Measuring Internet Traffic Manipulation

A Thesis presented

by

Anke Li

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Master of Science

in

Computer Science

Stony Brook University

December 2016

Stony Brook University

The Graduate School

Anke Li

We, the thesis committee for the above candidate for the

Master of Science degree, hereby recommend

acceptance of this thesis

Phillipa Gill - Thesis Advisor
Research Assistant Professor, Computer Science Department

Michalis Polychronakis - Thesis Committee Member
Assistant Professor, Computer Science Department

Samir R. Das - Thesis Committee Member
Professor, Computer Science Department

This thesis is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Thesis

Measuring Internet Traffic Manipulation

by

Anke Li

Master of Science

in

Computer Science

Stony Brook University

2016

There has been a surge of interest in measuring various aspects of Internet traffic manipulation. This report focuses on measuring two types of traffic manipulation:

- Traffic differentiation in mobile networks.
- Online information control.

Traffic differentiation—giving better or worse performance to certain classes of Internet traffic—is a well-known but poorly understood traffic management policy. There is active discussion on whether and how ISPs should be allowed to differentiate Internet traffic [1, 2], but little data about current practices to inform this discussion. Previous work attempted to address this problem for fixed line networks, but at the time of the publication of our work, there was no solution that works in the more challenging mobile environment. In terms of censorship measurement, despite the high perceived value and increasing severity of online information controls, a data-driven understanding of the phenomenon has remained elusive.

In this report, we first present the design, implementation and evaluation of Differentiation Detector, the first system and mobile app for identifying traffic differentiation for arbitrary applications in the mobile environment.

Next we introduce Information Controls Lab (ICLab)—a project focused on collecting and analyzing reliable information controls data on the Internet at scale—by comparing it with another design point in the space of Internet censorship measurement with particular emphasis on how they address the challenges of locating vantage points, choosing content to test, and analyzing results. We discuss the trade offs of decisions made by each platform and show how the resulting data provides complementary views of global censorship, as well as the lessons learned and open challenges discovered through our experiences.

Table of Contents

List of Figures

List of Tables

Chapter 1

Differentiation Detector

In this chapter, we present the design, implementation, and evaluation of the first system and mobile app for identifying traffic differentiation for arbitrary applications in the mobile environment (*i.e.,* wireless networks such as cellular and WiFi, used by smartphones and tablets). The key idea is to use a VPN proxy to record and replay the network traffic generated by arbitrary applications, and compare it with the network behavior when replaying this traffic outside of an encrypted tunnel. We perform the first known testbed experiments with actual commercial shaping devices to validate our system design and demonstrate how it outperforms previous work for detecting differentiation. We released our app and collected differentiation results from 12 ISPs in 5 countries. We find that differentiation tends to affect TCP traffic (reducing rates by up to 60%) and that interference from middleboxes (including video-transcoding devices) is pervasive. By exposing such behavior, we hope to improve transparency for users and help inform future policies.

1.1 Introduction

The rise in popularity of bandwidth-hungry applications (*e.g.,* Netflix) coupled with resource-constrained networks (*e.g.,* mobile providers) has reignited discussions about how the different applications' network traffic is treated (or mistreated) by ISPs. One commonly discussed approach to managing scarce network resources is *traffic differentiation*—giving better (or worse) performance to certain classes of network traffic—using devices that selectively act on network traffic (*e.g.,* Sandvine [3], Cisco, and others). Differentiation can enforce policies ranging from protecting the network from bandwidth-hungry applications, to limiting services that compete with those offered by the network provider (*e.g.,* providers that sell voice/video services in addition to IP connectivity).

Recent events have raised the profile of traffic differentiation and network neutrality issues in the wired setting, with Comcast and Netflix engaged in public disputes over congestion on network interconnects [4--6]. The situation is more extreme in the mobile setting, where historically regulatory agencies imposed few restrictions on how a mobile provider manages its network, with subscribers and regulators generally having a limited (or no) ability to understand and hold providers accountable for management polices.[1]

Despite the importance of this issue, the impacts of traffic differentiation on specific applications---and the Internet as a whole---are poorly understood. Previous efforts attempt to detect traffic differentiation (*e.g.,* [8--11]), but are limited to specific types of application traffic (*e.g.,* peer-to-peer traffic [8]), or focus on shaping behaviors on general classes of traffic as opposed to specific applications [10--12]. Addressing these limitations is challenging, due to the wide variety of applications (often closed-source) that users may wish to test for differentiation, coupled with the closed nature of traffic shapers. As a result, external observers struggle to detect if such shapers exist in networks, what exactly these devices do, and what is their impact on traffic. What little we do know is concerning. Glasnost [8] and others [9, 10, 13] found ISPs throttling specific types of Internet traffic, Cogent admitted to differentiating Netflix traffic [14], and recent work by Google and T-Mobile [15] indicated that unintentional interactions between traffic shaping devices has been known to degrade performance.

This chapter tackles three key challenges that have held back prior work on measuring traffic differentiation: (1) the inability to test arbitrary classes of applications, (2) a lack of understanding of how traffic shaping devices work in practice, and (3) the limited ability to measure mobile networks (*e.g.,* cellular and WiFi) from end-user devices such as smartphones and tablets. Designing methods that can measure traffic differentiation in mobile networks (in addition to well-studied wired networks) presents unique challenges, as the approaches must work with highly variable underlying network performance and within the constraints of mobile operating systems. By addressing these challenges, we can provide tools that empower average users to identify differentiation in mobile networks, and use data gathered from these tools to understand differentiation in practice.

This work makes the following key contributions:

- **The design and implementation of a system for detecting traffic differentiation in mobile networks (Sec. 1.3).** We design an application-layer trace record-and-replay system that enables testing of arbitrary network applications, and works even if these applications are closed source. Our solution does not require special privileges on client devices, making it readily deployable on a global scale. The key idea is to use a VPN proxy to record arbitrary application

[1] As we discuss throughout this chapter, FCC rules that took effect in June 2015 [7] now prohibit many forms of traffic differentiation in the US.

traffic, and later replay that traffic both with and without the VPN to identify differentiation. We implement this both as an Android and desktop application.

- **Validation of our approach using commercial shaping devices (Sec. 1.4).** To the best of our knowledge, we are the first study to validate that our detection system correctly triggers differentiation on commercial shaping devices. Using a testbed consisting of two such devices, we establish a ground truth to evaluate the accuracy of our approach and identify several pitfalls of previous approaches.
- **Evaluating statistical techniques for identifying differentiation (Sec. 1.5).** Previous work uses a variety of statistical techniques to determine when differentiation occurs. We use our testbed with commercial shaping products (with configurable shaping rates), a controlled network with no differentiation, and controlled loss with Linux Traffic Control (`tc`) to systematically evaluate the accuracy of different statistical techniques for identifying differentiation.
- **An Android app that conducts our differentiation tests and a measurement study of differentiation in mobile networks (Sec. 1.6).** We develop an Android app that conducts our differentiation tests on unmodified Android OSes, as part of an IRB-approved study of differentiation in mobile networks worldwide. We deploy our software in the US and four other countries, and find several instances of shaping, in addition to pervasive interference from middleboxes.

  In addition to being the first approach to reliably detect differentiation from mobile devices without special privileges, our study also identifies several instances of middleboxes violating end-to-end principles for a variety of popular apps. In particular, we find evidence of video transcoding, HTTP header manipulation in requests and responses, and proxies that modify TCP behavior. While previous work identified such behavior for Web traffic, we are the first to do so for arbitrary app traffic.

The following section discusses related work and defines differentiation that we detect. After detailing our key contributions listed above, we discuss issues with differentiation and its detection that are beyond the scope of this work (Sec. 1.7), and we conclude in Sec. 1.8.

1.2 Background

There is an ongoing debate about whether the network should be neutral with respect to the packets it carries [16]; *i.e.,* not discriminate against or otherwise alter traffic except for lawful purposes (*e.g.,* blocking illegal content). Proponents such as President Obama argue for an open Internet [1] as essential to its continued success, while some ISPs argue for shaping to manage network resources [8, 13].

The regulatory framework surrounding traffic differentiation is rapidly changing; between submission and publication of this manuscript, the FCC transitioned from imposing few restrictions on how mobile providers manage their network, to new rules that prohibit many forms of differentiation [7]. In this environment, it is important to monitor traffic differentiation in practice, both to inform regulators and enforce policies.

In the remainder of this section, we overview related work on measuring traffic differentiation and network neutrality and then define the types of traffic differentiation our system aims to detect.

1.2.1 Related Work

Comcast's blocking of BitTorrent [17] in the mid 2000s spurred research efforts to study traffic differentiation. This led to several studies that focus on applications or flows known to be subject to classification, and design tests to detect differentiation on those flows. For example, Glasnost focuses on BitTorrent [8] and NetPolice [9] tests five applications (HTTP, BitTorrent, SMTP, PPLive and VoIP). Bonafide [18] takes a similar approach to Glasnost in the mobile environment, including tests for HTTP, FlashVideo (YouTube), SIP, RTSP, BitTorrent, and VoIP H323. These approaches focus on *protocols* rather than *specific application implementations*. A key limitation is that the shaping devices we observed support application/provider granularity, *e.g.,* shape RTSP traffic for one content provider, but not other RTSP applications (see Sec. 1.4). Such application-based differentiation may not be detected by these systems. They also require manually implementing protocol details and thus have limited extensibility. We address these limitations in this work.

Furthermore, these projects treat traffic differentiation as a ''black box'' which they aim to detect by sending traffic and observing network performance metrics. Notably, these prior studies lack ground-truth information about how differentiation is implemented, and whether or not their detection mechanisms would trigger traffic shaping in practice (an issue we address in this work).

Another approach avoids testing specific applications and focuses on detecting differentiation for arbitrary traffic. NANO [10] uses Bayesian analysis to detect differentiation, while Zhang et al. [11] focus on identifying when it is feasible to detect and isolate differentiation. Other related projects focus on performance issues in ISPs [5, 19--21] and shaping of all of a subscriber's traffic [12], but do not detect differentiation itself.

Other related studies detect proxies on network paths, and conduct tests to identify a wide range of application-specific proxy behaviors [22--26]. A recent paper [27] specifically focuses on proxies and other middleboxes in mobile networks, and discusses how these devices vary according to mobile virtual network operator (MVNO). Zarinni et al. use Bonafide [18] to test for differentiation of BitTorrent, VoIP-H323 and RTSP [28] in MVNOs. The authors did not observe differentiation of this limited set of applications; in contrast, our approach identified differentiation in three of the networks they tested.

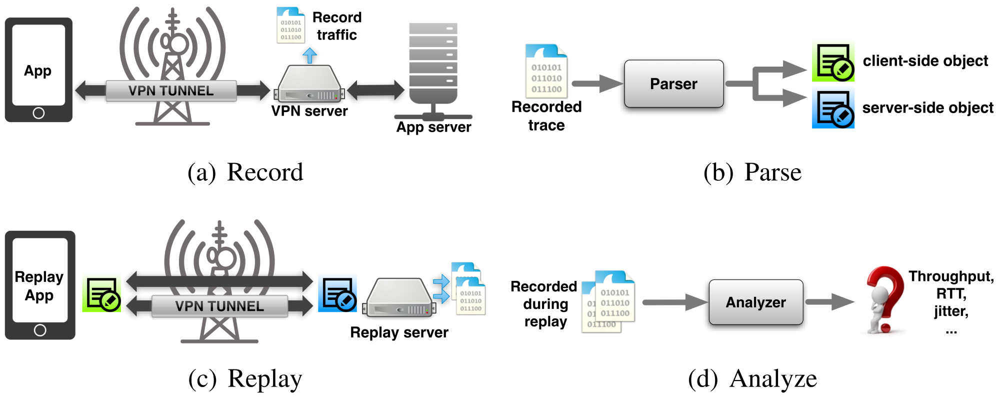
1.2.2 Differentiation considered in our work

In this work, we consider the problem of detecting differentiation caused by traffic-shaping middleboxes (rather than interconnection provisioning [4--6]). We focus on the mobile environment, in large part due to minimal regulation and scarce network resources in mobile data networks. However, our techniques also work in the fixed-line environment.

We focus on differentiation affecting network performance *as perceived by applications*. In our experiments, we observe that even high-bandwidth mobile applications such as video streaming do not necessarily exhaust the bandwidth available, possibly to avoid wasting data transfer in case of viewer abandonment. This is in contrast to protocols that previous approaches focused on (e.g., BitTorrent), which aim to saturate the link bandwidth. For applications that do not exhaust the available resources, we do *not* consider traffic shaping where the shaping rate is higher than the bandwidth demands to be differentiation. For example, if a shaper limited a given application to 200 KB/s, but the application never sent faster than 150 KB/s, we would not consider this differentiation (as the application would never be affected by it). This more conservative approach gives us a clearer picture of shaping that actually impinges on the network resource demands of applications.

We make the assumption that traffic differentiation is likely triggered by at least one of the following factors: IP addresses, port numbers, payload signatures, number of connections, total bandwidth, and time-of-day. We confirmed these assumptions are consistent with features listed in online manuals of deep packet inspection (DPI) devices and traffic shapers. We also run experiments with two commercial packet shapers and show that our assumptions hold for these devices (Sec. 1.4.2).

Nongoals. In this work, there are certain types of traffic management policies that we are not able to detect. First, we do not focus on detecting congestion at peering points, a tactic used by certain ISPs to extract payment from large bandwidth consumers [4]. Second, we do not focus on blocking (*e.g.,* censorship) or content modification (*e.g.,* transcoding) by middleboxes. While our methodology is able detect these types (and we found cases of both), they are orthogonal to the issue of shaping. Third, we currently do not support detecting differentiation based on the destination IP address contacted by clients. Our analysis of commercially deployed traffic shapers indicates that IP



(a) Record (b) Parse

(c) Replay (d) Analyze

Figure 1.1: System overview: (a) Client connects to VPN, that records traffic between the client and app server(s). (b) Parser produces transcripts for replaying. (c) Client and replay server use transcripts to replay the trace, once in plaintext and once through a VPN tunnel. Replay server records packet traces for each replay. (d) Analyzer uses the traces to detect differentiation.

addresses are not commonly used to identify applications, since none of our observed configurations use them for shaping. We speculate this is due to the fact that many applications contact servers with IPs shared by many services (*e.g.,* EC2, or Google frontends that serve both search and YouTube traffic), and the IPs used by any service may change over time. Thus, IP addresses are a poor classifier for shaping a specific application. Regardless, we are investigating how to use limited forms of IP spoofing to address this limitation.

1.3 Methodology

We use a trace record-replay methodology to reliably detect traffic differentiation for arbitrary applications in the mobile environment. We provide an overview in Figure 1.1. We record a packet trace from a target application (Fig. 1.1(a)), extract bidirectional application-layer payloads, and use this to generate a transcript of messages for the client and server to replay (Fig. 1.1(b)). To test for differentiation, the client and server coordinate to replay these transcripts, both in plaintext and through an encrypted channel (using a VPN tunnel) where payloads are hidden from any shapers on the path (Fig. 1.1(c)). Finally, we use validated statistical tests to identify whether the application traffic being tested was subject to differentiation (Fig. 1.1(d)). We discuss each of these steps below.

1.3.1 Recording the trace

Our method requires a packet trace to replay. While it is straightforward to gather one on desktop operating systems, recording packet traces on today's mobile operating systems requires ‘‘rooting’’ and/or ‘‘jailbreaking’’ the phone (potentially voiding the phone’s warranty). This is not a problem for small-scale testbed experiments, but traffic differentiation---and the applications it affects---is a moving target that requires running experiments for a variety of applications on a large set of devices in different networks worldwide. Thus, we need a way to enable end users to capture traces of mobile application traffic without requiring modifications to the OS.

To address this challenge, we leverage the fact that all major mobile operating systems natively support VPN connections, and use the Meddle VPN [29] to facilitate trace recording. Figure 1.1(a) illustrates how a client connects to the Meddle VPN, which relays traffic between the client and destination server(s). In addition, the Meddle VPN collects packet traces that we use to generate replay transcripts. When possible, we record traffic using the same network being tested for differentiation because applications may behave differently based on the network type, *e.g.,* a streaming video app may select a higher bitrate on WiFi.

Our methodology is extensible to arbitrary applications---even closed source and proprietary ones---to ensure it works even as the landscape of popular applications, the network protocols they use, and their impact on mobile network resources changes over time. Prior work focused on specific applications (e.g., BitTorrent [8]) and manually emulated application flows. This approach, however, does not scale to large numbers of applications and may even be infeasible when the target application uses proprietary, or otherwise closed, protocols.[2]

1.3.2 Creating the replay script

After recording a packet trace, our system processes it to extract the application-layer payloads as client-to-server and server-to-client byte streams. We then create a replay transcript that captures the behavior of the application’s traffic, including port numbers, dependencies between application-layer messages (if they exist) and any timing properties of the application (*e.g.,* fixed-rate multimedia).[3] Finally, we use each endpoint’s TCP or UDP stack to replay the traffic as specified in the transcript. Figure 1.1(b) shows how we create two objects with the necessary information for the client and server to replay the traffic from the recorded trace.

Logical dependencies. For TCP traffic, we preserve application-layer dependencies using the implicit *happens-before* relationship exposed by application-layer communication in TCP. More specifically, we extract two unidirectional byte streams s_{AB} and s_{BA} for each pair of communicating hosts A and B in the recorded trace. For each sequence of bytes in s_{AB}, we identify the bytes in

[2] The Glasnost paper describes an (unevaluated) tool to support replaying of arbitrary applications but we were unsuccessful in using it, even with the help of a Glasnost coauthor.

[3] This approach is similar to Cui et al.’s [30]; however, our approach perfectly preserves application payloads (to ensure traffic is classified by shapers) and timing (to prevent false positives when detecting shaping).

s_{BA} that preceded them in the trace. When replaying, host A sends bytes in s_{AB} to host B only after it has received the preceding bytes in s_{BA} from host B. We enforce analogous constraints from B to A. For UDP traffic, we do not enforce logical dependencies because the transport layer does explicitly not impose them.

Timing dependencies. A second challenge is ensuring the replayed traffic maintains the inter-message timing features of the initial trace. For example, streaming video apps commonly download and buffer video content in chunks instead of buffering the entire video, typically to reduce data-consumption for viewer abandonment and to save energy by allowing the radio to sleep between bursts. Other content servers use packet pacing to minimize packet loss and thus improve flow completion times.

Our system preserves the timings between packets for both TCP and UDP traffic to capture such behavior when replaying (this feature can be disabled when timing is not intrinsic to the application). Preserving timing is a key feature of our approach, that can have a significant impact on detecting differentiation, as discussed in Section 1.2.2. In short, it prevents us from detecting shaping that does not impact application-perceived performance.

Specifically, for each stream of bytes (UDP or TCP) sent by a host, A, we annotate each byte with *time offset* from the first byte in the stream in the recorded trace. If the ith byte of A was sent at t ms from the start of the trace, we ensure that byte i is *not sent before* t ms have elapsed in the replay. For TCP, this constraint is enforced after enforcing the happens-before relationship.

In the case of UDP, we do not retransmit lost packets. This closely matches the behavior of real-time applications such as VoIP and live-streaming video (which often tolerate packet losses), but does not work well for more complicated protocols such as BitTorrent's µTP [31] or Google's QUIC [32]. We will investigate how to incorporate this behavior in future work.

1.3.3 Replaying the trace

After steps 1 and 2, we replay the recorded traffic on a target network to test our null hypothesis that there is no differentiation in the network. This test requires two samples of network performance: one that *exposes* the replay to differentiation (if any), and a *control* that is not exposed to differentiation.

A key challenge is how to conduct the control trial. Initially, we followed prior work [8] and randomized the port numbers and payload, while maintaining all other features of the replayed traffic. However, using our testbed containing commercial traffic shapers (Section 1.4.2), we found that some shapers will by default label ‘‘random’’ traffic with high port numbers as peer-to-peer traffic, a common target of differentiation. Thus, this approach is unreliable to generate control trials because one can reasonably expect it to be shaped.

Instead, we re-use the Meddle VPN tunnel to conduct control trials. By sending all recorded traffic over an encrypted IPSec tunnel, we preserve all application behavior while simultaneously preventing any DPI-based shaper from differentiating based on payload. Thus, each replay test consists of replaying the trace twice, once in plaintext (exposed trial), and once over the VPN (control trial), depicted in Figure 1.1(c). To detect differentiation in noisy environments, we run multiple back-to-back tests (both control and exposed). Note that we compare exposed and control trials with each other and not the original recorded trace. We explore potential issues of using a VPN as a control in Sections 1.4.3 and 1.7.

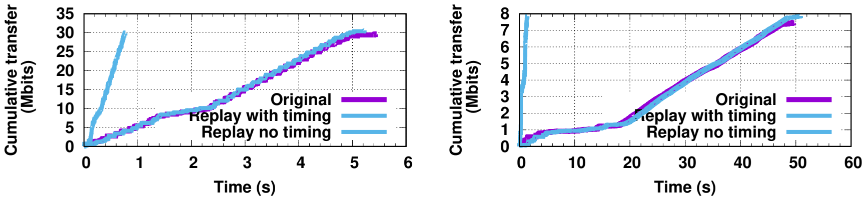
1.3.4 Detecting differentiation

After running the exposed and control trials, we compare performance metrics (throughput, loss, and delay) to detect differentiation (Fig. 1.1(d)). We focus on these metrics because traffic shaping policies typically involve bandwidth rate-limiting (reflected in throughput differences), and may have impacts on delay and loss depending on the queuing/shaping discipline. We base our analysis on server-side packet captures to compute throughput and RTT values for TCP, because we cannot rely on collecting network-level traces on mobile clients. For UDP applications, we use jitter as a delay metric and measure it at the client at the application layer (observing inter-packet timings).

A key challenge is how to automatically detect when differences between two traces are caused by differentiation, instead of signal strength, congestion, or other confounding factors. In Section 1.5, we find that previous techniques to identify differentiation are inaccurate when tested against commercial packet shaping devices with varying packet loss in our testbed. We describe a novel area test approach to compare traces that has perfect accuracy with no loss, and greater than 70% accuracy under high loss (Fig. 1.9).

1.4 Validation

We validate our methodology using our replay system and a testbed comprised of two commercial shaping devices. First, we verify that our approach captures salient features of the recorded traffic. Next, we validate that our replays trigger differentiation and identify the relevant features used for classification. To the best of our knowledge, this is first study to use commercial shapers to validate differentiation detection. Finally, we discuss potential overheads of using a VPN connection as a control and how we mitigate them.



(a) YouTube (b) Skype

Figure 1.2: Plots showing bytes transferred over time, with and without preserving packet timings for TCP (YouTube) and UDP (Skype) applications. By preserving inter-packet timings, our replay closely resembles the original traffic generated by each app.

Row	Changes in traffic	Detection result using:	
		Original ports	Different ports
1	No changes	YouTube	YouTube
2	Added a packet with 1 byte of data to the beginning of traffic	HTTP	P2P
3	Added one byte of random data to the beginning of first packet	HTTP	P2P
4	Replaced ‘‘GET’’ with a random string (same size)	HTTP	P2P
5	Replaced ‘‘youtube’’ string with a random one (first packet only)	HTTP	P2P
6	Replaced ‘‘youtube’’ string with a random one (first packet, HOST header only)	YouTube	YouTube
7	Added one byte of random data to the end of first packet	YouTube	YouTube
8	Added ‘‘GET ’’ to beginning of first packet	YouTube	YouTube

Table 1.1: Effect of different parameters on YouTube traffic detection for a popular commercial shaping device. IP addresses do not affect traffic classification, but ports and payloads have effects that vary.

1.4.1 Record/Replay similarity

Our replay script replicates the original trace, including payloads, port numbers, and inter-packet timing. We now show that the replay traffic characteristics are essentially identical to the recorded traffic. Figure 1.2 shows the results for a TCP application (YouTube (a)) and a UDP application (Skype (b)). As discussed above, preserving inter-packet timings is important to produce a replay that closely resembles the original trace (otherwise, we may claim that differentiation exists when the application would never experience it).

Figure 1.2(a) shows that our replay captures the behavior of the application, presenting the cumulative transfer over time for a YouTube trace collected and replayed over the Verizon mobile network. The figure shows the behavior for the original trace and two replays, one which preserves the inter-packet timing (overlaps with original) and one which transfers as fast as possible while preserving application-layer dependencies. Preserving inter-packet timings results in a replay that closely follows the recorded application behavior. Figure 1.2(b) shows similar results for Skype traffic.

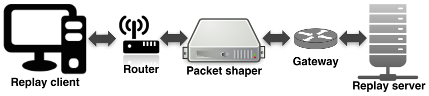


Figure 1.3: Our testbed for testing our differentiation detector.

1.4.2 Traffic shapers detect replayed traffic

We now validate that our replay traffic is properly classified for differentiation using commercial shaping products. We acquired traffic shaping products from two different vendors and integrated them into a testbed for validating whether replays trigger differentiation (Fig. 1.3). The testbed consists of a client connected to a router that sits behind a traffic shaper, which exchanges traffic with a gateway server that we control. The gateway presents the illusion (to the packet shaper) that it routes traffic to and from the public Internet. We configure the replay server to listen on arbitrary IP addresses on the gateway, giving us the ability to preserve original server IP addresses in replays (by spoofing them inside our testbed). We describe below some of our key findings from this testbed.

Differentiation testing must be extensible. One device lists more than 700 applications that it uniquely identifies, the other lists approximately 2,000 application filters. Further, both devices routinely update their classification rules, on timescales of months (if not shorter). Thus, testing only a small number of applications is insufficient. By allowing users to create their own traces to conduct differentiation tests, our approach is extensible to evolving differentiation targets.

Our replays trigger traffic shaping. For replays to be effective, they need to ‘‘look’’ like legitimate application traffic from the perspective of the traffic shaper, *i.e.,* replay traffic should be classified as the application that was recorded. We validate this for a variety of popular mobile applications: YouTube, Netflix, Skype, Spotify, and Google Hangouts. Figure 1.4 shows the YouTube and P2P policies on the shaper applied to our replay as we vary the application payload.

Reverse-engineering classification. We use our testbed to understand which features of the traffic were used to trigger shaping, and should be preserved in replays. We replay a recorded trace in our testbed multiple times, each time modifying a different feature of the traffic (destination IP, ports, and packet payloads) and observe its effect on classification.

Table 1.1 summarizes the results for running tests using YouTube. We find that regular expressions on packet payload are the primary signature, with YouTube being correctly identified when payloads were unmodified, despite changes to the server IP and ports (row 1). We modify three different aspects of the packet payload: the first payload byte, application-layer protocol details in the first packet, and arbitrary bytes after the first packet. The second row of the table shows that adding a one-byte packet caused the shaper to classify the replay as HTTP (when using

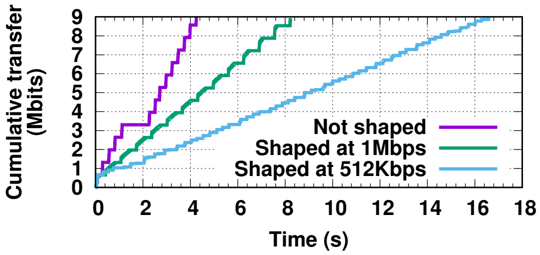


Figure 1.4: The shaper has rules to rate limit YouTube and P2P at 1Mbps and 512Kbps, respectively. *Shaped at 1Mbps* line is the YouTube replay in plaintext where shaper correctly detects as YouTube and shapes at 1Mbps. *Shaped at 512Kbps* is YouTube replay with randomized payload and ports, where shaper detects as P2P and limits at 512Kbps. The *Not shaped* line is YouTube replay with string ‘‘youtube’’ being replaced by a random string, which the shaper detects as ‘‘HTTP’’ and does not limit (since there are no rules for generic HTTP traffic).

port 80) or P2P (when using high, random ports). This behavior is identical if replacing the first byte, the ‘‘GET’’ command, or removing the string ‘‘youtube’’ from the payload of the first packet (rows 3-8). However, if the GET command was unmodified and at least one ‘‘youtube’’ string appeared in the first packet, the flow was classified as YouTube. This indicates that this shaping device is using regular expressions and falling back to port-based classification when regular expressions do not match. Interestingly, modifying any subsequent packets or payload after the GET command has no impact on classification.

We now summarize other key findings:

- **Server IP addresses do not affect classification.** While our shapers support IP-based policies, we found no evidence of IP-based classification rules.
- **Non-standard ports may still be subject to differentiation.** An important outcome of our testing is that traffic with random high port numbers may be classified as P2P. Glasnost’s [8] detection approach (among others) assumes that traffic sent on random ports will not be subject to differentiation, but traffic sent on standard ports will. However, our shaping device classifies traffic on random ports as P2P, which itself is often subject to differentiation.
- **Traffic shaping decisions are made early.** For HTTP requests, the first packet with payload, *i.e.,* request packet from client, is enough for classification. For UDP flows, *e.g.,* Skype, the shaper requires more packets for classification (*e.g.,* $\sim$10). This means that traffic differentiation tests can trigger differentiation using a small amount of data and only need to run long enough to observe steady state performance. This observation is particularly salient for mobile users who may be subject to data caps.
- **HTTPS does not preclude classification.** We expected that the shaper would not identify applications using HTTPS as their transport. However, we found that in practice the devices do

App	Avg packet size (bytes)	Avg throughput diff (%)
Youtube	705	1.13
Netflix	679	0.42
Hangout	435	2.06
Skype	234	15.64

Table 1.2: Minimal effect of VPN on our measurements. Skype packets are small, leading to larger throughput overhead compared to other apps.

detect applications, specifically via the SNI (Server Name Indication) field in the TLS handshake. Modifying this field results in detection as HTTPS (instead of the specific application).

A key question is whether our replay approach is effective for *all* shapers, not just those in our lab. While we can evaluate only the shapers we possessed, our replay approach successfully detects differentiation in operational networks (Section 1.6), indicating that our effectiveness is not limited only to the lab environment. It remains an open question whether there are other shaper models that are not accounted for in our design.

1.4.3 VPN overhead

Our control trials use a VPN tunnel, due to potential issues that arise when randomized ports and payloads are classified as P2P, or otherwise shaped. We now investigate the overhead of this approach.

VPN overheads can stem from (1) IPSec encapsulation and (2) latency added by going through the VPN (*e.g.,* if the VPN induces a circuitous route to the replay server). The overhead for IPsec encapsulation varies depending on the size of the payload, with smaller packets incurring higher overheads. The impact on throughput is relatively small, as shown in Table 1.2. For most applications, the difference in throughput with the VPN is 2% or less. However, for Skype, which has an average packet size of less than 300 bytes the throughput overhead is higher (15%). Note that we both record and replay traffic using an MTU that is small enough to prevent fragmentation in the VPN tunnel.

To minimize the impact of latency, we run the replay and VPN server on the same machine (currently running on Amazon EC2), which adds less than 2ms of latency compared to contacting the replay server directly (without a VPN tunnel). We argue these overheads are acceptably low, and represent a reasonable lower bound on the amount of differentiation we can detect.

1.5 Detecting Differentiation

In this section, we present the first study that uses ground-truth information to compare the effectiveness of various techniques for detecting differentiation, and propose a new test to address limitations of prior work. We find that previous approaches for detecting differentiation are inaccurate when tested against ground-truth data, and characterize under what circumstances these approaches yield correct information about traffic shaping as perceived by applications.

When determining the accuracy of detection, we separately consider three scenarios regarding the shaping rate and a given trace (shown in Figure 1.5):

- **Region 1.** If the *shaping rate is less than the average throughput* of the recorded traffic, a traffic differentiation detector should always identify differentiation because the shaper is guaranteed to affect the time it takes to transfer data in the trace.
- **Region 3.** If the *shaping rate is greater than the peak throughput* of the recorded trace, a differentiation detector should never identify differentiation because the shaper will not impact the application's throughput (as discussed in Section 1.2.2 we focus on shaping that will actually impact the application).
- **Region 2.** Finally, if the *shaping rate is between the average and peak throughput* of the recorded trace, the application may achieve the same average throughput but a lower peak throughput. In this case, it is possible to detect differentiation, but *the impact of this differentiation will depend on the application*. For example, a non-interactive download (*e.g.,* file transfers for app updates) may be sensitive only to changes in average throughput (but not peak transfer rates), while a real-time app such as video-conferencing may experience lower QoE for lower peak rates. Given these cases, there is no single definition of accuracy that covers all applications in region 2. Instead, we show that we can configure detection techniques to consistently detect differentiation or consistently *not* detect differentiation in this region, depending on the application.

In the remainder of this section, we use this region-based classification to evaluate three techniques for detecting differentiation. We describe them in the next section, explain how we calibrate their respective thresholds and parameters, then discuss how resilient these approaches are to noise (*e.g.,* packet loss), and how efficient they are in terms of data consumption.

1.5.1 Statistical tests

We explore approaches used in Glasnost [8] and NetPolice [9], and propose a new technique that has high accuracy in regions 1 and 3, and reliably does not detect differentiation in region 2.

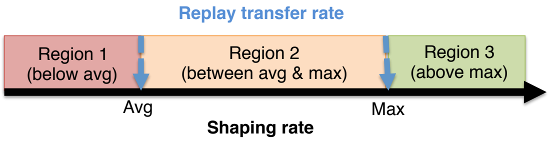


Figure 1.5: Regions considered for determining detection accuracy. If the shaping rate is less than the application's average throughput (left), we should always detect differentiation. Likewise, if the shaping rate is greater than peak throughput (right), we should never detect differentiation. In the middle region, it is possible to detect differentiation, but the performance impact depends on the application.

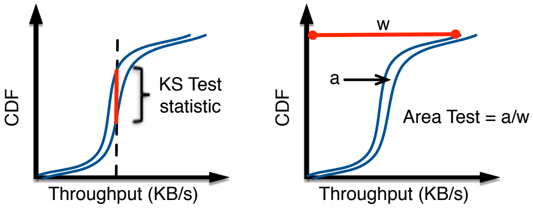


Figure 1.6: KS Test statistic (left) and Area Test statistic (right). In the former case, the difference between the distributions is small in terms of throughput, but the KS Test statistic is large. In the Area Test statistic, we find the area between the distributions and normalize it by the smaller of the peak throughputs for each distribution.

Glasnost: Maximum throughput test. Glasnost [8] does not preserve the inter-packet timing when replaying traffic, and identifies differentiation if the maximum throughput for control and exposed flows differ by more than a threshold. We expect this will always detect differentiation in region 1, but might generate false positives in regions 2 and 3 (because Glasnost may send traffic at a higher rate than the recorded trace).

NetPolice: Two-sample KS Test. As discussed in NetPolice [9], looking at a single summary statistic (*e.g.,* maximum) to detect differentiation can be misleading. The issue is that two distributions of a metric (*e.g.,* throughput) may have the same mean, median, or maximum, but vastly different *distributions*. A differentiation detection approach should be robust to this.

To address this, NetPolice uses the two-sample Kolmogorov--Smirnov (KS) test, which compares two empirical distribution functions (*i.e.,* empirical CDFs) using the maximum distance between two empirical CDF sample sets for a confidence interval α. To validate the KS Test result, NetPolice uses a resampling method as follows: randomly select half of the samples from each of the two original input distributions and apply the KS Test on the two sample subsets, and repeat this r times. If the results of more than $\beta\%$ of the r tests agree with the original test, they conclude that the original KS Test statistic is valid.

Our approach: Area Test. The KS Test only considers the difference between distributions

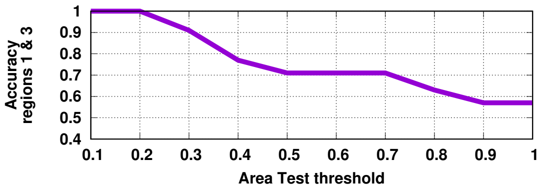


Figure 1.7: Calibrating Area Test for Youtube. We pick 0.2 as the threshold.

along the y-axis (as shown in Figure 1.6, left) even if the difference in the x-axis (in this case, throughput) is small. In our experiments, we found this makes the test very sensitive to small changes in performance not due to differentiation, which can lead to inaccurate results or labeling valid tests as invalid.

To address this, we propose an *Area Test* that accounts for the degree of differentiation detected: we find the area, a, between the two CDF curves (as shown in Fig. 1.6, right) and normalize it by the minimum of peak throughputs for each distribution, w. This test concludes there is differentiation if the KS Test detects differentiation *and* the normalized area between the curves is greater than a threshold t. We discuss how we select thresholds in Sec. 1.5.4.

1.5.2 Testbed environment

Our testbed consists of a commercial traffic shaper, a replay client, and a replay server (Figure 1.3). We vary the shaping rate using the commercial device to shape each application to throughput values in regions 1-3. Depending on the average and maximum throughputs for each trace, we vary the shaping rate from 0.1 Mbps to 30 Mbps (9% to 300% of peak throughput).

We emulate noisy packet loss in our tests using the Linux Traffic Control (`tc`) and Network Emulation (`netem`) tools to add bursty packet loss according to Gilbert-Elliott (GE) model [33]. We perform all our evaluations and calibrations under three conditions: 1) low loss (no added loss), 2) moderate loss (1.08%), and 3) high loss (1.45%). We found that correlated losses higher than 1.5% had disastrous effects on TCP performance and all detection techniques were inaccurate.

Evaluation method. We explore the parameter space by varying the application, loss rate, shaping rate, and number of replay repetitions for each statistical test. We present results for a few popular TCP-based applications (YouTube and Netflix) and UDP-based applications (Skype and Hangout); we also performed tests on a number of other applications (omitted for space). We repeat each test 10 times, where a test is a replay performed with a given set of parameters, with and without shaping. We present results that aggregate these tests to produce statistically significant results.

1.5.3 Evaluation criteria

We now describe the criteria under which we evaluate the three differentiation tests.

Overall accuracy: Using the taxonomy in Figure 1.5, a statistical test should always detect differentiation in region 1, and never detect differentiation in region 3. We define accuracy as the fraction of samples for which the test correctly identifies the presence or absence of differentiation in these regions.

Resilience to noise: Differences between two empirical CDFs could be explained by a variety of reasons, including random noise [10]. We need a test that retains high accuracy in the face of large latencies and packet loss that occur in the mobile environment. When evaluating and calibrating our statistical tests, we take this into account by simulating packet loss.

Data consumption: To account for network variations over time, we run multiple iterations of control and exposed trials back to back. We merge all control trials into a single distribution, and similarly merge all exposed trials. As we show below, this can improve accuracy compared to running a single trial, but at the cost of longer/more expensive tests. When picking a statistical test, we want the one that yields the highest accuracy with the smallest data consumption.

1.5.4 Calibration

We identify the most accurate settings of threshold values.

Glasnost: We used the threshold suggested by Glasnost, $\delta = 0.2$. This yields perfect accuracy in region 1, but generates 100% false positives in region 3 (as expected). Glasnost always detects differentiation in region 2. For applications that are sensitive to changes in peak throughput, this test will yield the correct result.

KS Test: We use thresholds suggested by NetPolice, *i.e.,* $\alpha = 0.95$ and $\beta = 0.95$. This yields good accuracy in regions 1 and 3, but inconsistent behavior in region 2. In other words, this test will sometimes detect differentiation in region 2, and sometimes not --- making it difficult to use for detection in this region. We also observed that even for tests with no added loss in our testbed over well-provisioned wired network, up to 8% of tests were considered invalid by KS Test due to the issue shown in Fig. 6, while the Area Test can correctly detect for differentiation (or no differentiation) in those cases.

Area Test: We find that $t = 0.1$ or $t = 0.2$ yield the best accuracy, depending on the application (Fig. 1.7). This yields good accuracy in regions 1 and 3, and consistent decisions of no differentiation for region 2. For apps that are insensitive to changes only to peak throughput (average throughput stays the same), this will yield the correct result.

App	KS Test		Area Test		Glasnost	
	R1	R3	R1	R3	R1	R3
Netflix	100	100	100	100	100	0
YouTube	100	100	100	100	100	0
Hangout	100	100	100	100	100	0
Skype	100	100	100	100	90	0

Table 1.3: Shaping detection accuracy for different apps, in regions 1 and 3 (denoted by R1 and R3). We find that the KS Test (NetPolice) and Area Test have similarly high accuracy in both regions (1 and 3), but Glasnost performs poorly in region 3.

App		Region 2	
	KS Test	Area Test	Glasnost
Netflix	65	28	100
YouTube	67	0	100
Hangout	40	0	100
Skype	55	10	92

Table 1.4: Percent of tests identified as differentiation in region 2 for the three detection methods. Glasnost consistently identifies region 2 as differentiation, whereas the Area Test is more likely to not detect differentiation.

1.5.5 Evaluation results

Summary results for low noise. We first consider the accuracy of detecting differentiation under low loss scenarios. Table 1.3 presents the accuracy results for four popular apps in regions 1 and 3. We find that the KS Test and Area Test have similarly high accuracy in both regions 1 and 3, but Glasnost performs poorly in region 3 because it will detect differentiation, even if the shaping rate is above the maximum. We explore region 2 behavior later in this section; until then we focus on accuracy results for regions 1 and 3 and the two KS Tests (as they do not identify differentiation in region 3)

Impact of noise. Properties of the network unrelated to shaping (*e.g.,* congested-induced packet loss) may impact performance and cause false positives for differentiation detectors. In Fig. 1.8 we examine the impact of loss on accuracy for one TCP application (Netflix) and one UDP application (Skype). We find that both variants of the KS Test have high accuracy in the face of congestion, but the Area Test is more resilient to moderate and high loss for Skype.

Impact of number of replays. One way to account for noise is to use additional replays to average out any variations from transient noise in a single replay. To evaluate the impact of additional replays, we varied the number of replays combined to detect differentiation under *high loss* and plotted the detection accuracy in Fig. 1.9.[4] In the case of moderate or high loss, increasing the number of replays improves accuracy, particularly for a short YouTube trace. In all cases, the Area

[4] Additional replays did not help when there is low loss.

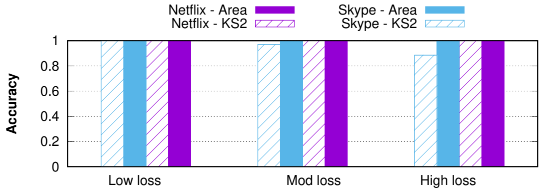


Figure 1.8: Accuracy against loss.

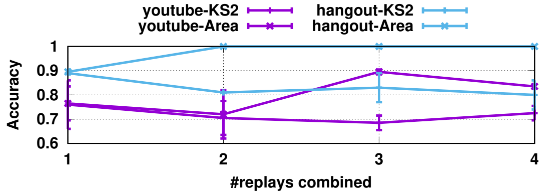


Figure 1.9: Effect of combining replays on accuracy in **high loss**.

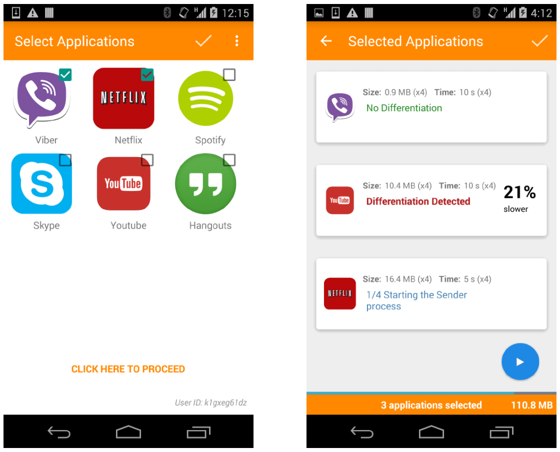
Test is more accurate than (or equal to) the KS Test. Importantly, the accuracy does not significantly improve past 2 or 3 replays. In our deployment, we use 2 tests (to minimize data consumption).

Region 2 detection results. Recall that our goal is to identify shaping that impinges on the network resources required by an application. However, in region 2, shaping may or may not impact the application's performance, which makes identifying shaping with actual application impact challenging. We report detection results for the three tests in region 2 in Table 1.4. We find that Glasnost consistently identifies differentiation, the Area Test consistently does *not* detect differentiation, and the KS Test is inconsistent. When testing an application that is sensitive to shaping of its peak throughput, the Glasnost method may be preferable, whereas the Area Test is preferred when testing applications that can tolerate shaping above their average throughput.

When detecting differentiation, we conservatively avoid the potential for false positives and thus do not report differentiation for region 2. Thus, we use the Area Test in our implementation because it will more reliably achieve this result (Table 1.4). Determining which apps are affected by region 2 differentiation is a topic of future work.

1.6 Measurement Study

We now use our calibrated detection approach to identify cases of differentiation using measurement study of production networks. For each network, we focus on detection for a set of popular apps that we expect might be subject to differentiation: streaming audio and video (high bandwidth) and voice/video calling (competes with carrier's line of business). The data from these experiments was



Size: 0.9 MB (x4) Time: 10 s (x4)
No Differentiation

Size: 10.4 MB (x4) Time: 10 s (x4)
Differentiation Detected
21%
slower

Size: 16.4 MB (x4) Time: 5 s (x4)
1/4 Starting the Sender process

3 applications selected 110.8 MB

Figure 1.10: Screenshot of our Android Differentiation Detector app.

initially collected between January and May 2015. We then repeated the experiments in August 2015, after the new FCC rules prohibiting differentiation took effect.

1.6.1 System implementation

We now describe the implementation of our client and server software. To promote transparency and guide policy for traffic differentiation, our source code and analysis results will be made continuously available at `http://dd.meddle.mobi`.

Client. We implement our replay client in an Android app called *Differentiation Detector*. It does not require any root privileges, and is available for download in the Google Play store.[5] The app consists of 14,000 lines of source code (LOC), which includes the Strongswan VPN implementation [34] (4,600 LOC). The app conducts differentiation tests and reports results to our replay servers as part of an IRB-approved study. Users undergo informed consent when the app first runs, and cannot run any tests unless they consent to participate in our study. An iOS implementation is under development.

The app (Fig. 1.10) is pre-loaded with replay transcripts for Viber, Netflix, Spotify, Skype, YouTube, and Google Hangouts, so that users can test for differentiation of these apps without recording traffic.[6] For each trace, the app follows the replay procedure described in Section 1.3 and repeats the replay tests twice[7] back to back.

[5] `https://play.google.com/store/apps/details?id=com.stonybrook.replay`

[6] While we currently only support Android, the approach should also work on any mobile OS that supports VPN connectivity, including iOS. We are currently developing support for users to record their own traces from mobile devices.

[7] Users can increase the number of back-to-back replays for more accuracy.

At the end of each replay, metadata such as carrier name, OS information, network type, and signal strength, is collected and sent to the server for analysis. To account for the case where background traffic might affect our detection results, future version of the app will measure data consumption during replays and discard results where we detect interference from other traffic. Users can also access their historical results through the app.

Server and analysis code The server coordinates with the client to replay traces, and records packet traces for analysis. This interaction is managed by side-channel connections that identify which trace is being replayed and what ports will be opened in the case of NAT traversal. For networks that allow it (*e.g.,* in our testbed), we support IP spoofing so our replay server can send packets using the IP addresses in arbitrary recorded traces. The server logic is 1,850 lines of code (Python).

Our analysis code implements tests for throughput, RTT, jitter, and loss differentiation. We implement KS Test and Area Test, and use simple scalar metrics (average, max) for loss. Our parsing script supports TCP and streaming UDP applications. It uses `tshark` to extract conversations. Together, these artifacts consist of 1,170 lines of code.

1.6.2 Challenges in operational networks

In this section we discuss several challenges that we encountered when attempting to identify differentiation in mobile networks. To the best of our knowledge, we are the first to identify these issues for detecting differentiation and discuss workarounds that address them.

Per-client management. To replay traffic for multiple simultaneous users, our replay server needs to be able to map each received packet to exactly one app replay. A simple solution is to put this information in the first few bytes of each flow between client and server. Unfortunately, as shown in Section 1.4.2, this can disrupt classification and lead to false negatives. Instead, we use side-channel connections out of band from the replay to supply information regarding each replay run.

NAT behavior. We also use the side-channel to identify which clients will connect from which IPs and ports. For networks without NAT devices, this works well. However, many mobile networks use NAT, meaning we can't use the side-channel to identify the IP and port that a replay connection will use (because they may be modified by the NAT). For such networks, each replay server can reliably support only one active client per ISP and application. While this has not been an issue in our initial app release, we are investigating other work-arounds. Moreover, to scale the system, we can use multiple replay servers and a load balancer. The load balancer can be configured to assign clients with the same IP address to different servers.

‘‘Translucent’’ HTTP proxies. It is well known that ISPs use transparent proxies on Web traffic [35]. Our system works as intended if a proxy simply relays exactly the same traffic sent by a replay client. In practice, we found examples of ‘‘translucent’’ proxies that issue the same HTTP requests as the client but change connection keep-alive behavior; *e.g.,* the client uses a persistent connection in the recorded trace and the proxy does not. To handle this behavior, our system must be able to map HTTP requests from unexpected connections to the same replay.

Another challenge with HTTP proxies is that the proxy may have a different public IP address from other non-HTTP traffic (including the side-channel), which means the server will not be able to map the HTTP connections to the client based on IP address. In these cases the server replies to the HTTP request with a special message. When the client receives this message, it restarts the replay. This time the client adds a custom header (X-) to HTTP requests with client's ID, so the server can match HTTP connections with different IP addresses to the corresponding clients. The server then ignores this extra header for the remainder of the replay. We have also observed that some mobile providers exhibit such behavior for other TCP and UDP ports, meaning we cannot rely on an X- header to convey side-channel information. In these cases we can identify users based on ISP's IP subnet, which means we can only support one active client per replay server and ISP, regardless of the application they are replaying.

Content-modifying proxies and transcoders. Recent reports and previous work highlight cases of devices in ISPs that modify customer traffic for reasons such as tracking [36], caching [35], security [37], and reducing bandwidth [38]. We also saw several cases of content-modifying proxies. For example, we saw Sprint modifying HTTP headers, Verizon inserting global tracking cookies, and Boost transcoding YouTube videos.

In our replay methodology, we assumed that packets exchanged between the client and server would not be not modified in flight, and trivially detect modification because packet payloads do not match recorded traces. In the case of header manipulation, we use an edit-distance based approach to match modified content to the recorded content it corresponds to. While our system can tolerate moderate amounts of modification (*e.g.,* HTTP header manipulation/substitution), if the content is modified drastically, *e.g.,* transcoding an image to reduce its size, it is difficult to detect shaping because the data from our control and exposed trials do not match. In our current system, we simply notify the user that there is content modification but do not attempt to identify shaping. We leave a more detailed analysis of this behavior to future work.

Caching. Our replay system assumes content is served from the replay server and not an ISP's cache. We detect the latter case by identifying cases where the client receives data that was not sent by the server. In the few cases where we observed this behavior, the ISPs were caching small static objects (*e.g.,* thumbnail images) which were not the dominant portion of the replay traffic

ISP	YT	NF	SF	SK	VB	HO
Verizon	M	M	M	-	-	-
T-Mobile	-	-	-	-	-	-
AT&T	F	F	F	-	-	-
Sprint	M/P	M/P	M/P	-	-	-
Boost	M	M	M	-	-	-
BlackWireless	60%	-	-	-	-	-
H2O	37%*	45%*	65%*	-	-	-
SimpleMobile	36%	-	-	-	-	-
NET10	P	P	P	-	-	-

Table 1.5: Shaping detection results per ISP in our dataset, for six popular apps: YouTube (YT), Netflix (NF), Spotify (SF), Skype (SK), Viber (VB), and Google Hangout (HO). When shaping occurs, the table shows the difference between average throughput (%) we detected. A dash (-) indicates no differentiation, (F) means IP addresses changed for each connection, (P) means a ‘‘translucent’’ proxy changed connection behavior from the original app behavior, and (M) indicates that a middlebox modified content in flight between client and server. *For the H2O network, replays with random payload have better performance than VPN and exposed replays, indicating a policy that favors non-video HTTP over VPN and streaming video.

(*e.g.,* when streaming audio) and had negligible effect on our statistical analysis.[8] Going forward, we expect this behavior to be less prominent due to the increased use of encrypted protocols, *e.g.,* HTTP/2 and QUIC. In such cases, an ISP may detect and shape application traffic using SNI for classification, but they cannot modify or cache content.

1.6.3 Differentiation results

In this section, we present results from running our *Differentiation Detector* Android app on popular mobile networks. This dataset consists of 4,786 replay tests, covering traces from six popular apps. We collected test results from most major cellular providers and MVNOs in the US; further, we gathered measurements from four international networks.

Our app supports both VPN traffic and random payloads (but not random ports) as control traffic. We use the latter only if a device does not support VPN connectivity or the cellular provider blocks or differentiates against VPN traffic. After collecting data, we run our analysis to detect differentiation; the results for US carriers are presented in Table 1.5. In other networks such as Idea (India), JazzTel (Spain), Three (UK), and T-Mobile (Netherlands), our results based on a subset of the traces (the traces that users selected) indicated no differentiation.

Our key finding is that our approach successfully detect differentiation in three mobile networks (BlackWireless, H2O, and SimpleMobile), with the impact of shaping resulting in up

[8] Boost was one exception, which we discuss later.

to 65% difference in average throughput. These shaping policies all apply to YouTube (not surprising given its impact on networks), but not always to Netflix and Spotify. We did not identify differentiation for UDP traffic in any of the carriers. Interestingly, H2O consistently gives *better* performance to port 80 traffic with random payloads, indicating a policy that gives relatively worse performance to VPN traffic and streaming audio/video. We tested these networks again in August 2015 and did not observe such differentiation. We speculate that these networks ceased their shaping practices in part due to new FCC rules barring this behavior, effective in June 2015.

We contacted these ISPs we tested for comment on the behavior we observed in their networks. At the time of publication, only one ISP had responded (Sprint) and did not wish to comment.

While attempting to detect differentiation, we identified other interesting behavior that prevented us from identifying shaping. For example, Boost transcodes YouTube video to a fraction of the original quality, and then caches the result for future requests. Other networks use proxies to change TCP behavior. We found such devices to be pervasive in mobile networks.

1.7 Discussion

This chapter focuses on detecting differentiation that impacts application performance. While investigating shaping, our technique revealed a number of other issues that are beyond the scope of this chapter.

Assigning blame. When differentiation is detected, we generally assign blame to the network from where the test is launched, *i.e.,* the access network. While in theory it might be caused by any network along the path between the access network and our replay server, we expect that transit networks will not have incentive to differentiate and we ensure that our replay hosting sites also do not differentiate traffic. As part of future work, we will investigate how to combine differentiation results from multiple vantage points (*e.g.,* using tomography) to identify cases when differentiation is not caused by the access network, but rather an intermediate network.

Differentiation against VPN traffic. A subtle challenge arises when recording traces of app-generated network traffic over a production network: what happens when recorded traffic is subject to differentiation (*e.g.,* shaping)? If the recorded and replayed traffic are both shaped, a naive detector will indicate that the network has no differentiation. One simple solution is to allow users to run traces captured from a network of the same type (*e.g.,* WiFi, 3G, LTE) where we know the recorded traces were not subject to differentiation (our current version of the app ships with traces from different applications recorded over a well-provisioned network that does not shape traffic).

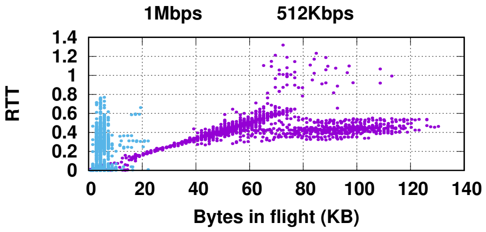


Figure 1.11: Effect of shaping on latency for a TCP stream. Shaping at 1 Mbps exhibits expected queuing delays that identify the size of the buffer (x=75 KB); however, shaping at a lower rate (512 kbps) leads to bimodal RTTs largely due to TCP never exiting slow start.

While this solves the problem, it does not tell us when there is a problem in the first place. For this, we need to detect that the VPN traffic itself is subject to differentiation. If there exists at least one class of traffic that is *not* shaped in our tests, then it will get higher performance outside the VPN tunnel compared to inside the tunnel. In this case, there will be at least one case where tunneled performance will be different from non-tunneled. As a result, we can reliably conclude that there is some kind of differentiation[9]; however, we cannot yet directly attribute it to shaping against a VPN.

One potential way to address this is to exploit the observation that popular shaping devices use token-bucket queues to enforce fixed-rate shaping. As shown in Fig. 1.11 (1 Mbps) shaping devices exhibit a characteristic queueing delay evident when plotting bytes in flight vs. RTT. If a traffic class is subject to differentiation, we will observe increased delays compared to a case that is not managed via a queue.

If indeed the control traffic is shaped, it limits our ability to test other types of traffic for differentiation---this is true of *any* differentiation detection approach. However, *detection of this behavior itself is important information*; for example, in the US even this behavior should be in violation of FCC rules effective June 12, 2015 [7].

Shaping configurations can have unintended consequences for TCP. In one sample configuration already used by an ISP (Fig. 1.11, 512 Kbps), we found that the shaping rate was so low that it prevented TCP from ever exiting slow start. Importantly, TCP increased its congestion window exponentially in time while the shaper's queue drained linearly in time, leading to substantial queuing and packet loss.

This is problematic for two reasons. First, this policy leads to wasted bandwidth at the network edge, which will cost the ISP running the shaper. Second, it is unclear how the user is being charged for this lost traffic. If users are billed based on bytes entering the network before

[9] More generally, we can draw the same conclusion if at least one class of traffic is shaped *differently* from VPN traffic.

hitting the shaper, the ISP will over charge the user for packets that were never delivered. Such policies, and their implications, merit further investigation as part of a future study.

Evasion. One of our goals is to improve transparency for network management practices in cellular data networks. ISPs wishing to avoid characterization may attempt to do so by blocking VPN traffic or selectively changing policies for our replay traffic. If ISPs choose to block our traffic, we can easily detect it and report this result to users and policy makers. For cases where the traffic is selectively given different treatment, we can attempt to disguise replay traffic, using different servers, ports and control traffic techniques (*e.g.,* different payload/port randomization scheme).

Data caps. Some providers have ‘‘unlimited’’ data plans with lower data rates after usage exceeds a certain cap. We do not expect to detect this from a single user, but may be able to identify it by combining results from multiple users or by monitoring data usage via Android APIs.

1.8 Conclusion

We presented and evaluated a new tool for accurately detecting differentiation using a VPN to record and replay traffic generated by apps to improve transparency in mobile networks. We found extensive use of middleboxes that break end-to-end principles in mobile networks, and identified their impact on applications.

From a network neutrality perspective, our findings are concerning in part due to observed shaping policies and third-party modification of traffic, both of which are pervasive. There is little research into the impact of such devices, and currently policymakers have essentially no guidance as to how to regulate this behavior. As part of our ongoing work, we are using a Web site to make our results public --- both to improve transparency for users and to guide future policies. In addition, we are investigating how to fingerprint middleboxes in mobile networks and developing techniques to understand their impacts on a wide range of applications.

Chapter 2

Information Controls Lab (ICLab)

In this chapter, we introduce Information Controls Lab (ICLab)—a project focused on collecting and analyzing reliable information controls data on the Internet at scale—by comparing it with another design point in the space of Internet censorship measurement with particular emphasis on how they address the challenges of locating vantage points, choosing content to test, and analyzing results. We discuss the trade offs of decisions made by each platform and show how the resulting data provides complementary views of global censorship. Finally, we discuss lessons learned and open challenges discovered through our experiences.

2.1 Introduction

The last five years have cemented the Internet as critical infrastructure for communication. In particular, it has demonstrated high utility for citizens and political activists to obtain accurate information, organize political movements, and express dissent. This fact has not gone unnoticed, with governments clamping down on this medium *via* censorship and information controls. Consequently, there has been a surge of interest in measuring various aspects of online information controls. More specifically, data obtained from such measurements has been used by (1) political activists to understand the motivation for and the impact of such government policies and (2) researchers to build safer and more secure censorship circumvention tools by understanding the techniques used to implement these policies [39].

While there have been numerous efforts to characterize online information controls [40--45], the data gathered or used by these measurements have limited scope due to the specificity of locations and time-periods considered. In order to gain a nuanced understanding of the evolution of Internet censorship, in terms of policy and techniques, a measurement platform needs to be able to gather longitudinal data from a diverse set of regions while performing accurate analysis using robust and well specified techniques. We present ICLab by comparing it with OONI-- that represent a different point in the censorship measurement design space.

In this chapter, we first identify three primary design decisions made in the development of censorship measurement platforms. Then, we describe how ICLab and OONI address these decisions, while considering the impact of these decisions on the measurement results produced by the systems. Finally, we show how ICLab and OONI, when used together, provide a unique insight into the current state of information controls around the globe.

2.2 Design Decisions

We now discuss the choices of (1) where measurements are run, (2) how measurements are run, and (3) how data is intepreted . These three design decisions are central to the design of a global censorship measurement platform.

2.2.1 Where measurements are run

There are two options when considering vantage points: crowd-sourced and dedicated infrastructure vantage points. Platforms using a crowd-sourced approach rely on volunteers running measurement software. They have the ability to turn citizens in any location into vantage points. Dedicated infrastructure, on the other hand are distributed and operated exclusively for the platform. Both approaches have their own benefits and drawbacks.

Cost and availability. A hurdle in setting up a dedicated infrastructure is distributing infrastructure globally. However, once this infrastructure is in place, it has the capability of performing on-demand measurements; limited only by the reliability and uptime of the infrastructure. Crowd-sourced platforms, on the other hand, incur no setup cost but are dependent on the availability of volunteers to execute measurements. As a consequence, crowd-sourced platforms are unable to provide a reliable flow of measurements from a region.

Representativeness and diversity of measurements. Crowd-sourced platforms have the potential to obtain a view of the Internet from a wide variety of networks (*e.g.,* residential, academic, and corporate). In contrast, dedicated infrastructure faces an uphill battle of distributing devices or may leverage existing infrastructure (*e.g.,* academic networks, or dedicated hosting networks). Vantage point location can impact conclusions drawn from their measurements. As an example, measurements conducted from the UK academic network (JANET) do not observe the ‘‘Great

Firewall of Cameron'' [46], since they are placed outside of its purview. Crowd-sourced platforms can also leverage public interest and news coverage to introduce additional vantage points.

Safety and risk. Information controls measurements using humans in the field poses a significant and hard to quantify risk. In many regions (*e.g.,* Syria) this risk has been determined to be too high for volunteers. Such risks impede crowd-sourced measurements, while infrastructure (such as VPN or hosting networks) allow for measurements while posing little to no risk to users.

2.2.2 Measurement autonomy

A censorship measurement platform can either use a central server to schedule experiments, or leave these tasks to each vantage point. This dichotomy has an impact on several capabilities of the platform.

Time and context sensitive measurements. The time and political context of measurements are important for understanding evolving and abrupt policy changes. As an example, during the rise of ISIS in 2014, the Indian government blocked (and subsequently unblocked) access to 32 websites including GitHub, Vimeo, and PasteBin for propagating ``Anti India content'' [47]. Centrally controlled measurement platforms have the advantage of being able to evolve existing and schedule new measurements in response to changing political and social situations. Locally controlled platforms, however, do not have this capability. Instead they are dependent on the update schedule of the local vantage point.

Infrastructure requirements. The ability to remotely schedule experiments and aggregate data centrally allows for the use of computationally constrained infrastructure, not needing technically savvy local maintenance efforts. This comes with the cost of bandwidth requirements associated with shipping unprocessed data to a central server. Locally controlled platforms require local management of the platform infrastructure to ensure up-to-date experiments, higher computational capabilities for processing gathered data, and lower bandwidth for communicating processed results of measurements.

2.2.3 Gathering and interpreting data

A censorship measurement platform must specify data collected and how it will identify censorship in this data.

Type and quantity of data gathered. A platform may record packet captures of entire tests or selectively gather data such as packet headers and responses. While complete packet captures are ideal for deep aposteriori analysis and to identify censorship not visible at the application layer. However, they require root privileges, high storage and bandwidth requirements, and may accidentally collect data of other system users.

Identifying censorship events. Another challenge that arises during the processing of gathered data is defining when ‘‘censorship’’ has occurred. This task is complicated by strange protocol implementations (*e.g.,* load balancers that cause gaps in TCP sequence numbers), server side blocking [48], and regular network failures.

2.3 The ICLab and OONI Platforms

In this section we describe the design decisions made during the development of the ICLab and OONI platforms.

2.3.1 Vantage points

The most fundamental difference between the ICLab and OONI platforms is the approach each system takes to recruit vantage points. ICLab relies on a dedicated infrastructure to perform measurements. This allows measurements that require permissions that may not be compatible with software to be run on end-user systems. As a consequence, the system has thus far focused on deployment on VPN vantage points and a limited deployment of Raspberry Pi’s installed with ICLab software. In contrast, OONI takes a lighter-weight software-based approach. and assumes some amount of technical savvy on the part of volunteers.

This leads to differences in the availability and representativeness of measurements from each platform. In Figure 2.1(a), we see that as a result of the decision to use VPN end points, ICLab is able to provide vantage points for measurements in significantly more countries than OONI (151 for ICLab and 46 for OONI in the last 100 days[1]). However, we found that that OONI’s crowd-sourced model is able to provide more AS-level diversity -- *i.e.,* OONI provides vantage points from an average of 3.15 different networks (ASes) in each country, compared to ICLab’s average of 1.46 networks per country.

[1] Since its release in 2012, OONI has received nearly 10M measurements from volunteers in 95 countries.

2.3.2 System architecture

In terms of system architectures, ICLab uses a central controller to schedule experiments while OONI processes data both on the vantage point and inside of its data processing pipeline. This introduces several key differences in how each platform handles its vantage points.

Measurement scheduling. ICLab takes a centralized approach to scheduling experiments, leveraging a single server that is able to schedule experiments on all deployed nodes (*e.g.,* VPNs, Raspberry Pis) or a subset thereof (e.g, a given country). This facilitates the execution of ongoing or one-off measurements. In contrast, OONI takes a decentralized approach. Recommended measurements are hard-coded into the OONI platform source-code and require vantage points (technically savvy volunteers) to regularly download updates in order to execute new measurements. Repetition of measurements is dependent on individual volunteer availability. Volunteers also have the option to add their own tests and modify inputs to existing tests (*e.g.,* they may change the set of URLs being used by a test).

Performing measurements. ICLab and OONI also differ in their approach to performing measurements. ICLab takes a ‘‘simple node’’ approach, with nodes largely being responsible for collecting data and transmitting it back to the central server for later analysis. This lowers the computational requirement of the vantage points but increased demands on bandwidth. In contrast, OONI performs measurements and analysis on the device and ships processed data back to a central server. Importantly, OONI allows volunteers to opt-out of submitting measurement reports to the OONI publishing server, while ICLab takes an informed approach with vantage points opting into participate in the system.

Figure 2.1(b) shows the impacts of these decisions on the platforms. OONI has a core set of vantage points that continuously measure and a few opportunistic measurements. ICLab on the other hand exhibits large coordinated testing as a result of its VPN vantage points.

2.3.3 Tests and analysis

Both platforms perform a battery of tests to identify censors that may be blocking or manipulating content. The ICLab test infrastructure is extensible, allowing new tests to be scheduled on vantage points without the need for updating their software. In addition to custom tests, the ICLab platform periodically schedules a baseline test on each vantage point. This baseline experiment tests connectivity to a set of URLs that are composed of the Alexa Top 500 websites and a country-specific list of potentially blocked URLs (obtained from the CitizenLab). In contrast, tests on the OONI platform are not scheduled remotely and new tests need to be obtained by software updates. Existing tests, however, do not require software updates to evolve the list of domains that they test connectivity to. The default experiments included in the OONI platform test connectivity to the global and country-specific lists of potentially blocked URLs (also obtained from the CitizenLab).

In terms of analysis, the ICLab platform does not perform analysis on the vantage points, rather it leaves all post-processing to the centralized servers. This allows ICLab to perform retroactive analysis on existing results. The OONI platform, on the other hand, performs data analysis on the vantage points. This allows independent and private deployments by in-country watchdog groups.

We now briefly describe the tests conducted to identify censorship by each platform.

DNS anomaly detection. For each URL to be tested on a given vantage point, the ICLab and OONI vantage points perform DNS name resolution queries for the domain name associated with that URL using both the default DNS resolver configured on the machine as well as Google's DNS at `8.8.8.8`. The ICLab platform concludes that an anomaly (*e.g.,* DNS injection, tampering, *etc.*) has occurred if a second DNS response is received within 2 seconds of the first. The OONI platform on the other hand, makes several requests at once and does not wait between requests. Requests are also made to control resolver that binds to a non standard DNS port. The client is able to report failures to resolve directly, and resolutions are included in the generated report to allow further analysis by the central analysis infrastructure.

HTTP tampering, proxy, and blockpage detection. For each URL to be tested on a given vantage point, the ICLab and OONI vantage points issue HTTP GET requests and record received responses, with ICLab to follow redirects. The responses received from these tests are processed to identify blockpages and evidence of HTTP tampering. The ICLab platform uses regular expression pattern matching to identify known blockpages and responses obtained by the same test executed from a censor-free vantage point in the US to identify instances of content manipulation. The OONI platform uses meta-data (*e.g.,* status codes, response sizes, *etc.*) obtained from a Tor control channel to identify HTTP tampering. Additionally, the OONI platform is also able to detect the presence of HTTP proxies. It does this by generating malformed HTTP requests that cause proxies on the vantage point network to reveal their presence (*e.g.,* by modifying the malformed headers). The data processing pipeline is then capable of identifying specific types of proxy software based on known fingerprints.

TLS man-in-the-middle detection. The ICLab platform also performs tests on HTTPS compatible URLs. For each such URL, a TLS handshake is performed and all server certificates that are received are checked for validity. If they are found to have been expired or signed by an untrusted certificate authority, a TLS anomaly is reported.

Sequence number, TTL, and RST anomaly detection. For each of the above tests, the ICLab platform analyzes raw data (packet captures) of TCP streams to identify inconsistent sequence number and TTL values in packet headers. Additionally, the presence of pre-mature RST packets is also recorded. If any of these are identified, the ICLab platform reports anomalies that may be the result of a censor injecting packets into a TCP stream.

TCP connectivity test. The OONI platform attempts to establish TCP connections to a specified set of hosts to validate that the handshake can be completed and detect instances of IP level blocking. In this test, the vantage point attempts to establish a connection to the end host directly and also via a Tor control channel. If the control channel succeeds, while the direct connection fails, an anomaly is reported.

Circumvention protocol tests. Finally, the OONI platform also includes a set of tests designed to detect the availability of several circumvention system by (1) mimicking the protocols involved and (2) by launching bundled instances of the actual tools and checking whether they are able to successfully complete connections. Currently the test considers the connectivity of all Tor pluggable transports (scramblesuit, meek, fteproxy, obfsproxy versions 2 to 4), Psiphon, Lantern, and the OpenVPN protocol.

2.4 Comparing results of ICLab and OONI

In this section, we provide results from our analysis of measurements generated by the ICLab and OONI platforms. In particular, we use measurements from each platform to (1) understand which countries are the least free -- *i.e.,* have the highest amounts of censorship and (2) to demonstrate challanges in finding ground truth when conducting censorship measurements. We use these results to demonstrate how the ICLab and OONI platforms can provide complementary insights into censor behavior.

2.4.1 Identifying the least free countries

Using the tests described in Section 2.3, we now report the countries found to be the least free based on measurements obtained from the ICLab and OONI platforms.

Figure 2.2 illustrates the fraction of URLs that were censored in each of the six least free countries -- Iran (IR), Saudi Arabia (SA), India (IN), Cyprus (CY), China (CN), and Russia (RU) -- based on tests conducted by the ICLab platform; and Iran (IR), Saudi Arabia (SA), India (IN), Greece (GR), Qatar (QA), and Turkey (TR) according to OONI's measurements. We note that the remaining countries only displayed marginal amounts of censorship -- *i.e.,* under 5% of all tested

URL	# of VPs	URL	# of VPs
battle.net	1459	uol.com.br	842
163.com	1417	alibaba.com	748
baidu.com	1350	yahoo.com	700
hao123.com	1333	directrev.com	564
youth.cn	918	roblox.com	415

Table 2.1: Websites with the highest number of TCP `RST` packets in ICLab.

URLs were censored. We find that both platforms find the most censorship in Iran, Saudi Arabia, and India.

In the specific case of Iran, both the ICLab and OONI platforms show comparably high levels of censorship. We see that both platforms are able to detect the large fraction of blockpages served by Iranian censors and that in addition to identifying the Iran blockpage, the TTL and RST anomaly detectors in the ICLab platform are also triggered. We attribute the extremely high levels of blocking observed to the fact that the measurements from both platforms were carried out around the same time period as the Iranian parliamentary elections. We investigate further using past data from the OONI platform and confirm that in October 2015, four months before the elections, a sharp rise in censorship was observed. This is illustrated in Figure 2.3. The URLs tested by the OONI platform during this time included content relating to political news and speech, social media, censorship circumvention tools, and pornography.

Analyzing the results for Saudi Arabia and India we find that measurements performed on the ICLab platform see significantly less information controls than on the OONI platform. In particular, while measurements from the ICLab platform detected a number of blockpages and RSTs in each of these countries, we find that it did not encounter the large number of DNS anomalies and incomplete TCP connections that are observed by the OONI platform. We attribute this to the fact that measurements from the OONI platform are usually obtained from residential networks where covert censorship is observed (*i.e.,* censorship without explicitly serving a blockpage).

Finally, the results from both platforms also provide an insight into the censorship infrastructure in place in each country. The presence of a single dominant method of censorship in Iran and Saudi Arabia are indicative of the presence of a central censorship apparatus, while the case of India -- where multiple equally dominant methods are observed -- is indicative of censorship being implemented by local ISPs rather than the central government.

2.4.2 The elusive ground truth

Ground truth plays a crucial role in analyzing censorship measurement data, and there are several challenges associated with gathering ground-truth censorship data at scale. Comparing measurement data collected in the field against a baseline collected in well-provisioned network settings (*i.e.,* in the lab) helps delineate censorship from server-side blocking caused by VPN blocking or automated measurements not looking like real user traffic. Table 2.1 shows websites with the highest number of TCP `RST` packets in their streams across ICLab's vantage points, pointing to possible server-side blocking.

Among the list of websites with many observed `RST`s are several websites hosted in China (*e.g.,* `163.com` and `baidu.com`) that exhibit anomalous TCP behaviors when queried by ICLab-- *i.e.,* the IPID values from `SYN` and `SYN-ACK` packets are different from the rest of the packets receieved, and sequence numbers overlapping between packets. This anomaly is hard to distinguish from anomalous traits that are caused by the Great Firewall of China. Similarly, gaming websites `roblox.com` and `battle.net` aggressively block VPN users, while `yahoo.com` and `directrev.com` (an ad marketplace website) do the same to a lesser degree. Other websites can also respond unexpectedly (*e.g.,* due to server misconfiguration) and trigger false alarms. As an example, the Iranian retail website `digikala.com` shows sequence number anomalies as tested by 587 of ICLab's vantage points, but is not censored in any of them.

OONI faces a similar problem in determining ground truth. Many of the 'control' measurements used by the local client to determine what sites should look like are conducted through Tor. In practice, many websites either fully deny, or display substantially different content to visitors through the Tor network, making it difficult for the probe to determine if the local result is correct or not.

An additional challenge arises due to websites that are suddenly unavailable for non-censorship reasons -- *e.g.,* a dead website with a registered domain and unavailable webserver. For these cases, the ICLab platform verifies if the webpage could be loaded from any one of its other vantage points. If the page was unable to be loaded successfully, it is discarded from the test outputs. Figure 2.4 illustrates the URLs that were censored in the 20 least free countries. We observe several vertical bands in this figure. These are indicative of dead websites, ones which could not be loaded from any vantage point.

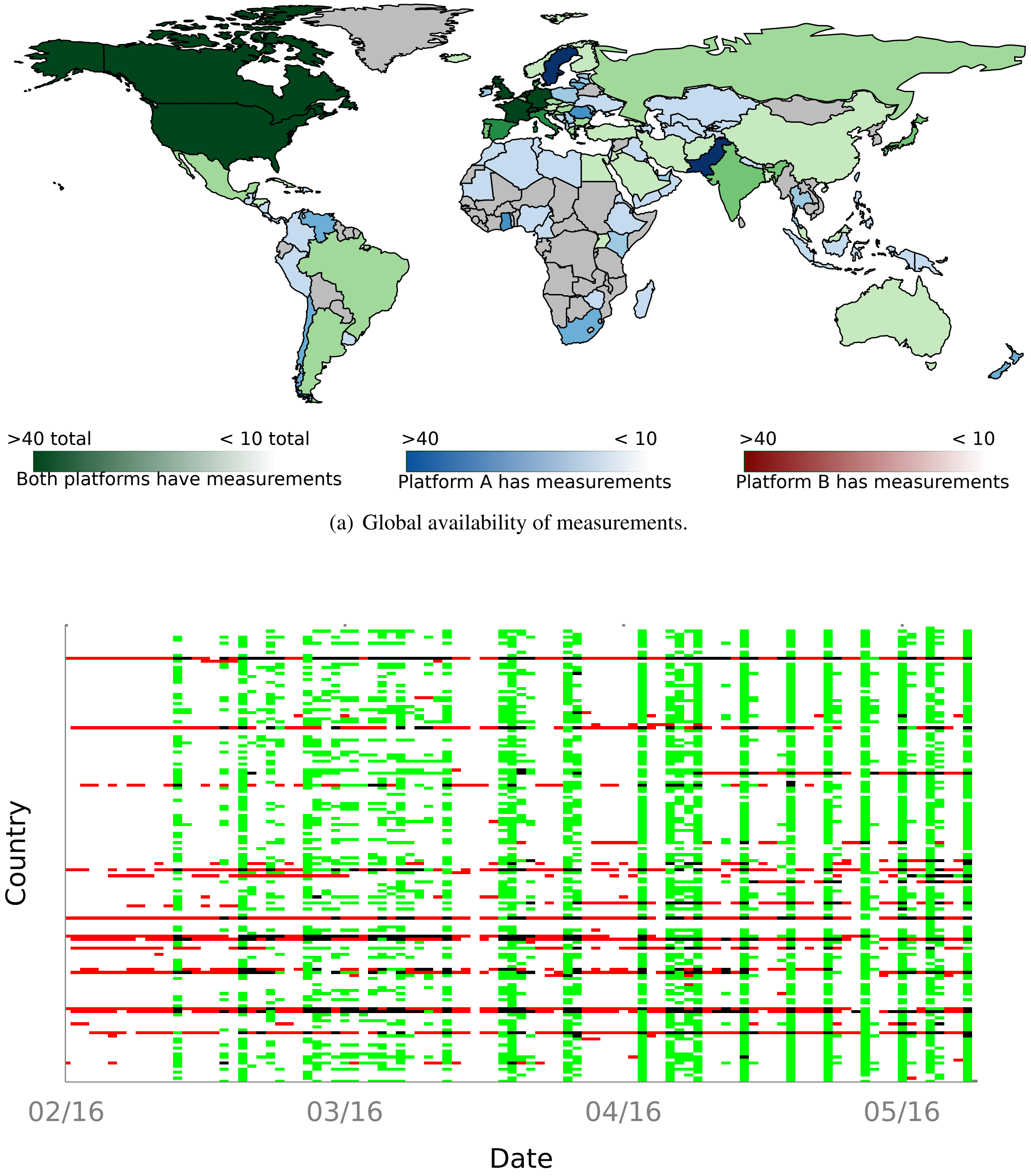
2.5 Conclusions

In this chapter we presented the fundamental design decisions faced by developers of large-scale longitudinal censorship measurement platforms -- where to obtain vantage points, how to use these vantage points to collect measurements, what measurements to collect, and how to analyze them. We then described the decisions that were made in development of the ICLab and OONI platform and their influence on the measurements obtained by these platforms.

In particular, we find that the ICLab platform is able to provide a more reliable and global picture of censorship by harnessing dedicated global VPN infrastructures in addition to on-the-ground volunteers. However, we also find that this dependence on VPNs can result in measurements being carried out on vantage points further away from residential networks which impacts the conclusions drawn from the platform. For example, the ICLab platform sees significantly less censorship than the OONI platform in India and Saudi Arabia. To this extent, it is important to work with representatives in affected countries, and the responsive nature of OONI has been successful in gaining support to measure several important political events. The challenge of obtaining representative, global, reliable, and response measurements remains a goal we continue to aspire to.

In addition, we showed how the results obtained from each of these platforms can be used to provide a deeper insight into understanding regional censorship at a global scale. By analyzing the types of censorship observed in several countries we were able to identify characteristics of the implemented censorship apparatus -- *i.e.,* results obtained by both platforms suggest the presence of a decentralized censorship infrastructure in India and a mostly centralized infrastructure in Iran and Saudi Arabia.

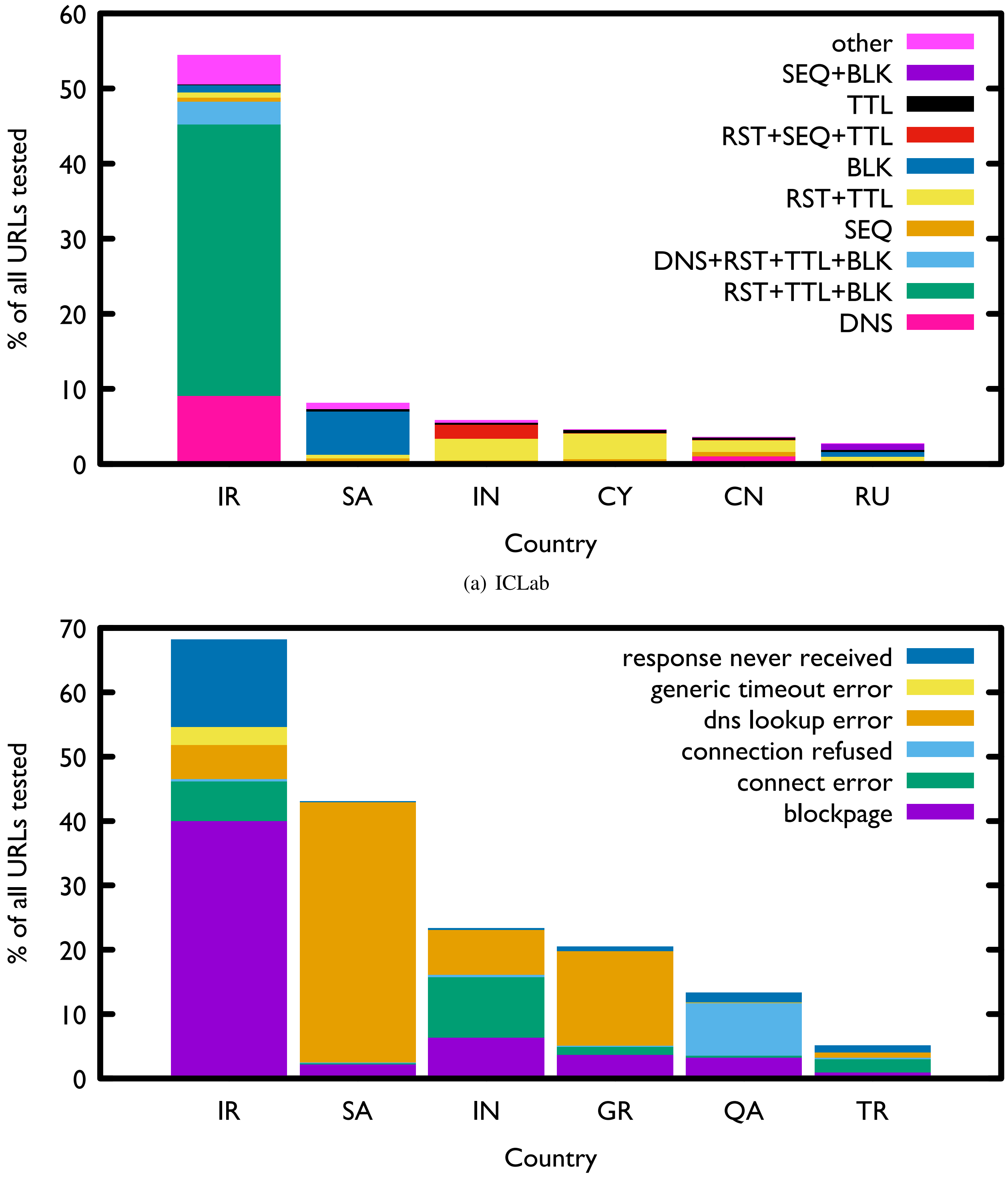
Finally, our current investigation also uncovers open challenges that remain in being able to distinguish censorship from anomalies that arise from phenomena such as misconfigured web servers, network outages, end-point discrimination, and unresponsive websites. Our platforms plan on addressing these limitations in future work.



(a) Global availability of measurements.

(b) Country-level temporal availability. Green and red indicate the availability of measurement data from ICLab and OONI, respectively. Black indicates the availability of measurements from both platforms from the same region on the given day.

Figure 2.1: Availability of measurements from the ICLab and OONI platforms in the last 100 days.



(a) ICLab

(b) OONI

Figure 2.2: The six most censored countries according to measurements from ICLab and OONI.

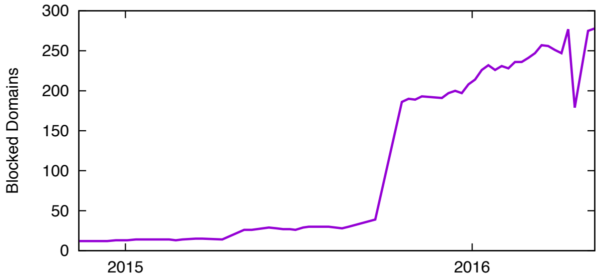


Figure 2.3: The number of domains seen blocked in Iran spiked in OONI measurement data in fall of 2015, several months before the election.

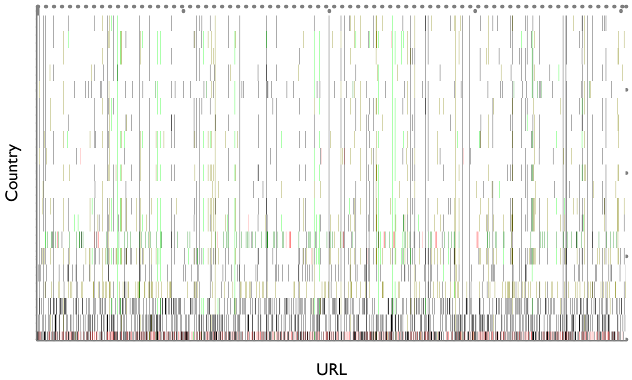


Figure 2.4: URLs censored in the 20 least free countries according to ICLab. Colors are indicative of the type of blocking observed (see Figure 2.2). The black vertical lines show websites that are either no longer available or have blocked access to all of the ICLab vantage points.

Chapter 3

References

[1] Barack Obama. Net neutrality: President Obama's plan for a free and open internet. http://www.whitehouse.gov/net-neutrality#section-read-the-presidents-statement.

[2] FCC. Order 10-201: Preserving the open internet. https://apps.fcc.gov/edocs_public/attachmatch/FCC-10-201A1_Rcd.pdf, December 2010.

[3] Sandvine - intelligent broadband networks. http://www.sandvine.com.

[4] Stacey Higginbotham. The Netflix--Comcast agreement isn't a network neutrality violation, but it is a problem. http://gigaom.com/2014/02/23/the-netflix-comcast-agreement-isnt-a-network-neutrality-violation-but-it-is-a-problem/, February 2014.

[5] Measurement Lab Consortium. Isp interconnection and its impact on consumer internet performance. http://www.measurementlab.net/blog/2014_interconnection_report, October 2014.

[6] Matthew Luckie, Amogh Dhamdhere, David Clark, Bradley Huffaker, and kc claffy. Challenges in inferring internet interdomain congestion. In *Proc. of IMC*, 2014.

[7] FCC. Protecting and promoting the open internet. https://www.federalregister.gov/articles/2015/04/13/2015-07841/protecting-and-promoting-the-open-internet, April 2015.

[8] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P. Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling end users to detect traffic differentiation. In *Proc. of USENIX NSDI*, 2010.

[9] Ying Zhang, Z. Morley Mao, and Ming Zhang. Detecting Traffic Differentiation in Backbone ISPs with NetPolice. In *Proc. of IMC*, 2009.

[10] Mukarram Bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. Detecting network neutrality violations with causal inference. In *CoNEXT*, 2009.

[11] Zhiyong Zhang, Ovidiu Mara, and Katerina Argyraki. Network neutrality inference. In *Proc. of ACM SIGCOMM*, 2014.

[12] P. Kanuparthy and C. Dovrolis. ShaperProbe: end-to-end detection of ISP traffic shaping using active methods. In *Proc. of IMC*, 2011.

[13] F. Sarkar. Prevention of bandwidth abuse of a communications system, January 9 2014. US Patent App. 14/025,213.

[14] Dan Rayburn. Cogent now admits they slowed down netflix's traffic, creating a fast lane & slow lane. `http://blog.streamingmedia.com/2014/11/cogent-now-admits-slowed-netflixs-traffic-creating-fast-lane-slow-lane.html`, November 2014.

[15] Jie Hui, Kevin Lau, Ankur Jain, Andreas Terzis, and Jeff Smith. How YouTube performance is improved in T-Mobile network. `http://velocityconf.com/velocity2014/public/schedule/detail/35350`.

[16] David Clark. Network neutrality: Words of power and 800-pound gorillas. *International Journal of Communication*, 2007.

[17] Peter Svensson. Comcast blocks some internet traffic. `http://www.washingtonpost.com/wp-dyn/content/article/2007/10/19/AR2007101900842.html`, October 2007.

[18] V. Bashko, N. Melnikov, A. Sehgal, and J. Schonwalder. Bonafide: A traffic shaping detection tool for mobile networks. In *IFIP/IEEE International Symposium on Integrated Network Management (IM2013)*, 2013.

[19] Ratul Mahajan, Ming Zhang, Lindsey Poole, and Vivek Pai. Uncovering performance differences among backbone ISPs with Netdiff. In *Proc. of USENIX NSDI*, 2008.

[20] Ashkan Nikravesh, Hongyi Yao, Shichang Xu, David R. Choffnes, and Zhuoqing Morley Mao. Mobilyzer: An open platform for controllable mobile network measurements. In *Proc. of MobiSys*, 2015.

[21] FCC announces ”Measuring Mobile America” program. http://www.fcc.gov/document/fcc-announces-measuring-mobile-america-program.

[22] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: Illuminating the edge network. In *Proc. of IMC*, 2010.

[23] N. Weaver, C. Kreibich, M. Dam, and V. Paxson. Here be Web proxies. In *Passive and Active Measurement (PAM)*, 2014.

[24] Nicholas Weaver, Robin Sommer, and Vern Paxson. Detecting forged TCP reset packets. In *Proc. of NDSS*, 2009.

[25] Switzerland network testing tool. https://www.eff.org/pages/switzerland-network-testing-tool.

[26] Neubot -- the network neutrality bot. http://www.neubot.org.

[27] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, Nicholas Weaver, and Vern Paxson. Beyond the radio: Illuminating the higher layers of mobile networks. In *Proc. of MobiSys*, 2015.

[28] F. Zarinni, A. Chakraborty, V. Sekar, S. Das, and P. Gill. A first look at performance in mobile virtual network operators. In *Proc. of IMC*, 2014.

[29] Ashwin Rao, Justine Sherry, Arnaud Legout, Walid Dabbout, Arvind Krishnamurthy, and David Choffnes. Meddle: Middleboxes for increased transparency and control of mobile traffic. In *Proc. of CoNEXT 2012 Student Workshop*, 2012.

[30] Weidong Cui, Vern Paxson, Nicholas Weaver, and Randy H. Katz. Protocol-independent adaptive replay of application dialog. In *Proc. of NDSS*, 2006.

[31] Arvind Norberg. uTorrent transport protocol. http://www.bittorrent.org/beps/bep_0029.html.

[32] Experimenting with QUIC. http://blog.chromium.org/2013/06/experimenting-with-quic.html.

[33] Gerhard Hasslinger and Oliver Hohlfeld. The Gilbert-Elliott model for packet loss in real time services on the Internet, 2008.

[34] Strongswan. www.strongswan.org.

[35] Xing Xu, Yurong Jiang, Tobias Flach, Ethan Katz-Bassett, David Choffnes, and Ramesh Govindan. Investigating transparent web proxies in cellular networks. In *Proc. PAM*, 2015.

[36] Jonathan Mayer. How verizon's advertising header works. http://webpolicy.org/2014/10/24/how-verizons-advertising-header-works/.

[37] Charles Reis, Steven D. Gribble, Tadayoshi Kohno, and Nicholas C. Weaver. Detecting In-Flight Page Changes with Web Tripwires. In *Proc. of USENIX NSDI*, 2008.

[38] Nicholas Weaver, Christian Kreibich, Martin Dam, and Vern Paxson. Here Be Web Proxies. In *Proc. PAM*, 2014.

[39] The Tor Project. Pluggable transports. https://archive.is/0vhLU.

[40] Abdelberi Chaabane et al. Censorship in the Wild: Analyzing Internet Filtering in Syria. In *IMC*, 2014.

[41] A. Sfakianakis, E. Athanasopoulos, and S. Ioannidis. Censmon: A Web censorship monitor. In *USENIX FOCI*, 2011.

[42] Hal Roberts, David Larochelle, Rob Faris, and John Palfrey. Mapping local Internet control. In *Computer Communications Workshop*. IEEE, 2011.

[43] Joss Wright, Tulio Souza, and Ian Brown. Fine-Grained Censorship Mapping: Information Sources, Legality and Ethics. In *FOCI*, 2011.

[44] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet censorship in Iran: A first look. In *Free and Open Communications on the Internet*. USENIX, 2013.

[45] Giuseppe Aceto, Alessio Botta, Antonio Pescapè, Nick Feamster, M. Faheem Awan, Tahir Ahmad, and Saad Qaisar. Monitoring Internet censorship with UBICA. In *Traffic Monitoring and Analysis*. Springer, 2015.

[46] EFF. The Great Firewall of Cameron: Why The UK's Filtering By Fiat Won't Work, and Won't Help, 2013.

[47] ZDNet. India blocks 32 websites, including GitHub, Internet Archive, Pastebin, Vimeo, 2014.

[48] Sheharbano Khattak, David Fifield, Sadia Afroz, Mobin Javed, Srikanth Sundaresan, Vern Paxson, Steven J Murdoch, and Damon McCoy. Do you see what i see? differential treatment of anonymous users. In *Network and Distributed System Security Symposium*, 2016.