[Ros14] Christian Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[SA14] Matt Sargent and Mark Allman. Performance Within A Fiber-To-The-Home Network. *ACM Computer Communication Review*, 44(3), July 2014.

[SBA14] Matt Sargent, Ethan Blanton, and Mark Allman. Modern Application Layer Transmission Patterns from a Transport Perspective. In *Passive and Active Measurement Conference*, March 2014.

[SCAB15] Matt Sargent, Jakub Czyz, Mark Allman, and Michael Bailey. On The Power and Limitations of Detecting Network Filtering via Passive Observation. In *Passive and Active Measurement Conference*, March 2015.

[Sch02] Paul Schmehl. The Microsoft UPnP (Universal Plug and Play) Vulnerability. `http://bandwidthco.com/sf_whitepapers/windows/The%20Microsoft%20UPnP%20(Universal%20Plug%20and%20Play)%20Vulnerability.pdf`, 2002.

[SCRA13] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. On Measuring the Client-Side DNS Infrastructure. In *ACM SIGCOMM/USENIX Internet Measurement Conference*, October 2013.

[SdDF+11] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband Internet Performance: A View From The Gateway. *SIGCOMM-Computer Communication Review*, 41(4):134, 2011.

[SH99] P. Srisuresh and M. Holdrege. RFC2663 - IP Network Address Translator (NAT) Terminology and Considerations, 1999.

[SLS09] A. Schulman, D. Levin, and N. Spring. "crawdad data set umd/sigcomm2008 (v. 2009-03-02)". http://crawdad.cs.dartmouth.edu/umd/sigcomm2008, March 2009.

[SMM98] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. In *ACM SIGCOMM Computer Communication Review*, volume 28.4, pages 315–323. ACM, 1998.

[Spa] The Spamhaus Project - PBL. http://www.spamhaus.org/pbl/.

[SSDA12] Matt Sargent, Brian Stack, Tom Dooner, and Mark Allman. A First Look at 1 Gbps Fiber-To-The-Home Traffic. Technical Report 12-009, International Computer Science Institute, August 2012.

[Sys14] C. Systems. Cisco Event Response: Network Time Protocol Amplification Distributed Denial of Service Attacks. http://www.cisco.com/web/about/security/intelligence/ERP-NTP-DDoS.html, February 2014.

[Tec13] Prolexic Technologies. An Analysis of DrDos SNMP/NTP/CHARGEN Reflection Attacks: Part II of the DrDos White Paper Series. http://www.prolexic.com/kcresources/white-paper/white-paper-snmp-ntp-chargen-reflection-attacks-drdos/An_Analysis_of_DrDoS_SNMP-NTP-CHARGEN_Reflection_Attacks_White_Paper_A4_042913.pdf, 2013.

[TH04] Hellinton H Takada and Ulrich Hofmann. Application and Analyses of Cumulative Sum to Detect Highly Distributed Denial of Service Attacks Using Different Attack Traffic Patterns. http://www.ist-intermon.org/dissemination/newsletter7.pdf, 2004.

[Uni] University of Oregon. Route Views Project. http://www.routeviews.org/.

[VH97] Vikram Visweswaraiah and John Heidemann. Improving Restart of Idle TCP Connections. Technical Report 97-661, University of Southern California, November 1997.

[WKB+10] Eric Wustrow, Manish Karir, Michael Bailey, Farnam Jahanian, and Geoff Houston. Internet Background Radiation Revisited. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*, IMC'10, 2010.

[WPD88] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol, November 1988. RFC 1075.

[XYLL12] Yang Xu, Chenguang Yu, Jingjiang Li, and Yong Liu. Video Telephony for End-consumers: Measurement Study of Google+, iChat, and Skype. In *ACM Internet Measurement Conference*, October 2012.

[Zal06] Zalewski, M. p0f: Passive OS Fingerprinting tool. http://lcamtuf.coredump.cx/p0f.shtml, 2006.

[Zim80] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *Communications, IEEE Transactions on*, 28(4):425 – 432, April 1980.

[ZMa] Zmap. https://zmap.io/.