IMPROVING CONTENT DELIVERY IN CELLULAR NETWORKS

by

Utkarsh Goel

A Proposal submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

March, 2016

ACKNOWLEDGEMENTS

First and foremost, I thank Professor Mike P. Wittie at Montana State University (MSU) for giving me the once-in-a-lifetime opportunity to pursue graduate studies in the United States of America. I have been fortunate to work under his direct supervision and to receive constant support from him in refining my research directions and to work on challenging problems.

I extend my acknowledgement to thank Dr. Moritz Steiner, Erik Nygren, Martin Flack, Stephen Ludin, and Dr. Ruomei Gao at Akamai Technologies for their gracious support in data collection on Akamai's global infrastructure, during my two internships in the Summers of 2015 and 2016.

I would also like to thank my collaborators from various academic institutions and industry without whose support much of my research may not have completed in time. The list of my collaborators include Ajay Kumar Miyyapuram at Cerner Corp., Kanika Shah at Yahoo!, Eben Howard at SEGA Games, Anish Babu Bharata at P2 Energy Solutions, Srinivas Prasad Gumdelli at IBM, Dr. KC Claffy at CAIDA, Dr. Andrew Lee at MintyBit, Dr. Lara Deek at Netflix, Professor Clemente Izureita, Professor Brendan Mumey, Professor Qing Yang, Professor Upulee Kanewala, and Professor Brittany T. Fasy at MSU.

Last, but not the least, I thank my friends Clint Cooper, Samuel Micka, Ahmad Yazdan Javaid, Syed Shabih Hasan, and Mohammed Abdul Qadeer for their continuous motivation and encouragement throughout my graduate school years.

Funding Acknowledgment

I thank National Science Foundation (NSF) to offer their generous financial support for working on several research problems through the grants NSF CNS-1527097 and NSF CNS-1555591. I also thank Western Transportation Institute (WTI) at MSU for partially supporting my graduate studies through NWP-OTIIS project funds.

Last, but not least, I thank several networking conferences, such as USENIX NSDI 2013, ACM SIGCOMM IMC 2013, ACM SIGCOMM IMC 2014, ACM CoNEXT 2014, IEEE ICCCN 2015, and academic institutions, such as University of Utah and Montana State University (MSU), for student travel grants on several occasions to attend conferences and meet some of the most influential researchers in the field of Computer Networking.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

ABSTRACT

The mobile ecosystem is growing faster than we have ever witnessed before. New technologies such as Internet of Things and Robots, collaborative gaming, interactive Artificial Intelligence systems, distributed computing, and remote equipment operations are changing the Internet originally envisioned several decades ago. The need to reliably exchange time critical information between automated machine or user devices has pushed our research community to expand the capabilities of the Internet. However, in spite of several years of research, the quest for (even) better performance remains. In this proposal, I argue for a shift in our current research directions from improving network layer technologies to improving application layer technologies. My conjecture in my PhD is that improvements to mobile Web performance can be realized if we move the network layer intelligence into the application layer, but without duplicating the core network functionality. To support this conjecture, I present several techniques that bring intelligence into application layer protocols, with the goal of (even) faster mobile Web. More specifically, my prior work in Internet measurement, customization of application layer protocols, and network infrastructure upgrades, motivates me to explore the further techniques to improve the capabilities of the Internet. I believe well-thought research directions for improving mobile Web performance will set another milestone in the growth and sustainability of the Internet to support new communication patterns for years to come.

1

INTRODUCTION

Motivation and Overview

The current mobile ecosystem is in the midst of an rapid growth. Internet Service Providers (ISPs) in developed countries inspire people to join the growing mobile population by introducing innovative technologies that offer higher quality of experience to their users than ever before. While in developing and under-developed countries, organizations such as Internet.org, Google, and others, encourage the still-unconnected population of the world to establish their online presence by offering free mobile data services [46, 48, 89]. But, as the Internet scales with more and more users joining the Internet, ISPs strive to accommodate all the users, and at the same time face challenges to ensure high quality of user experience. The networking research community has developed solutions to address challenging problems that ISPs face. For example, network infrastructure upgrades from IPv4 to IPv6 enable ISPs to accommodate far more connected devices than IPv4 could ever allow. However, despite several years of research to improve the user experience, user dissatisfaction with Internet speeds still remains a major challenge to be addressed.

Interactive applications that offer rich experience with Web browsing, video streaming, collaborative gaming, require low latency, high bandwidth, or both to provide high quality of user satisfaction. Such applications currently suffer from high Internet delays and fluctuations in the network connectivity. As a result, such applications experience constantly dropping adoption rates among users [75, 141, 165, 261]. Internet was not originally designed to support such low-latency and high-bandwidth demanding applications and therefore such applications often fail to perform reliably in networks with poor Internet connectivity. For example, ultra high definition 4K

and 3D on-demand videos, live video chat, cannot provide high quality of experience to their users in high latency and lossy environments, such as Wi-Fi and cellular radio. As a result, industry leaders and researchers at academic institutions have together developed numerous techniques to reduce the end-to-end latency and increase network bandwidth between users and servers.

Techniques to bring application content closer to users has shown promising results for reducing end-to-end latency [47, 215, 223], as such techniques allow Content Providers (CPs) to host their application data onto datacenters close to users, and in some cases, inside the user's home ISP. Such techniques are effective in reducing the latency as the content requested by the user is never fetched from outside the network, thus eliminating potential time spent on ISP-to-ISP peering and congested links. Other techniques such as selecting a closest server to a given user has further reduced end-to-end latency over the years [97, 151, 161, 272]. This is because such techniques allow Content Delivery Networks (CDNs) to load balance the load on their geographically distributed servers and redirect the user from already over loaded servers [143, 224]. Numerous other techniques allow ISPs, Web browsers, CDNs to efficiently cache the most popular content requested by their users [87, 117, 148, 322]. Such techniques are effective in reducing end-to-end latency as the content could be fetched from client's browser, from an in-network cache deployed by user's ISP, or from a CDN server hosted inside globally distributed datacenters.

Research community has also shown great interest in reducing the number of round trips required during a data transfer, as reducing round trips refers to reducing the overall latency to download an image, webpage, or a large movie file. TCP Fast Open is one of such techniques developed by Google that allows clients to send application data during the time of TCP connection establishement. [255]. Another technique from Google that also reduces round trip is called Quick UDP Internet

Communications (QUIC) [171]. QUIC reduces round trips by eliminating the need for TCP connection establishment if the two client and server has already connected in the past. Being inspired by these techniques, I developed OneTrip that reduces the number of round trips required in DNS resolutions [149]. My approach to reduce round trips is based on understanding how DNS servers deployed by ISPs behave when performing a lookup and thus adapt functionality of DNS Authorities accordingly, for faster DNS lookups (more details in Chapter 7).

Previous studies have shown that modifications to application layer are also effective in achieving in reducing end-to-end latency and improving the user experience. Techniques to compress application data, either by using popular compression encoding (such as WebP, GZIP, VP9, etc.), or by using compression proxies deployed worldwide by Google and Opera Software [51, 241]. Other techniques developed to reduce end-to-end latency from the application layer includes sending redundant DNS request and optimizing webpage structures, with the goal of faster webpage load time [306, 309].

As the above mentioned techniques improve quality of experience of experience for users and motivate them join the Internet userbase, ISPs often struggle to accommodate all of their users at the same time. As the Internet scales, the current Internet Protocol (IPv4) does not provide enough IP addresses that can be uniquely assigned to each user [113]. Therefore, new IP protocols (IPv6) have emerged and provided about 340 trillion trillion trillion IP addresses, a large enough IP address space to accommodate the current rate of growing users for hundreds of years to come [9, 160]. However, ISPs or organizations that still hold a significant number of unused IPv4 addresses show reluctance to transition to IPv6 because they do not anticipate a newar-future depletion of their IPv4 address space. My recent study on measuring the performance of IPv6 in cellular networks shows that IPv6 outperforms

legacy IPv4 in mobile Web [149, 273]. Thus, in my study I offer CPs and CDNs another reason to switch to IPv6 for improving the delivery of their application data.

Finally, large scale measurement of above mentioned techniques remains a major concern for reliable application performance. With this goal, the research and developer community has developed several measurement tools and services that allow measurement of application performance in production cellular networks [150]. Such tools, testbeds, and services allow for careful analysis of application traffic in different network environments.

Even with such a diverse collection of techniques to reduce end-to-end latency, the quest for even better performance still remains among researchers and application developers, as some of these techniques do not yet effectively utilize the capabilities of application layer protocols. Therefore, I argue for a shift in our current research directions from improving network layer technologies to improving application layer technologies. My conjecture in my PhD is that improvements to mobile Web performance can be realized simply by moving the network layer intelligence into the application layer, but without duplicating the core network functionality. For example, while routing is not a function of the application layer, but route choice by end hosts could improve application performance, akin to server selection open to users in many Web and online gaming applications [151, 165, 199, 271, 326].

Thesis Statement

The mobile ecosystem will continue to expand as the quest for innovative and interactive applications grows in the coming years. In order for these applications to sustain for a long time, current content delivery protocols need to be thoroughly evaluated under different networking environments. However, such protocols are currently tested in only in-lab or on a limited scale, which often results in poor user experience in untested scenarios. Mobile Web performance can be improved by careful measurement of different applications and network layer technologies, followed by suitable customizations in the content delivery protocols.

Proposal Organization

This proposal is organized into eight chapters that expose different aspects of Internet infrastructure and protocols to improve Web performance in cellular and wired networks. Figure 1.1 summarizes my previous and planned contributions in the areas of developing Internet measurement techniques, understanding Internet protocol behavior in different networks, customizing protocol, and improving application performance.

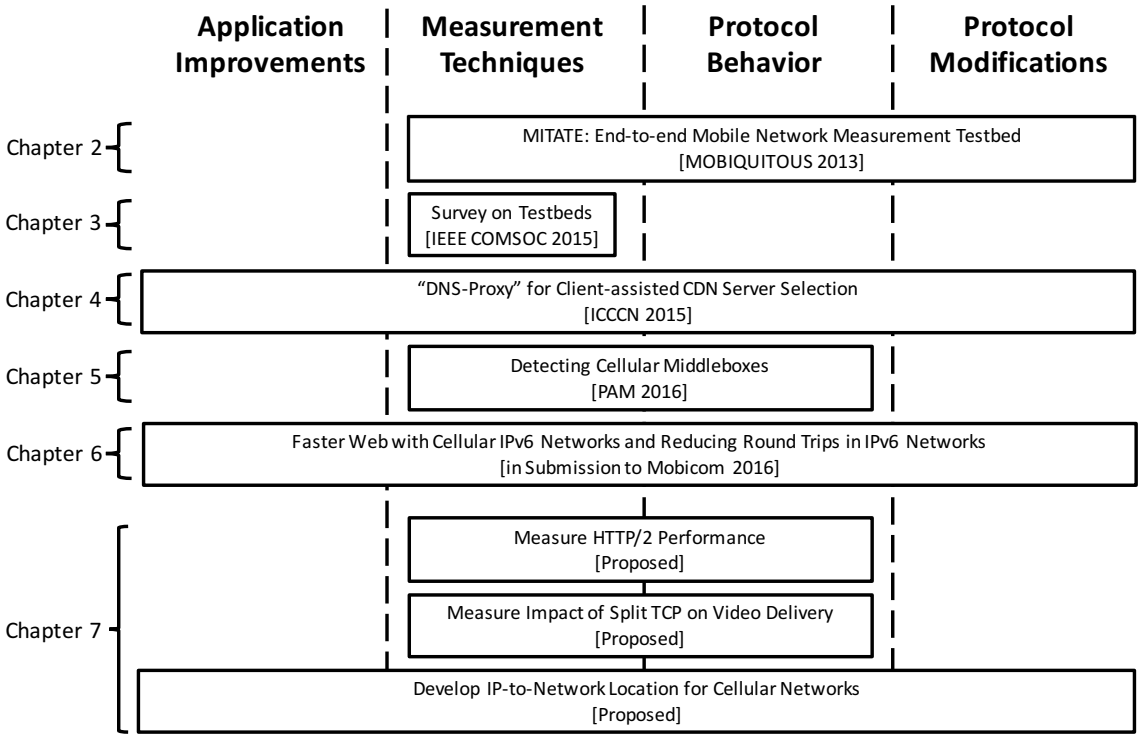


Figure 1.1: Proposal overview with individual projects categorized with respect to Internet measurement techniques, Protocol behavior, Protocol modifications, improving application performance, and by proposal chapter.

Chapter 2 presents my work on developing a mobile network measurement platform (known as MITATE) that allows application developers, researchers, and network operators to prototype their application traffic in production cellular networks. In my experience with MITATE, I identified that cellular ISPs in the US employ traffic shaping techniques to throttle the throughput of bandwidth-oriented network applications, such as Skype and BitTorrent.

Chapter 3 presents my work on surveying different mobile network measurement developed over years by developers and researchers from industries, academic institutions, and network operators. I also provide a comparative analysis of different measurement tools suggesting what tool might be more useful in measuring the performance of a given application.

Chapter 4 presents my work on improving CDN server selection techniques by developing intelligence into the DNS protocol. I developed a tool called DNS-proxy that allows clients to select fastest available servers based on the latency estimated by clients. I show that DNS-Proxy is effective in reducing the webpage load time, compared to using other techniques for server selection.

Chapter 5 presents my work on detecting cellular Web proxies using passive measurement techniques. I used Akamai's global infrastructure for content delivery to detect connection terminating proxies deployed by major cellular ISPs worldwide. I discovered that when Web proxies are present in a network, the CDN servers estimate low latency, whereas cellular clients estimate tens of milliseconds of latency. This work allows server operators to detect Web proxies using server logs and optimize TCP stack on their servers.

Chapter 6 presents my work on measuring the performance of IPv6 networks deployed by cellular networks in the US. My study shows that IPv6 networks outperforms the performance of legacy IPv4 networks. I recommend Content

Providers and CDNs to safely transition to IPv6 and host mobile content on IPv6-enabled servers for faster content delivery. During this study, I also developed OneTrip, a technique to reduce the number of round trips required during DNS lookup.

In Chapter 7, I propose three challenges related to improving mobile Web performance in cellular networks. First project aims to measure the performance of HTTP/2 protocol in cellular networks. The second project aims to measure the impact of connection splitting proxies on video streaming in cellular networks. And the third project aims to develop more accurate IP-to-network location services for CDNs and CPs to use. I then discuss potential technique that I seek to employ in order to address these challenges.

In Chapter 9, I draw conclusions from my prior research findings and expected impact of my proposed research towards improving mobile Web performance. And finally, in Chapter 10, I provide my anticipated timeline to complete the proposed work.

Contributions

The projects presented as part of this proposal make significant contribution towards improving Web performance in general by measuring the performance of content-delivery protocols and bringing intelligence into the application layer. Below I list individual contributions of my work categorized within my larger vision of developing end-to-end network measurement techniques, understanding Internet protocol behavior in different cellular and wired networks, customizing application layer protocol, and improving the application performance.

- **Application Improvements** In this category, I classify my work based on improvements in user experience related to Web performance on mobile and wired networks.
  - My work for developing DNS-Proxy shows significant improvements in webpage load time for five major content delivery networks, from different locations in the US [151].
  - My work towards developing PCP technique allows CDNs and application developers to more carefully measure the latency between Internet end-points [199].
  - My work towards developing OneTrip allows content providers and CDNs to eliminate round trips in DNS lookup and thereby save hundreds of milliseconds on page load time [149].
  - For the project related to developing IP-to-network location tool, **I propose to** show that network locations of cellular IP addresses could be more reliably identified by using cellular middleboxes as an approximate locations, when measured from widely distributed CDN servers.

- **End-to-end Network Measurement techniques** In this category, I classify my work based on new measurement techniques I developed.
  - I developed an end-to-end mobile network measurement testbed called MITATE (Mobile Internet Testbed for Application Traffic Experimentation) that allows prototyping of application traffic in cellular networks – a feature not currently available in any other existing cellular testbed [147].
  - I also conducted a survey on various mobile network measurement testbeds, tools, and services developed by researchers at industry and academia [150]. I brought awareness among different developer and research communities about different tools that they could use to accurately measure the performance of their application.
  - I developed a technique to measure the performance of different CDN servers returned in DNS responses from public, Open, and local DNS servers [151].
  - I developed a technique to estimate the end-to-end latency between any two arbitrary hosts on the Internet, by traversing latency to different CDN servers deployed by Akamai, Google, and Level 3 [199].
  - I developed three measurement techniques that server operators could use to passively detect the presence of TCP connection splitting proxies in the ISP networks [148]. These techniques are based on estimating client and server side latencies, measuring packet loss on servers, and identifying characteristics of TCP SYN packets received by servers.
  - I extended Akamai's Real user Monitoring System [41] to accurately extract Web performance metrics for mobile content hosted on IPv6-enabled content servers in US cellular networks [149].

  - For the project related to HTTP/2 performance, **I propose to** develop techniques to measure the performance of HTTP/2 protocol for cellular networks. My goal here would be to ensure that the measurement technique carefully isolates the possible interference between IP protocols and HTTP protocols.
  - For the project related to video streaming in cellular networks, **I propose to** develop techniques to first detect the presence of TCP terminating proxies for video content and then develop techniques to carefully measure the impact of such proxies on the quality of video streaming.
  - Finally, for the project related to developing IP-to-network location tool, **I propose to** develop techniques to estimate network locations of cellular clients (both IPv4 and IPv6), by using network distance between CDN servers and cellular middleboxes as an approximate of the cellular clients' locations.

.

- **Understanding Protocol Behavior** In this category, I classify my work based on using measurement techniques to detect the behavior of Application layer protocols and network middleboxes.
  - Using MITATE, I identified that cellular carriers in the US employ traffic differentiation techniques to throttle the throughput of bandwidth-hungry applications, such as Skype and BitTorrent [147].
  - I identified that DNS servers deployed by Google, Open DNS, and ISPs do not return same CDN server addresses for any given client

- I identified that middleboxes deployed by several cellular ISPs worldwide split TCP connections of HTTP traffic, likely with the goal of faster end-to-end connectivity [148]. I also identified that cellular carriers in France split both HTTP and HTTPS connections. I then identified that connection that are split by cellular middleboxes experience lower packet loss than connections not split by middleboxes. Finally, I identified that the TCP SYN characteristics of splitted connections are very similar, in terms of TCP timestamp, Maximum Segment Size, and Initial congestion window sizes.
- In a measurement study to evaluate the performance of IPv6 networks deployed by cellular carriers in the US, I identified that IPv6 is faster than the legacy IPv4 protocol being used for the last several decades in mobile Web [149].
- For the project related to HTTP/2 performance, **I propose to** identify performance of HTTP/2 protocol and investigate whether a transition to HTTP/2 improves Web performance for all types of websites.
- For the project related to video streaming in cellular networks, **I propose to** measure the impact of Web proxies on the quality of video streaming, in terms of video start up delay, number of buffering events, encoded bitrate, etc. across different cellular networks worldwide.
- Finally, for the project related to developing IP-to-network location tool, **I propose to** investigate on the stability of the results of my proposed technique.

- **Protocol Modifications** In this category, I classify my work based on modifications I introduced in different application protocols.

– My contribution towards developing MITATE allows developers and researchers to prototype their application traffic on production cellular networks [147]. Further, using MITATE, legacy application protocols can be modified and evaluated under different networking scenarios.

– I developed DNS-proxy to modify the legacy behavior of the DNS protocol to allow client-assisted CDN server selection, with the goal of improved Web performance [151].

– I developed OneTrip that reduces the number of round trips required during DNS lookup in IPv6-only cellular networks [149].

– For the project related to developing IP-to-network location tool, **I propose to** argue for a shift in current techniques to estimate geographic or network locations of cellular IP addresses.

14

MITATE - TESTBED FOR APPLICATION PROTOTYPING IN MOBILE NETWORKS

Abstract

This chapter introduces a Mobile Internet Testbed for Application Traffic Experimentation (MITATE). MITATE is the first programmable testbed to support the prototyping of application communications between mobiles and cloud datacenters. I describe novel solutions to device security and resource sharing behind MITATE. Finally, I show how MITATE can answer network performance questions crucial to mobile application design.

Introduction

Innovative mobile applications, such as multiplayer games and augmented reality, will require low message delay to provide a high quality of user experience (QoE) [99, 145]. Low message delay, in turn, depends on low network latency and high available bandwidth between mobile devices and cloud datacenters, on which application back-end logic is deployed. Unfortunately, mobile network performance can change rapidly [318]. Worse, traffic shaping mechanisms in cellular networks, such as as cap-and-throttle, traffic redundancy elimination, and deep packet inspection (DPI), can delay application messages without being reflected in standard metrics of network performance [177, 191, 333].

If innovation in the mobile space is to achieve broad adoption, new applications must deliver a high QoE across a range of network conditions. In other words, application communication protocols must be smart enough to adapt to changing network performance to keep message delay low. Such adaptations might include

changing packet size, or moving between server endpoints to deliver best traffic performance for a given client [318, 319].

To design and validate adaptive communication protocols developers need to prototype their implementations in production networks. The research community has produced several testbeds capable of application prototyping in the wired Internet [8, 10–12, 58, 74, 100, 102, 122, 194, 275, 284, 290]. To date, however, cellular network measurement platforms are not *programmable* in that they do not provide an foreign code execution environment [13,66,146,167,204,320]. Instead applications are evaluated in network simulators configured to reflect measurements of network performance [318]. While measurement-based simulation allows repeatable experiments, it misses the dynamic effects of competing traffic in cellular schedulers and of traffic shaping mechanisms.

The technical problem I address in this chapter is a lack of a programmable testbed for mobile application prototyping in production cellular networks. I have identified two challenges to building such a testbed. First, the personal nature of mobile devices creates user concerns over privacy, accountability for actions of foreign code being prototyped, and abuse of limited data plan and battery resources. Striking a balance between a flexible application prototyping environment and the safe execution of foreign code has been a difficult problem even in the more permissive wired environment [275,290]. Second, because mobile battery and data plan resources are limited, testbed participants need adequate incentives to share them. Difficulty in enlisting mobile users has limited measurement studies to small samples [146], high cost of testbeds based on dedicated hardware [71], and collection of only high level network performance metrics [204].

In this chapter I describe MITATE – a Mobile Internet Testbed for Application Traffic Experimentation made possible by novel solutions to the problems of security

and mobile resource sharing. MITATE is unique in that it allows programmable application traffic experiments between mobile hosts and back-end server infrastructure. MITATE provides strong client security by separating application code execution from traffic generation. MITATE also provides incentives and protections for mobile resource sharing through tit-for-tat mechanisms. MITATE's specialized traffic experiments can help developers answer questions crucial to mobile application design such as: "What is the largest game state update message that can be reliably delivered under 100 ms?," "Does my application traffic need to contend with traffic shaping mechanisms?," or "Which CDN provides fastest downloads through a particular mobile service provider's network peering points?"

The remainder of this chapter is organized as follows. 6 covers related research. In 2 I describe MITATE's architecture. 6 shows MITATE application prototyping capabilities. Finally, I conclude and present directions for future work in 5.

Related Work

The research community has produced several testbeds capable of application prototyping in the wired Internet [8, 10–12, 58, 74, 100, 102, 122, 194, 275, 284, 290]. To date, however, cellular network measurement platforms are not *programmable* in that they do not provide a foreign code execution environment [13, 66, 146, 167, 204, 320]. The result is a functionality gap: new applications are either evaluated on a small number of mobile devices, or in network simulators [144, 318]. While small scale studies capture real application performance, they miss variation across geographic areas, carriers, and devices. On the other hand, simulation studies configured to reflect aggregate measures of network performance miss the dynamic effects of traffic shaping and cellular schedulers [177, 191, 318, 333].

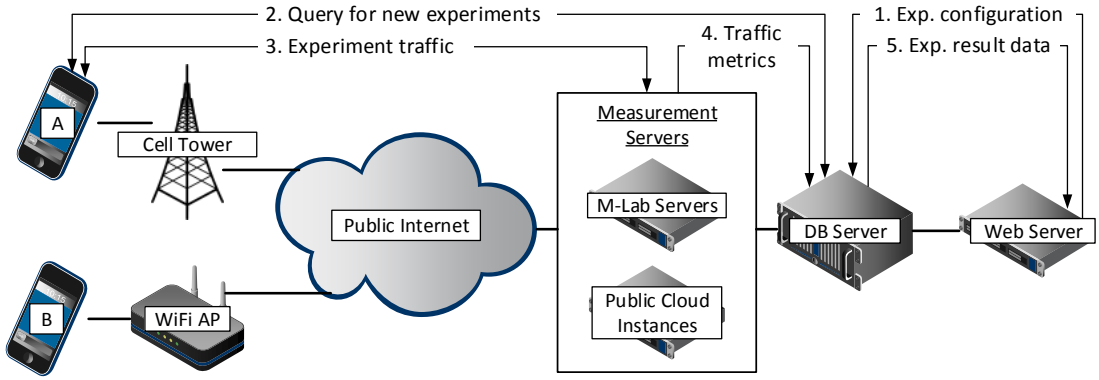


Figure 2.1: MITATE architecture and steps of a network traffic experiment.

Existing testbeds share some features with MITATE, such as criteria-based filtering of testbed devices [146], (limited) evaluation of application layer mechanisms such as HTTP and DNS [167], and an M-Lab[1] back-end [204]. Closest to our approach is Dasu, which provides a custom execution environment within an extension to a PC BitTorrent client [275]. SatelliteLab is also similar to MITATE in that prototyped application logic is not executed on edge devices [122].

One mobile testbed with programmable features is PhoneLab, which provides 200 participants with mobile phones and discounted data plans [71]. In exchange, participants agree to network experiments executed on their phones. However, PhoneLab relies on a custom OS, which limits its deployment to dedicated hardware, since installing an OS is a significant barrier to entry for most users [122].

MITATE

MITATE goes beyond current work and allows application prototyping on mobile devices in production cellular networks. MITATE offers the flexibility of Dasu and SatelliteLab, but without the security vulnerabilities of mobile code [122, 275]. To

[1] `http://measurementlab.org`

achieve wider adoption and easier access than the dedicated hardware model of PhoneLab, I adapt proven resource sharing incentives [71, 108]. In this section, I describe MITATE's architecture, application prototyping capabilities, and address the challenges of security and resource sharing on mobile devices.

Architecture and Traffic Experiments

To register a device with MITATE, a user downloads our mobile application and starts it as a background service with her login credentials, obtained by creating a MITATE account. Once her device is registered, a user can conduct traffic experiments, referring to 2.1, as follows: In Step 1, a user creates an experiment by uploading a configuration file, described in 2, via the Web interface. In Step 2, MITATE devices query the database for new experiments, whose criteria they meet. To reduce resource contention, as in SatelliteLab, I allow only one experiment at a time on a device [122]. If device A, for example, meets the geographic location and network type criteria of an experiment, A will begin, in Step 3, to transfer data defined by the experiment to the measurement servers. Experiment transfer traffic is timed at each endpoint (mobiles and measurement servers) and network performance metrics, together with metadata, are reported back to the database in Step 4. Finally in Step 5, a user may access the Web interface again to visualize, or download the experiment data collected by multiple devices. Based on the collected data, the user may refine her experiment and restart the process from Step 1.

Programmable Network Traffic Experiment Configuration

MITATE offers a flexible programming environment that supports evaluation and optimization of existing application traffic traces, as well as prototyping of adaptive application communication protocols. Existing network testbeds support

such flexibility through mobile code, whose potential security vulnerabilities result in designs based on dedicated testbed hardware [71, 102], or execution environments constrained by custom APIs [275, 290]. Neither solution is satisfactory. While the dedicated hardware limits adoption, custom APIs require application reimplementation in restricted, or non-standard programming environments.

I propose a secure and flexible network testbed design that eliminates the drawbacks of mobile code. MITATE experiments use multiple rounds of statically defined traffic transmissions. Processing between the rounds, i.e. mobile application logic, is implemented offline. Offline processing allows for the execution of unmodified application code inside an emulator[2] with message transmissions delegated to MITATE. Offline processing can also optimize communication protocol parameters, such as packet size, through binary parameter search, or a more powerful approach, such as CPLEX.[3] Finally, static experiment definitions allow static verification, which simplifies resource management (2) and testbed security design (2) and leads to a more accessible testbed.

Application Traffic Trace Experiments can help answer questions such as "What is the largest game state update message that can be reliably delivered under 100 ms?" An abbreviated MITATE experiment configuration XML file in 2.2 specifies two transfers, `t1` and `t2`. The transfers transmit the specified number of `bytes` between a MITATE mobile `client` and a datacenter server IP with MITATE backend logic.

The configuration file also specifies criteria definitions that `client` endpoints must meet before executing an experiment. In the 2.2 example, criteria `c1`, requires that a mobile be within 5000 m of geographic coordinates 45.666 -111.046

[2] `http://developer.android.com/tools/help/emulator.html`

[3] `www.ibm.com/software/commerce/optimization/cplex-optimizer/`

```
<experiment>
 <transfer>
  <id>t1</id>
  <src>client</src>
  <dst>54.243.176.74</dst>
  <prot>UDP</prot>
  <dstport>5060</dstport>
  <bytes>32</bytes>
 </transfer>
 <transfer>
  <id>t2</id>
  <src>54.243.176.74</src>
  <dst>client</dst>
  <prot>UDP</prot>
  <srcport>5060</srcport>
  <bytes>512</bytes>
 </transfer>
 <criteria>
  <id>c1</id>
  <latlong>"45.666 -111.046"<\latlong>
  <radius>5000<radius>
  <networktype>cellular</networktype>
  <starttime>12:00</starttime>
  <endtime>13:30</endtime>
 </criteria>
 <transaction count="10">
  <criteria>
   <criteriaid>c1</criteriaid>
  </criteria>
  <transfers>
   <transferid>t1</transferid>
   <transferid delay="40">t2</transferid>
   <transferid>t1</transferid>
  </transfers>
 </transaction>
</experiment>
```

Figure 2.2: MITATE XML configuration file.

(Bozeman, MT), be connected to a cellular network, and that device time be between noon and 1:30PM. MITATE will allow experimenters to specify a wide set of criteria, for example radio signal strength, location (eg. radius, bounding box, or set of ZIP codes), availability of GPS (indoor/outdoor), or device travel speed (for example over 55mph).

Finally, configuration files specify one, or more transactions that group criteria and transfers. In the 2.2 example, there is one transaction, which conceptually reflects a user request (transfer `t1`), game state update (transfer `t2`) after 40 ms of server processing `delay`, and an acknowledgement (transfer `t1`). This transaction will be executed by a mobile device if the device satisfies transaction criteria when polling MITATE servers, fewer than `count` devices have completed the transaction, and the user issuing the experiment has sufficient test data credit (see 2) to execute the entire transaction.

To find the largest game state update that can be delivered under 100 ms, multiple experiment rounds can perform binary parameter search, with MITATE reporting individual transfer and overall transaction delays. MITATE can also be

```
<transfer>
 <id>dns_req</id>
 <src>client</src>
 <dst>DNS</dst>
 <dstport>53</dstport>
 <prot>UDP</prot>
 <bytes><![CDATA[0x0100be07de55...]]></bytes>
 <response>1</response>
</transfer>
```

Figure 2.3: DNS query in MITATE.

used with sophisticated optimization tools, such as CPLEX, where performance of intermediate solutions are the reported metrics in each experiment round. Because MITATE traffic experiments use production networks they are not necessarily repeatable, and so decision metrics should be averaged over multiple trials. Finally, a `repeat` attribute can indicate that a transfer, or a transaction, should be executed multiple times. These `repeat` and `delay` attributes can be combined to configure periodic traffic, for example polling every 10 minutes for 24 hours.

Programmable Application Traffic Experiments can help answer questions such as "Which CDN provides fastest downloads through a particular mobile service provider's peering points?" To measure download times an experiment needs to issue a DNS lookup, followed by a download from the resolved server addresses. MITATE supports such experiments with two mechanism: explicit packet content and device-specific scheduling.

2.3 shows a configuration of transfer `dns_req` that represents a DNS lookup for a CDN server. The `bytes` tag contains the explicitly specified bytes of a well-formed DNS lookup request. When the `response` tag is set to `1`, the DNS reply packet will be included in the result data set, from which a user can parse out the resolved IP addresses.

To measure the download time of an image hosted on a particular CDN network, the user would configure a second experiment with a well-formed HTTP `GET` request to each resolved server IP. To make sure that each mobile device contacts only the IP addresses it resolved, each MITATE measurement contains the unique ID of the device that collected the result. That ID can be subsequently used as an endpoint address instead of the "`client`" keyword.

One downside of our approach is a potential for delay between each round of transmissions as experiments wait to be scheduled on mobile devices. I are working on integrating MITATE with the Android emulator to make the process of experiment configuration as easy as writing to a socket. Our integration will carefully modify emulator clocks, so that they advance only by measured transmission delay, excluding experiment scheduling delay. This mechanism will allow studies of adaptive communication mechanisms, such as server-host switching in online games, implemented in native application code running inside the emulation with only traffic transmissions being delegated to MITATE.

Deployment Incentives

One of the challenges faced by mobile network measurement platforms is how to assure sufficient resource capacity for scheduled experiments. The limiting resource is mobile data, subject to monthly caps.[4] To assure a supply of mobile bandwidth that matches the demand, a mobile testbed must, first, entice users to contribute resources and, second, protect contributed resources from abuse. MITATE jointly addresses both problems using a data *credit* exchange system inspired by BitTorrent tit-for-tat mechanisms [108].

[4]While battery power is also limited, it can be more easily replenished by charging.

The insight behind BitTorrent's tit-for-tat mechanisms is that they reward users for contributing bandwidth, as well as for merely being willing to do so. While in BitTorrent users make this assessment vis-a-vis each other, MITATE accounts for contribution and willingness to contribute with respect to the system as a whole. A MITATE user earns bandwidth credit for her experiments by allowing others' experiments to run on her device. A user is considered willing to contribute when her devices reliably ping MITATE servers for new experiments. The credit earned by the user, x_{earned}, is computed daily as:

$$x_{\text{earned}} = \alpha \times x_{\text{max}} \times \min\left(\frac{x_{\text{contributed}}}{x_{\text{max}}} + \frac{p_{\text{actual}}}{p_{\text{expected}}}, 1\right),$$

where x_{max} is the remaining amount of mobile data a user is willing to contribute during a monthly billing cycle divided by remaining number of days, $x_{\text{contributed}}$ is the volume of mobile data used by MITATE experiments on the user's data plan, p_{actual} is the number of pings reaching MITATE servers within 24 hours, and p_{expected} is the expected number of pings based on a system wide ping frequency setting. The parameter $\alpha < 1$ creates a mismatch between contributed resources and earned credit intended to ensure high experiment completion rates in areas with fewer participating devices, such as rural states. I recalculate user credit every 24 hours to prevent users from accumulating credit that, if used all at once, could deplete system resources on any given day. I expect that some participants will use MITATE sporadically and others on ongoing basis. Similar user participation takes place in BitTorrent, yet the system as a whole is able to maintain a sustained capacity [108].

Thus, MITATE credits users for contributed bandwidth, which allows them to use the bandwidth of others, keeping the two in a state of equilibrium. A final element of the mechanism to prevent resource abuse is that daily experiment bandwidth

requirements are computed at submission time, a process facilitated by the static XML experiment definition, and checked against submitting user's credit before being admitted to the system. I believe this approach is more predictable than resource caps enforced at run time that can lead to low experiment completion rates [290]. I also believe MITATE's credit based approach is simpler and more democratic than the delegated trust approach proposed in NIMI [245].

Security and Privacy

MITATE's goal of open-access necessitates a well thought out security design. With the contributed data plan resources protected by the incentive mechanisms, the security goals focus on protection of user privacy, the volunteered devices, and non-MITATE Internet resources.

Protecting User Privacy MITATE runs on personal mobile devices, which has the potential for violations of privacy if a device owner's activity and personally identifiable information were to become public. For example, user network and calling activity is not only private, but may itself contain personally identifiable information. Similarly GPS data becoming public can lead to legal challenges if traffic laws (speeding), or property laws (trespassing) were violated.

I have designed multiple levels of protection to preclude violations of user privacy. First, MITATE can only be used for active traffic experiments and cannot monitor non-MITATE traffic on a device. Second, while MITATE does collect GPS and accelerometer readings as metadata to accompany network performance metrics, users are asked to opt-in before starting the MITATE mobile app. Finally, third, I separate all data collected on devices from personally identifiable user account information. Each device registered with MITATE receives two random IDs: one to label traffic

metrics collected on the device, the other to keep track of credit data earned by the device for its owner. The dual ID system means that collected experiment data are never linked to a device owner's identifiable information.

Protecting User Devices Users who volunteer their devices for MITATE agree to cede some control over them. It is imperative that MITATE limit other user's actions on volunteered devices to within the bounds of that agreement. MITATE protects user devices with three mechanisms.

First, a user can set usage limits for mobile data, WiFi data, and battery level on their devices. These limits are consulted during experiment scheduling to disallow experiments that exceed remaining device resource allowance. Second, users never directly interact with others' devices. To submit an experiment, or download data, users authenticate and communicate with MITATE servers over encrypted connections. Mobile devices download experiments and upload collected metrics to MITATE servers also using encryption. Finally, third, our XML experiment configuration is static in that it does not allow conditional, nor jump statements. Such static definitions enforce the separation between the on-device functionality of data transmission and off-device processing. This separation allows for static checking of XML configurations using mature schema verification tools, which is simpler than dynamic code analysis and more lightweight than mobile code sandboxing. Static experiment definition also allows for the volume of each transfer in the XML file to be added up and compared against user credit and device resource limits.

Protecting non-MITATE Resources Our final goal is to protect non-MITATE resources, for example from DDoS attacks configured as MITATE experiments. ScriptRoute, designed from the ground up as a secure Internet measurement

system, considers two types of malicious experiments: *magic* packets and traffic amplification [290]. *Magic* packets can disrupt legitimate traffic, for example, when a spoofed `FIN` packet closes a TCP connection. Because MITATE allows experiments with explicitly defined packet content, I will make sure that these packets do not pose threats to other systems by matching them against signatures of known exploits using intrusion detection mechanisms.

Traffic amplification takes place when a malicious user leverages testbed nodes to monopolize the resources of a legitimate service, for example through a Smurf attack. Existing testbeds limit traffic amplification by placing a rate limit on the volume of data that can be generated by an experiment, which also constrains legitimate load testing. Instead, MITATE limits the total volume of experiment data to a user's earned credit. Although a MITATE user may request that multiple devices send data simultaneously, the user's credit will be rapidly depleted, and so even if the transmissions are malicious, they will be short-lived.

MITATE Application Traffic Prototyping Capability

To demonstrate MITATE's traffic emulation capabilities I present a set of network experiments and collected data. I show that MITATE can elicit various network performance phenomena useful to developers in answering a wide range questions about application traffic performance. The collected data includes traffic performance metrics and associated metadata. Prior to sending experiment traffic, MITATE calculates the clock offset between the mobile and measurement servers, which allows us to time unidirectional (unacknowledged) UDP transfers [200]. Experiments were performed on several Android phones and two different cellular

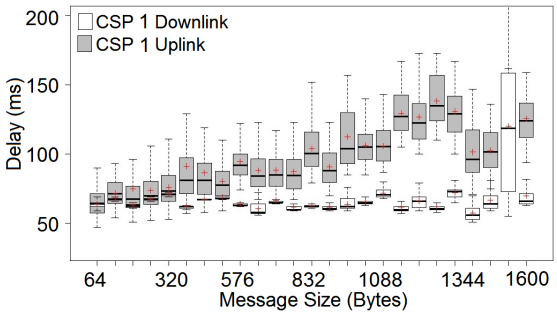


Figure 2.4: Message delay vs. message size at 10 AM on CSP 1 to a CA datacenter.

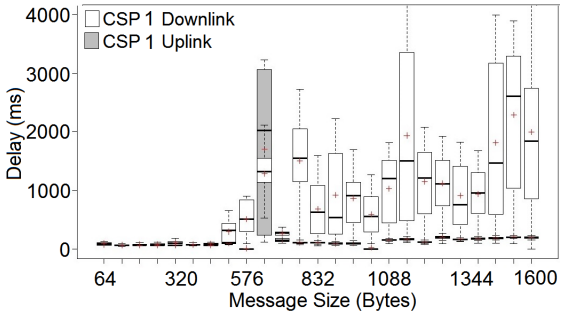


Figure 2.5: Message delay vs. message size at 2 PM on CSP 1 to a CA datacenter.

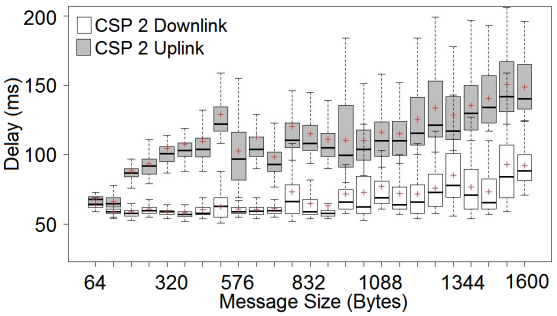


Figure 2.6: Message delay vs. message size at 10 AM on CSP 2 to a CA datacenter.

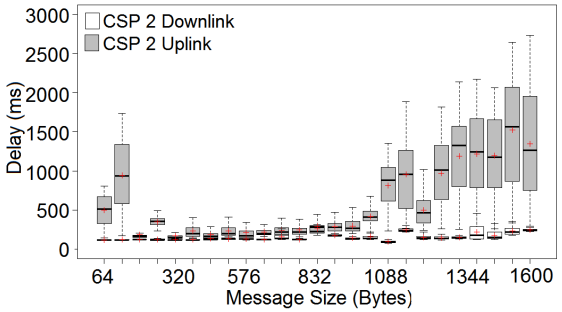


Figure 2.7: Message delay vs. message size at 10 AM on CSP 2 to a VA datacenter.

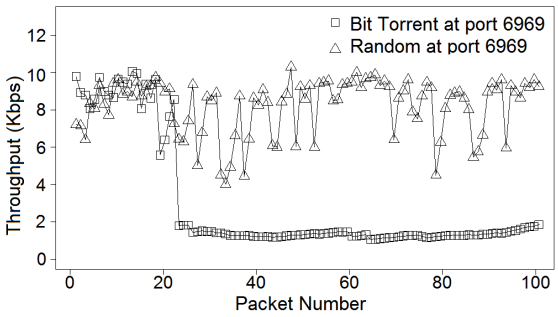


Figure 2.8: Per packet throughput of BitTorrent and random payloads on CSP 1.

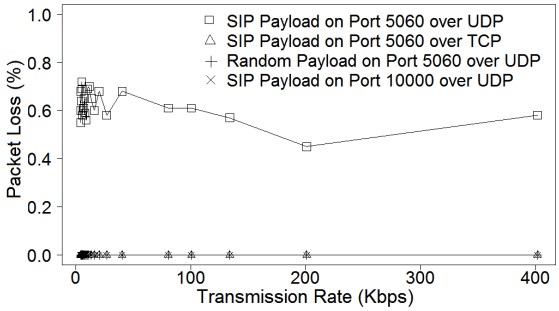


Figure 2.9: Packet loss of SIP and random payloads vs. flow data rate on CSP 1.

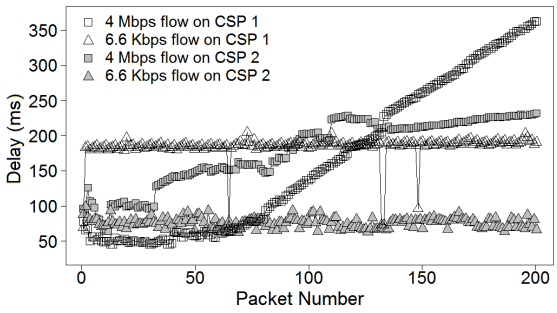


Figure 2.10: Delay of different data rate flows vs. on CSP 1 and CSP 2.

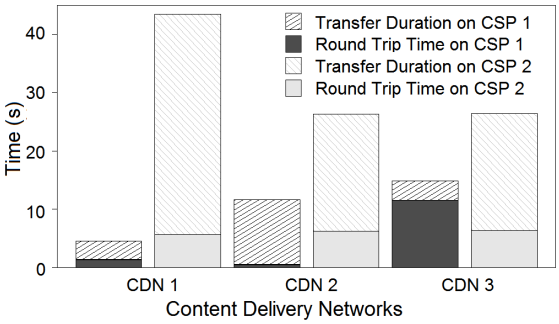


Figure 2.11: Round trip time and transfer time of 3 MB image from three CDNs.

service providers (CSP) networks in Bozeman, MT, and over connections to two different cloud datacenters. I anonymize the identities of CSPs and CDNs.

Effect of Packet Size on Message Delay

In gaming applications game state updates need to be delivered while their content is relevant. And so, game developers may want to know: "What is the largest game state update message that can be reliably delivered under 100 ms?" To answer that question I configure a MITATE experiment with transfers of increasing size (`bytes`). I plot the results in Figures 2.4–2.7, which show message delay as a function of message size during different times of day.

Our results show that message delay increases with message size and does so more rapidly on the uplink, likely due to asymmetric network provisioning. I also observe in 2.5 a high delay for larger messages on the downlink, likely due to mid-day network congestion. 2.6 shows a higher sensitivity of message delay to size on CSP 2. That effect is especially pronounced on connections to a datacenter located in Virginia, shown in 2.7.

From these experiments a developer might conclude that a message of 320 B can be delivered under 100 ms with high confidence to customers in Bozeman, MT on CSP 1, but a smaller message might be needed on CSP 2. Also, to keep message delay low, requests from Bozeman should not be directed to the Virginia datacenter.

Effect of Traffic Shaping

The degree to which FCC net neutrality rules apply to CSPs continues to be debated [57]. And so, application developers may want to ask: "Does my application traffic need to contend with CSP traffic shaping mechanisms?" To answer that question I configure a series of MITATE experiments, in which transfers of

specific content, on specific ports, and at different rates are used to detect traffic shaping [123, 177].

2.8 shows downlink throughput on CSP 1 of consecutive BitTorrent and random payloads transmitted over UDP on tracker port 6969. Our results show a drop in throughput for well-formed BitTorrent packets relative to random content, which likely indicates the presence of DPI mechanisms. I did not detect similar throughput drops on CSP 2. These results show that embedding of explicit packet payloads allows MITATE to detect content based traffic shaping.

2.9 shows downlink percent packet loss on CSP 1 of 1000 SIP packets transmitted on port 5060 over UDP and TCP versus transmission rate. Our results show that while SIP packets over TCP are undisturbed, same packets over UDP experience close to 60% loss rate. Because loss remains nearly constant across transmission rates, I believe that SIP packet loss over UDP is due to traffic policing, rather than traffic shaping.

2.10 shows per packet delay of uplink UDP flows transmitted at 4 Mbps and 6.6 Kbps on CSP 1 and CSP 2 versus packet number. The 4 Mbps flows experience an increase in delay, likely from queueing that results from the mismatch between sending and token bucket service rate limits [177]. The 6.6 Kbps flows, on the other hand, are sent below the service rate and avoid self-induced congestion. Testing different transmission rates allows developers to determine the maximum sending rate that will fall below token generation rate and avoid queuing delays. The experiments are useful for configuration of adaptive video stream encoding.

Measurement Based CDN Selection

Finally, dynamic content applications customize content for each user and have the opportunity to adapt to user's network conditions, for example, by embedding

links to static content in different CDNs. And so, application developers may want to ask: "Which CDN provides fastest downloads through a particular mobile service provider's network peering points?" To answer that question I configure a MITATE experiment that sends a well-formed HTTP `GET` requests, configured in the `bytes` tag, for an image hosted in three different CDNs.

2.11 shows the CDN response time for the first bit, or round trip time (RTT), and last bit, or transfer duration, of a 3 MB image delivered over the two CSP networks. Our results show a lower last bit delay for requests in CSP 1, but a higher RTT variation between CDNs, likely due to different CSP peering points that lead to CDN servers. From these experiments a developer might conclude that for users in Bozeman, MT CDN 2 provides the best combination of performance across the two CSP networks.

Discussion and Future Work

In this chapter I described MITATE, the first public testbed that supports prototyping of application communications between mobiles and cloud datacenters. MITATE separates application logic from traffic generation, which simplifies security and resource sharing mechanisms. I have presented data collected with MITATE experiments that demonstrates the system's capability in eliciting effects of cellular network performance on mobile application message delay.

Future work on the project involves deploying the current implementation onto M-Lab servers. In the meantime, I invite the community to use publicly available MITATE code[5] in private deployments. I also welcome community participation in evolving MITATE functionality in the areas of resource sharing models, GPS

[5] `http://github.com/msu-netlab/MITATE`

and accelerometer data anonymization, data visualization, and tools based on the MITATE platform.

SURVEY ON MOBILE NETWORK MEASUREMENT TOOLS

Abstract

Mobile (cellular) networks enable innovation, but can also stifle it and lead to user frustration when network performance falls below expectations. As mobile networks become the predominant method of Internet access, developer, research, network operator, and regulatory communities have taken an increased interest in measuring end-to-end mobile network performance to, among other goals, minimize negative impact on application responsiveness. In this survey I examine current approaches to end-to-end mobile network performance measurement, diagnosis, and application prototyping. I compare available tools and their shortcomings with respect to the needs of developers, researchers, network operators, and regulators. I intend for this survey to provide a comprehensive view of currently active efforts and some auspicious directions for future work in mobile network measurement and mobile application performance evaluation.

Introduction

Mobile (cellular) network applications deliver interactive services, generally supported by back-end logic deployed on cloud infrastructure. These applications support a wide breadth of functionality, such as live video, social gaming, communication services, and augmented reality [26, 107, 195, 265]. Future services will increasingly leverage cloud-based datasets and processing power for innovative applications of live speech translation, real-time video analysis, or other computationally intensive tasks [264, 281]. As the frequency of interactions between mobile devices and back-

end servers increases, application responsiveness will be increasingly tightly coupled with end-to-end network performance.

To innovate in the interactive mobile application space and provide a richer user experience, applications employ communication protocols with sophisticated data delivery optimizations that support responsive communications under a range of network conditions [52,88,128,158,329]. However, the deployment and configurations of these optimizations require detailed network performance data that is not readily available, which results in challenges across the cellular ecosystem. For example, **developers** evaluate performance of their mobile applications in device farms that lack geographic diversity and device mobility [20, 53, 56].

Researchers lack network performance data, or tools to acquire such data, in order to rapidly test hypotheses and focus on realistic network performance problems. **Network operators** need to monitor and troubleshoot end-to-end network performance without degrading base station throughput. Finally, **regulators** have a limited view of network performance, especially with respect to traffic shaping by network providers, impeding their ability to tackle performance challenges and roadblocks for sustained innovation in the mobile space [175, 259].

This chapter provides a comparative analysis of currently available network measurement platforms for end-to-end mobile network measurement, monitoring, and experimentation. (For a survey of wireline Internet measurement platforms see recent work by Bajpai and Schonwalder [68].) I further categorize measurement platforms as research **testbeds** for network experimentation, extensible distributed measurement **tools**, and **services** for widespread monitoring of networks performance. In the following sections describe the most salient features of each platform, and how some features differ across them. Table 3.1 compares the testbeds, tools, and services in

terms of their experimentation flexibility, device selection criteria, resource protection, and other features.

Based on our review of current measurement efforts, I observe that although existing approaches comprise only a patchwork of needed functionality, they already generate powerful insights to guide development, research, and regulatory actions. However, in spite of the relative maturity of several measurement platforms, daunting problems remain including support for wide-scale application prototyping and deployment, detection of traffic shaping, and long-term network performance monitoring. Most existing mobile measurement platforms have been developed in isolation, and one motivation for this survey is to foster more concerted and cooperative efforts at standardization of measurement libraries, privacy policies, and technology exchange [27, 105, 166, 328].

The rest of this chapter is organized as follows. Section 3 reviews goals of end-to-end mobile network measurement. Sections 3, 3, and 3 respectively discuss testbeds, tools, and services for end-to-end mobile network measurement. Section 5 presents directions for future work and concluding thoughts.

Goals of end-to-end mobile network measurement

Developers, researchers, network operators, and regulators share the broad goal improving the performance of mobile network communications. However, their perspectives and goals differ, and so the tools developed by these communities focus on different aspects of network performance.

Developers' View of Network Performance

Developers try to provide a responsive application experience users enjoy. Although much of the delay experienced by user requests is due to back-end processing and front-end rendering [128], as hardware and software processing speed improves, network performance becomes a dominant concern [158]. However, mobile network performance varies across providers, devices, geography, and time of day, and is generally hard to predict [111, 318]. To minimize the effects of network performance variation on application responsiveness, developers optimize content delivery by data batching [327], adaptive encoding [158], lowering communication frequency [196, 241], or strategically deploying application infrastructure near users [95, 174]. To understand which optimizations to apply and how to configure them, developers monitor application communication performance and evaluate what-if scenarios.

Application monitoring assesses the impact of edge network performance on application responsiveness through server-side monitoring [187], or with measurement and reporting code embedded within the application [96, 111, 153]. To evaluate what-if scenarios, developers use A/B testing of two versions of an application. Such studies generally target certain users, networks, or time intervals, and thus require expressive test device selection criteria. Although A/B testing may be implemented in the application itself, third party application libraries offer easier integration and a safer starting point [32, 40, 42]. Developers may also test changes without affecting users by deploying their application on test device farms [20, 53, 56]. However, these services currently provide access only to stationary cellular devices, which limits measurement realism in terms of geographic and network diversity. Academic testbeds and measurement libraries have the potential for greater reach and realism, but have only seen limited industry interest and involvement. Though perhaps the proliferation of network testbeds and tools presented in this chapter suggests their growing popularity.

Researchers' View of Network Performance

The research community has produced several testbeds that offer significant flexibility to execute a variety of network experiments [23, 70, 104, 132, 147, 183, 198, 203, 206, 212, 291, 293, 324, 332]. Yet, the availability of these testbeds and knowledge of how to use them often remains limited by practical barriers to collaboration across research groups. Researchers may need to set up their own infrastructure for data collection [275], obtain Institutional Review Board (IRB) approvals [212], or revive code that is no longer maintained [161,202]. Even when maintainers of a given testbed help to set up experiments, communication rounds take time, especially when software modifications are needed. As a result, researchers often decide it is more expedient to develop new tools, even when it duplicates others' efforts and achieves only a small scale evaluation [70, 212].

Several organizations are working to lower the barrier to entry and promote concerted development of network measurement tools. For example, M-Lab maintains a repository of measurement tools, including MobiPerf, WindRider, and NDT (Mobile client), discussed in sections 3, 3, 3 respectively [19]. One of M-Lab's goals is for new tools to leverage existing code base, for example the Mobilizer library [324]. M-Lab also supports the development of common ethical guidelines for network measurement data collection [328]. However, the continued flow of proposals for new, independently deployed cellular tools (five in 2014 [23, 70, 198, 269, 324], eight in 2013 [132, 135, 147, 203, 212, 239, 291, 332], three in 2012 [53, 104, 258], one in 2011 [206], one in 2010 [183], and two in 2009 [225,315]) suggests that more needs to be done to improve collaboration among different research groups.

The research community has also worked to decrease the need for and the cost of redundant experimentation and created several repositories of wireless network

measurement data [3, 33, 35]. While data repositories facilitate reproducibility of research results, they have their limitations. For example, to study current phenomena, such as changes in network traffic management policies expected after new FCC Net Neutrality regulations [83], researchers need new measurement data quickly, rather than waiting for a new dataset to be released after another group's publication. Additionally, data in repositories may be obfuscated, suitable for one experiment, but lacking in sufficient detail for another, or may be difficult to correlate when multiple datasets are collected at different times or under different conditions. For these reasons, live testbeds and measurement tools form a critical foundation of innovative research and education environments.

Network Operators' View of Network Performance

In addition to their operational monitoring of cellular network performance from base stations and other network elements, network operators are also interested in end-to-end network measurement from the device's perspective to provide responsive and reliable service at reasonable operating cost, including the cost of fielding customer support calls. Network operators also want to simplify and speed up the deployment of new access technologies and over-the-top services. A key element in these processes is the ability to troubleshoot network performance issues without affecting base station throughput.

However, industry insiders describe troubleshooting cellular networks as "an art with few scientific principles." To increase their insight into end-to-end network performance and network factors that may affect it, e.g, received signal strength, many network operators have deployed Carrier IQ on handsets in their networks [94, 301], and then faced customer backlash [6, 246] due to this application's approach (or lack thereof) to user privacy protection. Although network operators continue

to use Carrier IQ, users continue to uninstall it on rooted phones [190, 287]. As a result network operators, like ATT, are looking for new methods to monitor and troubleshoot user network performance that can match the scale and efficiency of embedded end-host monitoring provided by Carrier IQ [155].

Regulators' View of Network Performance

Finally, regulators need monitoring tools to inform their understanding of availability, reliability, and performance of mobile networks over time. Constrained network performance and delayed upgrades to next generation technologies, e.g., 4G, have long been seen as stifling innovation in the US [173, 242]. Further, traffic shaping mechanisms and anti-competitive behavior by some network providers impede deployment of new services [50, 59, 84, 85, 133, 178, 192, 259, 285]. Even developers of popular measurement tools struggle to create incentives for longitudinal and widespread measurement [135]. A few tools that have gained traction with users rely on user-initiated network tests, which limits measurement frequency and representativeness [183, 206].

Shared Challenges

Developers, researchers, network operators, and regulators face the same challenges in deploying end-to-end mobile measurement tools: incentivizing a statistically significant sample of users to install and run the tool; protecting those users' resources from abuse; and preserving user privacy.

To motivate user participation, measurement platform designers have used schemes such as bundling measurement code with other functionality [275], offering free devices [212], press coverage [135, 183], or simply appealing to user altruism and

curiosity [135]. These approaches result in either a narrowly focused user base or short-lived deployments, both of which limit platform utility.

The second challenge is how to protect contributed platform resources from abuse. Some peer-to-peer systems have used tit-for-tat mechanisms to ensure fair resource sharing [108], but mobile network measurement platforms thus far rely on user altruism on the one hand and conscientiousness on the other [104, 135, 206, 293]. Scaling and sustaining measurement platforms over the long term will require more rigorous resource protection methods in existing tools.

Finally, a measurement platform should isolate personally identifiable information from experimental data collected on a mobile device. Measurement platforms discussed in this chapter offer a range of solutions to maintain this separation. Google has supported the development of a proposed set of ethical guidelines for the design of mobile-based network measurement tools [328]. These guidelines have informed the design of some tools, specifically MITATE and Mobiperf, but the disparate legal frameworks for user privacy around the world make it difficult to create conformant tools for the global mobile Internet [105].

Network Testbeds

Mobile application developers need to know how well a network can deliver their application content. Custom network experiments that emulate communication protocols of their applications create performance profiles in different network settings to inform application design. End-to-end systems that support such functionality need to balance the flexibility of their feature set against potential abuse of contributed user resources and threats to user privacy. I divide systems according

to how they resolve this conflict for new experiments from external researchers into uncurated and curated approaches.

Uncurated Network Testbeds

Uncurated network testbeds allow users immediate access upon registration. Users experiments and changes to these experiments do not need to go through an approval process. Although their open nature allows these platforms to scale, they are limited in the type of personal information they collect without going through an Institutional Review Board (IRB) approval process.

MITATE Mobile Internet Testbed for Application Traffic Experimentation (MITATE), developed at Montana State University (MSU) in April 2013, enables experimentation with mobile application traffic in live mobile networks [147]. Experiments execute on user-volunteered devices that meet specified criteria, such as signal strength, geographic location, or network provider. Developers can use MITATE to evaluate the performance of mobile application communications under a wide range of conditions before their applications are deployed, or even fully developed. MITATE supports configurable active network measurements to detect network traffic shaping by ISPs, and integration with other tools, for example CPLEX to explore protocol configuration tradeoffs through parameter search and optimization [168].

Functionality: MITATE supports active network measurements on mobile devices. MITATE experiments are configured through XML files that describe the content of experiment data transfers, transport layer protocols, network endpoints, and timing. An XML configuration also describes criteria that volunteered devices must meet to execute an experiment, such as network type (cellular or Wi-Fi), signal strength,

geographic location, network carrier, minimum battery power, and device model. To ensure that experiments are defined correctly, MITATE servers validate new XML configuration files against an XML schema definition (XSD). Users interact with MITATE through an API that allows upload of XML configuration files and download of collected data.

Each mobile device polls a central MITATE server at MSU for new experiments whose criteria matches that device's capabilities. Devices download static traffic definitions that specify what traffic to exchange between the mobile device and back-end servers. MITATE mobile devices can interact with third party systems, for example DNS and CDN servers, through explicitly configured, well-formed request packets, and by recording reply content and delay. Although each MITATE experiment is a series of static transmissions, complex logic can be implemented across processing rounds, e.g., DNS lookups and ping transactions require two rounds. Such an experiment specifies a device ID as a criteria, which allows for the same device to issue DNS lookups in round one and subsequent pings in round two.

Data Collection: MITATE records the delay of each data transfer as well as metadata such as signal strength, accelerometer readings, and device location. This delay measurement allows calculation of 42 metrics, including uplink and downlink latency, throughput, jitter, and loss, as well as mobile sensor readings [201]. For example, an experiment estimates available bandwidth by dividing the size of a large transfer by its duration. MITATE experiments may also use a series of small transfers to estimate packet round trip time (RTT), loss, and jitter. At the start of an experiment, MITATE estimates the clock offsets between a device and each server endpoint, which allows separate measurement of uplink and downlink latency.

Collected data is available for download in the form of SQL insert statements to populate a local instance of a MySQL database for each user. MITATE allows users to download data only for their own experiments and those whose data is made public. Aggregate metrics, for example mean latency, are computed through queries to the local database instance. This design reduces the load on the MITATE database servers and allows users to run arbitrary queries over their experiment data.

Resource Incentives and Protection: MITATE is a collaborative framework built around incentives for user participation, inspired by BitTorrent's tit-for-tat mechanism [108]. MITATE users earn data *credit* by contributing their mobile resources. Users can then spend credit to run experiments on others' devices. Earned credit expires after 24 hours to prevent its accumulation and use for large experiments that might overwhelm available system-wide resources at any point in time.

MITATE's credit system encourages ongoing participation and protects contributed resources from abuse. Users can leverage MITATE resources in direct proportion to how much data they contribute to the system. MITATE does not rate-limit device transmissions (although users can set monthly data caps and battery limits on their devices), which permits realistic load-testing experiments. Although distributed denial of service (DDoS) attacks launched from multiple devices are technically possible in MITATE, they are destined to be short lived, because rapid transmissions from multiple devices will quickly deplete the malicious user's earned credit.

Privacy Protection: A significant challenge to expanding measurement systems on volunteered personal devices is the threat to user privacy. To limit the exposure of personally identifiable information, MITATE captures data only from active traffic

experiments and does not monitor non-MITATE device traffic. Collected data is also indexed by virtual device IDs, rather than personally identifiable phone and International Mobile Equipment Identity (IMEI) numbers.

Remaining Challenges: MITATE is still in active development; project goals for the next couple of years include: deployment on M-Lab, support for peer-to-peer transmission between mobiles (important for IoT and gaming experimentation), and iOS device support.

Seattle The Seattle testbed, originally developed in March 2009 at the University of Washington to support wired host experimentation, now also supports mobile application prototyping [93]. The design goal was to increase the diversity of testbed hardware to provide a more realistic prototyping environment than testbeds relying on dedicated hardware (e.g., PlanetLab, Emulab, or GENI [8, 102, 282]). Seattle runs on volunteered devices in last mile networks, and on institutional servers. As of 2015, Seattle includes about 800 mobile devices and over 10,000 nodes in total.

Functionality: Seattle experiments run on sandboxed virtual machines in a pared down implementation of Python called Repy. Seattle libraries support Repy functions such as data serialization, cryptography, and processing URLs, HTTP messages, and other protocols. Repy code is pushed to Seattle-registered through an API. Users can select devices by location and network type (Wi-Fi or cellular) to which device is connected, but Seattle does not support selection by device travel speed, provider, or model. Seattle also supports P2P communication among devices.

Data Collection: Seattle does not collect network performance data by default. Instead

users define their own metrics through experiments implemented in Repy. Seattle does not provide access to device sensors [323]. although sensor applications can make sensor data available to Repy programs through an API. The Sensibility testbed is an extension of Seattle, which allows Repy experiments to interact with mobile sensor data, but not to transmit or capture network traffic [24].

Resource Incentives and Protection: The Seattle incentive model is based on a tit-for-tat approach, where a user has access to ten volunteered devices for every device she registers with the system. While this policy makes sense in the wired setting, where devices are not generally restricted by monthly data caps, users who register wired hosts but experiment with others' mobile devices can deplete the mobile data cap. As a mitigating step, by default Seattle limits data transmissions to 10 Kbps, so even if the experiment fully uses that transmission rate, the owner can likely continue using their device. This limit prevents Seattle experiments from measuring available bandwidth and generating load-testing traffic – limitations not present in MITATE's credit-based model.

Privacy Protection: Seattle protects user privacy by allowing experiment code execution only in sandboxed virtual machines, which isolates experiment processes from each other and from non-Seattle processes.

Limitations: The authors of Seattle list several limitation of the current system, including inability for Seattle nodes to host services on ports below 1024, increase the transmission limit on donated resources, send ICMP traffic due to Repy restrictions, and put a limit on battery drain [332].

Emerging Systems PhantomNet, being developed at University of Utah, is an emerging testbed based on a network of small-cell base stations connected through a software-defined network (SDN) backbone [198]. Users will be able to not only experiment with end-to-end services, but also modify backbone traffic forwarding for their experiments. PhantomNet devices will have dual-radio interfaces, which will allow integration with a reseller network, for example through SciWiNet. PhantomNet also leverages management tools from other systems, notably Emulab and Seattle. Currently, PhantomNet remains under development.

Curated Network Testbeds

Curated network testbeds vet network experiments prior to deployment. In particular, vetting involves passive monitoring experiments that collect privacy sensitive data, such as users' traffic, or location history and may need to go through an IRB review. Other experiments may require changes to the testbed itself and need to be approved by the testbed's developer team [251].

PhoneLab PhoneLab is a programmable smartphone testbed, developed at the University at Buffalo in November 2013, to support flexible experimentation intended to emulate application deployment scenarios [212, 250]. PhoneLab experiments are implemented as mobile applications pushed to rooted Android smartphones given to student volunteers at the University at Buffalo. PhoneLab's model supports long-term, passive experiments that can record network transitions, battery drain, and use of other applications on the device.

Functionality: PhoneLab experiments are pushed to participants either via the Google Play Store, or separate as over-the-air updates. PhoneLab can benchmark

third-party mobile applications without modifications to their code, which may be required in other testbeds. PhoneLab mobile applications can run experiments in the background or interactively. PhoneLab also supports experiments at the OS level, with modifications to the Android runtime system. Platform experiments are vetted by the PhoneLab development team and go through pre-deployment testing. Researchers submit experiments as XML configuration files that specify background experiments to start or stop, log tags to collect, and where to upload collected data. The PhoneLab Conductor fetches configuration files from PhoneLab servers and pushes them to testbed devices.

Data Collection: PhoneLab data collection relies on the Android logging interface, which gives experiments access to device operational data (such as phone status, battery level, etc.), as well as custom application log data. All log data is uploaded to the central server when a device is charging. When their experiment completes, users receive an archive of data that matches experiment tags from all devices that participated in their experiment.

Resource Incentives and Protection: Unlike MITATE and Seattle, which rely on volunteered devices, PhoneLab provides phones with discounted data plans to its participants. In spite of this incentive scheme, the PhoneLab team has faced significant participant attrition, with only 43 of 191 volunteers continuing after the first year [212]. PhoneLab limits the number of simultaneously active of experiments on each device to balance device utilization against interference between experiments.

Privacy Protection: To protect user privacy, experiments submitted to PhoneLab need IRB approval or exemption. PhoneLab participants choose to participate in a

particular experiment after reviewing what information will be collected. Participants can opt-out of an experiment at any time.

Limitations: PhoneLab's use of data plan subsidy potentially limits the scalability of the testbed. Also if phones are not replaced frequently, testbed hardware will eventually lag behind phone models used by the general public. Finally, PhoneLab code is not publicly available, which precludes the possibility of private deployments [249].

SciWiNet Science Wireless Network (SciWiNet), being developed at Clemson University, is a NSF-funded re-seller of network infrastructure, based on Mobile Virtual Network Operator (MVNO) model, which provides the research community with a service on Sprint's cellular network infrastructure (and T-Mobile's infrastructure by late 2014) [23]. SciWiNet supports experimentation over 3G and 4G cellular networks, but without support for SMS, MMS, or voice services. SciWiNet provides additional infrastructure to the research community in the form of a shared pool of wireless devices (smartphones and USB LTE dongles), a common set of Android applications (WiFi hotspot, VPN tunnels, performance monitoring programs), and a set of wireless network services (VPN tunnel termination, secure database backend, performance monitor servers and backend).

Deployment: The SciWiNet project has two proposed project phases and is in phase-I as of September 2014. In phase-I, the project aims to determine the potential user community for SciWiNet infrastructure and investigate capabilities that it should support. In phase-II, the project will develop, deploy and operate the functional SciWiNet network infrastructure based on what was learned in phase-I.

Device support: Since SciWiNet uses Sprint's cellular network as its back-end cellular infrastructure, Sprint maintains a whitelist of mobile devices that are authorized to access SciWiNet's network and therefore eliminates the need to install a SIM card in every mobile device. Although SciWiNet records device MAC address, it does not make the device MAC publicly available. SciWiNet maintains a list of popular devices and blacklisted devices. iOS devices are excluded because they do not support re-seller networks [14]. SciWiNet helps researchers access testbed resources by providing them with 1-2 mobile devices and a prepaid data plan for a limited time, typically six months. Alternatively researchers can access SciWiNet from their own devices and SciWiNet covers part of the data usage costs.

Data Collection: SciWiNet Android app collects the following network measurements over cellular and Wi-Fi networks: throughput for TCP and UDP traffic flows, packet loss, and ping latency. It can also detect location-based services such as base station identity, location, and wireless signal strength.

Resource Incentives and Protection: Users can login to their account to check their data usage, or data contributed by others to their experiments. Data usage is limited by a leaky bucket rate limiter, where a user receives a number of tokens, which he can share among multiple devices. Once the data rate is exceeded, the device is temporarily restricted from accessing the SciWiNet network.

Remaining Challenges: As of September 2014, it is unclear how SciWiNet will provide access to its devices and network resources to the research and developer community.

One possibility is to offer incentives for user participation by providing free or discounted device access.

LiveLabs LiveLabs, designed at Singapore Management University in February 2014, is a mobile testbed intended to evaluate location-based services, such as commercial promotions to shopping mall customers [70]. LiveLabs has been tested on the campus of the Singapore Management University (SMU) and is currently being deployed at a large shopping mall near SMU campus, Singapore Changi International Airport terminal, and on the Sentosa resort island. The testbed is available to the three partnering venue operators, but not the general public.

Functionality: To facilitate evaluation of location-based services, LiveLabs supports device location discovery in indoor settings as well as characterization of user behavior. LiveLabs is designed for continual operation, thus the design has focused on low energy usage, for example by allowing multiple experiments to concurrently use sensor readings such as GPS, or WiFi signal strength. Researchers and participating companies use LiveLabs to evaluate location-based applications, for example real-time promotions to users at a shopping mall. LiveLabs is available for Android and iOS systems.

Data Collection: Unlike other testbeds discussed in this section, LiveLabs does not collect network performance metrics, but instead focuses on discovering user behavior, by recording device ID and a variety of sensor readings. The LiveLabs backend then supports higher level functions to detect and record user behavior, such as history of movement, group size, user physical queue length, and activities such as standing,

walking, or sitting. LiveLabs also records information about participating users, such as their nationality.

Resource Incentives and Protection: LiveLabs has three mechanisms for garnering user participation: rebates on users' monthly data bills; context-based apps that offer rebates on specific commercial services in deployment locations [189]; and a "lucky draws" lottery, though details of frequency and prizes are not specified [70].

Privacy Protection: Data collected by LiveLabs has the potential to disclose private user information, such as location, shopping patterns, and nationality. As such, experiments launched on LiveLabs go through SMU's IRB approval process [18]. Users are also asked to opt-in to data collection on their devices.

Limitations: LiveLabs is not designed for mobile network measurement (does not collect network metrics) and so it offers functionality distinct from MITATE, Seattle, and PhoneLab. At the same time, LiveLabs supports experimentation with new services in the mobile environment similarly to PhoneLab and has attracted participation of 30,000 users through its incentive model and business partnerships.

Measurement Tools

Mobile network performance characterization requires wide scale and ongoing measurement from a variety of devices across different networks and locations. Tools in this space, developed by industry, research, and regulatory communities, differ in how they obtain network metrics and how they select devices for measurement. Although network measurement tools presented in this section are not testbeds, in

that they only support a fixed set of experiments, these tools do support long-term and wide-scale network monitoring, which offers important insights to developers, researchers, and regulators.

Standalone Measurement Tools

Standalone measurement tools are ready-to-deploy solutions with pre-defined network measurement functionality. The open-source nature of these tools allows other to modify them, although many of the tools offer measurement customization options. Data collected by these tools is generally, though not always, publicly available.

FCC Speed Test The FCC Speed Test app, released in November 2013, was designed to provide insight to regulators and the public on the performance of mobile networks across the United States [135]. Developed in collaboration with SamKnows and major wireless service providers, the free application is available on Google Play Store for Android smartphones [138]. An iOS version of the application is also slated for release, though limitations of the iOS API prevent collection of some metadata that is collected by the Android version [106, 175].

Functionality: At the start of a measurement, the FCC Speed Test app pings available measurement servers to identify the one with lowest round trip time (RTT) to the mobile device. The selected server then sends a list of measurement instructions to the mobile device. If the mobile device is currently using less than 64 Kbps of bandwidth for other tasks, it starts the measurements, otherwise the device postpones measurement until its bandwidth usage drops.

The FCC Speed Test app supports active traffic measurements over four types of connections: single connection HTTP GET and POST, as well as multi-connection GET and POST. Multi-connection transfers test multithreaded download performance over three parallel downloads of 256KB data chunks. To measure packet loss and RTT, the FCC Speed Test app exchanges a series of UDP packets with the nearby server. Following a measurement, the mobile device uploads measurement data and associated metadata to an FCC server.

Data Collection: The FCC Speed Test app reports upload and download rates, packet loss, and RTTs based on HTTP and UDP transfers. Packet loss on a path is inferred based on failure to receive a UDP packet on that path within three seconds. The app records the number of packets sent each hour, the average RTT, total packet loss for performed tests, and throughput in 5-second intervals [136]. The app also collects device-related as well as network metadata, including signal strength reported by the device, connection type (3G/4G/Wi-Fi), location and ID of cell towers, GPS location, device model, OS version, network country code, SIM's operator ID, SIM's country code, network carrier, phone type (GSM/CDMA), and the device's roaming status.

Resource Incentives and Protection: To build nationwide measurement capacity the FCC Speed Test app relies on user curiosity about their network performance. Instrumental to the app's popularity and success was a press campaign [76,106,163,179,283], which was followed by application installation and measurements from more than 50,000 devices in about 1.5 years. These numbers have declined over the life of the system, so the effectiveness of a publicity-driven approach to support long-term network monitoring remains to be seen.

Privacy Protection: The FCC app collects measurement data on the mobile device in the application sandbox, as opposed to through the standard Android logging interface, so data is not visible to other applications. The collected data are uploaded to FCC servers over encrypted connections. Once the data are uploaded, or become stale, they are automatically deleted from the application's sandbox storage. The FCC Speed Test app does not collect personally identifiable information, such as phone number or IMEI [137].

Limitations: The FCC Speed Test app executes only experiments configured by the FCC, i.e., it does not support custom network measurement. As of October 2014, the configured tests do not detect traffic shaping in mobile networks, which is of increasing interest to regulators and the general public [50,59,84,85,133,178,192,259,285]. With respect to resources used on the device, the FCC application runs at startup and prevents the phone from sleeping, which can drain the phone battery.

WindRider Content-based traffic discrimination has recently been considered a threat to mobile application performance [50, 59, 84, 85, 133, 178, 192, 259, 285]. WindRider, a measurement tool developed in 2009 at Northwestern University, detects application and service-based traffic discrimination by mobile ISPs [293].

Functionality: WindRider supports active and passive measurement of traffic shaping [315]. Active measurements exchange traffic between a user's mobile devices and a randomly chosen M-Lab server. The mobile device initiates a series of uploads and downloads and records their observed performance. To detect port-based traffic shaping, WindRider compares delay of identical transfers to different ports on M-Lab

servers. Passive measurements record packet latency to well-known web servers during normal user browsing activity. To detect content-based traffic shaping, WindRider compares the observed packet delay to that reported by other devices in different carrier networks and locations to the same destinations. Active measurement results are stored on M-Lab servers, while passive measurement data, collected with user permission, are stored on WindRider servers.

Data Collection: The WindRider mobile application collects experiment-related data such as connection start time, connection establishment time, connection finish time, and number of inbound and outbound bytes [293]. WindRider also records metadata such as device IMEI, device location (as ZIP code), network carrier, and browsing history. WindRider also collects device hardware performance metrics that can help interpret observed traffic delays, such as CPU execution time, virtual memory size, page faults per minute, and other metrics as permitted by the OS API.

Resource Incentives and Protection: WindRider relies on user curiosity for its network measurements.

Privacy Protection: WindRider optionally collects device IMEI, which can be linked with a user's browsing history. To protect user privacy, users can choose whether to make this information available to the application.

Limitations: Although WindRider supports detection of traffic shaping in mobile networks, it has two significant limitations. First, the measurement traffic is sent only to M-Lab servers, but developers may want to investigate traffic shaping on other paths. Second, WindRider only detects content-based traffic shaping as

discrimination based on traffic sources, i.e., well-known Web servers, rather than type of traffic, for example BitTorrent.

MySpeedTest The MySpeedTest mobile application, launched in June 2012 by Georgia Tech, measures network performance of mobile devices with the goal of observing and explaining patterns of user behavior in mobile ISPs to application developers [139, 210]. Such analysis may allow developers and service providers to tune application performance [209]. The MySpeedTest mobile application is available on Google Play and has more than 900 active users from 115 different countries, as of February 2013 [209]. As of April 2013, MySpeedTest is in the process of sharing a subset of their data with Google's M-Lab to help researchers benefit from data collected by each others' experiments [210].

Functionality: MySpeedTest performs passive and active measurements. Passively, MySpeedTest records the total number of bytes sent and received by each active application since the device booted. Information such as package name, bytes transmitted and received, application status (active vs. background) helps users know which applications consume the most data and power, and which applications may affect performance of other applications on the device.

Active measurements include a recurring test to measure TCP uplink and downlink throughput, inter-packet delay, and packet loss. MySpeedTest also measures network latency with 40 parallel ICMP pings to five servers in the U.S. and Europe. These tests store the minimum, average, and maximum latency to each of the five servers. The collected data help researchers and developers understand the performance of paths to potential application servers [209].

TCP-based experiments can reduce the bandwidth available to other applications on the device, so MySpeedTest performs TCP-based experiments only on user request, in a single thread for about 20 seconds, and using the maximum-sized packets that will not be fragmented. MySpeedTest also gauges streaming data quality by measuring packet loss and jitter of UDP traffic flows. MySpeedTest servers generates a stream of 64-byte UDP packets, transmission at Poisson-sampled intervals, with timestamps and sequence numbers in the payload. The server sends 500 packets with a data rate less than 1 Kbps to avoid congestion. The client calculates packet loss and jitter from every 10 packets received. The client compiles all data collected on mobile device into the JSON format and sends it to the server for storage.

Data Collection: The MySpeedTest mobile application collects experiment-related data such as TCP upload and download throughput, ping latency, UDP jitter, UDP packet loss, and time to acquire a dedicated channel for data transmission [210]. MySpeedTest also collects device level data, such as cellular service provider, Android version, device manufacturer, connection type, radio firmware, hashed phone number, hashed IMEI, software version, SIM card state and serial number, latitude and longitude of base station, network operator ID, CDMA system ID, CDMA network ID, Wi-Fi signal strength, battery technology, status of battery charging, battery health, battery voltage, battery temperature, and device location.

Resource Incentives and Protection: Similar to the FCC Speed Test app, MySpeedTest relies on user curiosity about their network performance. MySpeedTest allows users to limit contribution of resources through a monthly data cap. To protect battery resources, MySpeedTest postpones experiments until the battery is above 5% and the device is attached to a network.

Privacy Protection: MySpeedTest collects personally identifiable information (phone number, IMEI, device location), which may expose private information, such as a user's location when a measurement occurred.

Limitations: Similar to MobiPerf and WindRider, MySpeedTest provides its users a limited network measurement capability between mobile devices and servers, as opposed to testbeds discussed in Section 3. MySpeedTest does not support transmission of custom traffic, such as tools to detect traffic-shaping based on content or port.

Akamai Mobitest Akamai's Mobitest application and Web service, released in March 2012 by Akamai Technologies, measures the performance of mobile Web sites [53]. The application uses the WebPageTest framework and is available for Android, iOS, Blackberry based smartphones, tablets and simulators [314].

Functionality: Mobitest platform relies on user participation to install Mobitest software on their mobile devices. Each Mobitest installation on a device acts as an agent to the WebPageTest framework, where such device executes experiments requested by other users through the Mobitest Web service [162]. To measure the page load time on a mobile device, a user enters a URL through the Akamai Mobitest Web interface and selects the mobile device hardware that will perform the download [53]. Mobile devices running Mobitest periodically poll WebPageTest servers to obtain pending URL download requests entered by Mobitest users. Each requested URL is then accessed from the default browser on each device over the Wi-Fi, or cellular network, depending on how the device is connected at the time.

Data Collection: Akamai Mobitest collects the total time to load a Web page, individual request headers, average Web page size, as well as screen shots of the loaded page and optionally video of the loading page [162]. The tool produces waterfall charts of requests and delays, and an HTTP archive (HAR) file [54, 126]. The collected data helps researchers and developers gain insight into the responsiveness of Web servers and browser rendering of different site implementations [252]. Mobitest allows users to reuse previously collected measurements by linking them to user accounts on Akamai Mobitest's site.

Resource Incentives and Protection and Privacy: The Akamai Mobitest app allows application developers to set the frequency at which pending experiments are downloaded from WebPageTest servers to be executed on their mobile devices. Additionally, Akamai Mobitest allows users to control device resource utilization through a number of configuration options. Specifically, users can set whether the app should poll for new experiments after restart, whether to restart the app after every experiment, whether to capture network traffic, and the frequency at which screenshots for loading pages are taken [7].

Limitations: Akamai Mobitest evaluates the webpage load time on mobile devices, but does not allow more general experiments with non-browser-based application traffic, including how to characterize traffic shaping of non-Web traffic. The WebPageTest framework requires rooted phones, which limits the tool's applicability outside of dedicated test farms.

RILAnalyzer RILAnalyzer, developed by the University of Cambridge and Telefonica in October 2013, is a client-side tool for monitoring of the mobile network control plane as well as the data plane [65, 296]. The application is available for rooted Android devices with Intel/Infineon XGold chipsets, which include the popular Samsung Galaxy S2/S3, Note 2, and Nexus devices.

Functionality: RILAnalyzer's focus is on discovering the promotions and demotions between the Radio Resource Control (RRC) states `IDLE` (no connection), `CELL_DCH` (dedicated communication channel), `CELL_FACH` (shared communication channel), and `CELL_PCH` (shared paging channel). Transitions between these states are triggered by control messages from the Radio Network Controller (RNC), which may themselves become a communication bottleneck [296]. As mobile devices consume different levels of energy in each of the RRC states, the devices themselves may use *Fast Dormancy* to reduce *tail-energy* and demote to lower energy states faster than through vendor and operator dependent timeouts [184].

RILAnalyzer implements a background tool that polls the device Radio Interface Layer (RIL) Daemon every second for the current RRC state. RILAnalyzer then obtains data plane network and transport headers using NetworkLog [4] to identify applications active during each RRC state.

Data Collection: RILAnalyzer collects RRC states at one second intervals, headers and timestamps of outgoing TCP and UDP packets from NetworkLog as reported by the Linux kernel.

Resource and Privacy Protection: RILAnalyzer is intended for small scale studies on dedicated devices, or devices operated by expert users [296]. As such the tool's design

has not made provisions to attract users with incentives, or to allow them to set limits on resource usage.

Limitations: RILAnalyzer is restricted to rooted phones on the Intel/Infineon XGold chipset. Although the authors of RILAnalyzer intend the tool for small scale studies, the specificity of hardware and overhead of reverse engineering RIL Daemon `OemCommands` commands does preclude large scale studies on diverse mobile hardware. RILAnalizer also puts a noticeable load on the CPU (∼10%), memory (<42%), and storage (with packet logs), which may limit the willingness of volunteers to run the tool on their phones.

Libraries for Mobile Network Measurement

Libraries for mobile network measurement may be embedded in other applications to add network measurement functionality. This approach is potentially easier to adopt by Developers than extending open-source code of a standalone measurement tool. As in the case of Mobilizer, a library may also form a basis of a measurement tool, i.e. the current version of MobiPerf.

MobiPerf The MobiPerf mobile application was developed as a collaboration of University of Michigan, Northeastern University, University of Washington, and Google's M-Lab to measure network performance and diagnose problems with application content delivery on mobile devices [206]. To allow the community to understand the impact of collected data across geographic locations, network carriers, and devices, MobiPerf allows a comparative study of past network measurements made by different users, but prevents users from running similar measurements to limit contention for testbed resources. New measurements are executed only if a

query for previously collected data comes back empty. The latest version of MobiPerf, released in August 2014, is based on Mobilizer – an open-source Android library for network measurement announced at AIMS 2014 [324].

Functionality: MobiPerf supports several types of network performance measurement, which can execute serially or in parallel [166]. Mobilyzer provides measurement isolation (only one experiment is active at a time), which avoids bandwidth contention and radio power state transitions across experiments. To measure throughput, Mobiperf transmits random data to and from a nearby M-Lab server for 16 seconds and computes uplink and downlink throughput from packet traces.

MobiPerf supports latency measurements on both IPv4 and IPv6 network paths, using ICMP ping when available, with fallback to a Java ping implementation and latency estimates from three-way TCP handshakes in HTTP transfers. Mobiperf measures the delay of DNS lookups using the default DNS server configured for the device, which limits the ability to measure performance of third-party open DNS infrastructure.

MobiPerf also supports measurement of uplink and downlink UDP packet loss, out-of-order delivery, and variation of one-way latency. To obtain these metrics on the uplink, a client device sends a group of UDP packets to a nearby M-Lab server, where the server calculates network metrics from packet arrival time and order. The same transmission repeats from server to client to calculate downlink metrics.

MobiPerf performs more complex measurements to discover fine-grained network policies and their effect on data plane performance. For example, MobiPerf measures radio resource control (RRC) state information of cellular networks to estimate the impact on packet latency [267]. Finally, MobiPerf measurements can execute in the background to support long-term monitoring of network performance.

Data Collection: Similar to other measurement tools, the MobiPerf application collects performance data such as TCP uplink and download throughput, HTTP download latency and throughput, traceroutes, path latency, and DNS lookup delay. Researchers and vendors may want to know how variation in mobile hardware affects application performance, so MobiPerf collects device-related data such as manufacturer, model, operating system version, Android API level, carrier, salted hash of device IMEI, coarse-grained cell ID location information, cell tower ID and signal strength, Location Area Code (LAC), local IP address, IP address seen by the remote server, GPS coordinates, ports blocked by cellular provider and network connection type (HSPA/LTE) [205, 331].

Resource Incentives and Protection: MobiPerf relies on user curiosity to support measurement, and users can limit the resources they contribute. Specifically, measurements do not execute when the device battery consumption, or MobiPerf application monthly data usage, exceed user-set thresholds.

Privacy Protection: MobiPerf currently records the users' e-mail address, if they choose to provide one, to access their historical measurement results. This information is secured by Google's account authentication mechanisms and is not made publicly available. To minimize any risk of exposing this potentially personally identifiable information, future versions of MobiPerf will store a salted hash of users' e-mail addresses instead.

Limitations: MobiPerf allows users to choose from only predefined measurements,

which limits the tool flexibility. For example, MobiPerf does not support transfers of custom content on arbitrary ports to detect network traffic shaping.

ALICE A Lightweight Interface for Controlled Experiments (ALICE) is a programmable network measurement library for Android devices developed by John Rula *et al.* at Northwestern University [269]. ALICE extends Dasu, a rule-based network testbed built as an add-on to the Vuze BitTorrent client [275], by enabling experiment definition in Javascript [270].

Functionality: The ALICE measurement library supports active and passive experiments on mobile devices. ALICE provides a programmable interface for the configuration of active network measurements, such as DNS resolution, ping, and iPerf. Tests can execute sequentially or in parallel. Although the sequence of tests and value passing between them is organized through a Javascript experiment definition, ALICE does not support custom traffic generation, and so is primarily a network measurement library. For serially scheduled experiments, ALICE allows one experiment on a device at a time; for parallel execution, ALICE allows a limited number of experiments to run at the same time – new experiments scheduled for a given device enter a queue until the device becomes available. ALICE chooses its test devices based on user-specified time of day, network provider, and network type (Wi-Fi/Cellular).

Data Collection: ALICE collects device location, radio signal strength (WiFi and cellular), WiFi access point name, device hardware address, IP address on each network interface, and number of bytes sent and received by other applications on the device. ALICE also collects performance metrics, including HTTP GET request

time, DNS lookup time, ping times, available bandwidth. ALICE records network diagnostic information provided by traceroute and NDT (Section 3.3.1).

Resource Incentives and Protection: As of September 2014, ALICE has been included in three different applications developed at Northwestern University and available through the Google Play store: Namehelp Mobile[1], Application Time (AppT)[2], and NU Signals v2[3]. The Northwestern team's deployment model of growing the tool through application deployments allows ALICE to benefit from popularity spikes of new applications. To protect device resources, developers can set quotas for bandwidth usage of individual measurements.

Privacy Protection: ALICE records hardware addresses of available network interfaces, which are unique to each device. In combination with the ability to record sent and received traffic payload of other applications, for example location reporting, ALICE creates a potential for privacy exposure, if user location, or other private data, is correlated to unique device ID.

Remaining Challenges: Currently ALICE does not support repeatable experiments on the same device, or set of devices, through device selection criteria. ALICE also does not support peer-to-peer experiments, or custom traffic transmissions, which limits the tool's support for application prototyping.

[1]Namehelp Mobile measures the DNS performance of Cellular ISPs and public DNS resolvers, including of CDN replicas [62]

[2]Application Time allows users to track their application usage on their mobile device [61].

[3]NU Signlas allows users to diagnose Wi-Fi problems [220].

Measurement Services

In addition to testbeds and tools there are many closed-source, proprietary measurement services for mobile networks. I divide these services into network monitoring and network discovery and diagnosis. The main goal of these is to collect data and provide insight to users based on their own device, but not necessarily make the data broadly available. Still, these services offer valuable insight to developers, researchers, regulators, and network operators able to access the data. Because the details of how these services are implemented and how they perform measurements is not widely available, I restrict our discussion, with few exceptions, to the commonalities and differences of what data these services collect.

Network Monitoring

Google Play Store and Apple App Store offer tens of applications for monitoring of network performance. Because of their relative similarity, I restrict our discussion to several popular and representative services.

Ookla SpeedTest Mobile Ookla's SpeedTest application for mobile devices, released in January 2009, measures the device's network performance over Wi-Fi and cellular links [225]. As of March 2015, the application support measurements against 3479 geographically distributed Ookla servers in about 80% of world's ISP networks, has over 10 million installations, and has successfully completed over 7 billion user initiated measurements on the Ookla infrastructure [39, 227]. Ookla also allows users to host an Ookla server To expand the capacity of their measurement infrastructure, Ookla also allows users to host an Ookla server [226]. The application is available for Android, iOS, Windows phone, and Amazon FireOS based smartphones [225].

Functionality: The application captures the device geographic location and uses it to identify a set of five nearby servers. If the device location is not available from the GPS, the application uses device's IP address and estimates the device's location using MaxMind's (approximate) IP-to-location database [197, 228, 231]. After identifying a pool of five nearby servers, the application sends a `hello` message to all five servers and selects the measurement server from the first received reply [229]. Users may also select a specific server based on criteria such as hosting ISP, distance from user, and city name.

This SpeedTest uses HTTP fetches of small files to measure round-trip time and compute uplink and downlink throughput [230]. To measure the ping latency, the application sends several HTTP requests and records the time when app receives responses from the server [230]. Ookla SpeedTest uses the computed connection throughput to estimate how much data it can download from the server within 10 seconds, and then uses up to four HTTP threads on a single persistent connection to download the estimated amount of data. To eliminate any influence on the throughput results from protocol overhead, buffering time on the device, CPU usage, the application first discards the fastest and slowest 10% of throughput values as outliers before computing the average throughput. The application then discards the slowest 20% of throughput values to prevent results from being influenced by TCP slow-start. Finally, the application calculates the downlink throughput for the experiment based on the average of the remaining throughput values. The uplink throughput test is similar to downlink throughput test.

Data Collection: Ookla's SpeedTest mobile application collects device location (GPS and network-based), radio signal strength, device ID, device phone number, call

status and remote phone number of an active call, names of devices on connected Wi-Fi network, local and public IP addresses, time at which the experiment was conducted, round-trip time, upload and download throughput, and connected network type (Wi-Fi or cellular). Ookla supports another application, *PingTest*, that collects network jitter and packet loss, to understand the suitability of the user's network for services such as VoIP audio, video streaming, and online gaming [253].

Resource Incentives and Protection: The application limits the number of HTTP threads to two when the observed throughput is less than 4 Mbps, otherwise the it uses four threads for throughput experiments.

Privacy Protection: The SpeedTest mobile application collects personally identifiable information (phone number, device ID, and device location), which may expose private information, such as a user's location when a measurement occurred. Users may delete previously collected data, or leave it on Ookla servers to compare with new data collected at a later time to discover changes in network performance over time.

Limitations: As of March 2015, the SpeedTest mobile application lacks a programming interface to allow users to automate and schedule experiments. Although, Ookla allows users to host SpeedTest experiments on their Web servers for in-house testing, via SpeedTest Mini, however, as of March 2015, the ability to run measurement against such servers is not supported on Speedtest's mobile application and is only supported with the Web version of Ookla SpeedTest [232]. The algorithm used by the application to measure the round-trip time relies on the time it takes to receive an HTTP response, which may include the time request spent in transport queue

and application processing at the server. Finally, the application does not support detection of traffic shaping.

RadioOpt Traffic Monitor The RadioOpt Traffic Monitor mobile application, released in April 2012 by RadioOpt GmbH, allows users to understand the performance, reliability, and utilization of their wireless and cellular networks [258]. Based on the information collected about the network, the application allows users to compare the performance of their wireless networks with other users in the same geographic region. The application is available for Android, iOS (iOS 7.0 or later), Blackberry, and Windows-based smartphones [256, 257]. As of March 2015, the application was installed over a million times.

Functionality: The RadioOpt Traffic Monitor mobile application uses CacheFly's CDN infrastructure. To identify a nearby server, the application sends a DNS query to the device's default DNS server for a CacheFly CDN domain name (`cdn2.speedtestsdk.com`). CacheFly uses TCP-anycast to direct users to the nearest CDN replicas [91]. Next, the application sequentially initiates downlink and uplink throughput tests to the selected server. To measure throughput, the application estimates the appropriate size of the data to exchange between the device and the server, similar to Ookla SpeedTest.

To measure latency, the application sends 15 ICMP ping requests to the server and records the time of each request/response pair. To measure the time to load a webpage on user's network, the application sends three HTTP GET requests and records the time to download the complete webpage, and other web objects such as CSS, image, JavaScript files embedded into the page.

Data Collection: The application computes parameters from measurement data such as the minimum, average, maximum ping latency to a nearby CacheFly server along with the standard deviation in latency and throughput, the amount of data uploaded and downloaded for the throughput tests, download time of a hosted web page and the web page size. The application also collects device-related information such as its location (including accuracy and device travel speed), web bookmarks and browsing history, names of devices connected to the same Wi-Fi network, signal strengths at different locations, number of SMSes sent and received, and incoming and outgoing voice minutes, device model and manufacturer, OS or firmware version, current time on the device, the time when the device was last rebooted, cellular access technology (2G/3G/4G), and network country code.

The application also collects information specific to applications on the device such as their names, duration of usage, cellular and Wi-Fi data consumption (only on Android based smartphones), memory consumption, traffic (per application) sent and received on the device over cellular and Wi-Fi networks, application type (OS service or background), and software packages used by the application.

RadioOpt collects device battery-specific information such as the battery state and charge remaining, voltage, temperature, technology, and charging state. Finally, the application collects Wi-Fi network related information such as the signal strength (latest, minimum, and maximum), network SSID and BSSIDs, the MAC address of the client, IP address of the client, and client-to-router link bandwidth.

Resource Incentives and Protection: RadioOpt relies on user curiosity to understand the performance of their own wireless and cellular networks. The app allows users to configure a monthly/weekly/daily cellular data cap, monitor their monthly data

traffic, SMSes received and sent, and voice minutes, and configure alerts when data, SMS, or voice minutes reach a threshold.

Privacy Protection: The application may discover user behavior since it collects information such as the user's Web browsing history, bookmarks, applications installed and their duration of usage, among others. However, any personally identifiable data collected by RadioOpt mobile application is not shared with RadioOpt servers without the user's consent.

Limitations: RadioOpt does not allow its users to understand whether their cellular ISPs are discriminating one traffic over the other. Further, the application does not support measurement experiments to be run against an arbitrary server.

OpenSignal The OpenSignal mobile application, released in March 2013 by OpenSignal, Inc., allows users to compare the quality and coverage of their cellular networks (on a Google Map's developer widget [154]) in different geographic areas and with other cellular networks available in the area [239]. The application rates for how well Web, Video, and VoIP based applications are likely to perform on the current cellular network. The application assist users to also find publicly available free and paid Wi-Fi hot-spots, and the walking directions for higher signal strength. The application has over 10 million installations and is available for Android and iOS based smartphones [237, 240]. As of March 2015, the application has garnered over 900,000 users and has performed several network measurements to collect information for over 800,000 cellular towers, 825 cellular networks, over 5B cellular signal readings, and over 1B Wi-Fi access points available in different countries [239].

Functionality: The OpenSignal mobile application supports several active and passive measurements to measure ping latency, download and upload throughput. The application performs periodic passive measurements, and publishes them to an OpenSignal server [92]. Before starting any measurement test, the application sends the device ID, OS, Android API version, and BSSIDs of nearby wireless networks to an OpenSignal server. Next, to measure latency, the application sends 3 HTTP HEAD requests to `www.google.com` [140, 238]. The application then records the time to receive the time to get the response for each request, followed by calculating the average of the three latency values.

To measure the download throughput, the application sends eight concurrent HTTP GET requests to download files of size 108 Mb each, from a CloudFront's CDN replica [92]. The download throughput test is performed for a fixed amount of time after which the application computes the average throughput. To measure the upload throughput, the application sends several concurrent HTTPS POST requests to upload several small image files of size 15 Mb in total, to an Amazon AWS server.

As a part of making the collected data available publicly and to encourage developers, researchers, regulators, and network operators to investigate and address network problems, OpenSignal provides two APIs [49]. The first API, known as NetworkStatus, allows developers to get signal strength, upload and download throughput, round trip latency, and network name, network ID, network type (2G/3G/4G), and network reliability for every measurement within certain distance of a specified geographic coordinate [233]. The second API, known as Tower Info, allows developers to get the cell ID, location area code, phone type (GSM/CDMA), and estimated latitude and longitude of a cellular tower [234].

To prevent misuse of their publicly available API, OpenSignal allows a maximum of five API calls every minute and 2000 API calls every month.

Data Collection: The OpenSignal mobile application collects device-related information such as SMS transmission and receipt timestamps, device location, ID, model name, OS, Android API level, IP address, behavior at different battery temperatures (hot, crashed, slow, fast), duration of OpenSignal sessions on the device, and whether the phone is engaged in a phone call during the measurement.

The application collects network-related information such as the active Wi-Fi SSID, names of devices connected to the Wi-Fi, SSIDs of other avaiable Wi-Fi, connection type (collected every 15 minutes), signal strength, upload and download throughput, and round trip latency to a Google server. For devices connected to GSM networks, the application associates cell towers by their cell id and location area code; for CDMA networks, by their Network ID, Base sub-station ID and system ID [240]. To understand the relationship between signal quality and battery consumption, the OpenSignal application collects the battery level, voltage and temperature [236].

Resource Incentives and Protection: OpenSignal does not provide any incentives for user participation to run measurement experiments on mobile devices.

Privacy Protection: Although the application collects information about phone calls and SMS messages, the application never reads them [235]. This is because the application only counts the total number of text messages received and sent from the device. Further, any personally identifiable information collected by the OpenSignal mobile application is never shared by any third party services [235]. However, OpenSignal does not take any responisbility of any data shared by the user on

online social networking websites through the OpenSignal application. Finally, the application does not put any obligation on the user to share the data collected with OpenSignal.

Limitations: The application does not detect the presence of traffic shaping in ISP networks. Further, to perform throughput measurement tests, the application requires an exchange of several hundred of megabytes between the mobile device and the server, which may not be suitable for users with low data plans [92].

Vodafone NetPerform The Vodafone NetPerform mobile application, released in June 2014 by Vodafone Sales and Services Limited, allows users to understand the performance of their cellular network in their region and compare with it with the performance that other users in the same region are experiencing [304]. The application also allows Vodafone to understand the amount of data that their customers use and as well as the trend in data usage by tracking the data usage from different applications installed on their customers' smartphones. Such knowledge of data usage allows Vodafone to resolve connectivity issues in their network, as well as, install higher capacity links to accommodate any customer demands to support interactive applications that require higher bandwidth. The data used by the Vodafone NetPerform mobile application is free for only Vodafone customers in Ghana, Ireland, and United Kingdom. However, users in other countries or non Vodafone customers may be charged for any data used by the Vodafone NetPerform application. The application is available for Android and iOS based smartphones [302, 303].

Functionality: Every hour, the application establishes a TCP connection with a

Vodafone server to verify whether the device has Internet connectivity. Conducting such a test every hour allows Vodafone to understand the network stability and any variation in end-to-end latency on their network over time. The application performs another hourly network measurement test to determine the uplink and downlink throughput against a nearby Vodafone server. The throughput tests execute for only 10 seconds, within which the application exchanges data with a Vodafone server [302]. The throughput is then calculated as the average of different throughput values sampled in 10 seconds.

Data Collection: The data collected by the Vodafone NetPerform mobile application is stored on Vodafone servers for only 14 months, which allows Vodafone to understand the changes in the seasonal use of the network usage by their customers. To understand and diagnose the network problems related to phone call connectivity, the application collects cellular tower ID to which the device is connected, signal strength, device location when the network is either limited or not available, the quality of 2G/3G coverage, device speed (if available through GPS), and time duration when the device uses cellular network, how the phone call ends (dropped or disconnected by the user) [305].

To understand and diagnose issues related to data services the application additionally captures whether the device can establish a connection with a Vodafone server, time taken to establish a connection with a Vodafone server, the MAC addresses of all available Wi-Fi access points along with their link bandwidth, hourly data usage of the device, data usage when the device is in standby mode, and the upload and download throughput [305].

To understand the types of Internet services that users are interested in and to allocate high capacity bandwidth for services that require high bandwidth, the

application captures the names of all applications installed on the device, the names of applications that the user uses everyday, the duration of application use, the amount of data is received and sent from each installed application.

Finally, to diagnose and resolve device related network issues, the application collects the device model and company, device IMEI (encrypted to maintain anonymity), the OS running on the device, firmware version, the OS language, battery status, memory in use, the time when the phone last rebooted [305].

Resource Incentives and Protection: Users do not get any incentives for running measurement tests on their devices. Instead, Vodafone relies on users' curiosity to understand the network performance and gathers data collected on users' devices to improve the quality of their voice and data services. With respect to protecting device resources, the application does not allow users to configure a monthly cap on the amount of cellular and Wi-Fi data that the application can use to run measurement tests. Further, since the application run throughput and latency tests every hour, the application prevents the device to turn off its radio, which drains the device's battery quickly [302].

Privacy Protection: The application does not collect any personally identifiable information such as the device phone number, the phone numbers of incoming and outgoing phone calls, incoming and outgoing SMS messages, and the names of available Wi-Fi hotspots. However, by collecting the names of application installed and when different applications are used, the Vodafone NetPerform application has a potential to discover user behavior, which might be unsuitable for some users.

Limitations: The Vodafone NetPerform mobile application does not allow users to

discover whether their cellular ISPs are performaing traffic discrimination. Further, the availability of the application only for users in a few countries in Europe restricts the network operators in other countries to gain insight of their network issues and performance.

Emerging Applications Many other network measurement services have been developed by independent developers to assist users to measure performance of wireless and cellular networks. Such applications include SpeedSpot [289], Sensorly [278], RootMetrics [266], NetworkCoverage [131], Internet Speed Test [295], Netradar [216], Cisco Data Meter [103], 4Gmark [300], and nPerf [219]. Because the details of how these services are implemented and how they perform measurements is not widely available, I restrict our discussion to the commonalities and differences of what data these tools collect and how. I also illustrate the similarities and differences between these emerging measurement services in Table 3.2.

Specifically, SpeedSpot, Sensorly, RootMetrics, and NetworkCoverage are similar to the OpenSignal mobile application in their capability for users to compare the performance of their wireless and cellular networks on a map and find nearby Wi-Fi networks. Internet Speed Test is similar to Ookla SpeedTest; it allows users to measure the latency and throughput to application's servers on user's network. Similar to RadioOpt, NetRadar and Cisco Data Meter applications run latency and throughput experiments against servers deployed on the cloud/CDN servers and allows users to monitor traffic sent and received by applications installed on their devices. 4Gmark and nPerf are similar to each other, in that, these applications not only allow users to measure the performance of network in terms of throughput and latency, but also measures the suitability and reliability of the network for streaming and Web applications.

Network Discovery and Diagnosis

While most of the previous projects focus on measuring end-to-end performance of mobile application communications, the following tools allow developers and researchers to learn more about the state of network infrastructure and configurations that affect transmission of application traffic. Pertinent features include the presence of proxy servers and other middleboxes, or complex multi-level DNS resolutions.

NDT (Mobile Client) The Network Diagnostic Test (NDT) system, developed by Internet2, evaluates the performance of mobile connections to diagnose problems that limit network bandwidth [169, 203]. NDT also detects problems associated with device misconfiguration and network infrastructure. NDT (Mobile) is currently hosted on Google's M-Lab and allows access to its backend through an Android mobile application.

Functionality: NDT measurements are performed from a mobile Web browser that issues requests to NDT servers, hosted by M-Lab. The server-specific tests diagnose observed network problems. After the measurement experiment completes, the server analyzes the results and returns them to the client device.

Data Collection: The NDT mobile application collects traffic performance information such as upload and download speed, round trip network latency (minimum, average, and maximum), jitter, TCP receive window size (current and maximum), packet loss, TCP retransmission timer, and number of selective acknowledgements received. The application also detects router cable faults, incorrectly set TCP buffers in the device, duplex mismatch conditions on Ethernet links, presence of NAT, and capacity limits.

Resource Incentives and Protection: The incentive model for the NDT mobile client is based on providing network diagnostic information in exchange for users running tests on their mobile devices. One issue for users who volunteer their device resources is that NDT requires permission to prevent the phone from going into power save mode, which may drain the battery quickly.

Privacy Protection: By default NDT records experimental data separately for each user, which allows users to privately diagnose their network problems. Data isolation also prevents malicious users from learning of open ports and interfaces in others' networks.

Limitations: The NDT mobile client executes experiments that evaluate network traffic only between a mobile device and its closest M-Lab server and not any arbitrary server.

Netalyzr The Netalyzr mobile application, developed as a collaboration of ICSI Berkeley, UC Berkeley, HIIT, and Aalto University, is a diagnostic tool that characterizes connectivity, performance anomalies, and network security issues [183, 298]. The tools measures network latency and bandwidth to reveal insight into not only performance to cloud servers, but also how middleboxes in the path affect the performance of traffic. As of March 2014, Netalyzr has run over 15000 times to diagnose 290 operators in 90 countries. Netalyzr is accessible via an Android mobile application available on the Google Play Store.

Functionality: Netalyzr identifies the presence of Network Address Translations (NATs),

proxy servers along a route, IP fragmentation, size of bottleneck buffers, reachability of services, and presence of HTTP proxies. When the Netalyzr application starts, it contacts the Netalyzr's Web server, which issues a DNS lookup request to redirect the user's request randomly to one of the twenty Netalyzr's back-end servers hosted on the Amazon cloud. Each back-end server supports twelve concurrent measurement sessions.

Netalyzr detects the presence of a NAT based on a difference between a user's local and public IP addresses. For clients behind a NAT, Netalyzr identifies how the network renumbers addresses and ports, i.e., whether the NAT uses fixed associations of local IP addresses to different public IP addresses, or if the NAT uses load-balancing.

To detect support for IP fragmentation, Netalyzr sends a 2 KB UDP packet (larger than 1500 B Ethernet maximum transmission unit (MTU)) to the server – a response from the server indicates the network supports fragmentation. If there is no response Netalyzr uses binary search to find the maximum packet size it can deliver without the packet being fragmented at the IP layer. The same test repeats from server to client to detect network support for fragmentation on the reverse path.

The sizing of bottleneck buffers affects user-perceived latency, and is measured based on the difference in latency during inactivity and during path throughput tests. Finally, queue drain time indicates the size of the buffer. To perform service reachability related experiments, the application attempts to connect to 25 different well known ports on a back-end server.

Netalyzr infers the presence of HTTP proxies if the public IP address in the request received by the back-end server is not the same as the client's public IP address. To detect the presence of in-path HTTP proxy, the client first sends an HTTP request to the server, the server then returns the request headers it received

in the request back to the client. The client then compares the headers it sent and the headers the server sent to the client for any added, deleted or modified fields. To detect the presence of caching policies, the application relies on the HTTP 304 Not Modified response from the server.

To detect the presence of a DNS-proxy server or firewall, the application sends a DNS request to Netalyzr's back-end server. If the client detects any change in the response (different transaction ID, or public IP address), then Netalyzr assumes an in-path DNS proxy exists. Netalyzr then makes invalid DNS requests to the back-end server. If the client receives an invalid response from the server, nothing is detected, but if the request is blocked, Netalyzer assumes a DNS-aware middlebox is blocking invalid DNS requests from leaving the network.

Data Collection: The Netalyzr mobile app records the presence of network interfaces, gateways, NAT detection, port renumbering, path MTU, packet fragmentation, DNS resolver, extension mechanisms for DNS (EDNS) support, port randomization, IPv6 support, hidden proxies, in-path caches, header manipulation, image transcoding, compression, HTTP type filtering, port filtering, traffic differentiation, IP fragmentation, signal-to-noise ratio, Wi-Fi/cellular configuration, network topology through traceroute, TLS handshake, UPnP vulnerabilities on Wi-Fi APs, clock drift, and TLS default certificates [298].

Resource Incentives and Protection: Netalyzr provides network diagnostic and troubleshooting information to users. Netalyzr requests user permission to modify system settings and to terminate other running applications in order to increase measurement accuracy. The Netalyzr mobile application asks users for permission to execute IP

traceroutes, since ICMP packet transmission on a mobile device requires access to raw sockets.

Privacy Protection: Netalyzr asks users to opt in to the data collection process before installing the application. Therefore, if users are uncomfortable with sharing the measurement results with Netalyzr, they may not install the application. However, when the user grants permissions to the application, Netalyzr could use GPS to get device location, read phone status and identity, and modify or delete the contents of USB storage to store or delete measurement related data on the device.

Limitations: Although Netalyzr provides a robust diagnostic set of end-to-end network measurements and helps users troubleshoot networks, unlike MITATE, or WindRider, Netalyzr does not detect traffic shaping in mobile ISPs.

PortoLan PortoLan is a network experiment testbed based on volunteered mobile devices that executes experiments submitted to back-end servers [156]. PortoLan is designed by Enrico Gregori *et al.* at Istituto di Informatica e Telematica, to discover Internet topology and build wide scale mobile network signal quality maps. The Android application for PortoLan is available on Google Play and allows users to run measurement tests like ping, traceroutes, maximum throughput, and detection of traffic shaping of BitTorrent traffic [22]. The PortoLan team intends to add capability to support active network experiments and access to mobile sensor data such as network signal strength, device location, network name, cell type, and roaming status. PortoLan relies on user altruism to build testbed capacity and support measurement. The PortoLan mobile application limits the device cellular bandwidth usage to 2 MB/day and postpones experimentation when battery drops below 40%.

Finally, the application does not collect personally identifiable information from the device and anonymously stores measurement data on backend servers.

Conclusions

This survey provides a comprehensive overview of the existing and emerging end-to-end mobile network measurement testbeds, tools, and services. In spite of the relative maturity of existing platforms, several functionality gaps remain with respect to the needs of developers, researchers, network operators, and regulators in assessing mobile network performance. First, existing tools do not adequately support detection of traffic shaping. As depicted in Table 3.1, testbeds such as MITATE, Seattle, PhoneLab, PortoLan, and WindRider can detect the presence of traffic shaping mechanisms in mobile ISPs, whereas, other testbeds do not. Second, device churn inherent in platforms based on ad-hoc user participation means that existing tools are not well-suited for long-term network performance monitoring. In fact, the popularity of tools such as PhoneLab and FCC has declined over time. Third, several testbeds enable developers to prototype the performance of their applications ahead of deployment. However there is significant disparity in how testbeds provide that functionality in terms of execution models and APIs. Finally, exchange of P2P traffic, network diagnostics, ICMP traceroutes, device selection criteria, and NAT traversal are also supported by different platforms. One significant axis of comparison between network measurement platforms not discussed in this survey is their accuracy. The variety of measurement methods used to obtain even the relatively standard network metrics, such as throughput, makes it difficult to compare the relative accuracy of the different platforms.

Based on the surveyed work, I believe the mobile network measurement community needs a more concerted effort among developers, researchers, network operators, and regulators to produce network measurement tools that meet the needs of all four communities. A more concerted effort would lead to greater adoption of (perhaps fewer) tools, as well as large-scale and long-term network monitoring. At the same time, funding agencies should support development of new measurement approaches and capabilities, especially when such improvements are aimed at enhancement of existing testbeds.

	Network Testbeds						Network Tools							Network Services						
	Uncurated			Curated			Standalones					Libraries		Network Monitoring				Network Discovery & Diagnosis		
	MITATE	Seattle	PhantomNet	PhoneLab	SciWiNet	LiveLabs	FCC SpeedTest	WindRider	MySpeedTest	Mobitest	RILAnalyzer	Mobiperf	ALICE	Ookla SpeedTest	RadioOpt	OpenSignal	NetPerform	NDT	Netalyzr	PortoLan
Measurement Capabilities																				
Traffic shaping/DPI	✓	✓		✓				✓												✓
Active measurements	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
Passive measurements				✓		✓		✓	✓		✓	✓	✓			✓	✓			
Measurement data publicly available	2			✓			2					✓	✓			✓		✓		
Custom packet content	✓	✓		✓						✓				1						
Peer-to-peer traffic	2	✓	✓																	
ICMP traceroutes	2			✓								✓	✓						3	✓
Programmable execution environment	4	✓	✓	✓		✓							✓						✓	
Access to mobile device sensor data	✓	2		✓	✓	✓		5	6		7	5,6	5,7	5	5,6, 7	5,6, 7	5,6, 7		5,7	✓
Experiments can be scheduled on specific clients	✓	✓	✓	✓		✓				✓			✓						✓	
IPv6 support	2	2										✓							✓	✓
Allow traffic on ports < 1024	3	3		✓																
Reports network problems												✓						✓	✓	
Supported mobile OS platform	8	8,9, 12	8	8	8	8,9	8,9	10	8	8,9, 11	8	8	8	8,9, 10,13	8,9, 10,11	8,9	8,9	8	8	8
Network coverage map																✓	✓			
Device Selection Criteria																				
Geographic location	✓	✓		✓	✓	✓										Q				
Device model	✓									✓		✓								
Device type (GSM/CDMA)																Q				
Battery charge level	✓	✓		✓												Q				
Carrier signal strength	✓			✓												Q				
Network carrier	✓												✓			Q				
Network type (Wi-Fi/Cellular)	✓			✓									✓			Q				
Time of day	✓	✓	✓	✓		✓				✓			✓							
Resource Usage Limit																				
Transmission rate		✓																		
Bandwidth cap	✓	✓			✓		✓		✓			✓	✓		✓					✓
Minimum battery charge	✓	2										✓								✓
Port restrictions		✓			✓															
Misc.																				
Measurement scheduling API	✓	✓	✓			✓						✓	✓							
Supports devices behind NAT/Wi-Fi	✓	✓		✓	✓		✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
Requires rooted phones			✓	✓						✓	✓									
Open to public	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
User incentive model	R	R,A		U		S,U	C	C	C	A		C,S	S	C	C,S	C,S	C,S	S	S	A
Experiments require IRB approval				✓		✓														
Open-source	✓	✓					✓	✓	✓	✓	✓	✓						✓		
Currently active	D	✓	D	✓	D	D	✓	✓	✓	✓	✓	✓	D	✓	✓	✓	✓	✓		✓
Records hardware specs	O				✓		✓	✓		✓		✓			✓	✓	✓			
Records hardware performance	O	O		O					✓	6					✓	✓				

Legends:

1 – measurements can be directed to specific Web servers.
2 – planned functionality.
3 – on rooted phones only.
4 – through multiple experiment rounds on the same device.
5 – GPS readings.
6 – battery readings.
7 – radio state.
8 – Android.
9 – iOS.
10 – Windows.
11 – Blackberry.
12 – Nokia.
13 – Amazon FireOS
A – user Altruism to support measurement capacity.
D – under Deployment.
O – Optionally.
C – user Curiosity to understand their own network performance.
Q – only at query time.
R – Reciprocal (tit-for-tat).
S – provides Service to clients other than measurement data.

Table 3.1: Experimentation flexibility matrix of end-to-end measurement testbeds, tools, and services.

	SpeedSpot	Sensorly	RootMetrics	NetworkCoverage	Internet SpeedTest	NetRadar	Cisco Data Meter	4GMark	nPerf
Support measurement tests									
Uplink throughput	✓	✓	✓		✓	✓	✓	✓	✓
Downlink throughput	✓	✓	✓	✓	✓	✓	✓	✓	✓
Latency	✓	✓		✓	✓	✓	✓	✓	✓
Signal coverage maps		✓	✓	✓		✓			
Uplink throughput tests									
Total uplink transfer	6 MB	10 MB	7 MB		8 MB	Random	900 KB	50 MB	20 MB
Probe method	P	P	P		P	TCP	P	P	P
No. of probes	1	2	1		12	Random	1	1	20
Duration	6 s	U	8 s		15 s	10 s	U	10 s	U
Downlink throughput tests									
Total downlink transfer	20 MB	400 MB	2400 MB	10 MB	400 MB	1 MB	2 MB	250 MB	10 GB
Probe method	G	G	G	G	G	TCP	G	G	G
No. of Probes	2	4	4	1	4	16	2	1	10
Duration	U	10 s	25 s	U	15 s	10 s	U	10 s	10 s
Latency tests									
Probe Method	G	G		T	P	G	T	G	G
No. of probes	10	6		Random	Random	2	30	3	10
Measure Application performance									
Webpage load time								✓	✓
Throughput of video streams								✓	✓
Application deployment									
Experiments run against servers hosted by	MaxCDN, Edge-Cast CDNs	OVH, Digital Ocean CDNs	Amazon AWS	Think Broad-band, Emanics Lab	v-speed	Amazon AWS, CacheFly CDN	Akamai CDN	4Gmark	nperf
Supported mobile OS plat-form	Android, iOS	Android, iOS	Android, iOS	Android	Android	Android, iOS, Windows, MeeGo, Symbian, NokiaX, Jolla, and Blackberry	Android, iOS	Android, iOS	Android, iOS
Misc.									
Developed by	Speed Spot	Sensorly	Root Metrics	Technische Universitt Darmstadt	V-Speed	Aalto University	Cisco Sys-tems	Trois Petits Points	nPerf
Released in	May 2013	Aug. 2012	Aug. 2011	Mar. 2015	Oct. 2014	Feb. 2013	May 2013	May 2013	Oct. 2014
Number of app installs	> 100 K	> 50 K	> 100 K	> 100	> 1 M	> 10 K	> 50K	> 500 K	> 50 K

Legend:
U – Until transfer completes.
G – HTTP GET.
P – HTTP POST.
T – TCP Connection Setup.

Table 3.2: Experimentation flexibility matrix of emerging end-to-end measurement services.

ROLE OF DNS IN CONTENT SERVER SELECTION

Abstract

Modern websites use Content Delivery Networks (CDNs) to speed up the delivery of static content. However, I show that DNS-based selection of CDN servers can be refined to fully deliver on the speedup of CDNs. I propose DNS-Proxy (`dp`), a client-side process that shares load-balancing functionality with CDNs by choosing from among resolved CDN servers based on last mile network performance. Our measurement study of CDN infrastructure deployed by five major CDN providers shows that `dp` reduces webpage load time by 29% on average. If `dp` has already resolved the domain, the reduction in webpage load time is as much as 40%. Finally, `dp` reduces the load time of individual static Web objects by as much as 43%. I argue that `dp` enables a more effective use of existing content delivery infrastructure and represents a complementary strategy to a continual increase of geographic content availability.

Introduction

The growing competition among Internet services such as online social networks, e-commerce, or streaming video, drives developers to improve the responsiveness of their applications. Content Delivery Networks (CDNs) play a vital role in reducing the request delay to improve the user experience [164]. To increase application responsiveness and attract content providers, CDNs invest significant resources to geographically distribute their content servers [141]. However, users' opinions on Web content delivery indicates that CDN performance could be improved to meet user expectations [86].

The speed at which static content is delivered is dominated by the network latency between clients and CDN servers from which the content is downloaded [77]

[157] [308]. Content-rich websites contain images large enough to require multiple round trips to download [319]. Further, website rendering on browsers includes dependencies, which means that static content such as image, advertisement, JavaScript, and CSS files are fetched over multiple request rounds [308] [193].

CDNs reduce the impact of network round-trip times (RTTs) on overall page load time by serving content from widely distributed servers in last-mile networks. CDNs balance the load on their servers through DNS-based server selection, where geographically distributed DNS servers resolve CDN URLs to IP addresses of nearby content servers [16] [25]. However, I discover that content distribution may not reduce the network latency as expected, because DNS-based server selection does not always direct clients to the closest available content server, for various reasons such as content availability, load-balancing, cost of bandwidth, etc.

I performed an extensive measurement effort to evaluate the performance of CDN infrastructure deployed by Akamai Technologies and Google Inc. and discovered the following four limitations of the current DNS-based server selection mechanisms.

- The end-to-end latencies between a client and the CDN servers returned in a DNS resolution may have a high variation.

- The end-to-end latency between a client and a CDN server resolved by one DNS server could be remarkably lower than the end-to-end latency between the same client and a different CDN server resolved by another DNS server. Although such differences in end-to-end latency are to be expected in general, I show that they are domain dependent, in that the same DNS does not always provide the fastest CDN server for a given client for every resolved Web domain.

- A DNS server may direct a client to unnecessarily distant CDN servers when closer CDN servers are available. Again, although such direction may result from intended load-balancing, I show how their negative impact may be avoided.

- DNS-based server selection may occasionally direct clients to CDN servers inaccessible from clients' network, which results in the failure to access static Web content.

I conclude that current implementation of DNS-based server selection should be improved, in that it should take full advantage of geographic server availability to reduce the impact of network RTT on overall webpage load time.

Based on our measurement study I argue that clients are best-positioned in the network to measure CDN performance and participate in the selection of the best CDN replica. It is therefore timely to explore solutions, where server selection is shared by clients and CDNs to both minimize the network delay and balance server load. I propose DNS-Proxy (`dp`), that complements DNS-based server selection with client measurement from the last-mile networks. `dp` implements a lightweight parallel probing mechanism to probe resolved CDN servers, to direct clients to the fastest available CDN server.

Our results show that, `dp` reduces the webpage load time by 29% on average. If a request for a domain is already resolved by `dp`, the webpage load time is reduced by as much as 40%. Finally, `dp` reduces the load time of individual static Web objects by as much as 43%. To the best of our knowledge, `dp` is the first step in this direction that extends the CDN replica selection functionality to client devices in last-mile networks on a per-domain granularity. Our experience with `dp` shows that load balancing can be shared effectively between CDNs and client devices in end-networks. I make `dp` available as an open source tool at `http://github.com/msu-netlab/dp`.

Although some CDNs may use anycast for global load-balancing [73], the goal of this study is to understand the impact of DNS-based server selection used by most major CDNs.

The rest of the chapter is organized as follows. In Section 4, I describe our experimental setup and discuss the impact of current DNS-based server selection techniques on Web performance. In Section 4, I discuss the implementation of `dp` as a tool for client-assisted server selection. Section 6 describes our evaluation results. In Section 5 I offer a discussion of `dp`'s path to deployment. In Section 4 I outline the related work on reducing network latency for Web applications. Finally, I conclude in Section 4.

DNS-based Load Balancing

To discover the limitations of current DNS-based server selection techniques and to understand their impact on Web performance, I configured several experiments on 123 devices, made available by the Dasu testbed, in different last-mile networks across different geographic areas [276]. Our measurement data contains 887 DNS resolutions from 386 DNS servers and 9,040 TCP and HTTP GET probes from clients on different continents to 1684 distinct CDN servers. I show the geographic distribution of our test devices in Table 4.1.

Experimental Setup

I measured the difference in end-to-end latency and download time of static content between clients and CDN servers returned by different DNS servers. I configured experiments on each device to download images from each of the resolved CDN servers, and record the time to establish TCP connection, time to receive the first bit of the HTTP response, and the time to download the image. Each device

CONTINENT	# CLIENT DEVICES
North America	54
Europe	35
Asia	14
Australia	10
Africa	6
South America	4

Table 4.1: Geographic distribution of Dasu nodes.

was configured to download a 77 KB image hosted on Akamai's CDN and a 118 KB image hosted on Google's CDN.

I configured each device to send a DNS query to its default (local) DNS server (LDNS), the Google's public DNS server `8.8.8.8` (GDNS), an open DNS server `208.67.222.222` (ODNS), and Level3's public DNS server `209.244.0.3` (L3DNS) to resolve domain names hosted by Akamai Technologies (`fbcdn-profile-a.akamaihd.net`) and Google Inc. (`lh3.googleusercontent.com`).[1] Next, each device recorded the resolutions from DNS servers, initiated a TCP connection with each CDN server in the DNS resolution, and recorded the time for TCP connection establishment. After a successful TCP connection, the device sent an HTTP HEAD request to warm up the CDN's cache and issued another HTTP HEAD request to record the time to receive first bit of HTTP HEAD response. Finally, each device sent an HTTP GET request to download the cached image and recorded the time to completely download the image.[2]

[1] I ensured that the LDNS configured on the device was not one of the open DNS servers used in our study.

[2] I ensured that the devices issued all the DNS queries and content fetches from CDN servers within a very small time window.

Impact of CDN Choice on Static Content Delivery

The DNS resolutions contain a list of IP addresses of CDN servers within a single subnet. The client operating systems selects the first IP address from the list. A DNS resolution contains a list of IP addresses of CDN servers, which might suggest that latency to these servers should be similar and that selecting any of the CDN IP addresses would not impact the download time. However, I show that there is a significant difference in the end-to-end latency between the client and each of the CDN servers/clusters returned by DNS.

In Figures 4.1-4.4 I show the minimum, average, and maximum difference in the end-to-end latency (measured as time to establish a TCP connection) among Akamai CDN servers resolved by LDNS, GDNS, ODNS, and L3DNS. Similarly, in Figures 4.5-4.8, I show latency difference among Google CDN servers. These graphs show that the end-to-end latency between the client and the CDN servers returned in the list has a high variation. For example, in Figure 4.1 I see that if clients always pick the server with the least end-to-end latency, 80 percent of the clients will have an end-to-end latency within 50 ms. However, since clients' operating systems pick the first CDN IP address from the list of resolved CDN servers, which results in a random CDN selection over time, I see that 80 percent of the clients will have an end-to-end latency within 100 ms. Therefore, if clients choose the first CDN IP present in the list they may not connect with the fastest server available, since the server with the lowest end-to-end latency with the client might not be the first server in the list.

Content providers are interested in understanding the impact of server selection on static content download times from CDNs. Therefore, next I show the impact of CDN choice on the image download time. In Figures 4.9-4.12, I show the minimum, average, and maximum image download time from Akamai CDNs returned by LDNS,

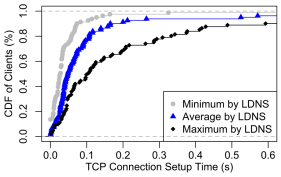


Figure 4.1: Latency to Akamai servers resolved by LDNS.

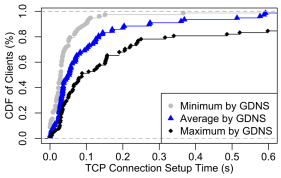


Figure 4.2: Latency to Akamai servers resolved by GDNS.

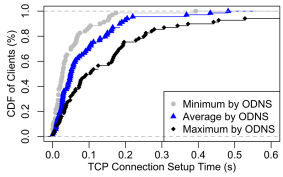


Figure 4.3: Latency to Akamai servers resolved by ODNS.

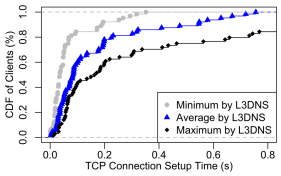


Figure 4.4: Latency to Akamai servers resolved by L3DNS.

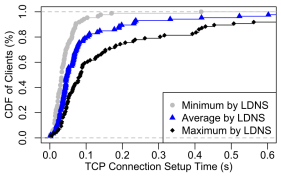


Figure 4.5: Latency to Google servers resolved by LDNS.

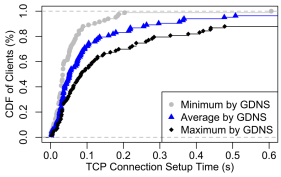


Figure 4.6: Latency to Google servers resolved by GDNS.

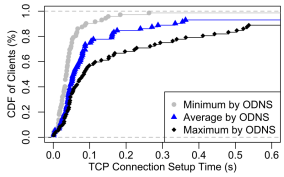


Figure 4.7: Latency to Google servers resolved by ODNS.

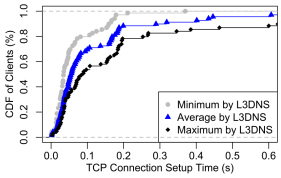


Figure 4.8: Latency to Google servers resolved by L3DNS.

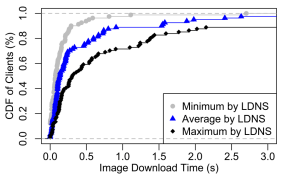


Figure 4.9: Download time for Akamai servers resolved by LDNS.

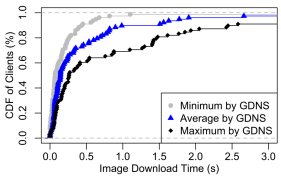


Figure 4.10: Download time for Akamai servers resolved by GDNS.

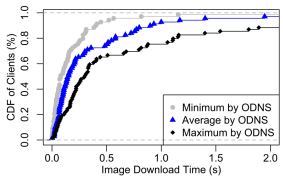


Figure 4.11: Download time for Akamai servers resolved by ODNS.

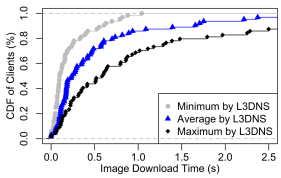


Figure 4.12: Download time for Akamai servers resolved by L3DNS.

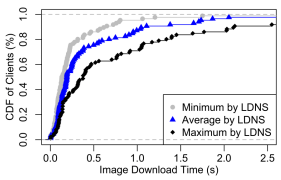


Figure 4.13: Download time for Google servers resolved by LDNS.

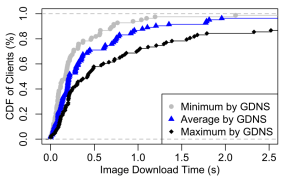


Figure 4.14: Download time for Google servers resolved by GDNS.

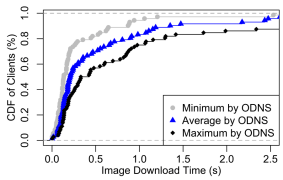


Figure 4.15: Download time for Google servers resolved by ODNS.

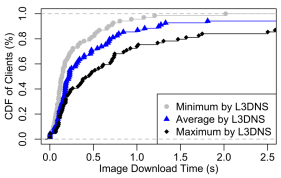


Figure 4.16: Download time for Google servers resolved by L3DNS.

GDNS, ODNS, and L3DNS. Similarly, in Figures 4.13-4.16, I show the minimum, average, and maximum image download time from Google CDNs returned by LDNS, GDNS, ODNS, and L3DNS. Similarly to previous graphs representing the variation in end-to-end latency, I show a variation in the download time of individual static Web objects. For example, in Figure 4.9, the minimum image download time for 80% of the clients is within 250 ms, however, the average image download time is within 750 ms. The difference in minimum and average image download time is within 500 ms for 80% of clients, which is due to the multiple round trips between clients and CDN servers involved.

Impact of DNS Choice on Static Content Delivery

Analogous to the choice among CDN servers, clients also have a choice between DNS servers. Clients have a number of options from which to chose their default DNS servers. Further, some DNS providers may process EDNS-based DNS requests [288], while others may strip out any information available in the EDNS payload [271]. Thus, the variation in the adoption of EDNS mechanisms may also introduce additional variation in the performance of the CDN servers returned in a DNS resolution, since DNS servers that process EDNS-based requests could use the client's IP address available in the EDNS payload to direct the user to a CDN server nearest to the client's subnet. Finally, using a client's IP address from the EDNS payload may direct the client to the closest CDN server, however, the client may not have the least end-to-end latency with that CDN server due to congestion in the network, large queues on the server, or circuitous routing. Therefore, it is important to understand the performance of CDN servers returned by different DNS providers.

In Figures 4.17-4.20, I compare the end-to-end latency and the time to download images from Akamai CDN servers returned by LDNS, GDNS, ODNS, and L3DNS.

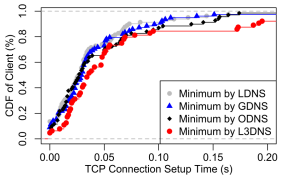


Figure 4.17: Min. end-to-end latency from Akamai CDNs.

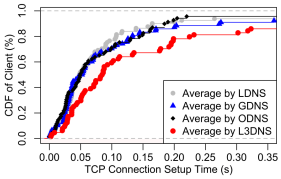


Figure 4.18: Avg. end-to-end latency from Akamai CDNs

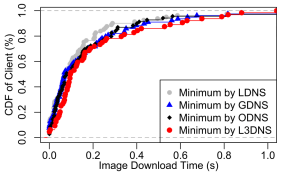


Figure 4.19: Min. Image download time from Akamai CDNs

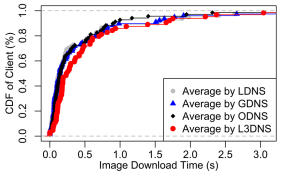


Figure 4.20: Avg. Image download time from Akamai CDNs

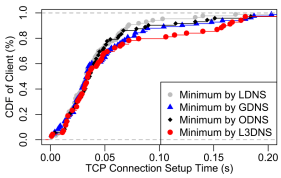


Figure 4.21: Min. end-to-end latency for Google CDNs.

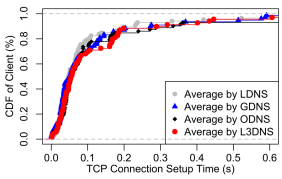


Figure 4.22: Avg. end-to-end latency for Google CDNs

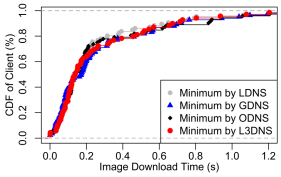


Figure 4.23: Min. Image download time for Google CDNs

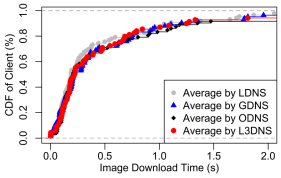


Figure 4.24: Avg. Image download time for Google CDNs

Similarly, in Figures 4.21-4.24, I compare the end-to-end latency and the time to download images from Google CDN servers returned by LDNS, GDNS, ODNS, and L3DNS. These figures show that the end-to-end latency between clients and CDN servers returned by one DNS server is lower than CDN servers returned by another DNS server. For example, in Figure 4.18 I see that for 80% of the clients that resolve Akamai CDN domains from LDNS, GDNS, ODNS, L3DNS have an average end-to-end latency within 100 ms, 125 ms, 150 ms, and 300 ms respectively. Similarly, in Figure 4.22 I see that for 80% of the clients that resolve Google CDN domains from LDNS, GDNS, ODNS, L3DNS have an average end-to-end latency within 100 ms, 150 ms, 175 ms, and 200 ms respectively. Such variation in the end-to-end latency to CDN servers is also reflected in the time to download images from Akamai and Google CDNs. For example, in Figure 4.24 I see that for 80% of the clients the time to

download image from LDNS, GDNS, ODNS, and L3DNS is 600 ms, 700 ms, 750 ms, and 800 ms respectively.

In Figure 4.18 I also see that for 35% of clients ODNS returns CDN servers with average end-to-end latency lower than CDN servers returned by GDNS. Similarly, for about 40% of the clients in Figure 4.22 I see that the average end-to-end latency to the CDN servers returned by all DNS servers are almost similar. As a result of such variation in DNS resolutions, it remains unclear which DNS server will direct the client to the fastest server at all times. I argue that clients could remain unaware of opportunities to reduce the Web latency when they rely on one DNS server.

Overall Performance Variation of CDN servers

Given that the Web performance is affected by the choice of a CDN server within a DNS resolution and by the choice of a DNS server, I now explore how these choices could impact the Web performance in combination. I argue this is important because many users opt for open and public DNS servers that offer faster resolutions and also because LDNS does not always resolve to best CDN servers, which I show later in Figure 4.41. In Figures 4.25 and 4.29 I compare the minimum, average, and maximum end-to-end latency between clients and the CDN servers resolved by any of the DNS servers used in our study. In these graphs I show that the minimum end-to-end latency to CDN servers, when resolutions from LDNS, GDNS, ODNS, and L3DNS are combined, is significantly lower than the average end-to-end latency. For example, in Figure 4.25, I show that for 90% of the clients the minimum end-to-end latency to Akamai CDN servers, resolved by any of the DNS servers, is less than 50 ms and the average end-to-end latency is more than 200 ms. I also show similar trend in end-to-end latency for Google CDNs and the image download time for Google and Akamai CDNs in Figures 4.29, 4.26, and 4.30. Therefore, client applications that rely

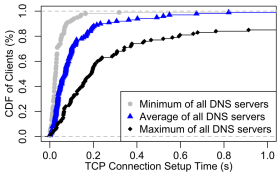


Figure 4.25: Latency variation with Akamai CDNs.

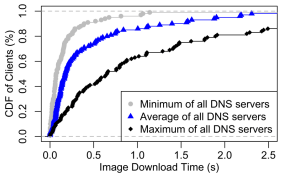


Figure 4.26: Download time variation with Akamai CDNs.

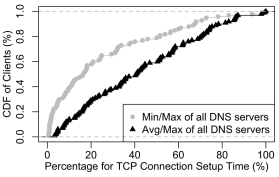


Figure 4.27: Latency variation for Akamai CDNs (%).

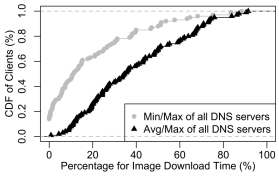


Figure 4.28: Download time variation for Akamai CDNs (%).

on resolutions from multiple DNS providers, as proposed by Vulimiri *et al.*, may not connect with CDN servers that have the least end-to-end latency to the client [306].

Next, in Figures 4.27 and 4.31 I show the ratio of minimum end-to-end latency to maximum end-to-end latency for Akamai and Google CDN servers returned by DNS servers in our study. For example, in Figure 4.27, I show that for the 85% of users connecting to Akamai CDN servers, the end-to-end latency for the fastest available server is 40% lower than the slowest server. Similarly, in Figure 4.31, the end-to-end latency to the fastest available Google CDN server is only about 30% lower than the slowest server.

I also show similar ratios for image download time from Akamai and Google CDN servers in Figures 4.28 and 4.32. For example, in Figure 4.28, I show that for 90% of the users the image download time from the fastest available Akamai CDN server is only 50% faster than the slowest server. Similarly, in Figure 4.32, the image download time from the fastest available Google CDN server is 30% faster than the slowest server.

Causes of CDN Performance Variation

In previous sections, I discovered the variation in the performance of CDN servers resolved by different DNS providers. While investigating the cause of this variation,

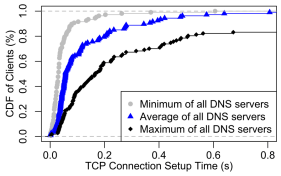


Figure 4.29: Latency variation with Google CDNs.

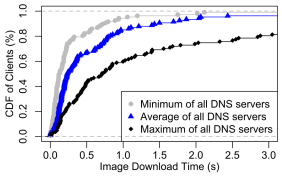


Figure 4.30: Download time variation with Google CDNs.

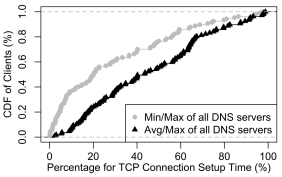


Figure 4.31: Latency variation for Google CDNs (%).

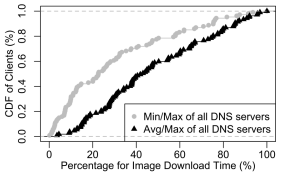


Figure 4.32: Download time variation for Google CDNs (%).

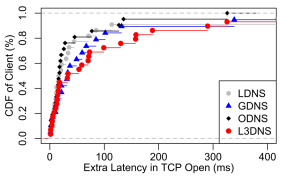


Figure 4.33: Extra latency to Akamai CDNs.

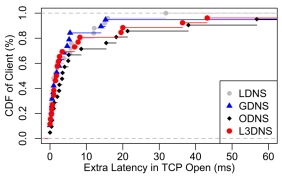


Figure 4.34: Extra latency to Google CDNs.

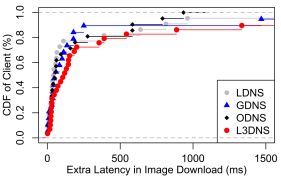


Figure 4.35: Extra download time to Akamai CDNs.

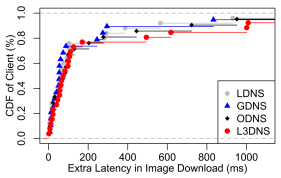


Figure 4.36: Extra download time to Google CDNs.

I also found that DNS resolutions often direct clients to CDN servers in different IP subnets and different network locations. For example, as much as about 45 and 40 percent of the clients were directed to CDN servers in different IP subnets when resolving Akamai and Google Web domains, respectively.

Next, I investigate the impact of inconsistency in DNS resolutions on client redirection. Due to inconsistency in DNS resolutions, some clients are often directed to servers in different geographic locations. For example, a client in Virginia was directed to servers in Cambridge, California, and Texas in three different DNS resolutions from the same DNS server. Therefore, I argue that when DNS resolutions are inconsistent, clients' Web requests may have to be served by CDN servers in locations farther than the closest available CDN server. Although this behavior is maybe due to routine load balancing, let us look at its impact on extra latency perceived by clients in connecting with CDN servers.

Our method to calculate extra end-to-end latency for each client is based on the difference in minimum and mean latency to servers in different IP subnets. In Figures 4.33 and 4.34, I show the extra end-to-end latency as perceived by clients to connect with Akamai and Google CDN servers, respectively. For example, In Figure 4.33 I show that for 15% of the clients that receive DNS resolutions to different IP subnets from LDNS and L3DNS for Akamai CDN domains, the extra end-to-end latency perceived by clients in every round trip is more than 100 ms and 150 ms respectively. Similarly, in Figure 4.34, I show that for 15% of the clients, that receive DNS resolutions to different IP subnets from LDNS and L3DNS for Google CDN domains, the extra end-to-end latency perceived by clients in every round trip is more than 15 ms and 20 ms respectively.

In Figures 4.35 and 4.36, I show the extra latency in downloading images as perceived by clients in connecting with Akamai and Google CDN servers, respectively. For example, in Figure 4.35 I show that for 80% of the clients, that receive inconsistent DNS resolutions from LDNS, GDNS, ODNS, and L3DNS, the extra latency in download image from Akamai CDN servers is within 100 ms, 200 ms, 300 ms, and 400 ms respectively. Similarly, in Figure 4.36, I show that for the same clients the extra latency in downloading an image from Google CDN servers is within 300 ms for LDNS, GDNS, and ODNS, and 500 ms for L3DNS. Therefore, if clients rely on resolutions from the regular DNS-based server selection, the penalty in terms of latency is very high. In Section 4, I show how `dp` eliminates the penalty while preserving load-balancing.

Unreachable CDN Servers

Client devices and end-networks are often configured with firewalls to block access to some IP addresses. Automated Intrusion Detection System (IDS) software scans for any activity that might abuse the network resources. Such IDS may

CLIENT COUNTRY	DNS	*#faulty* RESOLUTIONS	
		Akamai	Google
North America	LDNS	2	6
North America	GDNS	3	
North America	ODNS	5	3
North America	L3DNS		11
Europe	LDNS	2	
Europe	GDNS	1	1
Asia	LDNS	5	
Asia	GDNS	3	6
Asia	ODNS		4
Asia	L3DNS	1	
Australia	LDNS	2	
Africa	GDNS		3

Table 4.2: Faulty resolutions for Akamai and Google CDN servers.

occasionally block access to some CDN servers, a behavior I verified in MSU's campus network. DNS providers remain unaware of such network configurations and therefore when clients request DNS resolutions, they occasionally get directed to inaccessible servers, which results in failure to start the downloads of static Web content [86] [243]. Although server inaccessibility prevents the content from being downloaded all together, I believe that this could be avoided by vetting CDN servers.

Throughout our study of over a period of three months, I recorded the number of *faulty* DNS resolutions, that is, whether a DNS resolution contained at least one server address to which client could not connect. I show the number of *faulty* DNS resolutions for Akamai and Google CDNs by LDNS, GDNS, ODNS, and L3DNS for clients in different countries in Table 4.2.

Out of the 123 DASU devices used in our study, I found that only about 15 devices in different continents received at least one DNS resolution that had at least one inaccessible CDN server. Specifically, for such clients in North America I found

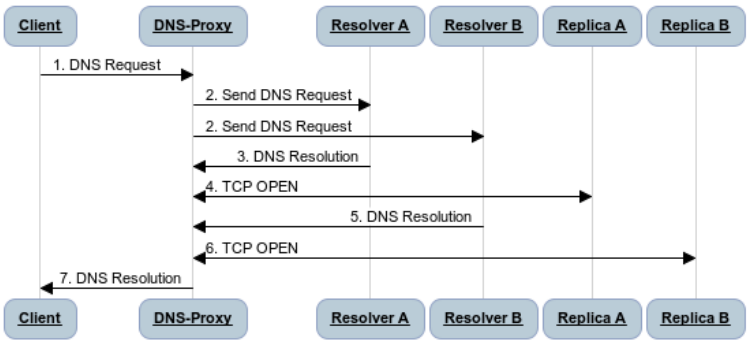


Figure 4.37: dp's resolution mechanism resolution for non-cached domains.

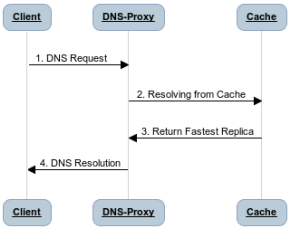


Figure 4.38: dp's resolution mechanism for cached domains.

that these clients were connected through Comcast Cable, Florida Cable, Bright House Networks, Cox Communications, VTX Broadband, and Time Warner Cable.

I show that the problem of faulty DNS resolutions exists for both Akamai and Google CDN infrastructure. Further, to mitigate client direction from faulty DNS resolutions, the use of multiple DNS providers may not be useful since I discovered that popular DNS providers such as GDNS, ODNS, and L3DNS and as well as clients' LDNS sometimes direct clients to the same inaccessible CDN servers. In Section 4, I show that dp eliminates this problem by making DNS resolutions aware of the performance and routing restrictions in the last-mile networks.

DNS-Proxy (dp)

Our measurement study has discovered that DNS-based load-balancing used by CDN infrastructures deployed by Akamai and Google often do not direct users to the closest CDN servers available. Therefore, I propose DNS-Proxy (dp), a client-side tool that selects best CDN servers with respect to performance of users' last mile networks. dp can also be used as a DNS server on network gateways to serve incoming DNS requests from multiple devices in the same subnet. dp runs as a virtual DNS server on

client devices and generates DNS resolutions that are most suitable for the variable performance of the user's network. `dp` receives DNS requests from client applications and fans them out to different DNS servers. `dp` then probes all resolved IP addresses in parallel and returns the CDN server with the lowest end-to-end latency to the client. I make `dp` available as an open source tool at `http://github.com/msu-netlab/dp`.

Client-assisted Server Selection

I depict sequence diagrams of `dp`'s approach to client-assisted server selection in Figures 4.37 and 4.38. `dp` runs on client devices and listens for incoming DNS requests on port 53, the standard port for DNS-based services. When `dp` receives a DNS request from a client, it forwards the request to a number of different DNS servers and waits for the DNS resolution replies. The DNS servers that the `dp` forwards the request to can be easily configured based on the user's preference. In our experiments, I configured `dp` to resolve DNS requests from LDNS, GDNS, ODNS, and L3DNS. As shown in Figure 4.37, after `dp` receives a DNS resolution, it sends TCP SYN packets, in parallel over raw sockets, to port 80 (standard port for hosting HTTP based services) and port 443 (standard port for hosting HTTPS based services) to each resolved CDN server. To prevent `dp` from inadvertently launching a SYN attack, `dp` sends a FIN packet after receiving a SYN/ACK for each SYN packet, or after a timeout.

The end-to-end latency to each CDN server is measured based on the time to receive the TCP SYN/ACK packet from the probed server. `dp` collects the end-to-end latency to each server and maps the domain name being resolved to the CDN server with the lowest end-to-end latency. `dp` then returns the server with the lowest end-to-end latency identified, as a resolution for the client's DNS request. Apart from directing clients to the fastest available CDN servers, `dp` prevents directing clients to the servers for which it never receives a TCP SYN/ACK packet. However, if a

DNS resolution contains only one IP address, `dp` directs the client to that IP address, regardless of it being inaccessible.

`dp` sends DNS responses to clients using one of two methods. `dp` resolves domain names either from its own cache of DNS entries (I refer it as *warm-cache*), or delays the DNS response for a domain not in its cache to probe for the fastest server on the fly (I refer it as *cold-cache*). As shown in Figure 4.38, when `dp` has a DNS entry for the domain being resolved in its cache, `dp` instantly replies to the clients with a DNS resolution, which also allows to reduce the overhead for name resolution [279]. However, when a DNS entry is not available in the cache, `dp` relies on a user configurable deadline (set to 30 ms by default) within which `dp` must reply the client's DNS request with the fastest identified CDN server. The user configurable deadline ensures that users' DNS requests do not have to wait if the domain being resolved is not available in `dp`'s cache, or if probing different CDN servers take a long time. While the use of a deadline in `dp` predictable response times, `dp` also continues to probe additional CDN servers after the deadline to refine its accuracy of server selection for future requests.

`dp` sets the time to live (TTL) value in the DNS response for each DNS resolution generated from the *cold-cache* to two seconds. A low TTL value in the DNS response allows clients to use a resolved server (from `dp`'s *cold-cache*) for only a short period of time before reissuing the DNS query. I expect `dp` to have probed all resolved CDN servers and identified the fastest available CDN server within two seconds. At this point `dp` will respond to the second DNS query from its *warm-cache*. However, the TTLs for DNS responses generated from the *warm-cache* are larger than two seconds, but lower than the actual TTL values present in responses from different DNS servers. Further, `dp` deletes DNS entries from its cache for any domain that has been cached for more than DNS TTL, which enables `dp` to proactively identify the best CDN

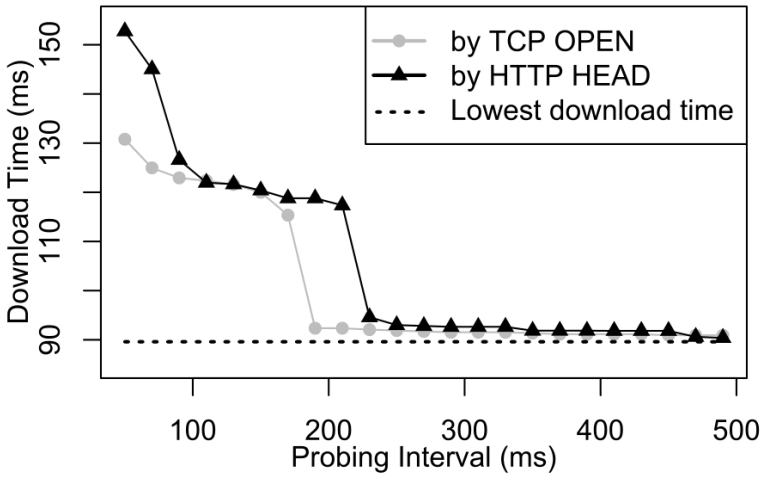


Figure 4.39: TCP OPEN vs HTTP HEAD for Akamai CDNs at different probing intervals.

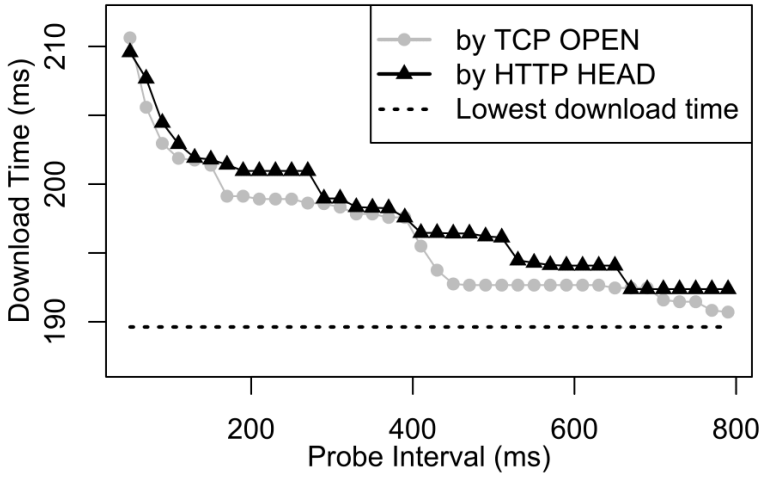


Figure 4.40: TCP OPEN vs HTTP HEAD for Google CDNs at different probing intervals.

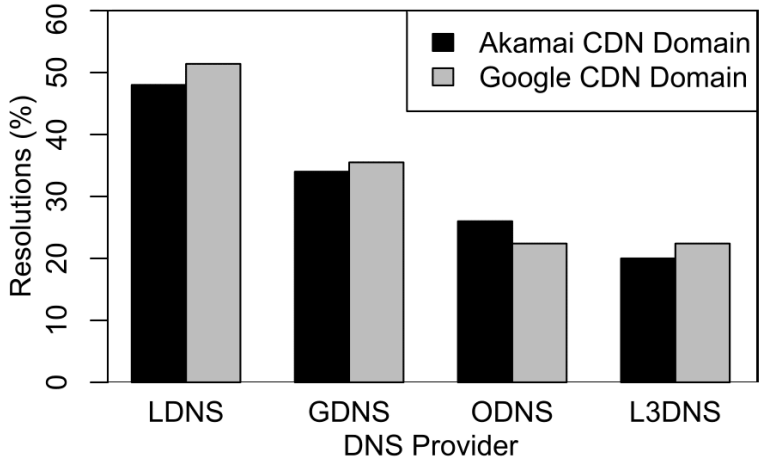


Figure 4.41: Frequency of different DNS servers resolving to fastest CDN servers, as identified by `dp`.

server, if the performance of the previously identified best server had changed since it was last probed [247].

Probing Metric

Our decision to use TCP connection setup time in `dp` as indicative of object download time from CDN server is based on the data collected from Dasu devices. Predicting the fastest available CDN server based on the actual least download time of static Web objects would reflect the true performance for a given CDN server, however, downloading Web objects from each resolved CDN server would introduce a high probing traffic and long probing delay. Therefore, I argue that the prediction of fastest available server should be based on a server selection mechanism that is faster and requires less probing traffic. I evaluate both the time to receive the first bit of HTTP response (I refer it as the HTTP HEAD method) from CDN servers and the time to establish a successful TCP connection with a CDN server (I refer it to as the TCP OPEN method).

To find an appropriate method for server selection, I compare the download time of fastest server identified by TCP OPEN and HTTP HEAD at different `dp` response deadlines. I refer to `dp`'s deadlines as probing interval before step 7 in Figure 4.37. In Figures 4.39 and 4.40, I compare the image download time from fastest CDN servers for Akamai and Google CDN domains, as identified using the TCP OPEN and HTTP HEAD methods. The x-axis shows `dp`'s probing interval following a client DNS request. The y-axis shows the average of image download times from the fastest CDN servers identified at different probing intervals of `dp`. The dashed horizontal line shows the download time from the fastest CDN server averaged over all clients, regardless of the server selection method and `dp`'s probing interval.

I show that as the probing interval increases `dp` continues to listen for additional resolved CDN servers to refine accuracy of client-assisted server selection for future requests. I also note that 1) At any given probing interval, the TCP OPEN method is more accurate on average than the HTTP HEAD method, because TCP OPEN receives higher percentage of probes back in a given interval, and 2) the line representing TCP OPEN method tends towards the dashed line, which indicates that the download time of the server chosen by the TCP OPEN method is approximately that of the server with the least image download time. For example, in Figure 4.39, the image download time of the fastest CDN server identified by the TCP OPEN method at the probing interval of 180 ms is 90 ms, whereas, the image download time of the fastest server identified by the HTTP HEAD method is 120 ms at the same probing interval.

Although probing may introduce an extra load on the client's network, it is important to note that `dp` caches probe results to avoid probing same servers in subsequent requests. Further, since `dp` reduces the number of DNS requests leaving the network or the client device by resolving them from it's cache, the overall network load is reduced. Finally, our evaluation of `dp` on a 24-hour DNS trace collected from the MSU's network shows that the network traffic sent and received by `dp` is less than 700 KB for every 500 resolved Web domains. I argue that in comparison to the benefits `dp` brings for clients, the `dp` probing traffic is reasonably negligible.

Results

Next I demonstrate, based on extensive measurement of `dp`, that client-assisted server selection is more effective at identifying fastest available CDN servers and reducing webpage load times across CDNs, last mile networks, and geographic locations, than the current DNS-based server selection techniques.

Identifying DNSs with Fastest CDN Servers

In Figure 4.41, I show whether the fastest CDN server chosen by `dp` was resolved from LDNS, GDNS, ODNS, or L3DNS. I show that for resolving Akamai CDN domains, only 48% of resolutions from LDNS contained the fastest CDN server, whereas, while resolving Google CDN domains, 51% of resolutions from LDNS contained the fastest CDN server. I also discovered that resolutions from different DNS providers may both contain one or more common IP addresses, which were identified as the fastest server by `dp` in some resolutions. For such resolutions, I increment the height of bar for each DNS provider that contained the fastest CDN server, which is indicative of the fact that the sum of heights of the bar plots do not aggregate to 100%. Since none of the DNS servers used in our study are reliable in directing clients to the closest available CDN replicas at all times, I argue that `dp` provides a complementary approach to DNS-based server selection by relying on multiple DNS providers and directing the clients to the fastest available CDN replicas from resolved CDN addresses at all times.

Faster Web through DNS-Proxy

I compare the webpage load time of websites hosted on Akamai, Google, Level 3, CloudFront, and Reflected Networks CDN servers, when clients use their ISP-provided LDNS and `dp`. I list the website addresses, number of Web objects, the hosting CDN, and the total page size in Table 4.3. I configured experiments on devices in California (CA), Montana (MT), Illinois (IL), and New York (NY) to load these websites 20 times each from CDNs resolved by LDNS and `dp`. To prevent object loading from browser cache, I loaded websites in Google Chrome browser's incognito window.

In Figure 4.42, I show the average webpage load time of different websites using LDNS and `dp` with *warm-cache* from CA, MT, IL, and NY. Above every bar, I

Webpage	Host CDN	# Web objects	Page Size
`huffingtonpost.com`	Akamai	374	3.4 MB
`developer.android.com/tv/`	Google	60	6.8 MB
`level3.com`	Level 3	58	1.4 MB
`chictopia.com`	CloudFront	90	1.8 MB
an adult "tube" site	Reflected N/w	70	7.6 MB

Table 4.3: Details of webpages loaded for comparison.

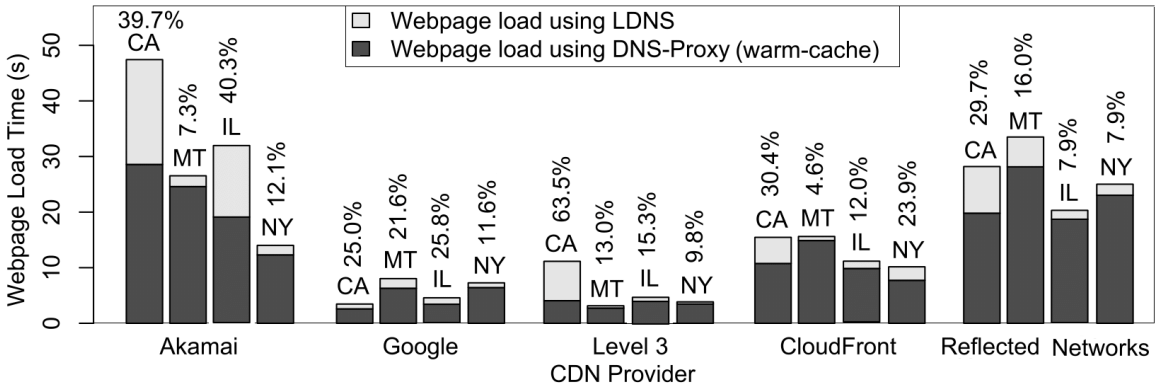


Figure 4.42: Comparion of Webpage load times using LDNS and DNS-Proxywith *warm-cache*.

show the reduction in webpage load time in percentage achieved when using `dp`. `dp` speeds up page loads across CDN providers and geographic locations. Specifically, for static content heavy website, such as *huffingtonpost.com*, `dp` provides a speedup of as much as 40%. For other websites with relatively fewer or smaller images, `dp` provides speedup close to 30% in the common case.

In Figure 4.43 I compare the average load times of individual Web objects, hosted on different websites, when loaded from servers resolved by LDNS and `dp` with *warm-cache*. I loaded a total of 35443 Web objects hosted by different CDN providers. I show the average object load time (which includes the DNS lookup time, TCP connection setup time, time to receive the first bit of HTTP response, and time to download the object) using LDNS and `dp` and label each bar with the average percentage reduction in individual object load time achieved by `dp` as compared to LDNS. I show that even when `dp` does not have a resolution for a domain in its cache, `dp` is effective in reducing the webpage load time by delaying DNS responses to identify fastest CDN servers. For example, `dp` reduces the load time of each web object hosted on CloudFront CDN servers by about 43%. For web objects hosted on other CDNs, `dp` reduces the load time for each object by about 20% on average.

Finally, in Figure 4.44, I compare the average of 20 webpage load times each from CDNs resolved by using LDNS and `dp` with *cold-cache*. For this comparison study, I configured `dp`'s DNS resolution deadline to 30 ms and cleared `dp`'s cache after each page load. I show that even when `dp` does not have a DNS resolution for a Web domain available in its cache, delaying DNS resolutions by 30 ms to identify faster servers helps to reduce the overall webpage load time. I show the average percentage reduction in webpage load times using `dp` with *cold-cache* on top of each `dp` bar in the graph. The one exception in our data is for webpages hosted on Google CDNs for which the average webpage load time with `dp`'s *cold-cache* was marginally higher than

the average load time with LDNS. I believe that in most cases `dp` could eliminate the *cold-cache* penalty through dynamic adjustment of DNS response deadline or to not delay DNS responses for domains where `dp` shows no gains for overall webpage load time – a subject of our future work.

Discussion

Two possible concerns come to mind with widespread adoption of `dp`. First, would `dp` probing introduce higher loads on CDN servers thereby increasing queuing delays? Because `dp` uses the light-weight TCP OPEN process and immediately closes the connections after they are established, I believe that the increase in network load, or CDN server resources due to probing is not significant.

Second, does client-based server selection disregards the existing DNS-based load balancing? Because `dp` selects CDN servers only from among the set resolved by DNS infrastructure, CDNs retain control over which replica server a client may connect to. As a result `dp` clients in different network locations will probe different sets of CDN servers. While it is possible that clients in the same location might select the same fastest CDN server and potentially increase its queuing delays, I plan to extend `dp` to avoid this situation by switching to HTTP HEAD-based probing, which takes server queuing delays into account.

Although I have demonstrated the benefits of `dp` in accelerating Web services hosted on CDNs, our method is applicable to other replicated server selection problems. To facilitate wide `dp` deployment I plan on integrating our method with `bind` DNS software commonly used in many last mile networks. Currently I make a standalone implementation of `dp` available at `http://github.com/msu-netlab/dp`.

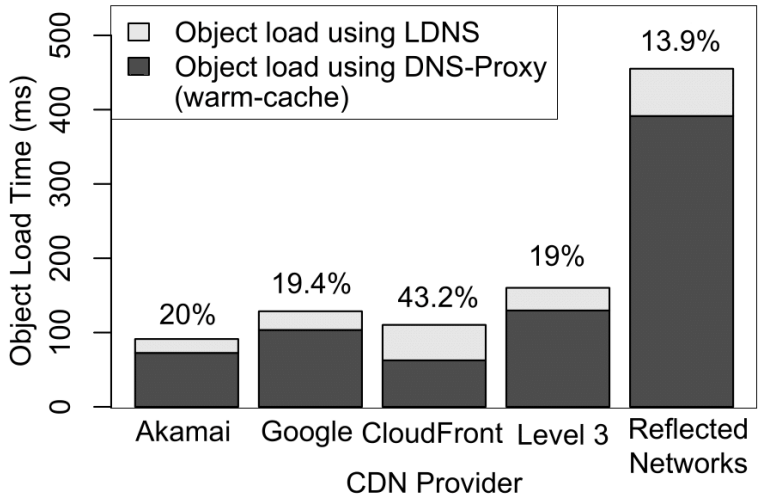


Figure 4.43: Comparison of Web object load times using LDNS and DNS-Proxy with *warm-cache*.

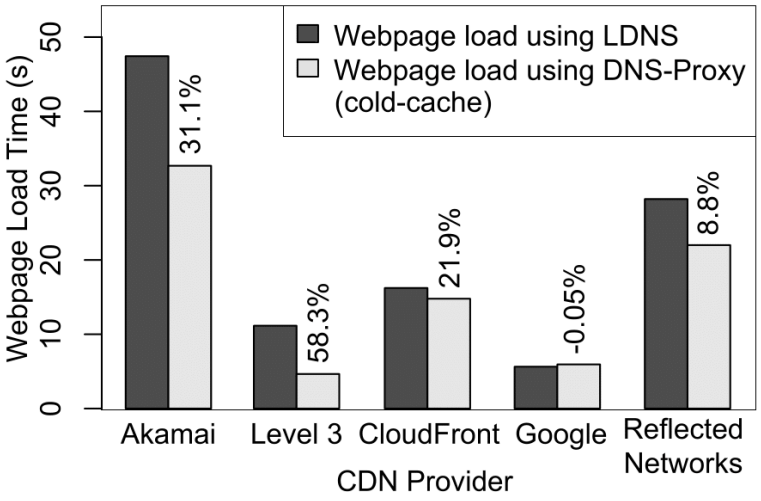


Figure 4.44: Comparison of Web object load times using LDNS and DNS-Proxy with *cold-cache*.

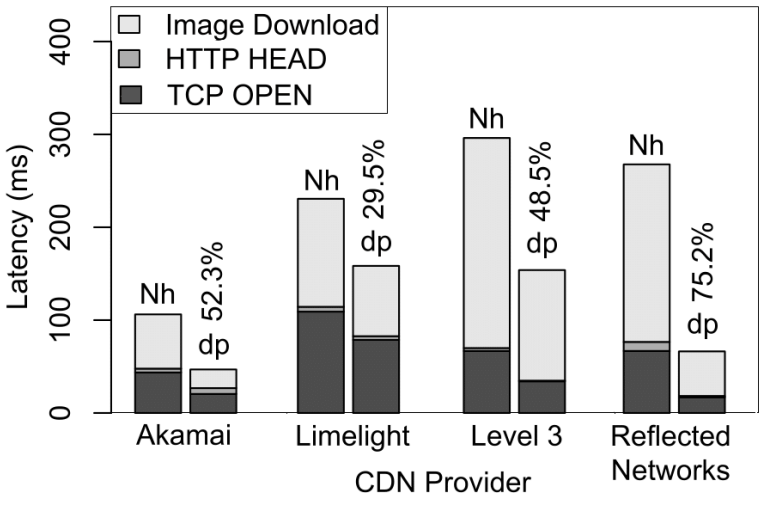


Figure 4.45: Comparison of object load times using CDN servers resolved by `dp` and Namehelp.

Related Work

In spite of several years of efforts to reduce network latency, CDNs and content providers strive to deliver a responsive experience to their users. In addition to evolving DNS-based server selection for Web applications, CDN server selection techniques for improving the delivery of live video have also been explored as an alternative to DNS-based load balancing [188] [211]. Although, server selection for video and Web content delivery are related to each other, I believe that the techniques to accurately identify the best CDN servers for both of these services need to consider different metrics (latency vs. available bandwidth), because of difference in application requirements. Therefore, I only discuss studies that aim to improve server selection for Web applications.

Reducing DNS Lookup Time

Vulimiri *et al.* proposed a client-side tool for sending DNS requests to multiple DNS servers and using the first received DNS resolution on the client [306]. CoDNS is a similar tool that distributes incoming DNS requests to multiple DNS servers to mask the delay in DNS lookups [244]. However, in Figure 4.18, I show that different DNS servers may direct clients to CDN servers with different end-to-end latencies. Therefore, directing clients to server in the first DNS resolution may not always direct clients to the closest available CDN server.

Shang *et al.* proposed a tool to reduce the DNS cache miss rate by exploiting similarity of requested Web domains to the domains that already have a DNS resolution cached [280]. DNS Pre-Resolve is another technique to eliminate DNS lookup delay by resolving domain names proactively during Web page rendering [118]. Although these two techniques reduce the impact of DNS on webpage load time,

they do not consider performance variation between CDN servers, nor accuracy of resolutions from different DNS servers.

DNS Server Selection from End-devices

Namehelp is a client-side tool to identify the DNS server that (on average) directs clients to the fastest available CDN servers [272]. Namehelp's mechanism to identify the best DNS server relies on user's Web browser history, in that, Namehelp resolves Web domains (accessed by the user in the past) from multiple DNS providers. Based on the average performance of CDN servers returned from different DNS servers for Websites used in the past, Namehelp configures the client's default DNS server to the DNS server that directed the client to servers with least latency on average. Namehelp is similar to `dp` in that it measures the client's latencies to multiple CDN servers resolved by different DNS servers for client-side server selection. However, I illustrate several differences in server selection techniques used by `dp` and Namehelp, and also show that `dp` direct clients to servers remarkably faster than servers resolved by Namehelp, on average.

- In our experience with Namehelp I discovered that in order to identify the best DNS server for a client, Namehelp sends over 27000 DNS requests (of size 80 bytes each) to different DNS servers every 15 minutes, followed by receiving a DNS response (of size 150 bytes each) for each request, and finally sending an HTTP HEAD request (of size 120 bytes each) to every resolved server. I argue that, unlike `dp`, Namehelp creates an undesired load on the last-mile network, which could potentially impact the performance of other applications running on the device.

- To improve the accuracy of server selection, Namehelp sends an HTTP HEAD request to every resolved server. However, based on our experience with `dp` I show in Figure 4.39 that using TCP OPEN delay is more accurate indicator of server performance and generates lesser probing traffic than sending HTTP HEAD requests.

- Unlike `dp`, Namehelp does not compare client's latencies to different CDN servers within a DNS resolution, which is important for minimizing the end-to-end latency between clients and servers (as discussed in Section 4).

- As shown in Figure 4.41, none of the DNS servers used in our study directed clients to servers with least end-to-end latency at all times. Therefore, I argue that the Namehelp's technique to resolve Web domains from an identified best DNS server (instead of identifying the best CDN server for each Web domain being resolved) might not ensure that clients will always be directed to fastest available CDN servers for all Web domains.

- Finally, unlike `dp`, Namehelp's resolution technique does not have a deadline within which it must resolve the requested Web domain.

In light of these differences between Namehelp (Nh) and `dp`, in Figure 4.45 I compare the average time to establish a TCP connection, time to receive the first bit of HTTP response, and the image download time from servers resolved by Namehelp and `dp`. For this comparison study, I configured clients to use Namehelp and `dp` one-by-one as their default DNS servers within a short time period. Next, I opened CDN URLs hosted on Akamai, Limelight, Level 3, and Reflected Networks CDNs on Google Chrome browser's incognito window (to prevent object loading from cache). I first resolved the Web domains from the configured DNS server, followed by sending an

HTTP GET request to download an image of size 82 KB, 53 KB, 207 KB, and 32 KB from the resolved Akamai, Limelight, Level 3, and Reflected Networks CDN servers respectively. I show the average percentage reduction in the overall object load time using `dp` on top each `dp` bar. For different CDN providers, I show that from an average of 25 DNS resolutions from Namehelp and `dp` each, the time to open TCP connection (TCP OPEN), time to receive first bit of HTTP response (HTTP HEAD), and image download time from servers resolved by `dp` are about 50% faster than the servers resolved by Namehelp on average.

CDN Load-balancing Techniques for Server Selection

A study by Shaikh *et al.* has shown benefits of providing client's IP address in the DNS request to Authoritative DNS servers, to allow clients to connect with nearby servers [279]. EDNS protocol has similar motivation in that it also enables CDN providers to direct users to nearby servers based on the client's IP subnet [288]. However, EDNS approach is still limited by how many DNS providers and ISPs support requests with EDNS [271].

A study by Kangasharju *et al.* compares the performance of different DNS redirection techniques, such as full redirection and selective redirection, used by CDN providers to reduce the impact of network latency on Web applications [176]. Their study shows that full redirection has superior performance over selective redirection, since selective redirection has an overhead to maintain which CDN server has what content in its cache, which may not be up-to-date and accurate at all times.

Relative Network Positioning for Server Selection

Previous studies have investigated the benefits of using network coordinate systems (NCS) to estimate the network latency between arbitrary end hosts [115]

[217] [321]. Such NCS techniques also enable CDN providers to estimate the network latency between clients and CDN servers to increase the accuracy of DNS-based server selection techniques [101]. However, a study by Choffnes *et al.* shows that network coordinate systems are often not accurate when used on edge networks [101].

CDN-based Relative Network Positioning (CRP), a tool by Su *et al.* shows that clients could be directed to closest CDN servers by comparing the cosine similarities between clients and different available CDN servers [172]. However, our other recent work on server selection shows that CRP technique is often not accurate in predicting the closest servers for clients [199].

Conclusions

Web application performance is affected by DNS resolutions to distant CDN servers. Although, DNS-based server selection may often direct clients to nearby CDN replicas, I show that current techniques could be improved to speed up the delivery of content. Therefore, I argue that clients are best positioned in the network to choose closest CDN servers. I propose DNS-Proxy (`dp`), a client-side tool that probes each resolved CDN address and directs clients to the fastest available servers. Effectively, `dp` shares load balancing functionality with CDNs by selecting from a set of resolved servers. Our measurement study on CDN infrastructure deployed by five major CDN providers shows that `dp` reduces webpage load time by 29% on average. If `dp` has already resolved a Web domain, the reduction in webpage load time is as much as 40%. Finally, `dp` reduces the download time of individual static Web objects by as much as 43%. Overall I believe `dp` enables a more effective use of existing CDN infrastructure and represents a complementary strategy to a continual increase of geographic content availability.

DETECTING CELLULAR MIDDLEBOXES USING PASSIVE MEASUREMENT TECHNIQUES

Abstract

The Transmission Control Protocol (TCP) follows the end-to-end principle – when a client establishes a connection with a server, the connection is only shared by two physical machines, the client and the server. In current cellular networks, a myriad of middleboxes disregard the end-to-end principle to enable network operators to deploy services such as content caching, compression, and protocol optimization to improve end-to-end network performance. If server operators remain unaware of such middleboxes, TCP connections may not be optimized specifically for middleboxes and instead are optimized for mobile devices. I argue that without costly active measurement, it remains challenging for server operators to reliably detect the presence of middleboxes that split TCP connections. In this paper, I present three techniques (based on latency, loss, and characteristics of TCP SYN packets) for server operators **to passively identify Connection Terminating Proxies (CTPs) in cellular networks**, with the goal to optimize TCP connections for faster content delivery. Using TCP and HTTP logs recorded by Content Delivery Network (CDN) servers, I demonstrate that our passive techniques are as reliable and accurate as active techniques in detecting CTPs deployed in cellular networks worldwide.

Introduction

The Transmission Control Protocol (TCP), Hyper Text Transport Protocol (HTTP) and secure HTTP (HTTPS) were originally designed with the assumption that clients communicate over end-to-end connections with servers. However, given

the different types of networks involved in an end-to-end connection between cellular clients and servers (such as the radio network, the cellular backbone, and the public Internet), optimizing communication for each of these networks independently improves the overall performance of the end-to-end connections between clients and servers [80] [152] [170]. One of the techniques used by cellular carriers to improve the communication performance in their networks is to deploy Connection Terminating Proxies (CTPs) that split TCP connections between clients and servers [134] [214]. CTPs allow cellular carriers to speed up TCP transfers between devices and the cellular gateways to the Internet through TCP optimization, content caching, and bandwidth throttling.

Content Distribution Networks (CDNs), cloud providers, or other server providers on the Internet are mostly unaware of specific CTPs deployed by individual cellular carriers. As a result, servers may not optimize their connections for CTPs, but optimize connections for the mobile device instead. I believe that if server providers are made aware of the presence of CTPs, TCP configurations could be fine-tuned to improve content delivery to the middlebox and to the end-user [125]. However, without expensive active network measurements on mobile devices, it remains challenging for server operators to reliably detect the presence of CTPs and optimize connections accordingly [322].

Carrier	**Latency**	**Packet Loss**	**TCP SYN**	**DH [16]**
AT&T	✓	✓	✓	✓
Verizon W.	✓	✓	✓	✓
Sprint	✓	✓	✓	✓
T-Mobile	✓	✓	✓	✓

Table 5.1: Comparison of results from our passive techniques with previous work [322] that uses active experiments, for cellular networks in the US.

In this study, I propose three techniques to **passively** detect the presence of CTPs in cellular networks, using TCP and HTTP logs recorded by Akamai's geographically distributed CDN servers. Our first technique compares **latency** estimated by clients and servers for TCP connections. The second technique compares the **packet loss** experienced by CDN servers for HTTP and HTTPS sessions. Our third technique analyzes characteristics of **TCP SYN** packets for connections to ports 80 (HTTP) and 443 (HTTPS). Although our evaluation is based on Akamai server logs, I argue that our techniques are not limited to CDN providers and also apply to other types of servers. The major contributions of this work are as follows:

- I perform the first large scale measurement study to passively detect the presence of CTPs deployed in cellular networks worldwide. Our study is based on data collected by Akamai CDN servers during January-July 2015. Our current dataset contains performance metrics from over a total of 14 million TCP connections from clients in different cellular networks.

- I propose three techniques for server operators to passively detect the presence of CTPs from TCP and HTTP server logs. Results from our measurements indicate that the use of CTPs is very popular among cellular carriers worldwide. In fact, carriers employ CTPs for splitting HTTPS sessions, in addition to splitting HTTP sessions.

- Using the collected data, I demonstrate that our techniques are reliable in detecting CTPs deployed in cellular networks across several countries. In Table 5.1, I compare the results of our passive techniques with the `Delayed Handshake (DH)` active measurement technique of CTP detection for cellular carriers in the US [322]. The tickmarks in the table indicate the presence of CTPs. I show that despite the fact that our passive measurement techniques do

not generate probing traffic, they correctly detect CTPs as detected by active experiments in `DH` [322].

The rest of the chapter is organized as follows. In Section 6, I discuss related work on detecting cellular middleboxes. In Section 5, I present our methodology. In Section 6, 5, and 5, I discuss how server operators could detect CTPs by using latency estimated by clients and servers, packet loss observed on the server-side, and inspecting `TCP SYN` packets, respectively. In Section 5, I offer discussion of our results. Finally, I conclude in Section 5.

Related Work

Several studies have investigated the characteristics, performance benefits and deployment locations of CTPs in cellular networks. Weaver *et al.* and Xu *et al.* investigated the characteristics of transparent Web proxies in cellular networks using active experiments on mobile devices [313] [322]. Other studies looked at the performance benefits of TCP splitting proxies to improve Web communications in cellular networks [81] [134] [214]. Ehsan *et al.* measured the performance gains of CTPs for Web caching and packet loss mitigation in satellite networks [127]. A study by Wang *et al.* characterized implications of cellular middleboxes on improving network security, device power consumption and application performance [311]. Our work, in contrast to these studies, focuses on detecting CTPs using passive measurement techniques, instead of active experiments.

Data Collection Methodology

To verify that our latency-based technique reliably detects CTPs in cellular networks worldwide, I used the webpage timing data collected by Akamai's Real User

Monitoring system (RUM) [41], which leverages the Navigation Timing API on the client browser [36]. The data includes the time to establish TCP connections for both HTTP and HTTPS sessions. Akamai's RUM also records TCP latency estimated by CDN servers for HTTP and HTTPS session. To investigate whether our packet loss-based technique reliably detects CTPs, I used TCP logs recorded by CDN servers deployed worldwide and extracted the number of packets retransmitted by the server for both HTTP and HTTPS sessions. Finally, to investigate whether our `TCP SYN`-based technique detect CTPs, I collected TCP-dumps on CDN servers for several hours and captured `SYN` packets for connection requests to port 80 (HTTP) and 443 (HTTPS).

Detecting CTPs from Client and Server-side Latency

When a CTP splits an end-to-end connection between clients and CDN servers, the latency estimated by clients should be higher than latency estimated by CDN servers. This is because the latency observed by the client will include the radio and cellular backbone latency (~tens of milliseconds [38]). Whereas the latency estimated by CDN servers would include the latency on the wired public Internet and is likely to be low (~5 ms), as CDNs have wide deployment of servers inside many cellular networks.

In this section I analyze the TCP latency estimated by clients and servers for TCP connections (both HTTP and HTTPS sessions) using two different methods. First, I compare the latency from both client and server endpoints to identify networks where the latency experienced by clients is significantly higher than latency experienced by servers – which indicates that a CTP is being used for a connection. Second, I compare the latency for HTTP and HTTPS sessions only from the server-side to

CC	Carrier	Protocol	Hits	Client RTT			Server RTT			Proxy?
				p25	p50	p75	p25	p50	p75	
US	AT&T	HTTP	1.7M	37	47	67	3	4	8	✓
US	AT&T	HTTPS	686K	45	60	89	52	75	114	X
US	Verizon W.	HTTP	1.9M	36	45	69	5	10	21	✓
US	Verizon W.	HTTPS	471K	44	60	87	48	65	87	X
US	T-Mobile	HTTP	2.1M	40	59	85	19	68	157	Limited
US	T-Mobile	HTTPS	459K	45	65	98	59	94	180	–
US	Sprint	HTTP	1.4M	39	52	78	3	12	28	✓
US	Sprint	HTTPS	275K	47	63	93	52	72	118	X
US	Clearwire	HTTP	96K	75	93	128	75	95	139	X
US	Clearwire	HTTPS	39K	75	92	137	82	100	143	X
CA	Bell Canada	HTTP	63K	38	50	69	49	78	151	–
CA	Bell Canada	HTTPS	17K	38	49	73	57	85	157	–
CA	Rogers	HTTP	97K	37	51	86	41	64	110	–
CA	Rogers	HTTPS	30K	37	52	87	48	72	119	–
CA	Telus	HTTP	65K	34	43	60	9	19	49	✓
CA	Telus	HTTPS	16K	43	58	83	47	66	104	X
CA	Sasktel	HTTP	10K	27	41	83	23	33	75	X
CA	Sasktel	HTTPS	2K	43	63	116	59	100	230	–
CA	Videotron	HTTP	7K	44	55	71	44	58	91	X
CA	Videotron	HTTPS	4K	46	58	86	50	70	120	X
MX	Uninet	HTTP	41 K	83	113	183	142	267	571	–
MX	Uninet	HTTPS	8 K	79	109	177	163	256	446	–

Table 5.2: Distribution of TCP latency estimated by clients (Client RTT) and servers (Server RTT) for IPv4-based cellular networks in North America.

identify networks where servers experience significantly different latencies for HTTP and HTTPS sessions – which indicates that a CTP is used for one type of connections.

In Table 5.2, I show the distribution (25th, 50th, and 75th percentile) of network latency measured by the client (**Client RTT**) and by the server (**Server RTT**) for major cellular networks in North America. The column **CC** represents the country code of each network. Column **Hits** represents the number of unique TCP connections behind latency distributions. The column **Proxy?** indicates whether our techniques detect CTPs for a given cellular carrier. For example, for AT&T network in the US, the **Client RTT** for HTTP sessions is almost 10 times the **Server RTT**, which

indicates that servers are communicating with a device only 4 ms away. Since 4 ms is too low for an end-to-end connection over a cellular network [38], I argue that servers communicate with CTPs deployed in AT&T network (as indicated by ✓ in the Proxy column). In the case of HTTPS sessions in AT&T, I observe that **Client RTT** and **Server RTT** are similar, which indicates that there is no CTP for HTTPS sessions in the AT&T network (as indicated by X in Proxy column). Further, when I look at only the **Server RTT** for HTTP and HTTPS sessions, I see that servers experience significantly higher latency for HTTPS sessions, which further confirms that AT&T does not employ CTPs for splitting HTTPS sessions. Tables 5.3, 5.4, and 5.5 show the application of the latency technique to detect CTPs in cellular networks in Asia, Europe, and Oceania and South America, respectively.

While employing our latency-based techniques to detect CTPs in cellular networks worldwide, I made five observations on the behavior of CTPs. First, I observe that for **p25** of HTTP sessions in T-Mobile USA network, the latency experienced by clients and servers is significantly different, which indicates a presence of CTPs HTTP sessions in T-Mobile network. However, for **p50** of the HTTP sessions, the two latencies are similar – indicating no presence of CTPs for HTTP sessions in T-Mobile network. To investigate this surprising behavior of T-Mobile network, I classified our data based on server locations and domain names. Table 5.6 shows the distribution **Client RTT** and **Server RTT** for HTTP sessions for different domain names across different locations in the US. I observe that for clients connecting to servers in CA and VA, CTPs are used on per domain basis. For example, the HTTP latency estimated by servers in CA to download webpages associated with a clothing website is significantly lower than latency estimated for a ticketing website. I see similar trends at other locations in the US and across several domain names. Next, I observe that T-Mobile employs CTPs for HTTP sessions only at a few locations

CC	Carrier	Protocol	Hits	Client RTT			Server RTT			Proxy?
				p25	p50	p75	p25	p50	p75	
CN	China Mobile	HTTP	85 K	34	61	101	46	77	128	X
CN	China Mobile	HTTPS	24 K	49	81	132	57	93	170	X
TW	HiNet	HTTP	53 K	33	48	70	35	50	91	X
TW	HiNet	HTTPS	18 K	33	48	77	38	58	103	X
CN	ChinaNet	HTTP	4 K	45	81	149	33	83	167	X
CN	ChinaNet	HTTPS	5 K	207	342	471	118	144	215	–
CN	China Unicom	HTTP	8 K	55	90	150	70	119	209	X
CN	China Unicom	HTTPS	4 K	70	109	187	82	127	213	X
HK	China Mobile	HTTP	9 K	32	53	93	34	60	110	X
HK	China Mobile	HTTPS	3 K	32	48	91	39	57	108	X
IN	Vodafone	HTTP	304 K	58	128	367	33	59	170	–
IN	Vodafone	HTTPS	191 K	80	131	349	102	244	553	–
KR	Korea Telecom	HTTP	28 K	29	35	43	30	40	51	X
KR	Korea Telecom	HTTPS	25 K	30	38	56	37	43	65	X
JP	SoftBank	HTTP	44 K	30	40	55	3	8	13	✓
JP	SoftBank	HTTPS	8 K	37	47	64	41	49	62	X
MY	TM Net	HTTP	13 K	57	75	120	65	113	397	X
MY	TM Net	HTTPS	3 K	60	82	129	83	136	380	X
AE	Eitc	HTTP	4 K	123	153	217	139	159	221	X
AE	Eitc	HTTPS	3 K	139	159	233	140	161	228	X
AE	Etisalat	HTTP	4 K	30	37	49	3	5	29	✓
AE	Etisalat	HTTPS	3 K	33	40	52	35	42	57	X

Table 5.3: Distribution of TCP latency estimated by clients (Client RTT) and servers (Server RTT) for cellular networks in Asia.

in the US. For example, in Table 5.6 the latency experienced by clients connecting to servers in TX indicate that T-Mobile does not use a CTP for terminating HTTP sessions for any domain name. Thus I argue that T-Mobile's deployment of CTPs in the US is different across different locations and domain names. Based on these observations, I label the **Proxy?** column in Table 5.2 as 'Limited'.

The second observation I make is that cellular networks in the US use CTPs for TCP connections over their IPv4 networks, but not over their IPv6 networks. Since I did not observe statistically significant IPv6 traffic from cellular carriers deployed outside of the US, I restrict this observation to cellular carriers in the US only. In

CC	Carrier	Protocol	Hits	Client RTT			Server RTT			Proxy?
				p25	p50	p75	p25	p50	p75	
DE	DTAG	HTTP	22K	39	50	75	5	8	14	✓
DE	DTAG	HTTPS	13K	53	79	125	34	46	93	–
DE	Vodafone	HTTP	57K	39	51	82	7	11	16	✓
DE	Vodafone	HTTPS	17K	49	64	100	53	70	128.5	X
ES	Telefonica	HTTP	65K	55	92	372	10	18	30	✓
ES	Telefonica	HTTPS	136K	108	149	218	14	22	35	Limited
ES	UNI2	HTTP	43K	41	57	96	38	62	141	X
ES	UNI2	HTTPS	121K	43	59	102	45	64	115	X
ES	Vodafone	HTTP	91K	30	43	72	6	15	30	✓
ES	Vodafone	HTTPS	223K	35	49	76	39	55	90	X
ES	Jazztel	HTTP	9 K	56	75	127	61	90	233	X
ES	Jazztel	HTTPS	17 K	56	73	109	66	87	147	X
FR	Bouygues	HTTP	75K	28	37	57	2	4	38	✓
FR	Bouygues	HTTPS	26K	30	39	59	35	47	79	X
FR	France Telecom	HTTP	37K	37	48	73	1	6	13	✓
FR	France Telecom	HTTPS	17K	40	56	94	1	7	39	✓
FR	SFR	HTTP	41K	37	50	82	3	7	33	✓
FR	SFR	HTTPS	15K	44	62	103	48	72	142	X
FR	Free	HTTP	23 K	43	59	92	40	59	90	X
FR	Free	HTTPS	10 K	45	63	116	26	42	71	–
GB	Telefonica	HTTP	186K	49	71	109	7	11	23	✓
GB	Telefonica	HTTPS	40K	59	85	150	48	72	115	X
GB	Vodafone	HTTP	115K	41	56	89	7	14	57	✓
GB	Vodafone	HTTPS	24K	49	68	111	54	76	145	X
IT	H3G	HTTP	49K	55	73	116	60	81	157	X
IT	H3G	HTTPS	14K	55	77	142	65	93	221	X
IT	Tim	HTTP	55K	39	57	94	6	12	41	✓
IT	Tim	HTTPS	13K	46	67	110	53	80	167	X
AT	France Telecom	HTTP	8 K	41	57	80	59	97	210	–
AT	France Telecom	HTTPS	3 K	43	59	87	66	101	219	–
AT	H3G	HTTP	9 K	40	57	79	58	94	205	–
AT	H3G	HTTPS	4 K	41	59	88	62	98	225	–
AT	T-Mobile	HTTP	10 K	33	48	72	5	15	48	✓
AT	T-Mobile	HTTPS	3 K	40	58	83	52	76	131	X
NL	Vodafone	HTTP	8 K	33	39	61	2	2	16	✓
NL	Vodafone	HTTPS	2 K	35	43	80	37	46	71	X
SE	Vodafone	HTTP	8 K	37	45	59	62	97	175	–
SE	Vodafone	HTTPS	39 K	37	46	58	65	89	142	–
TR	Turk Telecom	HTTP	34 K	54	83	150	39	80	143	X
TR	Turk Telecom	HTTPS	9 K	49	72	138	50	80	145	X
TR	Vodafone	HTTP	16 K	40	59	116	9	51	85	Limited
TR	Vodafone	HTTPS	4 K	55	92	128	64	102	152	X

Table 5.4: Distribution of TCP latency estimated by clients (Client RTT) and servers (Server RTT) for cellular networks in the Europe.

CC	Carrier	Protocol	Hits	Client RTT			Server RTT			Proxy?
				p25	p50	p75	p25	p50	p75	
AU	Vodafone	HTTP	106 K	31	40	62	2	3	13	✓
AU	Vodafone	HTTPS	64 K	38	51	94	36	48	87	X
NZ	Vodafone	HTTP	7 K	30	49	71	2	11	27	✓
NZ	Vodafone	HTTPS	6 K	38	59	99	37	61	115	X
BR	Telefonica	HTTP	560 K	51	108	273	58	120	309	X
BR	Telefonica	HTTPS	63 K	40	78	165	51	100	212	X
PY	Telefonica	HTTP	13 K	180	217	289	186	237	430	X
PY	Telefonica	HTTPS	3 K	184	221	297	202	262	428	X

Table 5.5: Distribution of TCP latency estimated by clients (Client RTT) and servers (Server RTT) for cellular networks in Oceania and South America.

Table 5.7, I show the distribution of TCP latency for IPv6 networks deployed by major US carriers, estimated by clients and CDN servers. I observe that clients in Verizon Wireless connecting to CDNs over IPv6 network experience latency similar to that estimated at the server for HTTP sessions. However, from Table 5.2, I observe that Verizon clients connecting to CDN servers over its IPv4 network experience much higher latency than experienced by the CDN servers, for HTTP sessions – indicating the presence of CTP for HTTP sessions in its IPv4 network. Therefore, I argue that Verizon employs CTPs for HTTP sessions in its IPv4 network and not in its IPv6 network.

The third observation I make is that some networks use CTPs to split HTTPS sessions. Using our measurement data, I identified a cellular carrier in France that employs CTPs to split HTTPS sessions. In Table 5.4, I show that for France Telecom, the **Server RTT** for HTTPS sessions is significantly lower than the **Client RTT**, therefore I believe that France Telecom uses CTPs to split HTTPS sessions. Telefonica in Spain is another cellular carrier for which I observe that CTPs split HTTPS sessions, as the latency estimated by CDN servers is lower than latency estimated by clients. Further, Telefonica's recent design of `mcTLS` protocol indicates

State	Domain Type	Client RTT			Server RTT			Proxy?
		p25	p50	p75	p25	p50	p75	
CA	Clothing website	37	51	75	2	3	3	✓
CA	e-Commerce website	40	56	80	2	2	3	✓
CA	Health Care website	40	56	90	40	80	175	X
CA	Ticketing website	37	49	65	43	93	186	X
VA	Clothing website	39	57	80	2	2	2	✓
VA	e-Commerce website	46	68	89	2	2	2	✓
VA	Health Care website	44	64	90	27	63	121	X
TX	Clothing website	54	72	96	49	93	204	X
TX	Health Care website	56	75	97	61	107	211	X
TX	Ticketing website	50	70	90	33	67	111	X
TX	Movies website	56	71	91	88	156	301	X

Table 5.6: Distribution of HTTP latency estimated by clients (Client RTT) and servers (Server RTT) for T-Mobile across different domains & locations.

that ISPs work towards deploying CTPs for HTTPS sessions [213], likely to support content caching and connection optimization for secure connections [292].

The fourth observation I make is that for some carriers, the **p75** of **Server RTT** is similar to **p25** of **Client RTT**, when the **p25** and **p50** of **Server RTT** indicate the presence of CTPs in that carrier. For example, the **p75** of **Server RTT** for HTTP sessions in Etisalat network in Table 5.3, suggests that CTPs may not be used for splitting all HTTP sessions. I speculate that when CTPs get overloaded, client requests are likely not sent to CTPs and instead sent directly to servers. As a result servers occasionally experience (unproxied) latency of end-to-end connections to mobile devices. To deal with such occasional instances, TCP stacks of servers should interpret such connections as direct connections to mobile devices.

Finally, the fifth observation I make is that for a few cellular carriers the **Server RTT** is either higher or lower than **Client RTT** by at least 80 ms for **p75**. Specifically, if I observe **Server RTT** to be higher than **Client RTT**, I speculate

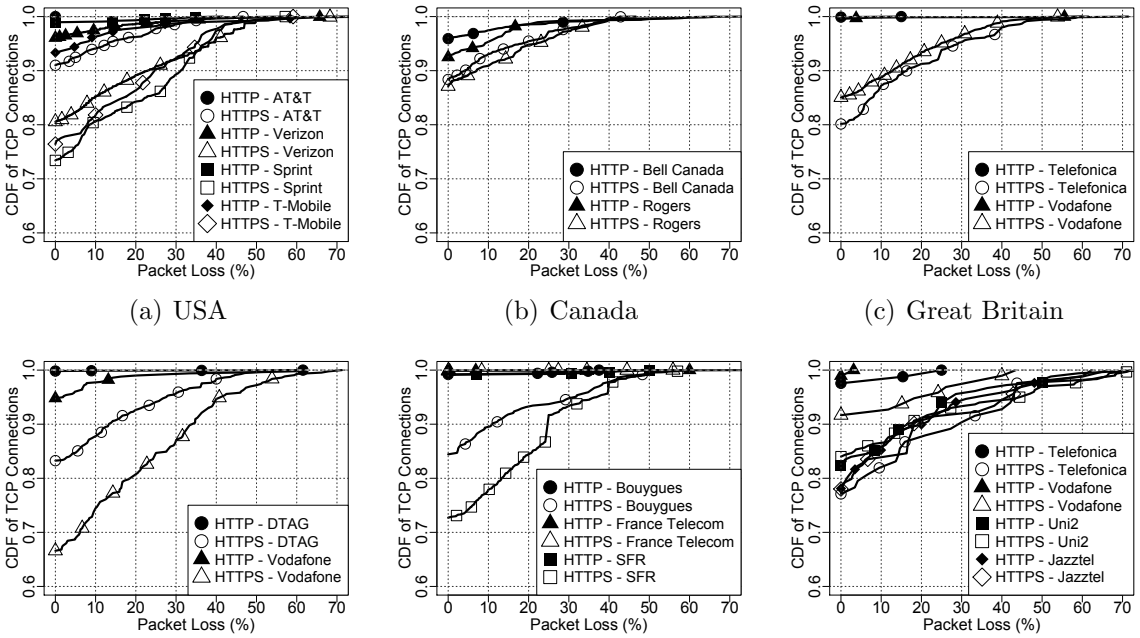
CC	Carrier	Protocol	Hits	Client RTT			Server RTT			Proxy?
				p25	p50	p75	p25	p50	p75	
US	AT&T	HTTP	15 K	37	45	60	2	3	16	✓
US	AT&T	HTTPS	4 K	43	58	87	47	62	93	X
US	Verizon W.	HTTP	232 K	46	62	84	43	66	83	X
US	Verizon W.	HTTPS	81 K	46	62	87	50	69	90	X
US	T-Mobile	HTTP	295 K	42	60	85	4	24	59	Limited
US	T-Mobile	HTTPS	82 K	47	68	96	49	67	99	X

Table 5.7: Distribution of TCP latency estimated by clients (Client RTT) and servers (Server RTT) for IPv6 cellular networks in North America.

that CTPs are deployed near the gateway and Internet egress points are far from the gateway. If I observe **Server RTT** to be lower than **Client RTT**, I speculate that CTPs are near to both egress points and gateways but clients connect to gateways far in the network. For such cellular carriers I place a **'-'** in the **Proxy?** column in Tables 5.2, 5.3, 5.4, and 5.5. I argue that for such cellular carriers, passive techniques in the following sections may be used to detect the presence of CTPs.

Detecting CTPs from Packet Loss on the Server-side

In previous section, I discussed how server operators could use latencies measurements by clients and servers to detect the presence of CTPs. In this section, I are interested in verifying another technique, based on packet loss, to passively detect CTPs across cellular networks worldwide using measurement data collected by Akamai CDN servers. Since I observe TCP latency estimated by CDN servers to CTPs is significantly low, I argue that CTPs and CDN servers are usually deployed within the same or nearby datacenters. Therefore, when a CTP is employed to split connections, the number of packets retransmitted by servers should be lower than packets retransmitted for connections where CTPs are not used. Following this assumption,



(a) USA (b) Canada (c) Great Britain

(d) Germany (e) France (f) Spain

Figure 5.1: Distribution of packet loss over HTTP and HTTPS sessions for cellular networks in different countries. For visibility, I reduced the number of symbols on each line.

in Figure 5.1, I show the distribution of packet loss observed during our tests for thousands of HTTP and HTTPS sessions. Our first goal is to identify networks where packet loss observed by CDN servers is higher for one type of connections and not others. I also aim to determine whether results from using packet loss correlate with our CTP detection in the previous section. Due to space limitations, I show distribution of packet loss for only a few cellular carriers in North America and Europe.

In Figure 5.1(a), I show the distribution of packet loss observed for HTTP and HTTPS sessions in four major cellular carriers in the US. Specifically, in the case of Verizon, AT&T, and Sprint networks, I observe that for HTTP sessions CDN servers experience low packet loss, whereas for HTTPS sessions CDN servers experience

significantly higher packet loss – indicating the presence of CTPs for HTTP sessions. The results for these networks agree with our observations from using latency-based technique. However, in the case of T-Mobile, I see that the packet loss for HTTP sessions is slightly higher compared to other networks. I speculate that the packet loss for HTTP sessions in T-Mobile network are influenced by T-Mobile's policy to employ CTPs at only a few locations and domain names in the US (Table 5.6).

Next, I compare the packet loss observed for connections in a network where CTP is not employed, the Rogers network in Canada, as detected by our latency-based technique in Table 5.2, with a network where our latency-based technique could not detect the presence of CTPs, the Bell Canada network in Canada. In Figure 5.1(b), I show that for both HTTP and HTTPS sessions in Bell Canada and Rogers networks, CDN servers observe similar packet loss. I speculate that either CTPs are not employed in the Bell Canada network or CTPs are present but CTPs experience same network conditions as Rogers network without CTPs.

I now extend our discussion and compare packet loss observed by CDN servers for connections in major cellular carriers in the UK, Germany, France, and Spain. Similarly to carriers in the US, in Figure 5.1(c) and 5.1(d), I show that packet loss observed by servers for HTTP sessions is significantly lower than packet loss observed for HTTPS sessions – indicating the presence of CTPs for HTTP sessions, similar to our observations from using latency-based technique. For cellular carriers in France in Figure 5.1(e), I observe that packet loss for HTTPS sessions in France Telecom network is similar to packet loss for HTTP sessions, with both being almost zero. This indicates that CTPs are employed by France Telecom for splitting both HTTP and HTTPS sessions – validating our observations from using latency-based technique.

Finally, in Figure 5.1(f), I show distribution of packet loss observed by CDN servers for major cellular carriers in Spain. I observe that for Vodafone and Telefonica

networks, the packet loss for HTTP sessions is much lower than packet loss for HTTPS session – indicating the presence of CTPs for only HTTP connections, similar to our observations from using latency-based technique. For Uni2 and Jazztel, however, I observe that packet loss for both HTTP and HTTPS is similar. This indicates that CTPs are used for both HTTP and HTTPS sessions, similar to our observations from using latency-based technique. One exception to our results is for Telefonica. Using the latency technique I identified that Telefonica could be a potential carrier where CTPs are used to terminate HTTPS sessions. However, the high packet loss for HTTPS sessions indicates that CTPs are not used for splitting HTTPS sessions. To disambiguate the presence of CTPs, I propose another technique that relies on analyzing the characteristics of TCP SYN packets, which I discuss next.

Detecting CTPs from TCP SYN Characteristics

Our third technique is based on analyzing `TCP SYN` packets to detect the presence of CTPs in cellular networks. Our active experiments on understanding characteristics of `TCP SYN` packets generated by different types of mobile devices have revealed that the advertised `Initial Congestion Window Size (ICWS)`, `TCP Timestamp` in the TCP options header, and `Maximum Segment Size (MSS)` values are different across different types of mobile devices. I also observed that these values are different even when the same device connects to Wi-Fi and cellular network. Based on this observation, our goal is to identify whether analyzing `TCP SYN` packets (captured passively for HTTP and HTTPS sessions) have the same `ICWS`, `MSS`, and an increasing TCP Timestamp value, which would indicate that `SYN` packets are likely being generated by a single machine (a CTP), instead of from multiple mobile devices with different hardware.

Results from our analysis of `TCP SYN` packets indicate that for all observed `TCP SYN` packets on port 80 from cellular carriers for which our latency and packet loss-based techniques suggest presence of CTP for HTTP sessions, the `ICWS` and `MSS` fields in the `TCP SYN` packets have the same value and the TCP Timestamp option have monotonically increasing values with a near constant skew – indicating the presence of CTPs for splitting HTTP sessions. For `TCP SYN` packets (generated from networks for which our latency and packet loss-based techniques suggest absence of CTPs for HTTPS sessions) to port 443 of CDN servers, I observed varying values of `ICWS`, `MSS`, and TCP Timestamp – indicating that the `TCP SYN` packets are likely generated by different mobile devices, instead of CTPs. I also verified our technique to be reliable for cellular carriers that employ CTPs for HTTPS sessions. For example, for France Telecom network in France I observed that the characteristics of all observed `TCP SYN` packets to port 443 were similar – indicating the presence of CTPs for HTTPS connections. For Telefonica in Spain, I did not observe similar characteristics of observed `TCP SYN` packets to port 443 – indicating absence of CTPs for splitting HTTPS sessions. Based on our findings on Telefonica's CTPs for HTTPS sessions from our latency, loss, and SYN-based techniques, I argue that active measurements may be needed to reliably detect CTPs. Finally, based on the data collected I did not find networks where `ICWS` and `MSS` values were similar but CTP was not detected using latency packet loss based techniques.

Discussion

I believe that one can leverage the use of our latency-based technique to identify the cellular latency offered by carriers where CTPs are present. I argue that for such carriers, **Client RTT** is a reliable indicator of the cellular latency, comprising of the

sum of radio latency and latency within the cellular backbone. Specifically, if 4G is widely deployed by a cellular carrier, the latency offered by 4G would be reflected in both **p25** and **p75** of **Client RTT**. Further, if 3G is more widely deployed than 4G, then the latency offered by 4G would be reflected in the **p25** and latency offered by 3G would be reflected in **p75** of **Client RTT**. For example, for Telefonica in Spain, Sensorly's [43] signal strength data suggests a wide deployment of 3G, but little deployment of 4G. Therefore, in Table 5.4, the **p25** of **Client RTT** for HTTP sessions (55 ms) reflects Telefonica's latency over its 4G network, whereas the **p75** latency of 372 ms reflects its 3G latency. Further, the Etisalat network in AE (in Table 5.3) has wide deployment of 4G (based on Sensorly data), thus the HTTP latency shown in both **p25** (30 ms) and **p75** (49 ms) of **Client RTT** represents the latency offered by Etisalat's 4G network. For other cellular networks with CTPs also, I verified that using Sensorly's data and **Client RTT** together allows cellular latency estimation in a given carrier.

Conclusions

Connection Terminating Proxies (CTPs) have been a great area of interest for many cellular carriers in the past. These proxies allow for optimizing TCP connections between servers and client devices. In this paper, I propose three techniques to passively identify the presence of CTPs, based on latency, loss, and `TCP SYN` characteristics. I also conduct an extensive measurement study based on Akamai server logs to demonstrate that our techniques can reliably detect CTPs in cellular networks worldwide. Based on our measurement results, I argue that server operators could use our suggested techniques to detect CTPs using server logs only

and optimize communications for different cellular networks with the goal of faster content delivery to end-users.

A CASE FOR FASTER MOBILE WEB IN CELLULAR IPV6 NETWORKS

Abstract

The transition to IPv6 cellular networks creates uncertainty for content providers (CPs) and content delivery networks (CDNs) of whether and how to follow suit. Do CPs that update their CDN contracts to allow IPv6 hosting achieve better, or worse performance in mobile networks? Should CDNs continue to host mobile content over IPv4 networks, or persuade to their CP customers the performance benefits of IPv6 content delivery?

In this paper we answer these questions through a comprehensive comparison of IPv4 and IPv6 mobile Web performance in cellular networks in the US from the point of view of Akamai's content delivery infrastructure. Our data show that IPv6 hosting outperforms legacy IPv4 paths in mobile Web. Our analysis leads to clear recommendations for CPs to transition to IPv6-hosted mobile Web. Finally, we propose new mechanisms, through which CDNs can safely transition mobile content to IPv6-enabled servers for improved content delivery.

Introduction

Despite many years of research to improve Web performance in mobile and wireless networks, users remain dissatisfied with lengthy webpage load times [263]. As Internet Service Providers (ISPs) upgrade their network infrastructure from IPv4 to IPv6, understanding the performance of mobile content delivery in cellular IPv6 networks is crucial. In this study, we take a novel approach to characterize the dynamically changing IPv6 ecosystem from the point of view of Akamai's content delivery infrastructure for cellular networks [224]. We argue that, unlike PlanetLab

and Amazon EC2 datacenters [2, 45], Akamai's content delivery servers are so deeply deployed inside several cellular ISPs' networks that the end-to-end communication between mobile devices and Akamai's servers need not, strictly speaking, touch the wired public Internet outside the cellular network. As a result, Akamai's unique content delivery infrastructure enables us to view the end-to-end cellular ecosystem between mobile devices and cellular gateways and evaluate how content is delivered over cellular IPv6 networks from the perspective of content providers (CPs), ISPs, and other content delivery networks (CDNs) [28, 55].

CPs, such as Facebook and others, care about the experience of users with their respective applications. To deliver application content from datacenters to users in a timely manner, CPs make contractual agreements with CDNs to ensure content has high availability, is secure, and is delivered to users through low latency connections. Some CPs sign contracts with CDNs for content delivery only over a cellular ISP's IPv4 network, while other CPs sign contracts for content delivery over IPv6 networks. Although, CPs are aware that ISPs are deploying IPv6 in their networks and CDNs are offering IPv6 hosting of mobile content, CPs remain uncertain whether upgrading to IPv6 will improve or worsen the performance of mobile content delivery.

CDNs, such as Akamai and others, care about the performance of content delivery to their mobile users. Although CDNs strive to upgrade their infrastructure to overcome IPv4 address scarcity and to make content available over IPv6 networks [34, 110, 113, 222, 262], one of their goals is to ensure that the performance of content delivery over an ISP's IPv6 network is as good as over that ISP's IPv4 network. CDNs generally act as a surrogate infrastructure for its many CPs, and for stability, reliability, and contractual implications, the configurations for each CP are often only changed with respective permission granted from the CP. Thus, adoption of new content delivery techniques, such as IPv6, is often a multi-year process as

its performance implications become better-understood in real-world conditions over time.

As cellular network operators adopt IPv6 addressing to resolve the challenges imposed by IPv4 address scarcity [113], the research community has shown interest in understanding different IPv6 deployment strategies within ISPs worldwide, and the adoption rate of IPv6 among mobile users and content providers [110,222,274]. Consequently, CPs and CDNs remain reluctant to embrace a wide-scale transition to IPv6, as they remain unaware of performance of IPv6 networks deployed by cellular carriers.

In this paper, we take a novel approach to investigate and expose performance of IPv6 and IPv4 ecosystems from a CDN's perspective – in between cellular ISPs and Content Providers. Our goal is to improve awareness within different networking communities about performance benefits (if any) of serving mobile content over IPv6, as opposed to IPv4. Our work precisely describes the current role of a CDN in connecting mobile users to content servers in the evolving cellular ecosystem in the US. To the best of our knowledge, our work is the most detailed investigation to compare mobile Web performance over IPv4 and IPv6 networks. We classify the four major contributions of this work as follows:

Dataset Richness: We conducted a large scale, comprehensive study to measure IPv6 performance in four major cellular carriers in the US to compare its native and NAT64/DSLite deployments. Using Akamai CDN infrastructure, we collected a rich dataset consisting of millions of data points of measured IPv6 and IPv4 performance, during the months of January - August in 2015.

Measurement: Our study investigates IPv6 performance across multiple factors that influence Web performance on cellular networks.

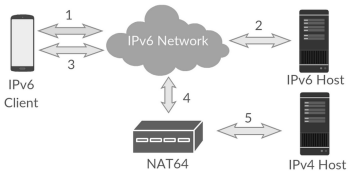


Figure 6.1: T-Mobile's IPv6 network.

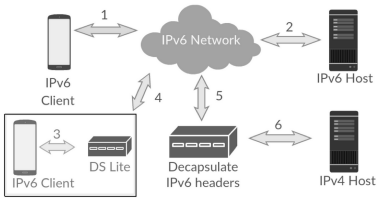


Figure 6.2: Verizon's IPv6 network.

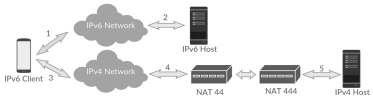


Figure 6.3: AT&T and Sprint's IPv6 net.

- We compare IPv6 and IPv4 networks through 1) round trip time between clients and CDN servers; 2) time to resolve domain names from cellular DNS; and 3) webpage load time.

- We extend Akamai's Real User Monitoring System (RUM) [41] to accurately extract Web performance metrics for mobile content hosted on IPv6-enabled content servers in US cellular networks.

Inferences Drawn: Our experience with Akamai's content delivery infrastructure shows that IPv6 networks deployed by cellular ISPs outperform their IPv4 networks.

- Our analysis includes recommendations for CPs to host mobile content on IPv6 for improved user experience.

- We also recommend that CDNs deliver mobile content over IPv6 to avoid in-path middleboxes for IP address translation deployed by cellular carriers.

- And finally, we suggest cellular network operators upgrade their network infrastructure to support IPv6, instead of continuing to deploy legacy IPv4 technologies in their network.

Problems Discovered: During our study, we discovered the following three problems related to how IPv6 content is delivered in cellular networks. We also propose several solutions we adopted to address these problems.

- We discovered that the DNS lookup process takes longer on Android devices with IPv6 capability than Android devices with IPv4-only capability. The lookup time is high because IPv6-capable devices wait for both `Type A` and `Type AAAA` (pronounced 'quad A') DNS queries to finish before establishing a TCP connection. Because of higher DNS lookup time for IPv6-capable clients, we observe that IPv6 clients in the Sprint network often experience slower webpage loads when connecting to IPv4 servers, than IPv4-only clients connecting to same IPv4 servers.

  Solution: To address this problem, we made four recommendations to the Google Android team to reduce long Round Trips Times (RTTs) to cellular resolvers. First, if the DNS lookup process on IPv6-capable Android devices could be modified to send `AAAA` and `A` DNS queries in parallel, lookup times could potentially become twice as fast. Second, we suggested that additional speedup in DNS lookup could be achieved by letting client browsers indicate to the mobile OS that they do not need to wait to get the `A` lookup back if they get an `AAAA` answer in DNS response. Third, in cases where mobile clients are on an IPv6-only network, the mobile OS could be modified to only send DNS `AAAA` queries, instead of both `AAAA` and `A`. Finally, the existence of `A` or `AAAA` only answers could be cached per-name for some time on the device, not withstanding the caching of the actual answers, which would enable the device to make a smarter query the following time that an IP address is needed.

- In the case of T-Mobile (and applicable to other major IPv6-only networks worldwide), we discovered that when IPv6-capable clients resolve an IPv4-only domain name, the cellular DNS introduces an extra round trip to the DNS Authorities. We discuss details in Section 6.

Solution: We develop and prototype OneTrip, a technique for DNS Authorities to eliminate the extra round trip in DNS lookups when resolving IPv4-only domains. Through in-lab simulations we show that DNS lookup times reduce significantly when DNS Authorities use OneTrip. We also experimentally verify that in T-Mobile's production cellular network, OneTrip maintains end-to-end connectivity.

- In a cellular network outside of the US, IPv6 packets were being routed via the US, which resulted in latency of end-to-end connections on IPv6 to be higher than IPv4 by 200 ms. While it is possible that IPv6 latency may be higher than IPv4 in some networks, we argue that it could be due to misconfigured routing policies.

  Solution: Based on our findings, Akamai's network team is actively working with that cellular carrier to resolve misconfiguration in its IPv6 routing.

The rest of the paper is organized as follows. In the next section, we offer a discussion on how IPv6 is deployed by different cellular ISPs in the US. An overview of different IPv6 deployment strategies will support our measurement techniques and research findings. In Section 6, we describe our data collection methodology. In Section 6, 6, and 6, we investigate the component differences of Web performance in IPv6 and IPv4 networks through measuring the round trip latency between clients and CDN servers, the DNS lookup time, and the webpage load time. In Section 6, we introduce OneTrip as a technique for DNS Authorities to eliminate unnecessary round trips from DNS lookups in IPv6-only networks. In Section 6, we discuss related work. Finally, we conclude in Section 6.

Overview of IPv6 Deployment in Cellular Networks

A cellular carrier that supports IPv6 addressing must provide a way for its IPv6 devices to connect with IPv4 and dual-stacked (both IPv4 and IPv6 addresses) Internet servers. Cellular network architectures are influenced by a variety of factors such as the capabilities of the existing infrastructure hardware to support IPv6 addressing, urgency to upgrade networks to IPv6, number of users with IPv6-capable devices, number of available IPv4 addresses, etc., which result in ISPs taking different approaches to IPv6 deployment [63]. In this section we provide an overview of how the different cellular carriers in the US have upgraded their IPv4 networks to provide IPv6 addressing to their users, based on the publicly available information from those carriers.

T-Mobile is an IPv6-only network for all phones with support for `464XLAT`, which includes all phones with Android version 4.3 and above [90]. For older versions of Android, as well as iPhone, Blackberry, and Windows devices, T-Mobile is an IPv4-only network. As a result, IPv6 devices in T-Mobile network always transmit IPv6 packets, whereas, IPv4 devices always transmit IPv4 packets. Thus, the choice of addressing (IPv4/IPv6) for devices is based on whether the device supports `464XLAT`.

In Figure 6.1 we depict the high level infrastructure of the IPv6 network deployed by T-Mobile. We show that when an IPv6 device communicates with an IPv6 server the packets are routed directly to the server through the IPv6 network without any NAT-based stateful middleboxes (Steps 1 and 2). However, when an IPv6 device communicates with an IPv4 server, the IPv6 packets generated by the device are routed to a stateful NAT64 middlebox (Steps 3 and 4). The NAT64 middlebox translates IPv6 packets to IPv4 address family and forwards the translated packets to the IPv4 server (Step 5). The NAT64 middlebox also converts reply IPv4

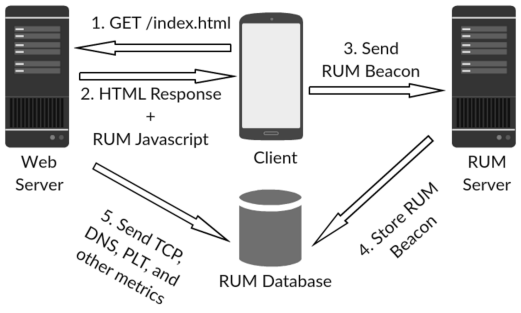


Figure 6.4: Sequence of Akamai's RUM interactions with client's browser.

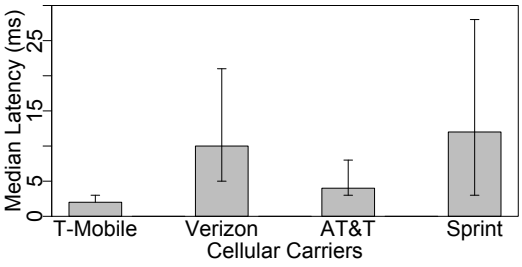


Figure 6.5: Round trip latency between Akamai CDN servers and cellular TCP Split proxies.

packets (from the IPv4 server) to IPv6 packets (forwarded to the mobile device), as shown in Steps 5, 4, and 3.

In summary, there are two ways in which T-Mobile routes packets from IPv6 devices: 1) via IPv6 network with no stateful middleboxes, and 2) via NAT64 middlebox, where IPv6 network is used between clients and NAT64 and IPv4 network is used between NAT64 and IPv4 servers.

Verizon Wireless (Verizon) provides both IPv6 and IPv4 addressing to all of its devices connected to its LTE network [60]. For devices with no IPv6 support or not connected to the LTE network, Verizon provides only IPv4 addressing – resulting in the devices using Verizon's IPv4 network. Thus, in the Verizon network, the choice of addressing on the device is based on whether the device is connected to the LTE network [260].

In Figure 6.2, we depict a high level infrastructure of Verizon's IPv6 network. Similarly to T-Mobile, when an IPv6 device communicates with an IPv6 server, the packets are routed to the server through Verizon's IPv6 network without any stateful NAT middleboxes (Steps 1 and 2). However, when an IPv6 device communicates with

an IPv4 server, the device uses the Gateway-Initiated Dual-Stack Lite (DS Lite), a software installed on the phone, to encapsulate IPv4 packets inside IPv6 headers (Step 3) [79,130]. The encapsulated packets are then forwarded to a middlebox that decapsulates the packet contents and strips out the IPv6 header (Steps 4 and 5). Finally, the IPv4 packets are forwarded to the IPv4 server (Step 6). One of the benefits of using a gateway initiated DS Lite is that encapsulating IPv4 packets in an IPv6 header allows Verizon to use IPv6 addressing for routing packets within the cellular network.

In summary, there are two different ways in which Verizon routes packets from IPv6 devices: 1) via IPv6 network with no stateful middleboxes, and 2) via IPv4-in-IPv6 tunnels using DS Lite software on the phone.

AT&T Mobility (AT&T) and Sprint provide both IPv6 and IPv4 addressing to only some of their IPv6-capable devices [5,64]. For other IPv6-capable and IPv4-only devices, both these networks provide only IPv4 addressing. Therefore, the choice of addressing for devices in these networks is neither dependent on `464XLAT` nor on the cellular technology and is rather likely to be decided by the carrier's respective network configurations.

In Figure 6.3, we depict a high level infrastructure of the IPv6 network deployed by AT&T and Sprint. We show that when an IPv6 device communicates with an IPv6 server, the packets are routed directly to the server, through the IPv6 network without any stateful middleboxes (Steps 1 and 2) [5,64]. However, when an IPv6 device communicates with an IPv4 server, unlike in T-Mobile and Verizon networks, the packets route through the IPv4 network, which consists of several stateful NAT middleboxes, such as NAT 44 and NAT 444, to convert private IPv4 addresses to public IPv4 addresses (Steps 3, 4, and 5).

In summary, there are two different ways in which both AT&T and Sprint route packets from IPv6 devices: 1) via IPv6 network with no stateful middleboxes, and 2) via IPv4 network with several stateful NAT middleboxes.

Data Collection Methodology

Recent studies on measuring CDN adoption rate among websites show that out of the most popular 1,000, 10,000, and 100,000 websites listed on Alexa [44], 77%, 35%, and 19% are hosted on different CDN infrastructures, respectively [15]. Further, a study on understanding the CDN market share indicates that Akamai CDN infrastructure leads in delivering content for majority of the popular websites, including several e-commerce, media, government, news, and social media websites [28] [55]. Specifically, out of the most popular 1,000, 10,000 websites listed on Alexa, as well as, top 500 websites listed on Fortune [17], Akamai delivers content for over 23%, 16.4%, and 32% websites, respectively [31]. Based on these results, we believe that our dataset on cellular network performance collected by globally distributed CDN servers of Akamai is representative of mobile Web performance in general.

We now shift our focus to organize our measurement data collection to accurately represent performance of native IPv6, legacy IPv4, NAT64, and DS-Lite sessions. First, we provide an overview of how we collect performance data from client devices and CDN servers. Next, we discuss techniques used to filter data generated by a number of independent sources, such as client's browser caching of content, Web proxies in the cellular network, and mobile device's operating system. Finally, we describe how we sanitize our measurement data to only contain RTTs, DNS lookup times, and webpage load times for pages loaded over end-to-end (E2E) IPv6 and

IPv4 sessions between clients and CDN servers, as well as pages loaded via NAT 64 middleboxes and DS-Lite.

Experimental Setup: To compare the Web performance perceived by end-users on IPv6 and IPv4 networks, we use Akamai's RUM system [41], as depicted in Figure 6.4. Akamai's Web servers inject JavaScript (RUM Javascript) into a small fraction of user requests for some of the customer-websites hosted on Akamai infrastructure (Steps 1 and 2). The injected JavaScript uses the browser exposed Navigation Timing API to capture the time to resolve domain names, time to establish TCP connections, and webpage load time, among several other metrics [36]. The JavaScript then sends the collected timing data in the form of a `RUM Beacon` to a dual-stacked RUM server after the page load completes (Step 3). The RUM server then sends the data in the `RUM Beacon` to the RUM database (Step 4). Finally, in Step 5, the Web server complements the data into the database with the TCP latency estimated by the Web server that served the webpage, the publicly routable IP addresses of the CDN server and the client, indicator of whether the webpage was available via IPv4-only, or dual-stacked GET requests, an indicator of whether the webpage was requested over IPv4 or IPv6, an indicator of whether the `RUM Beacon` was submitted over IPv4 or IPv6, and the cellular ISP name to which the client's IP address belongs as determined by Akamai's `EdgeScape` [1].

Note: Earlier CDN deployments only served some of the static webpage content. However, as the need for responsive Web performance increased, CPs moved their base pages and other page resources onto replica servers as well. DNS CNAMEing allows base page domain to resolve to a domain page hosted by a CDN [207].

Using Akamai's RUM, in Figure 6.5 we show the round trip TCP latency estimated by Akamai servers when connecting to TCP terminating proxies deployed by

cellular ISPs using measurement techniques developed by Goel *et al.* [148]. The height of each bar graph represents the median TCP latency, and the extreme ends of error bars represent the 25th and 75th percentile of the latency respectively. We show that the median latency between Akamai CDN servers and cellular gateways of T-Mobile and AT&T is only 2 ms and 4 ms, respectively. For Verizon and Sprint, the latency is less than 10ms. We argue that such a low latency is possible only when Akamai's CDN servers are in extreme proximity with the cellular gateways, as opposed to significantly higher latency estimated by Amazon EC2 and PlanetLab datacenters in the US [112]. Therefore, our view of the IPv6 ecosystem using Akamai's infrastructure allows us to isolate the performance differences if IPv4 and IPv6 within the cellular network (without introducing the confounding factor of the public Internet).

Data Sanitization: We filter our dataset to include performance numbers that pertain only to webpages loaded on Google Chrome browser on Android devices. Our choice to eliminate any influence of iOS was based on lack of IPv6 support on iOS devices at the time of this study. During this study, IPv6 support on iOS devices was not available and was only made available in late 2015 with the release of iOS 9 [185]. Therefore, we choose to consider measurement data for Android devices only. Next, in our dataset, we observe that the Web browsers, from which the webpages were mostly requested were Chrome Mobile on Android and Safari on iOS devices. The lack of support for Navigation Timing API on Safari browser installed on iOS version 8 and below motivated us to eliminate any RTT, DNS, and webpage load time related measurement data from our dataset [37].

Recent study on detecting Performance Enhancing Proxies (PEPs) in cellular networks has revealed that cellular networks in the US do not use PEPs for HTTPS traffic [148]. Therefore, to remove any influence of PEPs (in terms of Web content

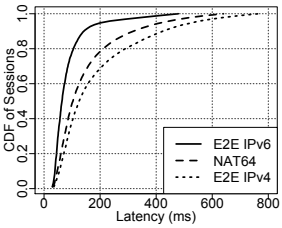


Figure 6.6: RTT distribution for T-Mobile clients.

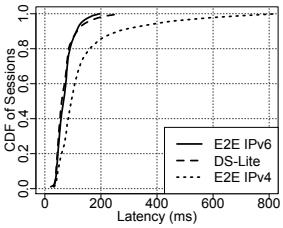


Figure 6.7: RTT distribution for Verizon clients.

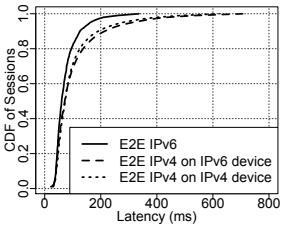


Figure 6.8: RTT distribution for AT&T clients.

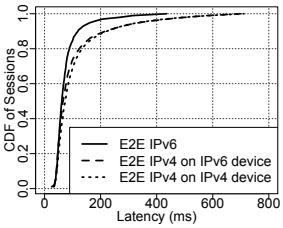


Figure 6.9: RTT distribution for Sprint clients.

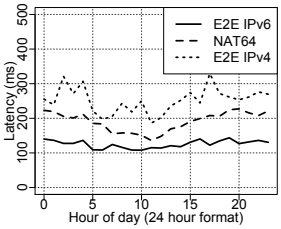


Figure 6.10: 24-hour RTT distribution for T-Mobile.

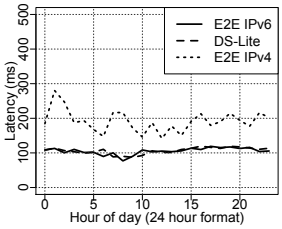


Figure 6.11: 24-hour RTT distribution for Verizon.

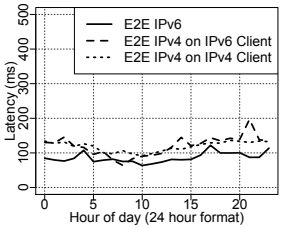


Figure 6.12: 24-hour RTT distribution for AT&T.

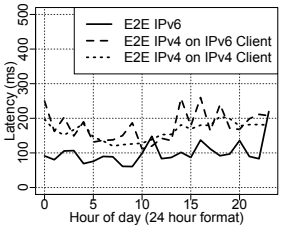


Figure 6.13: 24-hour RTT distribution for Sprint.

caching and TCP split connections) in our dataset, we consider latency for only HTTPS sessions. Latency for HTTPS sessions allows us to accurately estimate the latency between CDN servers and client devices and ensure that the estimated latency is **not** between servers and PEPs in cellular networks. In this work we focus on Web performance over native IPv6 and legacy IPv4 networks, and so eliminate factors, such as presence of PEPs that may confound our measurement data. Analysis of how PEPs in cellular networks impact Web performance is potential future work of this study.

Finally, to remove any influence of content caching in Web browsers on the measurement data, we consider data for only newly created TCP connections. We extract data for connections whose setup time is more than 20ms, which ensures that a TCP socket was created over the cellular network and that an existing connection was not used [38]. We employ a similar technique to extract DNS resolution times

that were resolved at the time of webpage load, thus eliminating the influence of any cached DNS resolutions in the browser.

Our sanitized dataset consists of measurement data for RTT and DNS lookup time for several million sessions between clients and Akamai CDN servers, and webpage load time from several hundred page loads.

Data Analysis: To record the latency over an IPv6 connection we use RTT to clients estimated by CDN servers for webpages requested over IPv6 network. To get the latency over connections via NAT64 (in T-Mobile) or DS-Lite (in Verizon), or by IPv6 clients using IPv4 network (in AT&T and Sprint), we use latency estimated by CDN servers for connections, where webpages were requested over IPv4 network and the `RUM Beacon` was submitted over IPv6 – indicating that the webpage loaded on an IPv6 client. To record the latency over IPv4 connections, we use latency estimated by CDN servers for webpages requested over IPv4 and where `RUM beacon` was also submitted over IPv4 – indicating that the webpage is loaded on an IPv4 client. We apply similar techniques to extract webpage load time.

To record DNS lookup times that pertain to domain names resolved by IPv6 clients, we extract data points for clients on which either websites were loaded over IPv6, NAT 64, or DS-Lite connectivity, or the RUM beacon was submitted to the RUM server over an IPv6. Similarly, to get DNS lookup times that pertain to domain names resolved by IPv4 clients, we extract data points for clients on which either websites were loaded over IPv4 or the `RUM Beacon` was submitted to the RUM server over IPv4 network.

Round Trip Latency over IPv6 and IPv4 Cellular Networks

The Round trip time (RTT) between clients and servers plays an important role in influencing Web performance [151]. In this section, we investigate whether serving mobile content over IPv6 results in lower latency between mobile clients and content servers. We also investigate whether the performance of IPv6 network as well as the performance gap between IPv6 and IPv4 remains same at peak and non-peak traffic hours of a day.

In Figures 6.6–6.9, we show the overall distribution of RTT between clients and CDN servers over IPv6 and IPv4 networks of different cellular carriers, collected in five months of 2015. The solid CDF lines in these graphs show the RTT when IPv6 clients connect to IPv6 servers, over the IPv6 network. The dashed CDF lines show the RTT when IPv6 clients connect to IPv4 servers via NAT64 middleboxes (in T-Mobile), via IPv4-in-IPv6 tunnel (in Verizon), or via the IPv4 network (in AT&T and Sprint). The dotted CDF lines show the RTT when IPv4 clients connect to IPv4 servers, over the IPv4 network. Additionally, in Figures 6.10–6.13, we show the RTT distribution for 24-hour period, averaged over two months (June and July in 2015).

In the case of T-Mobile in Figure 6.6, we observe that the RTT for sessions over IPv6 network is lower than the RTT over the IPv4 network. For example, for median and 80% of sessions, the RTT over IPv6 network is about 49% and 64% faster than RTT over IPv4 network, respectively. Even for sessions via NAT64 middlebox, the IPv6 RTT is lower than RTT over IPv4 network. For example, for connections that go through NAT64 middleboxes, the latencies for median and 80% of sessions are about 18% and 27% faster than RTT over the IPv4 network, respectively.

Further, to eliminate any effects of provisioning differences between IPv4 and IPv6 networks, we compare performance over IPv6 and IPv4 networks at peak and

non-peak traffic hours in Figure 6.10. We observe that the RTT over T-Mobile's IPv6 network outperforms latency over its IPv4 network at all times. In fact, NAT64 sessions also experience lower latency than IPv4 sessions at all times of the day. Although we see that the performance gaps between all three distributions is not same at all times, native IPv6 connectivity always provides least possible latency among others. Based on our observations, we argue that such differences in round trip latency in T-Mobile network arise from the elimination of overhead of NAT middleboxes deployed in the IPv4 network to perform address translation of client sessions. With no middleboxes in case of end-to-end IPv6 connectivity or one middlebox in case of NAT64 sessions, users experience lower latency.

In the case of Verizon in Figure 6.7, we observe that RTT for IPv6 sessions is similar to RTT for DS Lite sessions. We expect the two RTTs to be similar since in case of DS Lite (IPv4-in-IPv6 tunneling) all packets are sent from the device over the IPv6 network, resulting in similar RTT as end-to-end IPv6 sessions. The RTT over IPv4 network however is influenced by Carrier Grade NATs and Large Scale NATs and thus experience significantly higher latency than end-to-end IPv6 or IPv4-in-IPv6 tunneled sessions. For example, for median and 80% of sessions on Verizon, the RTT over IPv6 network is about 29% and 44% faster than RTT over its IPv4 network, respectively. Additionally, when comparing performance over 24-hour periods in Figure 6.11, we observe that IPv6 latency inside Verizon's network outperforms latency over its IPv4 network in both peak and non-peak traffic hours. Based on our observations, we argue that the reduced RTT over Verizon's IPv6 network is due to two major factors: 1) no stateful middleboxes in its IPv6 network, and 2) use of IPv6 connectivity only over LTE network, as opposed to use of IPv4 connectivity over 3G network.[1]

[1]We could not disambiguate our measurement data specific to sessions over LTE and 3G networks, because Akamai RUM uses JavaScript to collect client-side performance and at the time of our

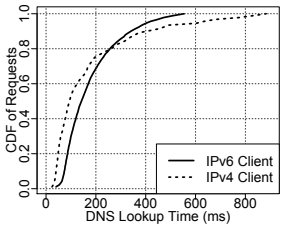


Figure 6.14: DNS Lookup time for T-Mobile clients.

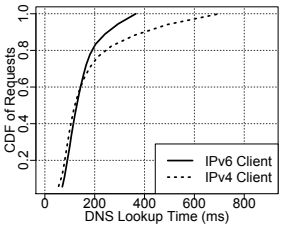


Figure 6.15: DNS Lookup time for Verizon clients.

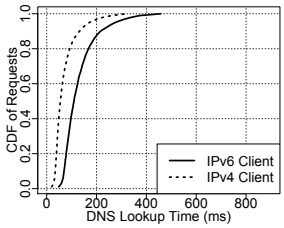


Figure 6.16: DNS Lookup time for AT&T clients.

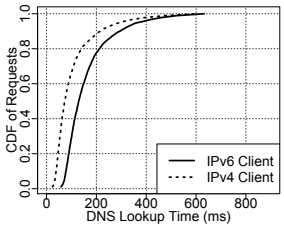


Figure 6.17: DNS Lookup time for Sprint clients.

Finally, in the case of AT&T and Sprint in Figures 6.8 and 6.9 respectively, we observe that RTT over IPv6 network is lower than RTT over their respective IPv4 networks, especially in the long tail. For example, for median and 80% of sessions on AT&T network in Figure 6.8, the RTT for IPv6 network is 17% and 24% faster than RTT over its IPv4 network respectively. Further, the RTT for sessions established by IPv6 clients with IPv4 servers is similar to RTT for sessions established by IPv4 clients with IPv4 servers. We expect the two RTTs to be similar because both AT&T and Sprint are dual-stacked networks and therefore both IPv6 and IPv4 clients must connect to IPv4 servers over their respective legacy IPv4 networks with NAT 44 and NAT 444 middleboxes – resulting in similar latency. Additionally, when comparing AT&T and Sprint's IPv6 network performance with their respective IPv4 networks over 24-hour periods in Figures 6.12 and 6.13, we observe that IPv6 sessions on both AT&T and Sprint experience lower RTT than latency experienced by IPv4 sessions. Therefore, based on our observations we argue that similarly to T-Mobile, IPv6 networks of both AT&T and Sprint outperform their respective IPv4 networks because there are no stateful middleboxes in their IPv6 networks.

measurement (Jan - Aug 2015), Chrome browser did not capture the cellular technology to which the client is connected when loading a webpage [114].

Discussion: Although, we observe that providing mobile content over IPv6 offers reduced latency for end-users, we identified a cellular network outside the US where, during our study, RTT over IPv6 network was higher than RTT over its IPv4 network by almost 200 ms. To investigate, we ran traceroutes from CDN servers to several IPv6 client IP addresses in that network and identified that IPv6 packets were being routed through another country, resulting in higher RTT over its IPv6 network. To investigate whether similar routing was applicable to IPv4 packets in that network at the time of our measurement, we ran traceroutes from the same CDN servers to IPv4 client IP addresses in that network and found that packets were **not** being routed via another country. While it is possible that IPv6 could be slower than IPv4 in some networks, it could be related to how IPv6 packets are forwarded on the Internet. Therefore, proximity of Akamai servers to cellular gateways eliminates the effects of misconfigured routing in the public Internet.

DNS Lookup Time for IPv6 and IPv4 Clients

in addition to RTT, DNS lookup time is another important factor which influences the Web performance in cellular networks [307]. In this section, we measure the DNS lookup time for both IPv6 and IPv4 clients resolving dual-stacked domain names (domains which can be resolved to both IPv4 and IPv6 addresses). In Figures 6.14–6.17, we show the distribution of DNS lookup times when IPv6 and IPv4 clients resolve dual-stacked domain names. The dotted CDF lines represent the lookup time when domains are resolved for IPv6 clients, whereas, the solid CDF lines represent lookup time when domains are resolved for IPv4 clients.

In general and contrary to the trends in the previous section, we see that the DNS lookup takes longer for IPv6 clients than IPv4 clients in T-Mobile, AT&T, and

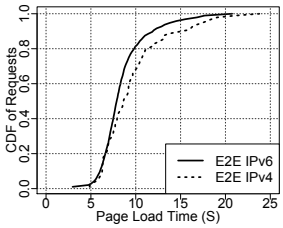


Figure 6.18: Dual-Stack webpage PLT for T-Mobile.

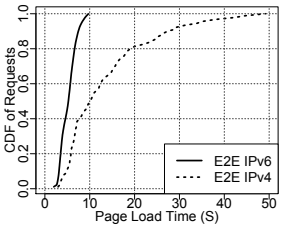


Figure 6.19: Dual-Stack webpage PLT for Verizon.

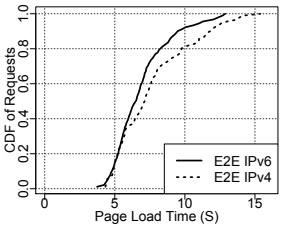


Figure 6.20: Dual-Stack webpage PLT for AT&T.

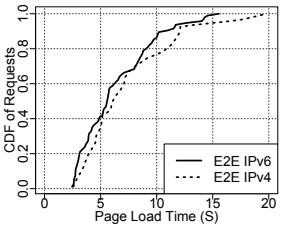


Figure 6.21: Dual-Stack webpage PLT for Sprint.

Sprint networks. However, for Verizon's IPv6 and IPv4 clients, the DNS lookup times are similar. DNS lookup times for IPv6 clients are influenced by their technique of DNS resolution. Client devices are unaware of whether content from a domain is available over IPv4 network or over IPv6 network. Therefore, clients must send both `AAAA` (IPv6) and `A` (IPv4) DNS queries to their local resolvers to resolve domain names. If an IPv6 address for the requested domain is available, Android clients prefer to connect with the IPv6 address, instead of the IPv4 address of the server [317]. Although the two DNS requests could be sent in parallel to reduce the time to perform the DNS lookups, we observe that regardless of the type of domain (IPv4-only or dual-stack), IPv6 Android clients always issue both `AAAA` and `A` DNS queries serially. Further, before returning the DNS response to the application the IPv6 clients wait until responses for both queries arrive, or the resolutions times out. Therefore, the DNS resolution on IPv6 Android clients require two round trips between clients and DNS server, whereas IPv4 Android clients wait for only one round trip for resolving the domain via type `A` query.

In the case of T-Mobile in Figure 6.14, we observe that the median DNS lookup time for IPv6 clients is 25.7% slower than IPv4 clients, because IPv6 clients wait for both type `AAAA` and `A` queries to finish, whereas IPv4 clients wait for only type `A`

queries. However, for about 20% of DNS requests, IPv6 clients experience faster resolution time than IPv4 clients. Since T-Mobile's IPv6 clients can only transmit IPv6 packets into its network, DNS lookups for IPv6 clients take place over T-Mobile's IPv6 network. Our earlier observation from Figure 6.6 shows that RTT over T-Mobile's IPv6 network is lower than IPv4 network, which likely helps about 20% of IPv6 DNS lookups to complete faster in spite of the additional RTT. Therefore, DNS lookups for some IPv6 clients outperform the lookup time for IPv4 clients, even though DNS lookup process for IPv6 clients waits for an additional DNS query to finish.

In the case of Verizon in Figure 6.15, we observe that about 60% of the DNS queries by IPv6 clients take same time as queries by IPv4 clients. Similarly to T-Mobile, IPv6 clients in Verizon network transmit IPv6 packets into the network, which results in DNS lookups over the IPv6 network. From Figure 6.7, we know that RTT over Verizon's IPv6 network is significantly lower than its IPv4 network, therefore the DNS lookup time for IPv6 clients is similar to lookup time for IPv4 clients.

In the case of AT&T and Sprint in Figures 6.16 and 6.17 respectively, we observe that the median DNS lookup times for IPv6 clients are about 38% slower than lookup times for IPv4 client. Since both IPv4 and IPv6 clients in these networks use their respective IPv4 networks to send DNS queries to local resolvers and that RTTs for IPv4 packets sent by IPv6 and IPv4 clients are similar (from Figures 6.8 and 6.9), IPv6 clients wait for responses for two DNS queries in serial, as opposed to IPv4 clients that wait for only one DNS lookup.

Discussion: We observe that the DNS lookup process takes longer for devices with IPv6 capabilities than devices with IPv4-only capabilities in AT&T, Sprint, and partly in T-Mobile networks. Following our observation, we argue that if the DNS lookup process on IPv6 capable Android devices could be modified to send `AAAA` and `A` DNS

queries in parallel, lookup time could be significantly reduced. Additionally, if the client browser could indicate to the mobile OS that they do not need to wait to get the `A` lookup back if they get an `AAAA` answer in DNS response, the DNS lookup time can be further reduced. And finally, for devices in T-Mobile network (and other IPv6-only networks such as Orange Poland and SK Telecom, Telenor [222]), if the OS installed on the device could identify whether the connected network is IPv6-only, the DNS lookup process could only resolve a `AAAA` DNS query, instead of the current implementation where both type `A` and `AAAA` DNS requests are resolved. Such a network specific DNS resolution process will allow clients connected to IPv6-only networks to speed up their DNS lookups. We are working with the Android team at Google to improve DNS resolution process.

Page Load Time over IPv6 and IPv4 Networks

Interactive webpages often require multiple DNS lookups and many round trips between clients and servers to download Web objects onto the client's browser. To investigate the overall impact on Web performance of IPv6's low RTTs and high DNS lookup times, we compare the webpage load time (PLT) over IPv6 and IPv4 networks, using the browser's Navigation Timing API. For this part of the study, we analyze measurement data for two types of webpages: 1) those available over both IPv6 and IPv4 networks (dual-stacked), and 2) those available over IPv4 network only. Our comparison of PLTs only includes page load requests, for which DNS lookups were performed by the client browser and the pages were loaded over newly established TCP connections. Our immediate goal is to eliminate any PLT data that includes DNS lookup from Web browser's cache and reuse of an existing TCP connection.

Table 6.1: Selected mobile device models with highest number of webpage load requests in different cellular networks.

Network	Device Model Name	Model ID
T-Mobile	Samsung Galaxy S5	SM-G900
Verizon	Samsung Galaxy S5	SM-G900V
AT&T	Samsung Galaxy S6 Edge	SM-G925
Sprint	Samsung Galaxy S6 Edge	SM-G925

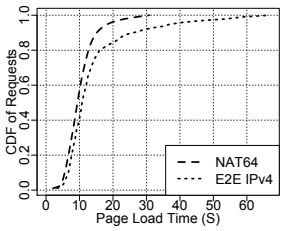


Figure 6.22: IPv4 webpage PLT for T-Mobile.

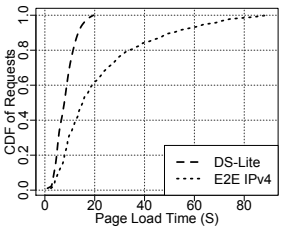


Figure 6.23: IPv4 webpage PLT for Verizon.

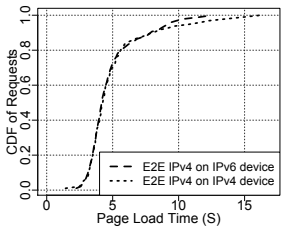


Figure 6.24: IPv4 webpage PLT for AT&T.

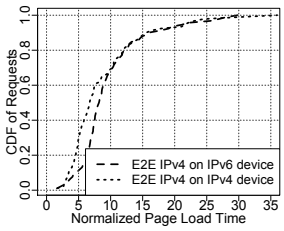


Figure 6.25: IPv4 webpage PLT for Sprint.

To mitigate the influence of mobile hardware on PLT, we consider PLTs from only one specific device model for each cellular network. For each carrier we selected the device model for which Akamai's RUM had highest number of requests for downloading webpage over both IPv4 and IPv6 networks of the corresponding carrier. We list the device models selected for each network from our dataset in Table 6.1.

Further, to accurately characterize the performance of webpage loads over IPv6 and IPv4 networks, we consider PLTs for only one Web URL loaded on one specific device model for each network. Similarly to our choice of device model, we selected the URL for which Akamai's RUM received highest number of requests from the selected device model. Interestingly, the URL for a major postal service website was the only dual-stacked webpage that was loaded the most number of times on all four cellular networks, with the page having 94 embedded Web objects in total and 0.83 MB in size. To prevent the influence of any in-network HTTP caches on PLTs, the page

Table 6.2: Details of IPv4-only webpages loaded over IPv4 networks of different cellular carriers.

Network	Webpage	#Object	Size
T-Mobile	Clothing	444	0.70 MB
Verizon	Internet Retailer	165	1.30 MB
AT&T	Home Improvement	63	0.67 MB
Sprint	Internet Retailer	165	1.30 MB

was loaded over Secure HTTP (HTTPS). Further, we identified different IPv4-only webpages for different carriers that had the most number of page load requests from the selected device. We list the details of IPv4-only URLs selected in each network for performance comparison in Table 6.2 respectively.

IPv6 webpage load time: In Figures 6.18–6.21, we show the distribution of page load time of one dual-stacked webpage loaded by IPv6 clients over networks and loaded by IPv4 clients over IPv4 network. The solid CDF lines show PLTs when the page was loaded over IPv6 network. The doted CDF lines show PLTs when the page was loaded over IPv4 network. In general, we observe that for all four US carriers, the PLTs of pages loaded by IPv6 clients over IPv6 networks are lower than PLTs of the same pages loaded by IPv4 clients over the respective carrier's IPv4 networks. Further, despite DNS lookup times being higher for IPv6 clients, we observe that PLTs are lower for IPv6 clients loading pages over IPv6 network. We argue that DNS lookup times for IPv6 clients influence the overall page load time by just one extra round trip and that the actual benefits of faster IPv6 network are observed when multiple Web objects are loaded over the IPv6 network in several round trips.

In case of T-Mobile in Figure 6.18, we observe that for median and 80% of page loads by IPv6 clients, the PLTs over IPv6 network are 9% and 14% faster than PLTs over T-Mobile's IPv4 network. In Figure 6.19, we observe similar reductions in PLTs

for pages loaded by Verizon's IPv6 clients over Verizon's IPv6 network. Specifically, we show that the median and 80% of the PLTs by IPv6 clients over Verizon's IPv6 network are 48% and 64% faster than PLTs over its IPv4 network, because of the differences in RTTs between Verizon's IPv6 and IPv4 networks as shown in Figure 6.7. For AT&T and Sprint as well, we observe that PLTs are lower over the IPv6 network.

IPv4 webpage load time: In Figures 6.22–6.25, we show distribution of PLTs of an IPv4 webpage, loaded by IPv6 and IPv4 clients. The dashed CDF lines show PLTs when IPv6 clients load an IPv4 webpage via NAT64 (in T-Mobile), via IPv4-in-IPv6 tunnel (in Verizon), or via IPv4 network (in AT&T and Sprint). The dotted CDF lines show PLTs when IPv4 clients load an IPv4 webpage over the IPv4 network. In general, we observe that T-Mobile and Verizon IPv6 clients experience reduced PLTs for IPv4 webpages, due to reduced RTT when using NAT64 in T-Mobile (as shown in Figure 6.6) and the use of LTE and IPv6 network in Verizon (as shown in Figure 6.7). For example, in Figure 6.23, we show that the median and 80% of the IPv4 page loads in Verizon, are about 49% and 67% faster than page loads over IPv4 network. In the case of IPv6 clients in AT&T and Sprint network in Figures 6.24 and 6.25 respectively, we observe that IPv4 webpage PLTs are similar (in AT&T), or occasionally slower (in Sprint) than PLTs experienced by IPv4 clients. We expect the two PLTs to be similar since both IPv6 and IPv4 clients use the IPv4 network to load IPv4 websites. However, in some cases we expect the PLTs by IPv6 clients to be slower than PLTs by IPv4 clients, since such clients wait for an additional DNS query for each domain in the webpage.

Although we only show performance gains with IPv6 for only one URL loaded on one type of device model, we also looked at other URLs loaded from other device models as well. We identified that IPv6 connectivity also improves the page load time

of other URLs, though due to space constraints we do not show PLT distributions for other URLs and devices.

Discussion: Based on our findings on webpage load times, we show that for IPv6 clients in T-Mobile and Verizon networks, both IPv4-only and dual-stacked websites will load faster than IPv4 clients loading the same websites over the carrier's respective IPv4 network. Although, we observe DNS lookups are slower for IPv6 clients in T-Mobile, AT&T, and Sprint networks, we observe their impact on the overall PLT over these networks to be minimal, since the RTT over IPv6 network is lower than RTT over IPv4 networks, which accounts for most of the round trips when loading a webpage [151].

For dual-stacked networks, such as AT&T and Sprint, IPv6 clients in such networks experience faster page loads for only dual-stacked websites. Further, websites available over IPv4 only may occasionally experience poor performance on IPv6 clients, likely due to slower DNS lookups that lower IPv6 latency cannot offset. Finally, we argue that since users are likely to visit websites from different networks, providing mobile content over IPv6 will not hurt the Web performance and in fact in some cases the Web performance will improve if pages are served over IPv6.

DNS Lookups in T-Mobile's IPv6-only Network

As cellular carriers upgrade their network infrastructure to IPv6 to mitigate IPv4 address scarcity, we find that the DNS protocol starts to introduce avoidable performance overhead. We observe an extra round trip in cellular ISPs, such as T-Mobile, Orange Poland, SK Telecom, and others [222], which use IPv6-only

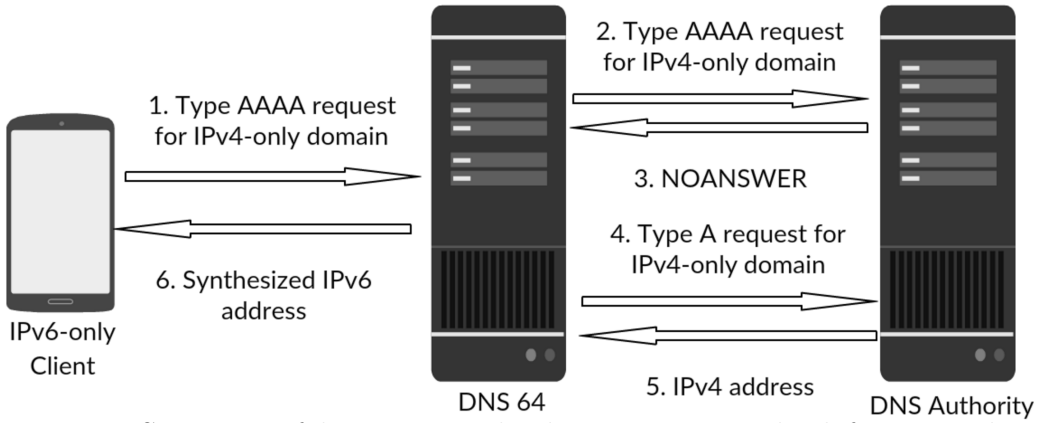


Figure 6.26: Sequence of how IPv4-only domains are resolved for IPv6-clients in IPv6-only networks.

addressing in their network [90]. In effect, ISPs that fully embrace IPv6 technology are penalized with slower domain name resolution.

We illustrate the problem scenario in Figure 6.26, which shows a sequence of DNS messages exchanged between a user device, T-Mobile's DNS 64 server, and a DNS Authority. An IPv6-only environment requires a mobile client to send a `AAAA` DNS query to the cellular DNS server to resolve an IPv4-only domain (Step 1). The DNS request then travels to the DNS Authority (Step 2). The DNS Authority replies with `NOANSWER` flag in the DNS response (Step 3), because an IPv6 address is not available for the IPv4 domain in question. Instead of returning the DNS response with `NOANSWER` back to the client, T-Mobile's DNS server sends a subsequent `A` DNS query for the same domain (Step 4), to which the DNS Authority replies with an IPv4 address (Step 5). After receiving the IPv4 address, T-Mobile's DNS server synthesizes an IPv6 address corresponding to the IPv4 address and sends the synthesized address to the client in response to the client's DNS request (Step 6). At this point, the mobile client is unaware whether the returned IPv6 address is a synthesized address, or a real address returned by the DNS Authority. Since T-

Mobile employs NAT 64 middleboxes to translate synthesized IPv6 addresses back to real IPv4 addresses (Figure 6.1), the synthesized address that the clients receive does not alter their end-to-end connectivity [67, 90].

Thus, if IPv6 addresses are available for a domain, the DNS lookup will finish in Step 3. However, when a domain has only IPv4 addresses available, the DNS lookup requires an extra round trip between the cellular DNS and the DNS Authority (Steps 4 and 5) [90]. The latency of this extra round trip could significantly influence the PLT when cellular DNS servers are not in proximity to DNS Authorities and the webpage requires multiple DNS lookups.

To address this overhead in DNS lookup process, we design OneTrip – a technique for DNS Authorities to eliminate the extra round trip from DNS lookup process in IPv6-only mobile networks. We show that for mobile clients in IPv6-only networks, a DNS Authority can proactively synthesize IPv6 addresses from IPv4 addresses, for all domains it holds mappings between domains and IPv4 addresses. The DNS Authorities could then reply with a synthesized IPv6 address, instead of a `NOANSWER`, to any IPv4-only domain name lookup from IPv6-only networks.

ONETRIP's Approach

To synthesize an IPv6 address from an IPv4 address, similarly to how T-Mobile's DNS 64 servers synthesize [90], OneTrip requires two types of datasets that identify the `/64` prefix used to synthesize an IPv6 address by cellular DNS servers. The first are mappings between client and cellular DNS server IP addresses. The second are mappings between cellular DNS server IP addresses and the `/64` prefix used by the NAT 64 middlebox to which their clients connect. The two datasets collectively allow OneTrip to map a client IP address to `/64` prefix, used by the NAT 64 middlebox associated with each DNS (DNS 64) server. OneTrip makes these mappings available

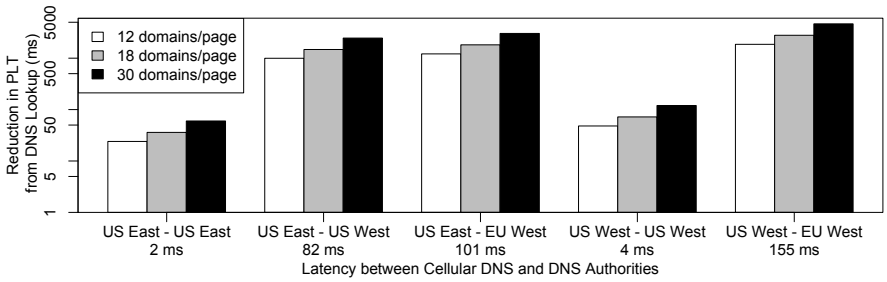


Figure 6.27: Reduction in DNS Lookup time when using OneTrip on DNS Authority.

to DNS Authorities so that when a cellular DNS server sends a request to the Authority, the Authority already knows (based on cellular DNS IP in the DNS query) the `/64` prefix of the NAT 64 middlebox to which the client is connected. The Authority then uses the corresponding `/64` prefix to synthesize the IPv6 address from the IPv4 address associated with the IPv4-only domain name in question.

Next, we describe OneTrip's approach to collect the above mappings using Akamai's global infrastructure of content delivery.

Collecting DNS IP – NAT 64 Prefix mappings: Using the crowd-sourced dataset collected by Netalyzr in over 11 months in 2013-2014 [183], we extracted mappings between cellular DNS server IP addresses and the `/64` prefixes used by `NAT 64` middleboxes deployed in T-Mobile's network [297]. The Netalyzr mobile application sends a `AAAA` DNS request to a cellular DNS server for resolving an IPv4-only domain name whose DNS Authority replies with the IP address of the DNS server that made the request. From the DNS response received by the client, we extract the IP address of the cellular DNS server by converting the last (least significant) 32 bits of the synthesized IPv6 address format from HEX to DEC. We also extract the `/64` prefix from the first (most significant) 64 bits of the synthesized IPv6 address.

Collecting Client IP – DNS IP mappings: For the purpose of this work, we modified our measurement system to make clients resolve a unique hostname that allows us to map the client IP addresses to the cellular DNS IP addresses in a manner similar to the above.

Finally, from the above two mappings we generate the associations between client IP addresses and `/64` prefixes that enable DNS Authorities to identify the NAT 64 middlebox, to which the clients connect. These mappings are also useful when DNS requests use the `EDNS0` extension (containing the IP address of the mobile client) for CDNs to perform server selection based on the client IP instead of the DNS server IP [98]. Specifically, when ISP resolvers send IPv6 client addresses in the DNS request, DNS Authorities search for the NAT 64 address, to which the client is currently connected, and then perform mapping based on the location of NAT64 address in the network. DNS Authorities may also combine NAT64 location information with the (limited) information about the location of client's IPv6 address.

Additionally, the mappings between client IP and NAT64 prefixes enable Web/proxy servers to reduce latency for HTTP transactions, as opposed to using synthetic IPv6 addresses only for the purposes of DNS lookups. Specifically, servers that currently embed static IPv4 addresses in HTTP headers or payload [90], can now embed synthesized IPv6 addresses to sidestep `464XLAT` software on resource limited mobile devices. This procedure reduces latency perceived by end users because the `464XLAT` software on the client first converts the IPv4 address (returned in HTTP response) into an IPv6 address. Next, IPv6 packets are forwarded to the NAT64 middlebox that converts them back to IPv4 packets for the IPv4 host. By allowing servers to embed synthetic IPv6 addresses in HTTP headers, or payloads, latency introduced by 464XLAT software on the client is eliminated.

Discussion: Based on the mappings we collect between clients and DNS server IP addresses, we find that each DNS IP is used by thousands of mobile clients to resolve domain names. Therefore, OneTrip's implementation on DNS Authorities can improve DNS lookup time for thousands of cellular clients for every IPv4-only domain name in resolution. We also observe from the data collected by Netalyzr that the DNS server address and `/64` prefix mappings were stable and that there were no changes over a period of 11 months in 2013-2014. We argue that such demonstrated historical mapping stability supports OneTrip's approach to reliably synthesize IPv6 addresses on the DNS Authorities, however there is a serious risk that doing this mapping outside of operator control could impair their ability to operate their network. The OneTrip mechanism is however a good way to identify the performance improvements available by improving the DNS protocol.

Speeding DNS Lookups with ONETRIP

We discussed OneTrip's approach for DNS Authorities to eliminate the extra round trip during DNS resolutions. However, it still remains unclear whether T-Mobile's cellular DNS would honor, or reject, the IPv6 addresses synthesized by an Authority outside of T-Mobile's network. To answer this question, we configured IPv6 clients in T-Mobile's network to send `AAAA` DNS requests to their respective cellular DNS servers. We also setup a DNS Authority, outside of T-Mobile's network, maintaining DNS records for a domain name with only IPv4 address. Using the two mappings collected by OneTrip, we configured the DNS Authority to reply with a synthesized IPv6 address to the DNS request for the IPv4-only domain name, instead of a `NOANSWER`.

Our experiments show that T-Mobile's DNS servers do not discard any DNS replies with IPv6 addresses synthesized by a DNS Authority outside of the T-Mobile's

network. Further, the mobile clients successfully connect to the IPv4 servers, using the synthesized IPv6 address.

Finally, we perform several experiments to understand the performance improvements that OneTrip brings to DNS content delivery when DNS Authorities are hosted at different proximity from cellular DNS servers. Specifically, our evaluation of OneTrip is based on several possible latency values that exist between cellular DNS servers and DNS Authorities. We expect geographically-distributed DNS Authorities to have lower latency to cellular DNS servers than a centralized DNS Authority. For example, a previous study has shown that Internet backbone round-trip time is around 82 ms between the East and West coast in the US [286]. The same study also provides latencies between different regions in the US and the EU. We use these latency values as representative of latency between cellular DNS servers and DNS Authorities hosted at these different locations.

A recent study by Varvello *et al.* shows that the 25th, 50th, and 75th percentile of websites (out of the top 9000 Alexa Websites that support HTTP/2) consists of over 12, 18, and 30 unique domain names, respectively [299]. We therefore argue that depending on the number of unique domain names in a given webpage and the latency between cellular DNS servers and DNS Authorities, the webpage load time could be significantly improved through OneTrip's removal of one round trip per resolution.

In Figure 6.27, using simulations we show the absolute reduction in DNS lookup time when using OneTrip on DNS Authorities. To measure the effectiveness of OneTrip, we perform in-lab simulations to emulate the behavior of T-Mobile's DNS 64 servers, where we perform DNS lookups from an emulated cellular DNS issuing name resolutions to a DNS Authority. On the x-axis, we represent five different latency values between cellular DNS and DNS Authority. On the y-axis, we show the

latency reduction in the DNS lookup time achieved when OneTrip is used on the DNS Authority.

In general, we see that as the network latency between cellular DNS and Authority increases, the gains increase as well because OneTrip shaves off the extra round in the DNS lookup process. The overall gain further increases when the number of unique domain names associated with different Web objects embedded in a webpage increases, because OneTrip reduces the latency for resolving each of these unique domain names. For example, when the latency between cellular DNS server and DNS Authority is about 82 ms and the number of unique domain names (that needs to be resolved from that Authority) on a webpage is 18, OneTrip reduces the overall DNS lookup time by about 1.4 seconds (18 * 82 ms). The latency reduction shown in Figure 6.27 is the maximum saving OneTrip offers when all domain name lookups are critical for webpage rendering; the minimum benefit is just one RTT saved on the lookup of the base page. In general, as OneTrip reduces the number of round trips between the cellular DNS and DNS Authorities from two to one, OneTrip offers a reduction of about 50% in the DNS lookup time between cellular DNS servers and DNS Authorities in IPv6-only network environments for IPv4-only content.

Discussion on ONETRIP's Approach

OneTrip relies on identifying the `/64` prefix associated with the NAT 64 middlebox, to which a client connects. Similarly, previous studies have developed several techniques to detect the presence of NAT 64 middleboxes in the network [120], including techniques to resolve an IPv4-only domain name with a `AAAA` DNS request and checking if an IPv4 answer is available [277]. Other studies have developed techniques to identify IPv4 addresses from synthesized IPv6 packets [72]. Some other techniques showcase new extensions to DNS protocol and a new Resource Record that

DNS servers can adopt to let clients know what the original IPv4 address is for the domain name in question [82,182,316]. Previous studies also investigated application layer protocols such as STUN to detect the NAT 64 prefix [268]. A study by Ding *et al.* offers a detailed comparison of different techniques used to identifying NAT 64 prefix in IPv6 networks [121]. The same study also showcase how the `EDNS0` extension could be used by DNS resolvers to send NAT 64 prefixes in the DNS request, together with the the technique to calculate the prefix by resolving an IPv4-only domain name.

OneTrip, in contrast to the previous techniques, is unique in that it enables DNS Authorities to detect and effectively use NAT 64 prefixes to reduce DNS lookup time, without the need of any support from the cellular ISPs, allowing measurement of the performance difference. Our motivation for OneTrip is to eliminate the extra round trip present in DNS lookups in IPv6-only networks, with the goal of faster mobile Web for the end-users. Similar round trip elimination has also been proposed in QUIC [171] and TCP Fast Open [255]. Although for eliminating the extra round trip, we also recommend the use of the `EDNS0` option in the DNS query for the cellular DNS servers to pass along the NAT 64 prefix that they are using, such that in the absence of an `AAAA` record DNS Authorities could synthesize one from the `A` record using the contents of the `EDNS0` option. However, we argue that support for such an `EDNS0` option may not be appealing for some cellular ISPs as it introduces additional operational overheads in ISPs' functionality. Therefore, we designed OneTrip for DNS Authorities maintained by Content Providers, Content Delivery Networks, and any independent server operator, which they can implement without any support from cellular ISPs. A strictly better option for Content Providers is to make their content available over IPv6 as this reduces the round-trip by making the `AAAA` record available. Finally, although OneTrip does not address performance overhead introduced by Android DNS lookup process when IPv6 clients wait for both `AAAA` and `A` replies (as

discussed in Section 6), our recommendations to Google's Android team would address the issue of sequential lookups.

Related Work

IPv6 adoption and deployment challenges: Previous studies have measured the adoption rate of IPv6 across different ISPs worldwide and indicate a significant growth in IPv6 traffic over the recent years [29, 110, 180, 222, 274]. While understanding the adoption of IPv6 is important, several communities (including network operators, CDNs, and content providers) have shown interest in discussing challenges faced by mobile ISPs and content providers to adopt IPv6 in their infrastructures in a panel at the @Scale conference held in 2015 [34]. Several case studies conducted at Akamai and Fortinet provide experiences with IPv6 deployment on a global scale [221] and the current state of the IPv6 in terms of information security [30, 142].

IPv6 performance measurement: A study by Dhamdhere *et al.* investigates the impact of BGP route changes on the performance of IPv6 networks [119]. Plonka *et al.* investigate the flow bit rates of IPv4 and IPv6 traffic at different times of the day [254]. A study by Donley *et al.* investigates the impact of several NAT middleboxes on IPv4 latency and offers a performance comparison between IPv4 and native IPv6 connectivity in wired networks [124]. Other studies investigate throughput, RTT, packet loss, and hop counts provided by IPv6 networks [186, 208, 312, 325, 330].

Addressing IPv6 connectivity issues: At the 2015 @Scale conference, a representative from Verizon Wireless suggested that browsers should not fallback on IPv4 when IPv6 connectivity is slow, because a fallback to IPv4 can mask critical performance issues related to IPv6 connectivity [34]. Collectively, several mobile operators suggested that application developers should support IPv6 connectivity to

help kickstart a transition from IPv4 to IPv6. This suggestion complements Apple's recent announcement of IPv6 support in all iOS9 applications [185].

Redirecting clients from broken IPv6 links: Several studies suggest methods to improve Web performance by selectively handing out answers to `AAAA` queries based on the performance of the current IPv6 connectivity is behind the client's resolver network [109, 159, 181, 221].

Our work, in contrast to these studies, focuses on understanding how different IPv6 deployment strategies in cellular networks influence the mobile Web performance, from the perspective of CDNs deployed in the middle of cellular networks and content providers. Based on our study, we argue that hosting mobile content on IPv6-enabled networks and content servers is another direction that CDNs and content providers could adopt to improve the end-user experience.

Conclusions

Content Providers (CPs) and Content Delivery Networks (CDNs) are not fully aware of the differences in mobile Web performance in cellular IPv6 and IPv4 networks. In this paper, we provide our experience with the changing IPv6 ecosystem in major cellular networks from the perspective Akamai's global infrastructure for content delivery. We perform extensive measurement to understand how different IPv6 technologies deployed by cellular carriers impact mobile Web performance. Our results indicate that cellular IPv6 networks outperform their legacy IPv4 networks. We argue to CPs and CDNs that the transition from IPv4 to IPv6 is another milestone for improving mobile Web performance.

THE PROPOSAL

In spite of several years of research to improve the mobile Web performance, quest for even better performance still remains. My conjecture in my PhD proposal is that improvements to application layer protocols can improve the mobile Web performance and thus the quality of user experience. Therefore, in this chapter, I introduce three of the most pressing problems in the area of mobile Web performance that remain unaddressed till date.[1]

1. HTTP/2 Performance in Cellular Networks

The new HTTP protocol (HTTP/2) was recently introduced by IETF after 16 years of extensive analysis of the legacy HTTP/1.1 protocol [78]. HTTP/2 is primarily designed to reduce the webpage load time. The HTTP/2 protocol is built on the legacy HTTP/1.1 protocol and introduces the following changes in the HTTP protocol:

- **Elimination of the Head-of-line blocking:** The Head of line blocking happens in the HTTP/1.1 protocol because only one HTTP request could be processed by the Web server at any given point in time. The HTTP/2 protocol eliminates the head-of-line blocking by allowing multiple independent HTTP streams to be created over a simgle TCP connection.

- **Request Prioritization:** In HTTP/1.1, Web browsers were not allowed to indicate to the Web server the priority in which the Web browser would like to receive the contents of the webpage. In HTTP/2, the Web browser is enabled

[1] Although, I introduce three challenging problems related to improving mobile Web performance in cellular networks, however, I plan to work on only one of the three problems towards my PhD dissertation.

to indicate a priority number to the Web server so that that most important Web object can be downloaded before other objects with low priority.

- **HTTP Header compression:** In HTTP/1.1, the Web browsers are required to send client's meta-data in each HTTP request. Such meta-data includes the information such as the client's browser name and version, operating system, client's compression capabilities of the payload, etc. Further, such meta-data was redundant across HTTP requests sent for the same webpage to the same web server. In HTTP/2, however, these redundant meta-data have been removed and only new or modified HTTP headers are transmitted. The headers are then compressed using HPACK algorithm to ensure further reduction in the header size [248].

With such features, HTTP/2 is expected to bring significant improvements to Web performance. In fact, several studies have already been conducted to investigate the performance of the HTTP/2 protocol, but only on either wired networks or under simulated environments. Some of these studies show that HTTP/2 does not improve Web performance for all the websites and in fact for some websites, the performance gets degraded [116,129,310]. While some other studies show that HTTP/2 is effective in improving the webpage load time for all the websites [299]. However, disagreement among results from these studies, it remains unclear as to whether serving Web content over HTTP/2 brings any improvements to webpage load time or not. For example, does upgrading a website content to IPv6 improves the performance or upgrading to the HTTP/2 protocol, or both?

In this study, I seek to perform a detailed and large scale measurement on HTTP/2 performance in cellular networks worldwide. My goal for this study is to understand the performance improvement or degradation that the HTTP/2 protocol

brings for cellular clients, in comparison to the legacy HTTP/1.1 protocols. My criteria for comparison between HTTP/2 and HTTP/1.1 protocol performance is based on the time the protocol takes to perform Transport Layer Security handshakes, time to receive the first bit of the HTML DOM, and the overall page load time. Further, I seek to investigate whether or not the performance of HTTP/2 varies for clients in different cellular networks. I also seek to investigate whether the HTTP/2 performance varies across Web browsers, mobile devices, and different times of the day. One of the questions I am interested to look at is, which combination of IP protocol (IPv4 or IPv6) and HTTP protocol (HTTP/2 or HTTP/1.1) works best for improving the mobile Web performance. In other words, would it be beneficial to upgrade a website's content to IPv6 and remain on HTTP1.1? or to remain on IPv4 but upgrade the website content to HTTP2? Would for some websites, upgrading website content to both IPv6 and HTTP/2 improves the overall performance? I believe my contributions in this work would set another milestone in understanding mobile Web performance, similar to my IPv6 study [149].

2. Impact of Web Proxies on Video Streaming Quality in Cellular Networks

Cellular networks deploy connection terminating proxies to isolate the performance issues of the cellular radio channels from the wired backbone of the Internet [80, 148]. Such proxies allows cellular carriers to improve the end-to-end performance of the two connections together [80]. Using these proxies cellular networks can also cache popular content in their network and serve the client requests from its local network, instead of forwarding the request to a host outside of the network. Several simulation studies have investigated the performance of these connecting terminating proxies on Web performance and have found that these proxies are effective in reducing webpage load time under simulated cellular environments and

real world wired networks [81, 127, 134, 214]. However, the video streaming quality is not measured in terms of page load time and therefore the conclusions from these studies cannot be applied directly towards improving the video stream quality. The videos streaming quality is measured in terms of the video startup delay, number of buffering events, time to fill the buffer, encoded video bitrate, total waiting time/total watching time, total video length, etc.

In this study I seek to investigate the impact of connection terminating proxies on video streaming by understanding the extent to which these factors are affected by a use of proxies, especially in cellular networks. My approach is to compare the video streaming quality when proxies are used against cases when proxies are not used. For this part of the study, I plan to use the techniques I developed previously to detect connection terminating proxies in cellular networks worldwide [148]. I then seek to investigate whether similar video streaming quality persists across different cellular networks and mobile devices. One of the other questions I am interested in investigating is, whether bypassing connection terminating proxies can result in better performance than letting the proxy split the connections? If so, I would then argue that CPs and CDNs should use Secure HTTP to transfer video chunks over cellular networks, as Secure HTTP connections are currently not split by cellular ISPs [148]. I believe that this study would set another milestone in understanding and improving the quality of video streaming experience for mobile users.

3. IP-to-Location Services for Cellular Networks

IP-to-location services have been a great area of interest among developers and the research communities, as such services allow for optimized delivery of the request content. Some popular areas for which IP-to-location services are used are as follows:

- **Infrastructure Planning:** Content Delivery Networks care about deploying sufficient capacity close to where the most requests come from the users. As a result, CDN providers use IP-to-location services to identify the locations where users requests come to their CDN servers, based on the client's IP address. Such knowledge allows CDNs to monitor the usage of their current deployment and analyze whether more capacity would be required at any given deployment location.

- **DNS-based Server selection:** DNS-based server selection techniques are most popular among CDN providers, as such technique allow CDNs and CPs to find a server closest to the IP address of the DNS server from where the requests are coming for resolution. Therefore, DNS Authorities hosted by CDNs and CPs use IP-to-location services to identify a geographic region where the client's DNS server is hosted. Although, an accurate location of the DNS server in this case may not needed, but an IP-to-location service could provide accurate country, state, and city level information to the DNS Authorities.

- **End-user Mapping:** Many DNS service providers have emerged in recent years, such as Google public DNS, Open DNS, etc. As DNS providers grow, clients get several options to configure the DNS on their devices. Therefore, as clients configure for DNS servers not hosted by their home ISP, the notion of clients being close to their DNS becomes invalid. Therefore, End-user was introduced as an alternative to DNS-based mapping, where CDNs and CPs could find servers near to the location of client's IP address, instead of the DNS IP address. To achieve this goal, CDNs and CPs use IP-to-location services to get an approximate location of where the clients might be in the network.

- **Business Modelling:** Online business companies care about the growth of their userbase. They also show interest in where the client requests are coming, who are their major customers, from which geographic locations they see high popularity of their content. Therefore, online business companies use IP-to-location services to map their users to different locations, with the goal of providing faster, appropriate, and targetted content to user population in different areas of the world.

- **Customer Billing:** Generally, customers of a CDN provider are billed based on how many requests the CDN served for a given customer. Some CDN providers also have different prices for serving content from different locations. For example, if cost of electricity for running a datacenter is cheaper in Europe, then the cost of serving the content from Europe datacenters could be cheaper than serving from any other place. Therefore, when a billing cycle appears, CDN providers calculate the number of requests seen for a customer from different areas of world using IP-to-location services.

- **Data Security:** Finally, application developers and content providers may desire to generate content based on the geographic locations. For example, some advertisements or content related to an event may be relevant or need to be accessed from only a few places in the world. Therefore, content providers use IP-to-location services to identify whether the client requesting the content belongs to one of the white-listed locations.

IP-to-location services have become important in recent years, because of different benefits they bring to the online business companies. Although, such services are reliable and often provide up-to-date information when polling them for clients IP addresses on the wired Internet, however, these services still lack accuracy when

polling for information to clients IP addresses in cellular networks. This is because of the following five reasons:

- First, cellular carriers employ dynamic assignments of IP addresses among different packet gateways deployed in a given country [21]. Therefore, anytime there is a change in IP address pool that any gateway holds, the cellular ISP needs to communicate the change to all of its gateways. Consequently, these changes needs to be propagated to all such tools that seek to maintain IP address to location for cellular IP addresses. However, when cellular ISPs frequently reassign IP address blocks, it becomes extremely difficult for such IP-to-location services to maintain an up-to-date information for cellular IP addresses.

- Second, a cellular ISP may use a single block of IP subnet for users at geographically different locations to support reuse of their limited IPv4 address space, via NAT middleboxes [69, 294]. In such cases, locating cellular users becomes even more challenging.

- Third, mobile roaming users are often identified at locations far away from their actual locations, because such users connect to their home gateway, instead of the foreign gateway of the cellular network [218, 294].

- Next, cellular networks dynamically forward user requests to different gateways to either distribute the load on their overall cellular infrastructure or to allocate more bandwidth to some areas in the network [21]. For example, a study on AT&T's DNS protocol behavior shows that for AT&T clients in Washington, AT&T encoded IP address in the DNS request that belonged to California.

- And finally, with IPv6 seeing significant adoption among cellular carriers, it becomes extremely difficult to use standard IP-to-location techniques to scan the

complete IPv6 address space. This is because the IPv6 address space consists of 340 trillion trillion trillion IP address and scanning such a huge address space is practically not possible in even hundreds of years to come [9,160]. As a result, it becomes difficult for content providers, CPs, and IP-to-location service providers to reliably identify the location of cellular clients. Therefore, the the benefits of using IP-to-location services when it comes to cellular networks are minimized. Therefore, what is needed is an application layer technique that does not rely on expensive and time consuming measurements to identify locations of IPv4 and IPv6 addresses.

As cellular network ecosystem changes dynamically in terms of how cellular ISPs assign IP addresses to their mobile clients, the need to monitor these network changes and the growth of cellular carrier becomes important for CDN operators. In this study, I seek to investigate the stability and consistency of the IP address assignments in cellular networks deployed in the US. More specifically, from my work on detecting cellular middleboxes, I identified that CDN servers of Akamai estimate round trip latencies of less than 5 ms when connecting splitting proxies are present in the network. I argue that since CDN servers only communicate with these Web proxies, what matters the most to CDNs is to identify the locations of these Web proxies. Specifically, I argue that Web proxies in cellular networks are good surrogate for clients' locations, as connections from cellular clients are terminated by these Web proxies. Therefore, approximating a client's locations based on how far the Web proxy (currently in use by the client) is from the CDN server, in terms of latency, should accurately reflect clients location in the network. For example, if for a given set of IPv4 or IPv6 connections received by a CDN server, the latency is 2 ms and we know the location of that CDN server, we could then confidently say that the

observed IP addresses belong to the location same as (or in close proximity) to the CDN server. Based on this conjecture, I plan to investigate how different IPv4 and IPv6 addresses could be clustered and mapped to different network locations. I also plan to investigate whether meaningful mappings between clusters of IPv4 addresses and IPv6 addresses could be generated, so that one could identify locations of IPv6 addresses using location traces of IPv4 addresses. I believe such a measurement study would set another milestone in monitoring and understanding the growth of the mobile ecosystem.

CONCLUSIONS

The mobile ecosystem is growing rapidly. In order to ensure significant growth of innovative applications, in terms of scalability and usability among mobile users, we need to ensure that network and application layer protocols are efficiently used. In this proposal, I argue for a shift in our current research directions from addressing network layer challenges to application layer. Specifically, I present several techniques that bring intelligence into the application layer protocols and thus improve the end-user experience. I then present three of the most challenging problems that remain unaddressed in the mobile space and argue that work in the proposed direction would enable the Internet to scale and sustain in the years to come.

RESEARCH TIMELINE

I plan to complete my PhD dissertation by April 2017. The following is the timeline for the proposed work:

Table 9.1: Research Timeline.

Begin	**End**	**Goal**
-	March 2016	PhD Proposal
April 2016	Feb 2017	Work on a proposed problem
Feb 2017	April 2017	Write and defend PhD dissertation

REFERENCES CITED

[1] Akamai EdgeScape. http://uk.akamai.com/dl/brochures/edgescape_service_description.pdf, Mar. 2002.

[2] PlanetLab. https://www.planet-lab.org/, Jun. 2007.

[3] UMass Trace Repository. http://traces.cs.umass.edu/, Dec. 2009.

[4] NetNetwork. https://github.com/pragma-/networklog, 2011.

[5] Sprint to Participate in World IPv6 Day. http://newsroom.sprint.com/news-releases/sprint-to-participate-in-world-ipv6-day.htm, Apr. 2011.

[6] The 'secret' app installed on millions of mobile phones that records your keystrokes, your browsing and reads your messages. http://www.dailymail.co.uk/sciencetech/article-2068225/Secret-app-installed-millions-Android-phones-reads-messages.html, Dec. 2011.

[7] Akamai Mobitest: Mobile Web Performance Measurement Agents. https://code.google.com/p/mobitest-agent/source/browse/trunk/mobitest-agent/Android/BZAgent/README?r=2, Mar 2012.

[8] Geni. http://www.geni.net/, Oct. 2012.

[9] Just how many IPv6 addresses are there? Really? http://rednectar.net/2012/05/24/just-how-many-ipv6-addresses-are-there-really/, May 2012.

[10] Archipelago measurement infrastructure. http://www.caida.org/projects/ark/, June 2013.

[11] FIRE: Future Internet Research and Experimentation. http://www.ict-fire.eu/, July 2013.

[12] Ripe atlas. http://atlas.ripe.net/, July 2013.

[13] ROOT Metrics. http://www.rootmetrics.com/, July 2013.

[14] Devices Supported by SciWiNet. http://sciwinet.org/SciWiNet-Devices.html, 2014.

[15] Dyn Research: CDN Adoption By The Numbers. http://dyn.com/blog/dyn-research-cdn-adoption-by-the-numbers/, Jun. 2014.

[16] Everything You Need To Know About CDN Load Balancing. http://www.webtorials.com/main/resource/papers/Dyn/paper1/CDN-LoadBalancing.pdf, Sept. 2014.

[17] List of Fortune 500 companies and their websites. http://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites, Jul. 2014.

[18] LiveLabs Registration. http://athena.smu.edu.sg/livelabs_register/, Oct 2014.

[19] M-Lab Tests. http://www.measurementlab.net/tests, 2014.

[20] Perfecto Mobile. http://www.perfectomobile.com/, 2014.

[21] PointRoll IP. http://www.pointroll.com/wp-content/uploads/2015/02/IP-Geo-Location-FAQ.pdf, Jul. 2014.

[22] Portolan network tools. https://play.google.com/store/apps/details?id=it.unipi.iet.portolan.traceroute&hl=en, May 2014.

[23] SciWiNet. http://sciwinet.org/, 2014.

[24] Sensibility testbed. http://sensibilitytestbed.com, July 2014.

[25] Traffic Director. http://dyn.com/traffic-director/, Sept. 2014.

[26] USTREAM. http://www.ustream.tv/, July 2014.

[27] What is Measurement Lab? http://www.measurementlab.net/about, 2014.

[28] Akamai Facts and Figures. https://www.akamai.com/us/en/about/facts-figures.jsp, Aug. 2015.

[29] Akamai's State of the Internet. http://www.wsta.org/wp-content/uploads/2013/11/Q1-2015-SOTI-Security-Report-Low-Res.pdf, 2015.

[30] Akamai's State of the Internet/Security. https://www.stateoftheinternet.com/downloads/pdfs/2015-q1-state-of-the-internet-report.pdf, 2015.

[31] Alexa and Fortune 500 CDN Marketshare. http://blog.cloudharmony.com/2015/03/cdn-marketshare-alexa-fortune-500.html, Mar. 2015.

[32] Apptimize. http://apptimize.com/, May 2015.

[33] CRAWDAD: A Community Resource for Archiving Wireless Data At Dartmouth. http://crawdad.org/, Jan. 2015.

[34] Facebook: IPv6 is here and You're Hurting your Users. https://www.youtube.com/watch?v=_7rcAIbvzVY, Sept. 2015.

[35] M-Lab. http://www.measurementlab.net/, Mar. 2015.

[36] Navigation Timing. http://w3c.github.io/navigation-timing/, Aug. 2015.

[37] Navigation Timing API. http://caniuse.com/#feat=nav-timing, Jun. 2015.

[38] NSF Workshop on Achieving Ultra-Low Latencies in Wireless Networks. http://inlab.lab.asu.edu/nsf/files/WorkshopReport.pdf, Mar. 2015.

[39] Ookla. http://www.ookla.com/, 2015.

[40] Optimizely. http://www.optimizely.com/, May 2015.

[41] Real User Monitoring. https://www.akamai.com/us/en/resources/real-user-monitoring.jsp, Aug. 2015.

[42] Splitforce. http://splitforce.com/, May 2015.

[43] Unbiased Wireless Network Information. http://www.sensorly.com, Aug. 2015.

[44] Alexa: Your Complete Web Analytics Toolkit. http://www.alexa.com/, Jan. 2016.

[45] Amazon EC2 - Virtual Server Hosting. https://aws.amazon.com/ec2/, Feb. 2016.

[46] Free Basics by Facebook. https://info.internet.org/en/story/free-basics-from-internet-org/, Feb. 2016.

[47] Google Peering and Content Delivery. https://peering.google.com/, Feb. 2016.

[48] Project Loon. https://www.google.com/loon/, Feb. 2016.

[49] Johanna. OpenSignal Blog. http://opensignal.com/blog/2015/01/09/our-academic-partners/, Jan. 2015.

[50] C. Aaron. Net Neutrality Is Dead. Here's How to Get It Back. http://www.savetheinternet.com/blog/2014/01/14/net-neutrality-dead-heres-how-get-it-back, Jan 2014.

[51] V. Agababov, M. Buettner, V. Chudnovsky, M. Cogan, B. Greenstein, S. McDaniel, M. Piatek, C. Scott, M. Welsh, and B. Yin. Flywheel: Google's data compression proxy for the mobile web. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2015)*, 2015.

[52] S. Agarwal and J. R. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *ACM SIGCOMM*, August 2009.

[53] Akamai. Free Mobile Web Performance Measurement Tool. `http://mobitest.akamai.com/m/index.cgi`, 2012.

[54] Akamai. Test Your Website Performance On A Mobile Device. `http://www.akamai.com/html/awe/login.html?campaign_id=F-MC-16282&curl=/html/awe_auth/mobitest.html`, 2012.

[55] Akamai. Our Customers. `https://www.akamai.com/us/en/our-customers.jsp`, Jun. 2015.

[56] Amazon. Aws device farm. `https://aws.amazon.com/device-farm/`, Aug. 2015.

[57] M. Ammori. The next big battle in Internet policy. `http://www.slate.com/articles/technology/future_tense/2012/10/network_neutrality_the_fcc_and_the_internet_of_things_.html`, Oct. 2012.

[58] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Experience with an evolving overlay network testbed. *SIGCOMM CCR*, 33(3):13–19, July 2003.

[59] N. Anderson. Huge ISPs want per-GB payments from Netflix, YouTube. `http://arstechnica.com/tech-policy/2011/01/huge-isps-want-per-gb-payments-from-netflix-youtube/`, Jan 2011.

[60] APNIC 34. IPv6 at Verizon Wireless. `https://www.hpc.mil/images/hpcdocs/ipv6/vzw_apnic_13462152832-2.pdf`, Aug. 2012.

[61] Aqualab. Application Time (AppT). `https://play.google.com/store/apps/details?id=edu.northwestern.aqualab.behavior.research`, Apr 2014.

[62] Aqualab. Namehelp. `https://play.google.com/store/apps/details?id=edu.northwestern.aqualab.namehelp&hl=en`, Apr. 2014.

[63] J. Arkko and F. Baker. Guidelines for Using IPv6 Transition Mechanisms during IPv6 Deployment. `https://tools.ietf.org/html/rfc6180`, May 2011.

[64] AT&T Public Policy Blog. It's World IPv6 Day. http://www.attpublicpolicy.com/administration/it's-world-ipv6-day/, Jun. 2011.

[65] A. Aucinas, N. Vallina-Rodriguez, Y. Grunenberger, V. Erramilli, K. Papagiannaki, J. Crowcroft, and D. Wetherall. Staying online while mobile: The hidden costs. In *ACM CoNEXT*, CoNEXT '13, pages 315–320, New York, NY, USA, Dec. 2013. ACM.

[66] M. Austin and M. Wish. The official story on AT&T Mark the Spot. http://www.research.att.com/articles/featured_stories/2010_09/201009_MTS.html, Oct. 2010.

[67] M. Bagnulo, P. Matthews, and I. van Beijnum.

[68] V. Bajpai and J. Schonwalder. A survey on internet performance measurement platforms and related standardization efforts. *IEEE Communications Surveys Tutorials*, PP(99):1–1, Apr. 2015.

[69] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian. Where's That Phone?: Geolocating IP Addresses on 3G Networks. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, 2009.

[70] R. K. Balan, A. Misra, and Y. Lee. Livelabs: Building an in-situ real-time mobile experimentation testbed. In *Workshop on Mobile Computing Systems and Applications (HotMobile)*, Feb. 2014.

[71] R. Baldawa et al. PhoneLab: A large-scale participatory smartphone testbed. In *USENIX NSDI poster session*, Apr. 2012.

[72] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li. IPv6 Addressing of IPv4/IPv6 Translators. https://tools.ietf.org/html/rfc6052, Oct. 2010.

[73] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. Known Content Network (CN) Request-Routing Mechanisms. https://tools.ietf.org/html/rfc3568, Jul. 2003.

[74] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: Realistic and controlled network experimentation. In *ACM SIGCOMM*, Aug. 2006.

[75] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in Unreal Tournament 2003®. In *ACM workshop on network and system support for games*, August 2004.

[76] K. Bell. FCC Launches iOS 'Speed Test' App. http://mashable.com/2014/02/25/fcc-speed-test-app-ios/, Feb 2014.

[77] M. Belshe. More Bandwidth Does not Matter (much). https://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2, Apr. 2010.

[78] M. Belshe, R. Peon, and E. M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). https://tools.ietf.org/html/rfc6146, May. 2015.

[79] Ben Parker. IPv6 Transition for VzW. https://sites.google.com/site/ipv6implementors/2010/agenda/14_Parker_VerizonWireless.pdf, Jun. 2010.

[80] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. https://tools.ietf.org/html/rfc3135, Jun. 2001.

[81] A. Botta and A. Pescape. Monitoring and measuring wireless network performance in the presence of middleboxes. In *Conference on Wireless On-Demand Network Systems and Services*, Jan. 2012.

[82] M. Boucadair and E. Burgey. A64: DNS Resource Record for IPv4-mapped IPv6 Address. https://tools.ietf.org/html/draft-boucadair-behave-dns-a64-01, Oct. 2010.

[83] J. Brodkin. FCC votes for net neutrality, a ban on paid fast lanes, and Title II. http://arstechnica.com/business/2015/02/fcc-votes-for-net-neutrality-a-ban-on-paid-fast-lanes-and-title-ii/, Feb. 2015.

[84] S. Buckley. Cogent and Orange France fight over interconnection issues. http://www.fiercetelecom.com/story/cogent-and-orange-france-fight-over-interconnection-issues/2011-08-31, Aug 2011.

[85] S. Buckley. France Telecom and Google entangled in peering fight. http://www.fiercetelecom.com/story/france-telecom-and-google-entangled-peering-fight/2013-01-07, Jan 2013.

[86] R. L. Burt. Why are images not loading? https://www.facebook.com/help/community/question/?id=10100214862890089&ref=notif&notif_t=answers_answered, Jun. 2013.

[87] J. Butler, W. Lee, B. McQuade, and K. Mixter. A Proposal for Shared Dictionary Compression over HTTP. http://lists.w3.org/Archives/Public/ietf-http-wg/2008JulSep/att-0441/Shared_Dictionary_Compression_over_HTTP.pdf, Sept. 2008.

[88] J. Butler, W. Lee, B. McQuade, and K. Mixter. A Proposal for Shared Dictionary Compression over HTTP. http://lists.w3.org/Archives/Public/ietf-http-wg/2008JulSep/att-0441/Shared_Dictionary_Compression_over_HTTP.pdf, Sept. 2008.

[89] I. by Facebook. Connecting the world. https://info.internet.org/en/, Feb. 2016.

[90] C. Byrne. 464XLAT: Breaking Free of IPv4. https://conference.apnic.net/data/37/464xlat-apricot-2014_1393236641.pdf, Feb. 2014.

[91] CacheFly. Technology and Infrastructure. http://www.cachefly.com/cachefly-cdn/technology/, Mar. 2015.

[92] J. Cainey, B. Gill, S. Johnston, J. Robinson, and S. Westwood. Modelling download throughput of LTE networks. In *Local Computer Networks Workshops (LCN Workshops), 2014 IEEE Conference on*, Oct. 2014.

[93] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. In *ACM SIGCSE Bulletin*, Mar 2009.

[94] Carrier IQ. Vodafone Portugal Pioneers Innovative Mobile Broadband Experience Management Architecture Using Carrier IQ Technology . http://carrieriq.com/wp-content/uploads/2014/08/PR.CarrierIQandVodafonePortugal.20090730.pdf, 2009 July.

[95] Cedexis. Openmix. http://www.cedexis.com/openmix/, Aug. 2015.

[96] Cedexis. Radar. http://www.cedexis.com/radar/, Aug. 2015.

[97] F. Chen, R. K. Sitaraman, and M. Torres. End-User Mapping: Next Generation Request Routing for Content Delivery.

[98] F. Chen, R. K. Sitaraman, and M. Torres. End-User Mapping: Next Generation Request Routing for Content Delivery. In *ACM Sigcomm CCR*, Apr. 2015.

[99] K.-T. Chen, P. Huang, and C.-L. Lei. Effect of network quality on player departure behavior in online games. *Parallel Distributed Systems*, 20:593–606, May 2009.

[100] D. R. Choffnes, F. E. Bustamante, and Z. Ge. Crowdsourcing service-level network event monitoring. In *ACM SIGCOMM*, Aug. 2010.

[101] D. R. Choffnes, M. A. Sanchez, and F. E. Bustamante. Network Positioning from the Edge: An empirical study of the effectiveness of network positioning in P2P systems. In *IEEE INFOCOM*, Mar. 2010.

[102] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Computer Communications Review*, 33(3):3–12, July 2003.

[103] Cisco Systems, Inc. Cisco Data Meter. `http://ciscovni.com/data-meter/index.html`, May 2013.

[104] K. Claffy. The 5th Workshop on Active Internet Measurements (AIMS-5) Report. In *ACM SIGCOMM Computer Communication Review, Volume 43, Number 3*, July 2013.

[105] k. claffy, D. D. Clark, and M. P. Wittie. The 6th Workshop on Active Internet Measurements (AIMS-6) Report. *Sigcomm CCR*, October 2014.

[106] J. Clover. FCC Launches 'FCC Speed Test' iPhone App to Measure Mobile Broadband Performance. `http://www.macrumors.com/2014/02/25/fcc-speed-test/`, Feb 2014.

[107] J. Codorniou. Whats next for social mobile games? `http://techcrunch.com/2012/12/22/whats-next-for-social-mobile-games/`, Dec. 2012.

[108] B. Cohen. Incentives build robustness in BitTorrent. In *International workshop on Peer-To-Peer Systems (IPTPS)*, Feb. 2003.

[109] L. Colitti. Broken IPv6 Clients. `https://sites.google.com/site/ipv6implementors/2010/agenda/BrokenIPv6clients.pdf`, Jun. 2010.

[110] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice. Evaluating IPv6 Adoption in the Internet. In *Passive and Active Measurement*, Apr. 2010.

[111] Conviva. Internet tv: Bringing control to chaos. `http://cdn2.hubspot.net/hub/468871/file-2377674682-pdf/drop/Internet_TV-Bringing_Control_to_Chaos-Conviva.pdf`, Aug. 2015.

[112] N. Craig. A simple and rough comparison of Akamai and Cloudfront CDN's. `https://thedotproduct.org/a-simple-and-rough-comparison-of-akamai-and-cloudfront-cdns/`, Mar. 2014.

[113] J. Curran. ARIN IPv4 Free Pool Reaches Zero. https://www.arin.net/announcements/2015/20150924.html, Sept. 2015.

[114] M. Cceres, F. J. Moreno, and I. Grigorik. Network Information API. https://w3c.github.io/netinfo/, Dec. 2015.

[115] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. ACM SIGCOMM, Sept. 2004.

[116] H. de Saxc, I. Oprescu, and Y. Chen. Is HTTP/2 really faster than HTTP/1.1? In *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, pages 293–299, April 2015.

[117] G. Developers. Pre-Resolve DNS. https://developers.google.com/speed/pagespeed/service/PreResolveDns, Apr. 2015.

[118] G. Developers. Pre-Resolve DNS. https://developers.google.com/speed/pagespeed/service/PreResolveDns, Apr. 2015.

[119] A. Dhamdhere, M. Luckie, B. Huffaker, k. claffy, A. Elmokashfi, and E. Aben. Measuring the Deployment of IPv6: Topology, Routing and Performance. In *ACM IMC*, Nov 2012.

[120] A. Y. Ding, J. Korhonen, T. Savolainen, Y. Liu, M. Kojo, S. Tarkoma, and H. Schulzrinne. Reflections on Middlebox Detection Mechanisms in IPv6 Transition. In *IAB Workshop on Stack Evolution in a Middlebox Internet (SEMI)*, Jan. 2015.

[121] Y. Ding, T. Savolainen, J. Korhonen, and M. Kojo. Speeding up IPv6 transition: Discovering NAT64 and Learning Prefix for IPv6 Address Synthesis. In *IEEE International Conference on Communications (ICC)*, June 2012.

[122] M. Dischinger, A. Haeberlen, I. Beschastnikh, K. P. Gummadi, and S. Saroiu. SatelliteLab: Adding heterogeneity to planetary-scale network testbeds. In *ACM SIGCOMM*, Aug. 2008.

[123] M. Dischinger, M. Marcon, S. Guha, K. P. Gummadi, R. Mahajan, and S. Saroiu. Glasnost: enabling end users to detect traffic differentiation. In *USENIX NSDI*, Apr. 2010.

[124] C. Donley. IPv6-IPv4 Performance Comparison: The Effect of NAT. https://www.nanog.org/sites/default/files/tuesday.general.donley.ipv6performance.17.pdf, 2013.

[125] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An Argument for Increasing TCP's Initial Congestion Window. *SIGCOMM CCR*, 40(3), Jun. 2010.

[126] A. Dyke. What is a HAR File and what do I use it for? http://www.speedawarenessmonth.com/what-is-a-har-file-and-what-do-i-use-it-for/, Aug 2012.

[127] N. Ehsan, M. Liu, and R. J. Ragland. Evaluation of Performance Enhancing Proxies in Internet over Satellite, Jan. 2003.

[128] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a SPDY'Ier Mobile Web? In *ACM CoNEXT*, Dec. 2013.

[129] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan. Towards a SPDY'Ier Mobile Web? In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, 2013.

[130] F. Brockners, S. Gundavelli, S. Speicher and D. Ward. Gateway-Initiated Dual-Stack Lite Deployment. https://tools.ietf.org/html/rfc6674, Jul. 2012.

[131] F. Kaup, F. Jomrich, and D. Hausheer. Demonstration of NetworkCoverage – A Mobile Network Performance Measurement App. *IEEE NetSys*, March 2015.

[132] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio. Network sensing through smartphone-based crowdsourcing. In *Embedded Networked Sensor Systems (SenSys)*, Nov. 2013.

[133] M. Fahey. Why Gamers Should Care About Net Neutrality. http://kotaku.com/5512448/why-gamers-should-care-about-net-neutrality, Apr 2010.

[134] V. Farkas, B. Hder, and S. Novczki. A Split Connection TCP Proxy in LTE Networks. In *Information and Communication Technologies*, Aug. 2012.

[135] FCC. Measuring Broadband America. http://www.fcc.gov/measuring-broadband-america/mobile, Nov 2013.

[136] FCC. Measuring Mobile Broadband Methodology - Technical Summary. http://www.fcc.gov/measuring-broadband-america/mobile/technical-summary, Nov. 2013.

[137] FCC. FCC Speed Test App Tip Sheet. https://www.fcc.gov/guides/mobile-speed-test-tip-sheet, 2014.

[138] FCCAPPs. FCC Speed Test App. https://play.google.com/store/apps/details?id=com.samknows.fcc&hl=en, Dec 2013.

[139] N. Feamster. My Speed Test Mobile Performance Measurement Tool Released. http://noise-lab.net/2012/06/02/my-speed-test-mobile-performance-measurement-tool-released/, June 2012.

[140] FierceWireless. 3G/4G wireless network latency: Comparing Verizon, AT&T, Sprint and T-Mobile in February 2014. http://www.fiercewireless.com/special-reports/3g4g-wireless-network-latency-comparing-verizon-att-sprint-and-t-mobile-feb, Mar. 2014.

[141] B. Forrest. Bing and Google Agree: Slow Pages Lose Users. http://radar.oreilly.com/2009/06/bing-and-google-agree-slow-pag.html, Jun. 2009.

[142] Fortinet. IPv6: Network Security and the Next Generation of IP Communication. http://www.fortinet.com/sites/default/files/solutionbrief/SG-IPV6.pdf, 2015.

[143] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber. Pushing CDN-ISP Collaboration to the Limit. *SIGCOMM Comput. Commun. Rev.*, 43(3), July 2013.

[144] J. Gao, A. Sivaraman, N. Agarwal, H. Li, and L. Peh. DIPLOMA: Consistent and coherent shared memory over mobile phones. In *International Conference on Computer Design (ICCD)*, Sept. 2012.

[145] D. Geerts, I. Vaishnavi, R. Mekuria, O. van Deventer, and P. Cesar. Are we in sync?: synchronization requirements for watching online video together. In *SIGCHI Conference on Human Factors in Computing Systems*, May 2011.

[146] A. Gember, A. Akella, J. Pang, A. Varshavsky, and R. Caceres. Obtaining in-context measurements of cellular network performance. In *ACM IMC*, Nov. 2012.

[147] U. Goel, A. Miyyapuram, M. P. Wittie, and Q. Yang. MITATE: Mobile Internet Testbed for Application Traffic Experimentation. In *Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous)*, Dec. 2013.

[148] U. Goel, M. Steiner, M. P. Wittie, M. Flack, and S. Ludin. Detecting Cellular Middle-boxes using Passive Measurement Techniques. In *Passive and Active Measurements Conference (PAM)*, 2016.

[149] U. Goel, M. Steiner, M. P. Wittie, E. Nygre, R. Gao, M. Flack, and S. Ludin. Measuring Cellular IPv6 Networks for Web Performance. In *Submission*, Mar. 2016.

[150] U. Goel, M. Wittie, K. Claffy, and A. Le. Survey of End-to-End Mobile Network Measurement Testbeds, Tools, and Services. *Communications Surveys Tutorials, IEEE*, 18(1):105–123, Firstquarter 2016.

[151] U. Goel, M. P. Wittie, and M. Steiner. Faster Web through Client-assisted CDN Server Selection. In *IEEE International Conference on Computer Communications and Networks (ICCCN)*, Aug. 2015.

[152] C. Gomez, M. Catalan, D. Viamonte, J. Paradells, and A. Calveras. Web browsing optimization over 2.5G and 3G: end-to-end mechanisms vs. usage of performance enhancing proxies. *Wireless Communications and Mobile Computing*, Feb. 2008.

[153] Google. Analytics for mobile apps. `http://www.google.com/analytics/mobile/`, Aug. 2015.

[154] Google Developers. Google Maps Developer Documentation. `https://developers.google.com/maps/documentation/`, Mar. 2015.

[155] V. Gopalakrishnan, L. E. Li, G. Ricart, J. Breen, J. Martin, Y. Xin, C. Elliott, A. Banerjee, J. Cho, M. Munakami, A. Chowdhary, N. Alsrehin, I. Alsmadi, D. Grunwald, E. Eide, R. Ricci, and M. Wittie. SDN and NFV Report-Out. `https://phantomnet.org/workshop/sdnnfv.pdf`, Feb. 2015.

[156] E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio. Sensing the Internet through crowdsourcing. In *Proceedings of the Second IEEE PerCom Workshop on the Impact of Human Mobility in Pervasive Systems and Applications (PerMoby)*, May 2013.

[157] I. Grigorik. Latency: The New Web Performance Bottleneck. `https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/`, Jul. 2012.

[158] I. Grigorik. *High Performance Browser Networking*. O'Reilly, 2013.

[159] B. I. T. A. Group. IPv6 AAAA DNS Whitelisting. `http://www.bitag.org/documents/BITAG_TWG_Report-DNS_Whitelisting.pdf`, Sept. 2011.

[160] C. Grundemann. IPv6 Security Myth IPv6 Networks are Too Big to Scan. `http://www.internetsociety.org/deploy360/blog/2015/02/ipv6-security-myth-4-ipv6-networks-are-too-big-to-scan/`, Feb. 2015.

[161] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *SIGCOMM Workshop on Internet Measurement*, Nov. 2002.

[162] Guy Podjarny. Open-Sourcing Mobitest. `https://blogs.akamai.com/2012/03/open-sourcing-mobitest.html`, Mar 2012.

[163] Z. Honig. FCC Speed Test app for iOS lets the government track your iPhone's network performance. http://www.engadget.com/2014/02/25/fcc-speed-test-app-ios/, Feb 2014.

[164] T. Hopkins. What Are The Benefits Of Using a CDN? http://www.rackspace.com/knowledge_center/frequently-asked-question/what-are-the-benefits-of-using-a-cdn, Sept. 2012.

[165] E. Howard, C. Cooper, M. P. Wittie, S. Swinford, and Q. Yang. Cascading impact of lag on user experience in multiplayer games. In *ACM NetGames*, Dec. 2014.

[166] J. Huang, C. Chen, Y. Pei, Z. Wang, Z. Qian, F. Qian, B. Tiwana, Q. Xu, Z. M. Mao, M. Zhang, and P. Bahlc. MobiPerf: Mobile Network Measurement System (Technical report). Technical report, University of Michigan and Microsoft Research, 2011.

[167] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *ACM MobiSys*, June 2010.

[168] IBM. CPLEX optimizer. www.ibm.com/software/commerce/optimization/cplex-optimizer/, 2013.

[169] Internet2. Network Diagnostic Tool (NDT). http://software.internet2.edu/ndt/, 2013.

[170] M. Ivanovich, P. Bickerdike, and J. Li. On TCP performance enhancing proxies in a wireless environment. *IEEE Communications Magazine*, Sept. 2008.

[171] J. Iyengar and I. Swett. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00, Jun. 2015.

[172] A. jan Su, D. Choffnes, F. E. Bustamante, and A. Kuzmanovic. Relative Network Positioning via CDN Redirections. In *ICDCS*, Jun. 2008.

[173] J.D. Power. Overall wireless network problem rates differ considerably based on type of service. http://www.jdpower.com/press-releases/2013-us-wireless-network-quality-performance-study-volume-2, Aug. 2013.

[174] Jeff Barr. Multi-Region Latency Based Routing now Available for AWS. https://aws.amazon.com/blogs/aws/latency-based-multi-region-routing-now-available-for-aws/, Mar. 2012.

[175] W. Johnston. Measuring Broadband America. http://www.caida.org/workshops/aims/1403/slides/aims1403_wjohnston.pdf, Mar. 2013.

[176] J. Kangasharju, K. W. Ross, and J. W. Roberts. Performance Evaluation of Redirection Schemes in Content Distribution Networks. In *Computer Communications*, 2000.

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

[178] T. Karr. Verizon's Plan to Break the Internet. http://www.savetheinternet.com/blog/2013/09/18/verizons-plan-break-internet, Sept 2013.

[179] J. Kastrenakes. FCC releases Android speed test app to gather data on cell carrier performance. http://www.theverge.com/2013/11/14/5105090/fcc-launches-android-mobile-speed-test-app, Nov 2011.

[180] R. Kisteleki. Measuring IPv6 usage at web clients and DNS resolvers. https://sites.google.com/site/ipv6implementors/2010/agenda/07_robert-kisteleki-measure-v6.pdf, Jun. 2010.

[181] E. Kline. IPv6 Whitelist Operations. https://sites.google.com/site/ipv6implementors/2010/agenda/IPv6_Whitelist_Operations.pdf, Jun. 2010.

[182] J. Korhonen and E. T. Savolainen. EDNS0 Option for Indicating AAAA Record Synthesis and Format. https://tools.ietf.org/html/draft-korhonen-edns0-synthesis-flag-02, Feb. 2011.

[183] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM SIGCOMM Conference on Internet Measurement*, Nov. 2010.

[184] N. S. N. S. Labs. Understanding smartphone behavior in the network. http://www.nokiasiemensnetworks.com/sites/default/files/document/Smart_Lab_WhitePaper_27012011_low-res.pdf, 2011.

[185] P. Lakhera. Your App and Next Generation Networks. https://developer.apple.com/videos/wwdc/2015/?id=719, June. 2015.

[186] Y.-N. Law, M.-C. Lai, W. L. Tan, and W. C. Lau. Empirical Performance of IPv6 vs. IPv4 under a Dual-Stack Environment. In *IEEE International Conference on Communications (ICC)*, May 2008.

[187] Librato. One Platform. Unlimited Metrics. Monitoring Zen. http://metrics.librato.com/, Sept. 2014.

[188] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. In *ACM SIGCOMM*, Aug. 2012.

[189] LiveLabs. Participation. `http://livelabs.smu.edu.sg/participant/`, Mar. 2013.

[190] Lookout Labs. CarrierIQ Scanner & Protection. `https://play.google.com/store/apps/details?id=com.lookout.carrieriqdetector&hl=en`, May 2013.

[191] X. Lu, W. Cao, X. Huang, F. Huang, L. He, W. Yang, S. Wang, X. Zhang, and H. Chen. A real implementation of DPI in 3G network. In *Global Telecommunications Conference (GLOBECOM)*, Dec. 2010.

[192] A. Lynn. Cable Companies' Big Internet Swindle. `http://www.freepress.net/blog/2009/11/24`, Nov 2009.

[193] M. Belshe and R. Peon. SPDY Protocol. `http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00`, Feb. 2012.

[194] J. Manweiler, S. Agarwal, M. Zhang, R. Roy Choudhury, and P. Bahl. Switchboard: A matchmaking system for multiplayer mobile games. In *ACM MobiSys*, June 2011.

[195] P. Marupaka. The future looks bright for augmented reality. `http://www.siggraph.org/discover/news/future-looks-bright-augmented-reality`, May 2014.

[196] Matthew Prince. Mirage 2.0: Solving the Mobile Browsing Speed Challenge. `https://blog.cloudflare.com/mirage2-solving-mobile-speed/`, June 2013.

[197] MaxMind. GeoIP2: Industry Leading IP Intelligence. `https://www.maxmind.com/en/geoip2-services-and-databases`, 2012.

[198] K. V. d. Merwe. PhantomNet: An end-to-end mobile network testbed. `http://www.caida.org/workshops/aims/1403/slides/aims1403_jvandermerwe.pdf`, Mar. 2014.

[199] S. Micka, U. Goel, H. Ye, M. P. Wittie, and B. Mumey. Internet Latency Estimation Using CDN Replicas. In *IEEE ICCCN*, Aug. 2015.

[200] D. L. Mills. Network Time Protocol (version 2) specification and implementation. Network Working Group Request for Comments: 1119, Sept. 1989.

[201] MITATE. MITATE : Mobile Internet Testbed for Application Traffic Experimentation (User Manual). http://mitate.cs.montana.edu/sample/MITATE_Documentation_v1.0.pdf, Nov 2013.

[202] MLAB. WindRider. http://www.measurementlab.net/tools/windrider, 2009.

[203] MLAB. NDT (Mobile Client). http://www.measurementlab.net/tools/ndt-mobile, 2013.

[204] MobiPerf. Welcome to MobiPerf. http://www.mobiperf.com/home, Feb. 2012.

[205] MobiPerf. Data Collection and Privacy Policy. http://www.mobiperf.com/privacy, 2014.

[206] MobiPerf. Welcome to MobiPerf. http://www.mobiperf.com/, 2014.

[207] P. Mockapetris. Domain Names - Implementation and Specification. Nov. 1987.

[208] A. M. Moreiras. Performance comparison between IPv4 and IPv6 on the Internet. http://www.ceptro.br/pub/CEPTRO/ArquivoNoticiaIPv6Performance/ipv6xipv4performance-napla.pdf, May 2011.

[209] S. Muckaden. MySpeedTest: Active and Passive Measurements of Cellular Data Network Performance. http://www.caida.org/workshops/isma/1302/slides/aims1302_smuckaden.pdf, Feb. 2013.

[210] S. Muckaden. *MySpeedTest: Active and Passive Measurements of Cellular Data Networks*. PhD thesis, Georgia Institute of Technology, 2013.

[211] M. K. Mukerjee, J. Hong, J. Jiang, D. Naylor, D. Han, S. Seshan, and H. Zhang. Enabling Near Real-time Central Control for Live Video Delivery in CDNs. In *ACM SIGCOMM*, Aug. 2014.

[212] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen. PhoneLab: A Large Programmable Smartphone Testbed. In *Workshop on Sensing and Big Data Mining*, Nov. 2013.

[213] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste. Investigating Transparent Web Proxies in Cellular Networks. In *ACM SIGCOMM*, Aug. 2015.

[214] M. Necker, M. Scharf, and A. Weber. Performance of Different Proxy Concepts in UMTS Networks. In *Wireless Systems and Mobility in Next Generation Internet*, Jun. 2004.

[215] Netflix. Netflix Open Connect Content Delivery Network. https://www.netflix.com/openconnect.

[216] NetRadar. What's my mobile operator's coverage? http://www.netradar.org/en, Mar. 2015.

[217] T. S. E. Ng and H. Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *IEEE INFOCOM*, Apr. 2001.

[218] I. Nir. Latency in Mobile Networks The Missing Link. http://calendar.perfplanet.com/2012/latency-in-mobile-networks-the-missing-link/, Dec. 2012.

[219] nPerf.com. Whats nPerf? How does it work? http://www.nperf.com/en/, Nov. 2014.

[220] NUStudents. NU Signals v2. https://play.google.com/store/apps/details?id=edu.northwestern.nux, May 2014.

[221] E. Nygren. Implementing IPv6 on a Global Scale: Experiences at Akamai, Dec. 2011.

[222] E. Nygren. Three years since World IPv6 Launch: strong IPv6 growth continues. https://blogs.akamai.com/2015/06/three-years-since-world-ipv6-launch-strong-ipv6-growth-continues.html, Jun. 2015.

[223] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network: A Platform for High-performance Internet Applications. *ACM SIGOPS Operating Systems Review*, 44(3), Aug. 2010.

[224] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network: A Platform for High-Performance Internet Applications. In *ACM SIGOPS Operating Systems Review, Vol. 44, No.3*, July 2010.

[225] Ookla. Ookla SpeedTest Mobile Apps. http://www.speedtest.net/mobile/, 2014.

[226] Ookla. Host a Speedtest Server. http://www.ookla.com/host, 2015.

[227] Ookla. Speedtest.net. https://play.google.com/store/apps/details?id=org.zwanoo.android.speedtest, Mar. 2015.

[228] Ookla SpeedTest. How do I correct my location? https://support.speedtest.net/hc/en-us/articles/203845660-How-do-I-correct-my-location-, Oct. 2012.

[229] Ookla SpeedTest. How does the Begin Test button select a server? https://support.speedtest.net/hc/en-us/articles/203845410-How-does-the-Begin-Test-button-select-a-server-, Jan. 2012.

[230] Ookla SpeedTest. How does the test itself work? How is the result calculated? https://support.speedtest.net/hc/en-us/articles/203845400-How-does-the-test-itself-work-How-is-the-result-calculated-, Jan. 2012.

[231] Ookla SpeedTest. Mobile Test Server Selection. https://support.speedtest.net/hc/en-us/articles/203845480-Mobile-Test-Server-Selection, Oct. 2012.

[232] Ookla SpeedTest. Ookla SpeedTest Mini. http://www.speedtest.net/mini.php, 2014.

[233] OpenSignal Developers. NetworkStats API. http://developer.opensignal.com/networkrank/, 2014.

[234] OpenSignal Developers. Tower Info API. http://developer.opensignal.com/towerinfo/, 2014.

[235] OpenSignal, Inc. OpenSignal Blog. http://opensignal.com/blog/2012/11/29/new-permissions-in-version-1-99-and-how-to-check-whether-an-app-is-malici Nov. 2012.

[236] OpenSignal, Inc. How phone batteries measure the weather. http://opensignal.com/reports/battery-temperature-weather/, Aug. 2013.

[237] OpenSignal, Inc. OpenSignal - Signal Finder and 3G/4G/Wifi Coverage Maps. https://itunes.apple.com/app/opensignal/id598298030, Mar. 2013.

[238] OpenSignal, Inc. The State of LTE. http://opensignal.com/reports/state-of-lte/, Feb. 2013.

[239] OpenSignal, Inc. OpenSignal. http://opensignal.com/, 2014.

[240] OpenSignal.com. OpenSignal WiFi map, speedtest. https://play.google.com/store/apps/details?id=com.staircase3.opensignal&hl=en, Mar. 2015.

[241] Opera Software ASA. Opera Turbo. http://www.opera.com/turbo, Mar. 2015.

[242] C. Osborne. The state of LTE 4G networks worldwide in 2014 and the poor performance of the US. http://www.zdnet.com/the-state-of-lte-4g-networks-worldwide-in-2014-and-the-poor-performance-of-t Feb. 2014.

[243] V. N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye. A Study of End-to-end Web Access Failures. In *ACM CoNEXT*, Dec. 2006.

[244] K. Park, V. S. Pai, L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *USENIX OSDI*, Dec. 2004.

[245] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large scale Internet measurement. *IEEE Communications*, 36(8):48–54, Aug. 1998.

[246] M. Peckham. Carrier IQ Wiretap Debacle: Much Ado About Something? http://techland.time.com/2011/12/01/carrieriq-wiretap-debacle-much-ado-about-something/, Dec. 2011.

[247] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush. From Paris to Tokyo: On the Suitability of Ping to Measure Latency. In *ACM IMC*, Oct. 2013.

[248] R. Peon and H. Ruellan. HPACK: Header Compression for HTTP/2. https://http2.github.io/http2-spec/compression.html, May 2015.

[249] PhoneLab. Overview. http://participate.phone-lab.org/info/, 2013.

[250] PhoneLab. PhoneLab A Programmable Smartphone Testbed. http://www.phone-lab.org/, 2013.

[251] PhoneLab. PhoneLab Experimenter Agreement. http://experiment.phone-lab.org/terms/, 2013.

[252] M. Piatek. Measurement @ Google. http://www.caida.org/workshops/aims/1403/slides/aims1403_mpiatek.pdf, Mar. 2014.

[253] PingTest.net. Measuring Network Quality. http://www.pingtest.net/learn.php, 2014.

[254] D. Plonka and P. Barford. Assessing performance of Internet services on IPv6. In *IEEE Symposium on Computers and Communications (ISCC)*, July 2013.

[255] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. TCP Fast Open. In *ACM CoNEXT*, Dec. 2011.

[256] RadioOpt GmbH. Download Traffic Monitor. http://www.trafficmonitor.mobi/en/download/, 2014.

[257] RadioOpt GmbH. Traffic Monitor & 3G/4G Speed. http://www.trafficmonitor.mobi/en/download/, 2014.

[258] RadioOpt GmbH. RadioOpt. https://www.radioopt.com/, Mar. 2015.

[259] Readwrite. Net Neutrality: What Happens Now That Verizon Has Vanquished The FCC. http://readwrite.com/2014/01/15/net-neutrality-fcc-verizon-open-internet-order, Jan 2014.

[260] B. Reed. LTE devices must support IPv6, says Verizon. http://www.networkworld.com/article/2257267/lan-wan/lte-devices-must-support-ipv6--says-verizon.html, Jun. 2009.

[261] V. W. Reporter. THE VALUE OF A MILLISECOND: FINDING THE OPTIMAL SPEED OF A TRADING INFRASTRUCTURE . http://www.tabbgroup.com/PublicationDetail.aspx?PublicationID=346.

[262] P. Richter, M. Allman, R. Bush, and V. Paxson. A Primer on IPv4 Scarcity. In *Sigcomm CCR*, Apr. 2015.

[263] V. Rideout and V. S. Katz. Opportunity for all? Technology and learning in lower-income families. http://www.joanganzcooneycenter.org/wp-content/uploads/2016/01/jgcc_opportunityforall.pdf, Jan. 2016.

[264] I. Rimington. Leave your wallet at home and pay with your profile picture. https://www.paypal.co.uk/Blog/Leave-your-wallet-at-home-and-pay-with-your-profile-picture/, Aug. 2013.

[265] K. Rogers. What's next after WhatsApp: a guide to the future of messaging apps. http://www.theguardian.com/technology/2014/feb/21/whatsapp-facebook-messaging-apps-viber-kik, Feb. 2014.

[266] RootMetrics. The RootMetrics testing methodology. http://www.rootmetrics.com/us/methodology, Mar. 2015.

[267] S. Rosen, H. Luo, Q. A. Chen, Z. M. Mao, J. Hui, A. Drake, and K. La. Discovering Fine-grained RRC State Dynamics and Performance Impacts in Cellular Networks. In *ACM Mobicom*, Sept. 2014.

[268] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). https://tools.ietf.org/html/rfc5389, Oct. 2008.

[269] J. Rula. ALICE - Mobile Experiment Engine. http://aqualab.cs.northwestern.edu/projects/alice, Aug. 2014.

[270] J. Rula. ALICE - Technical Description. http://aqualab.cs.northwestern.edu/262-details-alice, Aug. 2014.

[271] J. P. Rula and F. E. Bustamante. Behind the Curtain - Cellular DNS and Content Replica Selection. Nov. 2014.

[272] J. P. Rula and F. E. Bustamante. Namehelp Mobile. http://aqualab.cs.northwestern.edu/projects/237-namehelp-mobile, Aug. 2014.

[273] P. Saab. Facebook V6 World Congress 2015. https://www.youtube.com/watch?v=An7s25FSK0U, Mar. 2015.

[274] P. Saab. Facebook V6 World Congress 2015. https://www.youtube.com/watch?v=An7s25FSK0U, Mar. 2015.

[275] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing experiments to the Internet's edge. In *USENIX NSDI*, Apr. 2013.

[276] M. A. Sanchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing Experiments to the Internet's Edge. In *USENIX NSDI*, Apr. 2013.

[277] T. Savolainen and J. Korhonen. Discovery of a Network-Specific NAT64 Prefix using a Well-Known Name. https://tools.ietf.org/html/draft-savolainen-heuristic-nat64-discovery-01, Feb. 2011.

[278] Sensorly. Unbiased Wireless Network Information. From people just like you. http://sensorly.com/, 2013.

[279] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of dns-based server selection. In *IEEE Infocom*, Apr. 2001.

[280] H. Shang and C. E. Wills. Piggybacking Related Domain Names to Improve DNS Performance. *Computer Network*, 50(11), Aug. 2006.

[281] K. Shubber. Microsoft kinect used to live-translate sign language into text. http://www.wired.co.uk/news/archive/2013-07/18/sign-language-translation-kinect, July 2013.

[282] C. Siaterlis, A. Garcia, and B. Genge. On the use of emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys Tutorials*, 15(2):929–942, Feb. 2013.

[283] S. Silbert. FCC launches speed test app for Android, looks to collect mobile broadband performance data. http://www.engadget.com/2013/11/14/fcc-launches-speed-test-app-android/, Nov 2011.

[284] C. R. J. Simpson and G. F. Riley. NETI@home: A distributed approach to collecting end-to-end network performance measurements. In *ACM PAM*, Apr. 2004.

[285] R. Singel. Mobile Carriers Dream of Charging per Page. `http://www.wired.com/business/2010/12/carriers-net-neutrality-tiers/2/`, Dec 2010.

[286] N. Smith. Latency between Amazon Web Services' Regions. `http://www.nsmith.net/articles/2011-08/latency-between-amazon-web-services-regions/`, Aug. 2011.

[287] sn707. ATT LG G3 Carrier IQ Removal Guide. `http://forum.xda-developers.com/att-lg-g3/general/att-lg-g3-carrier-iq-removal-guide-t2819295`, July 2014.

[288] S. Souders. Extension Mechanisms for DNS (EDNS(0)). `http://tools.ietf.org/html/rfc6891`, Apr. 2013.

[289] SpeedSpot. SpeedSpot: Pioneering Hotel WiFi Speed Test. `http://speedspot.org/`, Mar. 2015.

[290] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public Internet measurement facility. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2003.

[291] A. Striegel, S. Liu, L. Meng, C. Poellabauer, D. Hachen, and O. Lizardo. Lessons learned from the netsense smartphone study. In *ACM Workshop on HotPlanet*, HotPlanet '13, Aug. 2013.

[292] M. Thomson. Blind Proxy Caching. `https://httpworkshop.github.io/workshop/presentations/thomson-cache.pdf`, Jul. 2015.

[293] I. Trestian, R. Potharaju, and A. Kuzmanovic. Closing the Loop: Feedback at Your Fingertips. `http://www.cs.northwestern.edu/~ict992/docs/draft.pdf`, 2009.

[294] S. Triukose, S. Ardon, A. Mahanti, and A. Seth. Geolocating ip addresses in cellular data networks. In *Passive and Active Measurement Conference (PAM)*, 2012.

[295] V-Speed. Cloud Managed Speed Test. `http://www.v-speed.eu/`, Mar. 2015.

[296] N. Vallina-Rodriguez, A. Auçinas, M. Almeida, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. RILAnalyzer: A Comprehensive 3G Monitor on Your Phone. In *ACM IMC*, IMC '13, pages 257–264, New York, NY, USA, Oct. 2013. ACM.

[297] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson. CRAWDAD dataset icsi/netalyzr-android (v. 2015-03-24). Downloaded from http://crawdad.org/icsi/netalyzr-android/20150324/middleboxes, Mar. 2015. traceset: middleboxes.

[298] N. Vallina-Rodriguez, N. Weaver, C. Kreibich, and V. Paxson. Netalyzr for Android: Challenges and opportunities. In *Workshop on Active Internet Measurements (AIMS)*, Mar 2014.

[299] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, Alessandro, Finamore, and K. Papagiannaki. Is The Web HTTP/2 Yet? In *Passive and Active Measurements Conference (PAM)*, Mar. 2016.

[300] Veloxity, Inc. 4Gmark Mobile performance test. `http://www.4gmark.com/`, Nov. 2014.

[301] J. Vijayan. ATT, Sprint confirm use of Carrier IQ software on handsets. `http://www.computerworld.com/article/2499667/application-security/at-t--sprint-confirm-use-of-carrier-iq-software-on-handsets.html`, Dec. 2011.

[302] Vodafone. Vodafone Net Perform. `https://play.google.com/store/apps/details?id=com.vodafone.netperform.full`, June 2014.

[303] Vodafone. Vodafone Net Perform. `https://itunes.apple.com/ie/app/vodafone-net-perform/id946160163?mt=8`, June 2014.

[304] Vodafone. Take control of your data and Wi-Fi usage – Vodafone NetPerform. `http://www.vodafone.co.uk/discover-vodafone/apps-and-downloads/vodafone_netperform/`, Mar. 2015.

[305] Vodafone. Vodafone Net Perform – Terms and Conditions. `https://www.vodafone.co.nz/legal/terms-conditions/netperform/`, Mar. 2015.

[306] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low Latency via Redundancy. In *ACM CoNEXT*, Dec. 2013.

[307] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Low Latency via Redundancy. In *ACM CoNEXT*, Dec 2013.

[308] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystify page load performance with wprof. In *USENIX NSDI*, Apr. 2013.

[309] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying Page Load Performance with WProf. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2013.

[310] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. How Speedy is SPDY? In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, 2014.

[311] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An Untold Story of Middleboxes in Cellular Networks. In *ACM SIGCOMM*, Aug. 2011.

[312] Y. Wang1, S. Ye, and X. Li. Understanding Current IPv6 Performance: A Case Study from CERNET. In *Symposium on Computers and Communications (ISCC)*, 2005.

[313] N. Weaver, C. Kreibich, M. Dam, and V. Paxson. Here Be Web Proxies. In *Passive and Active Measurements Conference*, Mar. 2014.

[314] Webpagetest. Test a website's performance. `http://www.webpagetest.org/`, 2008.

[315] WindRider. WindRider A Mobile Network Neutrality Monitoring System. `http://www.cs.northwestern.edu/~ict992/mobile.htm`, Oct 2009.

[316] D. Wing. Learning the IPv6 Prefix of a Network's IPv6/IPv4 Translator. `https://tools.ietf.org/html/draft-wing-behave-learn-prefix-04`, Oct. 2009.

[317] D. Wing and A. Yourtchenko. Happy Eyeballs: Success with Dual-Stack Hosts. `https://tools.ietf.org/html/rfc6555`, Apr. 2012.

[318] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *USENIX NSDI*, Apr. 2013.

[319] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao. Exploiting locality of interest in online social networks. In *ACM CoNEXT*, November 2010.

[320] M. P. Wittie, B. Stone-Gross, K. C. Almeroth, and E. M. Belding. MIST: Cellular data network measurement for mobile applications. In *Conference on Broadband Communications, Networks and Systems (BROADNETS)*, Sept. 2007.

[321] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: A Lightweight Network Location Service Without Virtual Coordinates. ACM SIGCOMM, Aug. 2005.

[322] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan. Investigating Transparent Web Proxies in Cellular Networks. In *Passive and Active Measurements Conference*, Mar. 2015.

[323] yanyan. Using Sensors In Seattle. `https://seattle.poly.edu/wiki/UsingSensors`, Apr 2012.

[324] H. Yao, A. Nikravesh, Y. Jia, D. R. Choffnes, and Z. M. Mao. Mobilyzer: A Network Measurement Library for Android Platform. In *Workshop on Active Internet Measurements (AIMS)*, Mar 2014.

[325] R. Yasinovskyy, A. Wijesinha, R. Karne, and G. Khaksari. A comparison of VoIP performance on IPv6 and IPv4 networks. In *IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, May 2009.

[326] S. Yaw, E. Howard, S. Poudel, B. Mumey, and M. P. Wittie. Collaborative group provisioning with QoE guarantees in multi-cloud deployments. In *submission*, 2014.

[327] YUI Team. Combo Handler Service Available for Yahoo-hosted JS. `http://yuiblog.com/blog/2008/07/16/combohandler/`, July 2008.

[328] B. Zevenbergen, I. Brown, J. Wright, and D. Erdos. Ethical Privacy Guidelines for Mobile Connectivity Measurements. `http://www.oii.ox.ac.uk/research/projects/?id=107`, 2013.

[329] C. Zhang, C. Huang, P. A. Chou, J. Li, S. Mehrotra, K. W. Ross, H. Chen, F. Livni, and J. Thaler. Pangolin: speeding up concurrent messaging for cloud-based social gaming. In *ACM CoNEXT*, Dec. 2011.

[330] X. Zhou, M. Jacobsson, H. Uijterwaal, and P. Van Mieghem. IPv6 Delay and Loss Performance Evolution. *Int. J. Commun. Syst.*, 21(6), June 2008.

[331] Y. Zhou. Mobiperf. `http://www.caida.org/workshops/isma/1302/slides/aims1302_yyzhou.pdf`, Feb 2013.

[332] Y. Zhuang, A. Rafetseder, and J. Cappos. Experience with Seattle: A Community Platform for Research and Education. In *GENI Research and Educational Workshop (GREE)*, Mar. 2013.

[333] E. Zohar, I. Cidon, and O. O. Mokryn. Celleration: loss-resilient traffic redundancy elimination for cellular data. In *Workshop on Mobile Computing Systems (HotMobile)*, Feb. 2012.