

Improving Content Delivery and Service Discovery in Networks

Suman Srinivasan

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2016

©2016

Suman Srinivasan

All Rights Reserved

ABSTRACT

Improving Content Delivery and Service Discovery in Networks

Suman Srinivasan

Production and consumption of multimedia content on the Internet is rising, fueled by the demand for content from services such as YouTube, Netflix and Facebook video. The Internet is shifting from host-based to content-centric networking. At the same time, users are shifting away from a homogenous desktop computing environment to using a heterogeneous mix of devices, such as smartphones, tablets and thin clients, all of which allow users to consume data on the move using wireless and cellular data networks.

The popularity of these new class of devices has, in turn, increased demand for multimedia content by mobile users. The emergence of rich Internet applications and the widespread adoption and use of High Definition (HD) video has also placed higher pressure on the service providers and the core Internet backbone, forcing service providers to respond to increased bandwidth use in such networks.

In my thesis, I aim to provide clarity and insight into the usage of core networking protocols and multimedia consumption on both mobile and wireless networks, as well as the network core. I also present research prototypes for potential solutions to some of the problems caused by the increased multimedia consumption on the Internet.

First, this thesis provides details about data usage and working of core protocols (DNS, HTTP, service discovery) and video traffic on networks through measurements and studies that I performed. This will help us understand network usage of data and content consumption happening in emerging computing devices such as smartphones, and the rapidly changing field of content networking, particularly service discovery and content delivery. I present the study of existing service protocols, particularly Zero Configuration Networking (ZeroConf) and multicast DNS (mDNS), on wireless networks. The findings of certain

shortcomings in the ZeroConf protocol, as well as a proposal of an improved architecture and implementation that improves existing service discovery protocols are included.

Secondly, the design and implementation of new software architectures and implementations needed to alleviate problems resulting from existing protocols on wireless networks is presented. Difficult networking problems such as service discovery are addressed through building and extending a suite of applications, called Seven Degrees of Separation (7DS), and BonAHA, a library developed on top of a ZeroConf implementation that makes it easier for software developers to develop applications for wireless opportunistic networks.

The third contribution of this thesis is the research and implementation of new mechanisms to handle increasing data demands at the network core and satisfy the growing appetite for multimedia by end users. The thesis covers my work on on-path content-delivery networks (CDNs), as well as a new distributed and dynamic CDN architecture called ActiveCDN. Both of these CDN projects are part of the NetServ project, which aims at developing a service-virtualization architecture for next-generation core networks.

Fourth and finally, this thesis also describes research prototypes for a new model of networking driven by content, aptly called content-centric networking. I designed research prototypes for content-centric networks, particularly in adding dynamic services and service scalability in such networks. This work involved bridging the content-centric and host-centric paradigms of Internet communication through using IPv6 as a common layer, which is described in detail in the thesis. In addition, I performed an evaluation of real-world video traffic patterns on the Internet, as well as how the growth in video traffic can be combined with analysis of content consumption to enable us to more efficiently perform video and content networking on the Internet of today and tomorrow.

Table of Contents

List of Figures	viii
List of Tables	xii
1 Introduction	1
1.1 Glossary	4
I Wireless Networks	6
2 Introduction: Wireless and Cellular Networks	7
3 7DS - Information Exchange in Opportunistic Networks	9
3.1 Introduction	9
3.2 Problem Statement	11
3.3 Architecture of the 7DS System	12
3.3.1 Zero Configuration Networking Setup	12
3.3.2 Proxy Server	13
3.3.3 Local Web Server	13
3.3.4 Search Engine	15
3.3.5 Cache Manager	16
3.3.6 Multicast Query Engine	16
3.4 Mail Transport Engine	17
3.5 Implementation	18
3.5.1 Experimental Setup	18

3.5.2	Porting 7DS to WRAP	18
3.5.3	7DS on Linux, Mac OS and Windows	21
3.6	Performance Evaluation	21
3.6.1	E-Mail Performance	23
3.6.2	Content Sharing	24
3.7	Security and Privacy	24
3.8	Related Work	25
3.9	Conclusion	27
4	BonAHA: Service Discovery Framework for Opportunistic Network Ap- plications	28
4.1	Introduction	28
4.2	Motivation	29
4.3	Service Discovery	31
4.3.1	Bonjour	33
4.4	BonAHA	34
4.4.1	Architecture	35
4.4.2	BonAHA API	36
4.5	Sample Applications	37
4.5.1	Location Finder	37
4.5.2	Tic Tac Toe	38
4.6	Scalability	41
4.7	Security and Privacy	42
4.8	Related Work	42
4.9	Conclusion	44
5	Measuring and Improving Service Discovery Protocols on Wireless Net- works	45
5.1	Measurements of Service Discovery Performance in Wireless Networks . . .	45
5.2	Improving Service Discovery in Wireless Networks	46
5.3	Multicast Service Discovery in a Campus-Wide Wireless Network	47

II	Delivering Video Content—CDNs and Beyond	49
6	Introduction: Content Delivery Networks and Content Networking	50
7	Unveiling the Content-Centric Features of TCP	52
7.1	Introduction	52
7.2	CDNs for Content-Centric Networking	53
7.2.1	Traditional Approach	53
7.2.2	On-path Content Delivery	55
7.2.3	Advantages of On-Path Content Delivery	56
7.3	Mechanism Details	56
7.4	Prototype Implementation	58
7.5	Performance Evaluation	60
7.5.1	Experimental Setup	60
7.5.2	Round-trip Time and Latency	61
7.5.3	Processing Overhead	62
7.6	Implementation Alternatives	62
7.7	Related Work	63
7.8	Conclusion	64
8	NetServ: Dynamically Deploying Software Defined Networking	65
8.1	Introduction	65
8.2	Technology Overview	67
8.2.1	Click Modular Router	67
8.2.2	OSGi Framework	68
8.3	NetServ Implementation	69
8.4	Evaluation	72
8.5	Related Work	73
8.6	Conclusion	76
9	ActiveCDN: Cloud Computing Meets Content Delivery Networks	77
9.1	Introduction	77

9.2	Motivation	79
9.3	ActiveCDN on the Edge Node	80
9.4	ActiveCDN on the Client	84
9.5	Implementation	89
9.6	Security and Privacy	90
9.7	Related Work	90
9.8	Conclusion	95
10	Trends in Online Internet Video Usage	97
10.1	Introduction	97
10.2	Summary of Results	98
10.3	Distribution of Video Popularity	99
10.3.1	Popularity Distribution Based On Time	99
10.4	Video Length and Popularity	100
10.4.1	Are shorter videos more popular?	101
10.4.2	Are Shorter Videos More Popular On Mobile?	102
10.4.3	Video Length Viewed Across Countries	103
10.5	Popularity Distribution Across Domains	104
10.6	Desktop, Mobile and Tablet Usage	106
10.7	Country Viewing by Day and Hour	109
10.8	OS Viewership by Day	111
10.9	Play and Open Rate by Day of Week	113
10.10	State Transitions in Video Playing	114
10.11	Related Work	116
10.12	Conclusion	118
III	Information Centric Networks	119
11	Introduction: Information-Centric Networks	120

12 CCNxServ: Dynamic Service Scalability in Content-Centric Networks	122
12.1 Introduction	122
12.2 Scaling Services Dynamically in CCNx	124
12.3 Architecture	126
12.4 Implementation	127
12.5 Challenges and Incentives	131
12.5.1 Economic Incentives	131
12.5.2 Technical Challenges	132
12.6 Related Work	133
12.7 Conclusion	134
13 IPv6 Addresses For Naming Content	136
13.1 Introduction	136
13.2 Motivation	137
13.3 IPv6 to the Rescue	139
13.4 Implementation	140
13.5 Mapping Content Names to IPv6, and Vice Versa	141
13.6 Potential Improvements	142
13.7 Implementation	143
13.8 Related Work	145
13.9 Conclusion	146
IV Conclusions	147
14 Conclusions	148
14.1 Opportunistic Networks	149
14.2 Content delivery	150
14.3 Content Centric Networks	151
14.4 Real world video traffic patterns	151

V	Bibliography	152
	Bibliography	153
VI	Appendices	177
A	BonAHA Applications	178
A.1	ONEChat: Enabling Group Chat and Messaging in Opportunistic Networks	178
A.1.1	Introduction	178
A.1.2	Related Work	179
A.1.3	Implementation Of ONEChat	181
A.1.4	Introduction to ONEChat	181
A.1.5	User and Message Discovery Using BonAHA	181
A.1.6	Messaging using RTP and RTT	183
A.1.7	Private and Secure Messaging Using ONEChat Groups	184
A.1.8	Messages in ONEChat	185
A.1.9	Creating, Joining and Leaving a Group	186
A.1.10	Enter-Network Notification	188
A.1.11	Leave-Network Notification	189
A.1.12	Performance Evaluation	189
A.1.13	Future Work	190
A.1.14	Conclusion	190
A.1.15	Acknowledgment	191
A.2	BBS-ONE: Bulletin Board and Forum System for Mobile Opportunistic Networks	191
A.2.1	Introduction	191
A.2.2	Forums and BBS Software	194
A.2.3	BBS in Opportunistic Networks	194
A.3	Service Model	195
A.3.1	Data Management	195
A.3.2	Deployment	196

A.3.3	Networking	197
A.4	Design and Implementation	198
A.4.1	Architecture Overview	198
A.4.2	Implementation	199
A.4.3	Related Work	201
A.4.4	Discussion and Future Work	202
A.4.5	Conclusion	202
A.5	FileXChange: Drag-and-Drop File Sharing in Opportunistic Networks . . .	203
A.5.1	Acknowledgment	204

List of Figures

3.1	The overall structure of the 7DS system, showing the search, multicast and transport engines	10
3.2	The algorithm of the 7DS proxy server for handling HTTP requests	14
3.3	The 7DS search page shows results for a keyword search. The results correspond to matching files in the local cache.	15
3.4	The inside of the WRAP platform. The hand and the pen in the picture show how small the WRAP board is.	19
3.5	7DS running on the WRAP platform. The screenshot shows a Windows HyperTerminal terminal interface with the WRAP board through a serial port connection. The WRAP board runs the <i>query scheduler broadcast</i> which broadcasts packets with the queries.	20
3.6	The cache manager component running on a Mac OS X system	22
4.1	The overall architecture of BonAHA networking from the developer's perspective. The API allows the developer to treat the network as a set of objects with associated metadata. Network events such as devices entering and leaving the network can be handled through simple event handling functions. The diagram shows how events are announced and browsed (1), how metadata is set and obtained (2), and how network communication is performed based on the above operations (3).	30
4.2	The state diagram for the Bonjour API.	32
4.3	The state diagram for the BonAHA API.	34

4.4	A screenshot of the Tic-Tac-Toe game developed using BonAHA. While it looks similar to any regular networked two-player Tic-Tac-Toe game, it is much simpler to develop due to the BonAHA framework.	41
5.1	Service loss rate for different residence times and browsing intervals for ZeroConf in opportunistic networks.	47
5.2	Left: the number of packets per second for the various multicast and broadcast packets in the campus-wide wireless network. Right: the average hourly rate of packets per second sent over a period of 24 hours.	48
7.1	The handshakes and networking messages used in the TCP-interception method of on-path CDNs.	55
7.2	The details of implementing the handshakes and networking messages in the TCP-interception method.	57
7.3	TCP latency of the four test nodes.	61
8.1	A minimal Click configuration.	68
8.2	NetServ prototype architecture.	70
9.1	How on-demand content caching using ActiveCDN works.	82
9.2	A visual depiction of what happens in our ActiveCDN implementation. Our implementation allows multiple users or clients to request video from the content server. The content server is able to control which nodes ActiveCDN is instantiated on, and redirect users to the node of its choice.	83
9.3	A screenshot of the original video playing on a user's browser. This is the video that is directly served by the origin server.	85
9.4	A screenshot of the processed video playing from one of the ActiveCDN nodes. The video, in addition to being cached and served from the ActiveCDN node, has been processed, adding the local weather information.	86
9.5	Client processing: A screenshot of the video with a scrolling news ticker overlaid on the video. The video is unmodified, and the news ticker information comes from the SMIL file that was sent to the client.	88

10.1 Popularity distribution of the top 20 videos for August 2013, broken down by week.	100
10.2 Popularity distribution of the top 10 domains serving video in May 2013 based on streams.	107
10.3 Popularity distribution of the top 100 domains serving videos in May 2013 based on streams.	107
10.4 Changes in percentage terms of mobile devices (compared to overall device usage) used to access video on websites over a period of two years.	108
10.5 Percentage of video accesses for four device types (two each of smartphone and tablet) shows a growth trend, from June 2010 to January 2012.	109
10.6 The number of daily video streams served using just the free version of the popular JW Player. These statistics include only player versions that are HTML5 compatible or higher (version 5.3 and above).	110
10.7 Breakdown of viewership by country, by day of week.	110
10.8 Breakdown of video viewership by country set #1, by hour, for the United States (with time zones adjusted), Brazil, Vietnam and France.	111
10.9 Breakdown of video viewership by country set #2, by hour, for Germany, Taiwan, United Kingdom, Italy, Canada and Turkey.	112
10.10 Device usage by time of day. The numbers indicate the hours of day according to a 24-hour clock, so 0 is midnight, 1 is 1 AM, etc. Blue indicates late night / early morning hours, pink indicates morning hours, yellow is afternoon / early evening, and green is late evening / night.	113
10.11 Device usage by day of week. Light green is Monday to Thursday, dark green is Friday, and blue is weekends. (Connected devices, as explained earlier, refer to living room devices such as the PlayStation.)	114
10.12 A breakdown of open and play rates across the days of the week.	115
10.13 Video state transitions across the multiple states that we measure (load, play, seek, stop, progress). The numbers indicated have been normalized to the number of people who are in the starting ("ready") state which is set to 100.	117
12.1 Overall CCNxServ architecture.	124

12.2	The architecture of our CCNxServiceProxy implementation and how it interacts with the various CCNx utilities.	128
12.3	A screenshot showing the processing of CCNx content after the Content Router interprets the content request.	130
12.4	The transformed content, with an overlay containing weather information at the bottom right of the video, being played in VLC player.	130
13.1	The architecture diagram of our IPv6 content addressing system. In our system, the regular browser makes a HTTP request through a proxy, which translates HTTP requests to an IPv6 content addressing system. The request is sent out over the network, until a router on path that has the content responds to the request. The proxy then translates the retrieved content back into a HTTP response to the user's browser.	138
A.1	Tom and Kate chat within group <i>Room 2</i>	182
A.2	Tom is informed that Kate created group <i>Room 2</i>	187
A.3	The sequence of the creating, joining and leaving a group.	188
A.4	Usage Scenario for the BBS-ONE. A mobile node moves from one isolated opportunistic network to another one, carrying desired information to a location where no connection to the infrastructure. Dotted lines indicate the direction of the movement of the node in and out of an area with wireless ad-hoc connectivity.	193
A.5	System Architecture of the BBS-ONE system. The diagram shows the networking, service discovery and rule validators, the post and node managers and how the components interact.	194
A.6	A screenshot of the FileXChange system.	203

List of Tables

10.1	Video length and number of plays	101
10.2	Video length and average number of views	102
10.3	Comparing desktop and mobile plays	103
10.4	Comparing high and low bandwidth countries	104
A.1	Usage of the four BonAHA functions in ONEChat	183
A.2	Usage of the five ONEChat system messages	185

Acknowledgments

I would like to primarily thank Prof. Henning Schulzrinne, who has guided me throughout the course of my doctoral research. I would also like to thank my committee members (Prof. Vishal Misra, Prof. Augustin Chaintreau, Prof. Gil Zussman), as well as Prof. Gail Kaiser and Prof. Steven Bellovin, all of whom have provided invaluable assistance.

I would like to thank my external committee member, Dr. Volker Hilt, who was also my mentor during my summer internship at Bell Labs. I would also like to thank my research colleagues and fellow members of the Internet Real Time (IRT) Lab, particularly Drs Jae Woo Lee, Se Gi Hong, Salman Baset and Arezu Moghadam for working with me and constantly providing me with help throughout the course of my research.

I would like to thank the following companies and institutions for their financial and other support with my thesis and research work: the National Science Foundation (Grants #04-54288 and #08-31912), Alcatel-Lucent (Bell Labs), DOCOMO Communications Laboratories Europe GmbH and Longtail Video.

Finally, I would like to thank my family, particularly my father and mother, who have patiently supported me throughout my life and helped me out with anything that I've ever needed help with.

Dedicated to my mother and father

Chapter 1

Introduction

We are in the middle of technology changes that impact how society and culture consume data and multimedia content on computer networks. Smartphones, tablets and connected devices (such as Playstations) have brought networking and computing to billions of people who may have never interacted with computers before. In addition to providing basic computing features, these devices can access the Internet and online services through an array of wireless and cellular data networking technologies as well as traditional wired networks. These changes require us to have a deeper understanding of the core issues facing networking and protocols today.

In this thesis, I aim to cover research topics in a broad and diverse array of networking technologies and protocols, and show how some of these core protocols can be improved upon to provide a much better experience to users and consumers, while at the same time making the core network and protocols more efficient.

The exponential growing popularity of mobile devices results in a proportionately growing demand for data on the move, particularly multimedia and video content and data. Cisco estimates that data accessed via “Wi-Fi and mobile devices will account for 66 percent of IP traffic” by 2019 [Cisco, 2015]; the study states that data access on mobile networks was 46% of all IP traffic in 2014. In regards to video traffic, Cisco’s study says that “consumer internet video traffic will be 80 percent of all consumer Internet traffic in 2019, up from 64 percent in 2014.”

Service discovery and content delivery are networking research problems that are well-

known and studied. But with the networking landscape changing rapidly in recent years, new approaches are necessary to tackle emerging problems in these two areas of wireless networks as well as the core of wired, always-on networks.

The rapid growth of smartphones, tablets and the hybrid “phablets” (smartphones with a form factor comparable to tablets) has enabled users to consume data on the move and look for always-connected functionality, either in the presence or absence of cellular and other wireless networks. Such users, in the absence of other forms of networking, would often want to communicate with peers in the local network using *opportunistic networks*, i.e., disconnected network islands made of wireless nodes in close range, but with high mobility and hence high churn rate [Shen *et al.*, 2010].

Meanwhile, at the network core, the increasing consumption of multimedia content on the existing wired networks and in mobile systems is putting a strain on the network core. Cisco Systems estimates that video will constitute 90% of all Internet traffic by 2019 [Cisco, 2015]. While content distribution networks (CDNs) are becoming more and more popular as a means of efficiently distributing multimedia content to end-users on the Internet [Tom Leighton, 2009], current CDN networks have to work on top of the existing Internet architecture using redirection mechanisms such as DNS redirection and pre-deployed hardware and networking routes.

In addition, the emergence of a new class of networking known as content-centric networking [Jacobson *et al.*, 2009b] argues that content, not hosts, are the cornerstone of the Internet today. While content-centric networking has become an active research topic in the networking field, questions remain about this new class of networking: can we use content-centric networking on today's Internet, or will it need an entirely new Internet architecture?

First, given that opportunistic networks are a fairly new class of wireless networks, it is necessary to measure and analyze how existing protocols work in such networks. In Chapter 5 I will describe the work that Dr. Se Gi Hong and I performed to measure and analyze a popular service discovery protocol called Zero Configuration Networking (ZeroConf) [Zeroconf Working Group, 2008] on wireless networks. We discovered problems facing service discovery protocols in highly mobile and transient networks, and worked on an implementation to enhance the ZeroConf implementation and allow it to function correctly

in opportunistic networks. We also evaluated the increasing use of service discovery and content delivery protocols, particular multicast DNS (mDNS), in real-world networks. I also touch on my work on WORKIT, a wireless toolkit for measuring and analyzing network protocols in next-generation wireless and cellular networks.

Second, in addition to analyzing service discovery, I worked on the Seven Degrees of Separation (7DS) [Srinivasan *et al.*, 2007][Moghadam *et al.*, 2008] suite of applications which aims to provide end-user application services in disruption-tolerant mobile networks where data packets could get delayed during transit. 7DS provides the necessary transport and application layer functionality for mobile nodes to exchange information using store-carry-forward communication. This is described in detail in Chapter 3.

Third, developing mobile applications that function properly in opportunistic networks is a difficult process, since such opportunistic network islands do not follow the client-server model of operation. Even peer-to-peer models of networking do not work in such a network due to the high churn rate of mobile nodes in such networks. Chapter 4 describes a library I have written, called BonAHA (Bonjour for Ad-Hoc Applications) [Srinivasan *et al.*, 2009b], which provides an API framework for building applications that run in such opportunistic networks: it aims to be easy and intuitive, provide a level of abstraction to opportunistic networking and at the same time provide flexibility to the developer to develop applications that run in opportunistic networks.

My fourth contribution involves the first prototype for NetServ [Srinivasan *et al.*, 2009a], a research effort to design an extensible architecture for core network services for the next generation Internet. In Chapter 8, I describe our first NetServ prototype, which uses the Click router [Kohler *et al.*, 2000] and the Java-based OSGi module system. NetServ enables service virtualization at the Internet core, thus allowing efficient use of applications such as content delivery networks (CDNs).

My fifth contribution is my work on on-path content delivery networks, which can dynamically intercept and redirect requests for content as well as serve content from a local cache. In Chapter 7, I present the work on my prototype, as well as measurements and analysis of this implementation using popular content providers on a test network.

My sixth contribution is ActiveCDN, a NetServ module which can dynamically deploy

CDN modules and serve content from participating NetServ nodes that are located near the edge of the network. This work, described in Chapter 9, was selected and demonstrated as part of the NetServ modules at the National Science Foundation’s 8th and 9th GENI Engineering Conferences (GEC8 and GEC9). ActiveCDN allows for content providers to dynamically deploy CDN nodes across the Internet based on demand, thus alleviating traffic load on the core networks.

My seventh contribution is dynamic services on content-centric networking. While content-centric networks allow for efficient distribution of content and make content the center of the networking stack, they do not properly or correctly handle the issues of services. In Chapter 12, I describe the architecture and implementation of a prototype I built that allows for building services on top of a pure content-centric networking stack. The chapter also includes an analysis of how service composition and dynamic scaling can be achieved by classifying services correctly.

My eighth and final contribution of this thesis is the evaluation of video traffic on the Internet using real-world data. Chapter 10 contains this evaluation, which can help us understand the nature of real-world video traffic patterns and can allow us to optimize content delivery strategies as Internet content grows.

1.1 Glossary

This section formalizes the terms used in the rest of this thesis.

1. **Opportunistic networks** are network islands that are not connected to the Internet, consisting of wireless nodes in close range, but with high mobility and hence high churn rate. They are also called mobile ad-hoc networks (MANETs) which are defined as self-configuring networks of mobile devices connected by wireless links. They are also called mobile mesh networks, or sometimes simply ad-hoc networks. [Ibing and Boche, 2012]
2. **Network core**, or core network, refers to the Internet backbone, which are principal data routes between large, strategically interconnected networks and core routers in the Internet.

3. **Active networks** are networks in which the traffic flowing through the network (defined as either services or packets) can execute actions that change the state of the network. [Wikipedia, 2015b]
4. **Content Distribution Networks (CDNs)** are nodes set up in order to serve content to end-users in a highly distributed and efficient manner, meant to provide end-users with the best content experience in terms of bandwidth and lower latency. [Rackspace, 2015]
5. **Content-centric networking (CCN)**, also known as **Information-Centric Networking (ICN)**, is a networking model that focuses on content as being centric to network operations. Xerox PARC defines it as an architecture that “operates by addressing and delivering Content Objects directly by Name instead of merely addressing network end-points.” [PARC, 2015]
6. **Host-based networking** refers to how networking on the Internet currently works, where IP (Internet Protocol) packets are addressed and delivered to specific nodes identified by their IPv4 or IPv6 addresses. [Wikipedia, 2015a]
7. **Wireless networks** refers to the IEEE 802.11 protocol based wireless networks that are setup at the homes of end users or in company offices [Cisco, 2015]. These wireless networks allow both computers and mobile devices with a 802.11 hardware and software stack get connected to the Internet via IP. These networks are usually limited by a distance of a few dozen meters at most, but provide up to 54 Mbps bandwidth.
8. **Cellular networks** refers to network services provided by cell-phone carriers such as Verizon and AT&T in the United States [Cisco, 2015]. Cellular networks allow devices (mostly mobile, but including laptops with cellular connectivity) to access the Internet through IP running on 3G or 4G signals from cell phone towers that are in the vicinity. These allow for a much larger range of access than 802.11 wireless networks, and allow a user to transparently switch from one cellular tower to another to ensure seamless connectivity.

Part I

Wireless Networks

Chapter 2

Introduction: Wireless and Cellular Networks

The popularity of smartphones and tablets running Apple's iOS and Google's Android operating systems have taken wireless and cellular data networking mainstream. Even prior to the smartphone revolution though, the popularity of WiFi (IEEE 802.11) brought the ease of wireless networking to a large number of homes and offices.

But the growing popularity of these mobile devices results in a proportionately growing demand for data on the move, particularly multimedia and video content and data. Cisco's 2015 report on Internet traffic [Cisco, 2015] notes that "global mobile data traffic will grow three times faster than fixed IP traffic from 2014 to 2019. Global mobile data traffic was 4 percent of total IP traffic in 2014, and will be 14 percent of total IP traffic by 2019." The growth of faster mobile networking connections indicate that this growth spurt will continue.

This complicates networking for a worldwide networking architecture of wireless and cellular hardware and software that was designed for smaller amounts of traffic. As a result, there are a variety of interesting engineering problems that need to be solved in order to deliver data and content more efficiently over wireless and cellular networks.

This thesis focusses on several areas of wireless networking, particularly in terms of service discovery and network communication across a specific class of wireless networks

called opportunistic networks. Even as cellular wireless and data standards such as 3G and 4G become more and more popular, there are still large areas where mobile users are unable to connect to the Internet directly.

Two projects detailed in this thesis - the 7DS application suite and the BonAHA framework - address this problem, and allow mobile users to communicate with one another locally through wireless (802.11) networks without requiring a wide-area network connection.

Chapter 3 presents the 7DS application suite, which is a set of functional end-user applications including a web browser, search application, e-mail client, file synchronization client and a content search and exchange application.

Chapter 4 introduces the BonAHA framework, a library that allows application developers to develop their own applications that run in opportunistic networks. BonAHA exposes an easy-to-use API to program such disconnected applications. In addition to the BonAHA API, this chapter and related appendices also present some useful applications that we built with BonAHA, such as a instant messaging tool, file transfer program and a Bulletin Board System (BBS) application.

In addition to BonAHA and 7DS, the thesis introduces the WORKIT project to build a experimental wireless testbed. Finally, an analysis of traffic patterns in wireless 802.11 networks, particularly for service discovery protocols. are discussed in Chapter 5.

Chapter 3

7DS - Information Exchange in Opportunistic Networks

3.1 Introduction

Imagine a scenario where there are several people in the vicinity of each other, but without connectivity to the Internet. Assuming that they all have mobile devices and are on the move, it should be possible for data to be transferred among them, as well as in and out to the Internet as people move from disconnected to connected mode, whether connected via wireless or cellular networks. The 7DS project described in this chapter aims to make such a scenario possible.

In the 7DS (Seven Degrees of Separation) [Papadopouli, Maria and Schulzrinne, Henning, 2001] project, we have been investigating how to emulate two core Internet services, namely web access for information retrieval and email for delivering messages from mobile nodes to the Internet. We have implemented and evaluated a 7DS prototype system that leverages search, feedback and propagation limits to build a scalable system that can deliver data to and from mobile nodes.

7DS makes data exchange in opportunistic networks possible by providing an application-level set of protocol services that enables exchange of information between peer devices. It enables dynamic information exchange by using a proxy server, a multicast query system, a search engine, and a Mail Transport Agent (MTA) [Crocker, 2009]. With these entities,

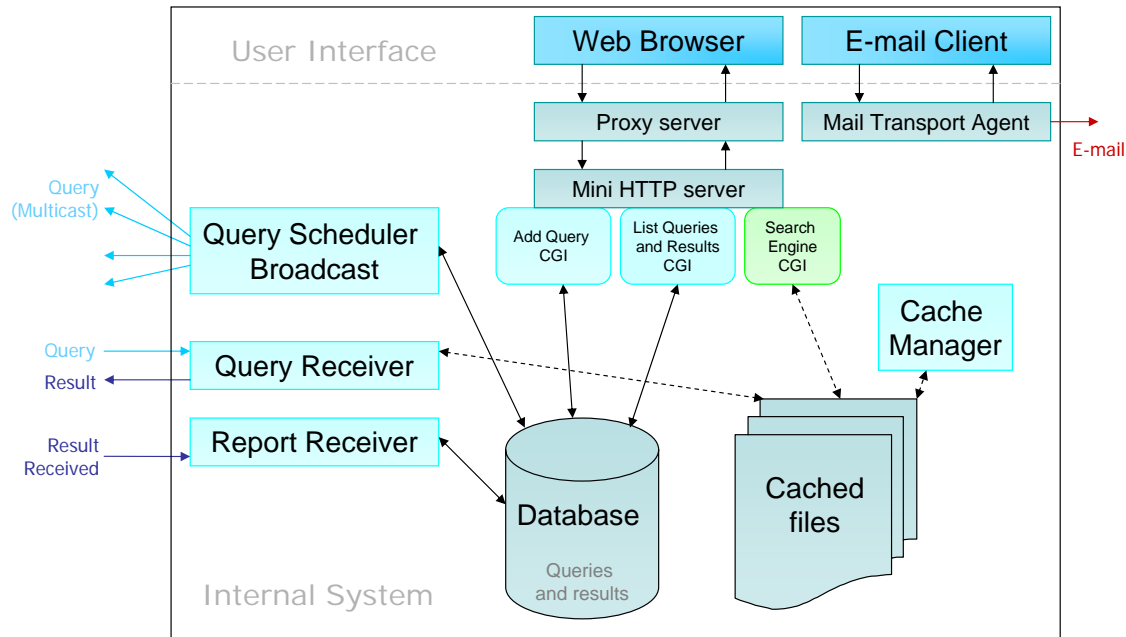


Figure 3.1: The overall structure of the 7DS system, showing the search, multicast and transport engines

7DS can perform efficient and transparent data exchange among peers in the absence of a network connection. Data exchange with the larger Internet occurs when peers encounter an Internet-connected node.

When users connect to each other over opportunistic networks, 7DS enables information transfer from user queries which are broadcast over a multicast query. Any device on the network that has content corresponding to the query will return that information to the querying node, which is relayed to the user and presented in the form of a web-based UI on the mobile browser. All nodes in the 7DS network are capable of both querying for and returning data, thus allowing users to ask for and receive information in the network.

7DS also enables e-mail transport by implementing a Mail Transport Agent (MTA) that receives the e-mail and broadcasts it to the peers. When a peer reaches the Internet, it forwards the accumulated e-mail to an SMTP server that delivers the e-mail to the destination.

The 7DS system is described in more technical detail in the rest of the chapter, which

is organized as follows. In Section 3.2, we introduce the problem of opportunistic networks and how we approach to this problem and how 7DS is implemented. Section 3.3 details the architecture of the 7DS system. In Section 3.5, we describe how the 7DS system was implemented and ported to embedded devices running Linux. We evaluate the performance of the system in Section 3.6, and present related work in Section 3.8.

3.2 Problem Statement

To facilitate information flow in a opportunistic network, devices need to be running a protocol that enables data exchange within this opportunistic network. In order to enable data exchange among peers in a opportunistic network, a peer-to-peer (P2P) data sharing system is needed. Current P2P file-sharing protocols such as Gnutella [Gnutella, 2004] and BitTorrent [Cohen, Bram, 2004] are built to run on connected networks with high-bandwidth. These protocols are too heavy-weight for opportunistic networks that are constrained by bandwidth and connectivity issues.

The 7DS system should be capable of setting up a peer-to-peer network that uses very little bandwidth and is also very robust. It should be able to work seamlessly in a highly mobile scenario where users are moving in and out of the opportunistic network. It also has to be interoperable, platform-independent and use computing and networking resources sparingly to enable it to run on a variety of devices, from mobile devices and embedded systems to laptop computers.

The 7DS platform that we have implemented uses a very lightweight protocol involving simple XML messages for exchanging queries and responses with peers. It works seamlessly and transparently: in the absence of an Internet connection, the 7DS platform automatically queries its peers, retrieves the requests and presents the user with the data he has requested. Finally, the 7DS discovery service handles service discovery as well as discovery of neighboring nodes very efficiently, enabling the system to work robustly in dynamic scenarios.

The 7DS platform was designed as an application-layer solution running as daemons. All the components of the 7DS system can be used by popular existing software that support

protocols supported by 7DS by simply changing a few settings. Examples of these are a proxy server setting on the browser for HTTP traffic (say a Squid reverse proxy at a company), or the SMTP mail server on the e-mail client for e-mail traffic (e.g., sendmail). Further, the 7DS components may, if required, be implemented as transparent proxies so that no reconfiguration of the client software will be needed at all.

3.3 Architecture of the 7DS System

The 7DS system consists of application-level services and CGI executables. The components of the 7DS system consist of a proxy server, web server, search engine, multicast engine and a transport engine.

The proxy server provides the intelligence to the 7DS system, routing requests to the Internet or peers depending on whether Internet connectivity is present or not. The web server provides the user interface and also allows files to be exchanged using the HTTP protocol. The search engine enables local content searches, while the multicast engine enables searches across peers.

These binaries were developed at Columbia University's Internet Real-Time Lab and tested on Windows, Mac OS and Linux platforms.

3.3.1 Zero Configuration Networking Setup

The discovery protocol of the 7DS system is mainly based on the Zero Configuration Networking specification (ZeroConf) [Zeroconf Working Group, 2008]. ZeroConf enables devices to obtain IP addresses for network connectivity without a central DHCP server. It uses multicast DNS (mDNS) [Cheshire and Krochmal, 2013b] for name resolution, and either DNS Service Discovery (DNS-SD) [Cheshire and Krochmal, 2013a], Simple Service Discovery Protocol (SSDP) [Goland *et al.*, 1999] or Service Location Protocol (SLP) [Guttman *et al.*, 1999] for service discovery.

In our implementation, IP addresses are allocated through a discovery protocol with a cross-platform implementation of Zeroconf called Howl [Howl, 2003]. (We will be using Apple's Bonjour [Bonjour, 2005] in future versions.) The 7DS discovery program uses Howl

to publish a service description using ZeroConf publishing services to all the clients that are listening for the publish message. The program also acts as a ZeroConf subscriber so that it can receive messages that are being published. As services are removed and added, the discovered services are stored in memory. This enables the system to find services and their locations without a discovery server such as DNS Service Discovery.

3.3.2 Proxy Server

The proxy server listens to incoming HTTP requests. Based on the type of request and whether the device is connected to the Internet or not, the proxy server decides to serve the request from the cache, the Internet or through querying other 7DS system nodes via the 7DS multicast engine. The proxy server serves as the interface between the user, the Internet and other 7DS peers.

The proxy server, based on the incoming query, retrieves the data object most relevant to the user's request from the local cache or the Internet, in that order. The proxy server uses the libcurl [libcurl, 2005] library in order to retrieve files over the network.

The algorithm used by the 7DS server to decide how to serve the client's request is outlined in Fig. 3.2. A separate service thread is created to handle each client.

3.3.3 Local Web Server

The web server on the 7DS system serves two functions. First, it runs the web-based user interface to the 7DS system. Secondly, it works together with the proxy server to display local cached results in the absence of Internet connectivity.

The web server should be capable of running on embedded devices. One such small open-source web server is tthttpd [tthttpd, 2006]. The tthttpd binary is only 49 KB in size, making it suitable for embedded devices. Another web server that is slightly larger in size but has more features is called lighttpd [lighttpd, 2006]. In addition, any web server that supports CGI and PHP can be used in conjunction with the 7DS system.

The 7DS system uses a folder where shared files are placed. This directory can be searched and indexed by the 7DS system components.

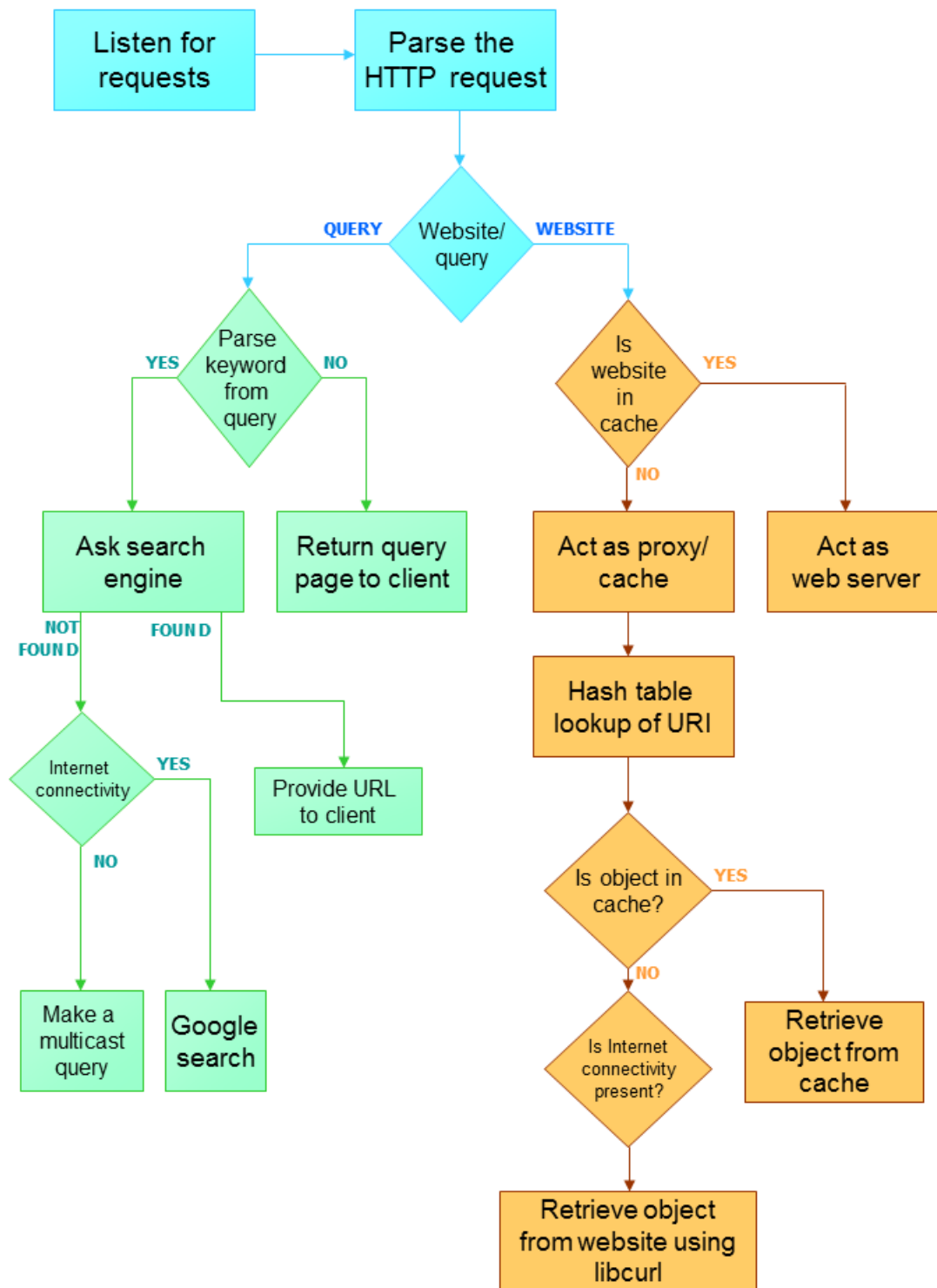


Figure 3.2: The algorithm of the 7DS proxy server for handling HTTP requests

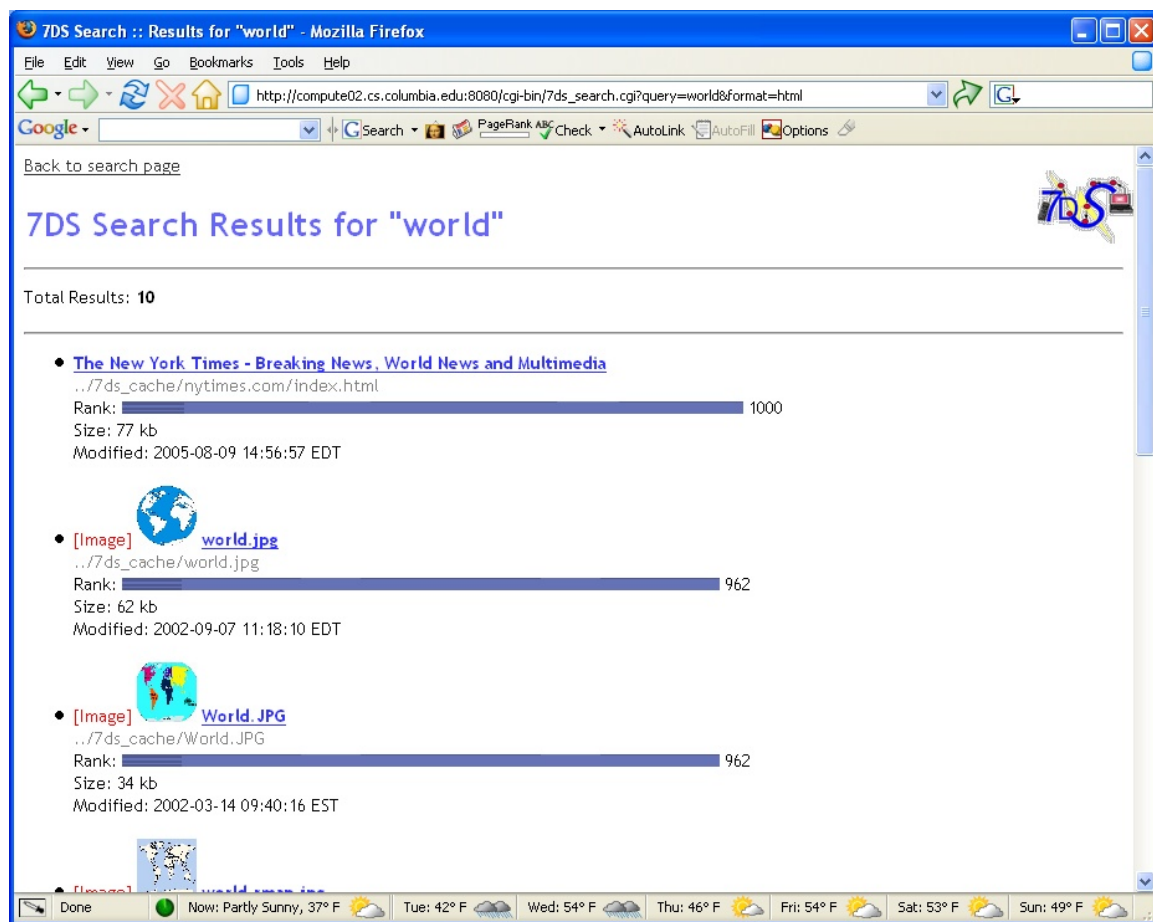


Figure 3.3: The 7DS search page shows results for a keyword search. The results correspond to matching files in the local cache.

3.3.4 Search Engine

The functioning of the search engine and the multicast engine are shown in Fig. 3.1. The search engine is built using the Swish-e library [swish-e, 2006]. It indexes HTML and XML files for keyword searches. Other file formats, such as Microsoft Word, Adobe PDF documents, and popular image formats such as JPEG, PNG and GIF, can also be indexed through a plugin architecture, and enabled us to build plugins for these file types.

The search engine is a CGI binary that runs on the local web server. It provides the user the ability to find files corresponding to the requested keyword that exist in the device's internal database.

A screenshot of the 7DS search engine in operation is shown in Fig. 3.3.

3.3.5 Cache Manager

The cache manager is a daemon that periodically runs in the background. The default interval is 20 seconds, but this can be modified through the configuration files. The cache manager checks if there have been any updates to the cache where the shared files reside and updates the indexes used to search the cache if necessary. If there have been no updates to files in that directory, then it just goes to sleep without taking any action.

3.3.6 Multicast Query Engine

For the multicast communication and sharing engine, we decided to use a keyword queries instead of entire URLs. The reason is as follows: if a user does not have a specific page and requests it, there is a high likelihood that his or her peers do not have that exact material or URL stored in their cache, either. So instead of querying for specific URLs, we query for keywords that represent the closest match to what the user is searching for.

The multicast query engine is used to exchange information among peers in the network. The user first enters a query through the 7DS web-based user interface. This query is added to the device's internal database, and then multicast to other nodes, with the results being returned asynchronously to the querying node.

The user is then presented with a dynamic page that lists the results corresponding to the user's query. This dynamic page, which is generated by a CGI binary, refreshes every 10 seconds and provides the user with an updated result list.

For the multicast system to work seamlessly, the following components are needed.

The queries, results and corresponding peers are stored in a **SQLite query database**, which is a small-footprint, open-source database engine [sqlite, 2006]. Unlike the larger and more popular database engines, SQLite does not require a daemon to handle SQL requests and is hence very suitable for our project.

The **query scheduler broadcast** engine broadcasts the query list in an XML-encoded string to the network. It reads the list of queries, encodes them in an XML-formatted string and broadcasts the string on a multicast packet. It sleeps for a small interval (20 seconds

by default) and then resumes and broadcasts again.

The **query receiver** listens for incoming packets. Upon receiving a query list, it runs a local search on the device using the search engine. If related information is present, it encodes it in a RSS-based XML format [RSS, 2006] and sends the XML as a response in UDP packets to the requesting peer.

The **report receiver** listens on a UDP port for packets sent by the query receiver. Upon receiving the XML packet containing the response, it decodes and parses the XML. It adds the information about the queries, corresponding results and peers to the database table while avoiding duplication.

In addition to the daemon components that are running on the device, several CGI programs invoked by the web server provide the user interface to allow the user to add queries and to view the results. The CGI query page allows a user to add a query to the database. The CGI results page lists the queries that were made and also shows the results corresponding to each query. The results page automatically refreshes at regular intervals in order to return the latest results to the user.

3.4 Mail Transport Engine

In addition to functioning as a query/response system, 7DS is also designed to perform e-mail gathering and delivery. The core communication protocol for this part is SMTP [Klensin, 2001].

The SMTP server listens to incoming messages and transfers those messages which should be propagated through the network to the local Message Transfer Agent (MTA). The MTA unit later relays them to its neighboring MTAs. The SMTP server also takes care of managing and storing all the received e-mails in each 7DS mobile node.

The SMTP server receives the e-mail content from the other nodes broadcasting to it and creates a SHA1 hash of the email and recipient information. When the 7DS node meets another node, its MTA goes through the hash-table and the email directory, reads all the stored emails and sends them to the peer's MTA. When the node is connected to the Internet, the Transport Engine sends the e-mails to the intended recipient. Because

of problems with e-mail duplication, we will explore the possibility of filtering the e-mails through a single server in future versions.

The library used to implement SMTP functionality is libESMTP [libesmtplib, 2006].

3.5 Implementation

The first version of the 7DS system written in C was completed in 2006 and tested on several platforms. A running 7DS system can be downloaded from the 7DS project web page [7DS Homepage, 2005]. 7DS has been compiled and tested on regular desktop versions of Linux, a small-footprint Linux running on an embedded hardware platform called WRAP [PC Engines, 2007], as well as Windows and Mac OS X.

3.5.1 Experimental Setup

We ran the 7DS code on several computers in order to test the different parts of the system and see whether they performed as expected in providing Internet services in an opportunistic network. Files from a few test websites were placed in 7DS caches of the different computers. Searches were performed and if results were found, they were transferred successfully.

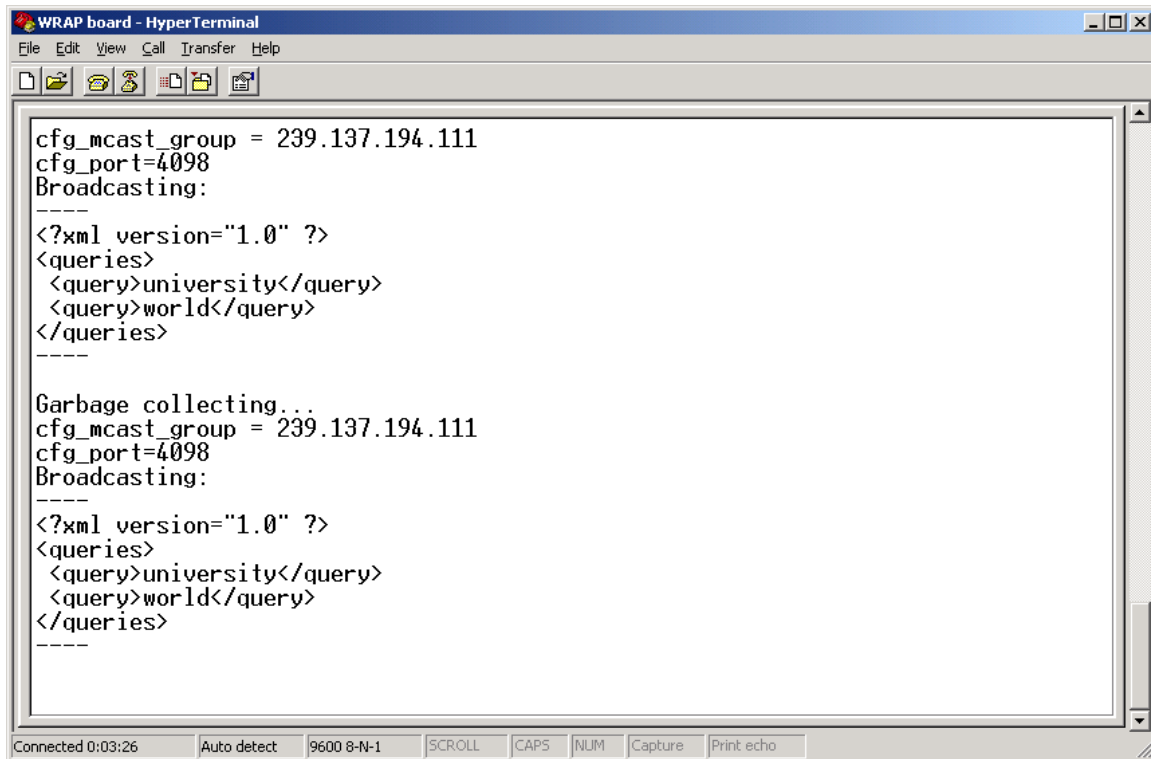
The computers and devices that ran the 7DS code were all set up and tested in two wired networks (Computer Science department and Electrical Engineering department), one wireless network (Columbia University's Engineering School) and one ad-hoc network (in Columbia's COMET Lab). Our pilot test ran on Red Hat Linux, Windows, Mac OS X as well as an embedded system running a small-footprint Linux operating system (LEAF).

3.5.2 Porting 7DS to WRAP

This section has details about how 7DS was packaged to run on PC Engines' WRAP (Wireless Router Application Platform) hardware platform. As of 2015, this product has reached end-of-life, according to the manufacturer's website [PC Engines, 2007], but when we built our 7DS application in 2005-2006, it was one of the leading models for developing embedded applications.



Figure 3.4: The inside of the WRAP platform. The hand and the pen in the picture show how small the WRAP board is.



```
WRAP board - HyperTerminal
File Edit View Call Transfer Help

cfg_mcast_group = 239.137.194.111
cfg_port=4098
Broadcasting:
-----
<?xml version="1.0" ?>
<queries>
  <query>university</query>
  <query>world</query>
</queries>
-----

Garbage collecting...
cfg_mcast_group = 239.137.194.111
cfg_port=4098
Broadcasting:
-----
<?xml version="1.0" ?>
<queries>
  <query>university</query>
  <query>world</query>
</queries>
-----

Connected 0:03:26  Auto detect  9600 8-N-1  SCROLL  CAPS  NUM  Capture  Print echo
```

Figure 3.5: 7DS running on the WRAP platform. The screenshot shows a Windows HyperTerminal terminal interface with the WRAP board through a serial port connection. The WRAP board runs the *query scheduler broadcast* which broadcasts packets with the queries.

The WRAP system is a board that is approximately the size of an adult's palm. The processor is a National Semiconductor Geode x86. It also has a network interface card and a wireless card. Our WRAP board boots off a 32 MB Compact Flash (CF) card attached to the board's card reader.

We installed a popular embedded version of Linux called Linux Embedded Application Firewall (LEAF) [LEAF, 2006] on the CF card and configured it to boot from the CF card on the WRAP board. LEAF is a stripped down version of Linux that boots off most IDE, memory or other devices with a small-footprint kernel. Packages are added via LEAF Repository Package (LRP) files which contain the binaries and metadata for running them.

The 7DS system was repackaged for LEAF by packaging the binaries in the LRP format. Also, because of the absence of libraries necessary for running 7DS on the LEAF platform, we also repackaged glibc 2.3, libcurl, swish-e, Howl and other Linux libraries in the LRP format.

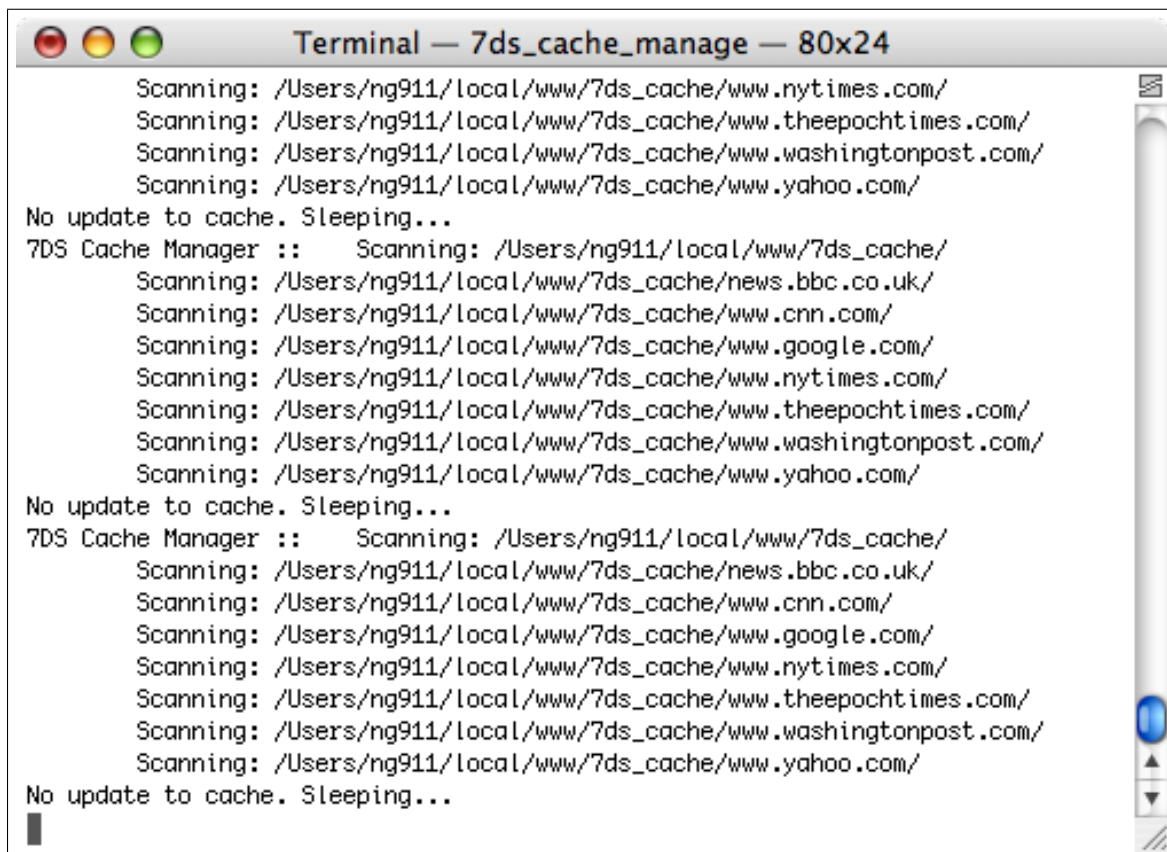
3.5.3 7DS on Linux, Mac OS and Windows

7DS was initially built on the Linux platform, and now exists as a GNU-style source distribution that can be built with `configure` or `make` commands. We have ported the 7DS system to run on the Mac OS X and Windows platforms as well.

For Mac OS X, a packaging system called DarwinPorts [DarwinPorts, 2007] needs to be installed. DarwinPorts provides an open source packaging management system for the Mac OS platform. A screenshot of 7DS components running under Mac OS X is shown in Fig. 3.6. 7DS has also been ported to the Windows platform using the Cygwin [Cygwin, 2007] shell and GNU utilities that come with Cygwin.

3.6 Performance Evaluation

Yuen and Schulzrinne [Yuen and Schulzrinne, 2006] have carried out an analytical study of the feasibility and performance of the 7DS system in opportunistic networks. In particular, they have compared time-based and hop-based Time to Live (TTL) schemes during message transfer between nodes. Some of their results are summarized below in the subsection



```
Terminal — 7ds_cache_manage — 80x24
Scanning: /Users/ng911/local/www/7ds_cache/www.nytimes.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.theepochtimes.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.washingtonpost.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.yahoo.com/
No update to cache. Sleeping...
7DS Cache Manager :: Scanning: /Users/ng911/local/www/7ds_cache/
Scanning: /Users/ng911/local/www/7ds_cache/news.bbc.co.uk/
Scanning: /Users/ng911/local/www/7ds_cache/www.cnn.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.google.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.nytimes.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.theepochtimes.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.washingtonpost.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.yahoo.com/
No update to cache. Sleeping...
7DS Cache Manager :: Scanning: /Users/ng911/local/www/7ds_cache/
Scanning: /Users/ng911/local/www/7ds_cache/news.bbc.co.uk/
Scanning: /Users/ng911/local/www/7ds_cache/www.cnn.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.google.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.nytimes.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.theepochtimes.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.washingtonpost.com/
Scanning: /Users/ng911/local/www/7ds_cache/www.yahoo.com/
No update to cache. Sleeping...
```

Figure 3.6: The cache manager component running on a Mac OS X system

analyzing performance evaluation of the 7DS e-mail system.

An important factor that would affect performance is the presence and density of wireless Access Points (AP). 7DS itself will benefit nodes in opportunistic networks, allowing them to retrieve or send information through peers to the Internet, but performance improvements (measured by lowest delay in reaching the final end-point) in some applications - like e-mail sending - will depend on the AP density. If performance measurement is done in a state like North Dakota, where the population density - and hence wireless AP density - is low, the delay in e-mails reaching the server is much larger. However, in a densely populated area like Manhattan, 7DS will be of more use and help in reducing delays dramatically. A study performed in February 2002 [Public Internet, 2002] shows that there were over a total of 13,000 wireless APs deployed in the areas of Manhattan covered by the study. Even though not all of them are open access points, a recent project called Cable WiFi [Networks *et al.*, 2014] allows users of the most popular cable networks in the country to be able to connect to other cable providers' wireless access points.

3.6.1 E-Mail Performance

We will look at e-mail performance in detail. The most critical part of the e-mail delivery process is the amount of time the e-mail spends in the 7DS network itself. Once the e-mail reaches the Internet, delays are minimal, in the order of seconds. Hence, we will attempt to quantify the performance boost due to 7DS in terms of improvement in delay while the node is in the 7DS network.

Yuen and Schulzrinne [Yuen and Schulzrinne, 2006] find that message delivery in most of their target scenarios is of the order of 100 seconds, which is quite reasonable. They also find a e-mail queue storage size of 50 messages on the node when the wireless AP is seventeen minutes away, 65 messages when the AP is thirty-four minutes away and 127 messages when the AP is eighty-three minutes away. Given the moderate size of e-mail messages, and the vastly increased storage in mobile devices today, we believe that the storage-delay tradeoff is quite worthwhile.

Without 7DS, each node would have had to wait to get to the AP itself, and the delay would have been five to ten times as large. Even though more e-mails are stored on behalf

of peers, the storage costs are a small price to pay for the reduction in delay.

3.6.2 Content Sharing

Unlike sending e-mail, sharing of webpages or content is much more dependent on the data present in the local network. Hence, 7DS will improve deliverability when websites or other forms of content are requested in an opportunistic network. Studies have shown that distribution of webpages in terms of popularity follows Zipf's law [Jelenkovic, Predrag and Radovanovic, Ana, 2003] [Almeida, Virgílio and Bestavros, Azer and Crovella, Mark and Oliveira, Adriana de, 1996]. Since the most popular content will be requested by most of the nodes in the network, several of the nodes would have an updated version of the requested content in their cache and could return them to the node that requests the webpage. Even though the content may be slightly outdated, the retrieval of this information is still more useful than having to wait to get Internet connectivity.

3.7 Security and Privacy

In developing 7DS, we need to consider the security and privacy of the users exchanging information. Even though the user might be sharing public content from a website, it could be content that the user has cached since they visited the site frequently. By making carefully crafted queries to the network and obtaining cached information from peers, it may be possible for a rogue user to find out website visit patterns for various users on the network. While 7DS itself does not identify the user who has shared the data, it may be possible for a rogue user to install a packet sniffing software and duplicate or spoof 7DS packets to monitor traffic on the network and analyze website visit information from peers. To alleviate this, we could block users who are sending spurious query packets, or too many queries. In order to prevent sniffing, we could also encrypt the exchanges using a pair-wise key such as ones generated by the Diffie-Hellman protocol. If encryption is not possible, the adversary will at least not be able to guess the private information of the user who is sharing content, and will only be able to sniff publicly available content. 7DS also ensures that we don't cache content loaded over SSL or anything that requires a login, and hence

does not store or share private information such as financial or medical information which are always behind an authenticated server.

It may also be possible for a user to send data that has been modified (which results in the requesting user getting wrong information), or worse, data infected with malware in the guise of legitimate data (which results in the requesting user getting infected by malware). This same problem also happens in peer-to-peer file sharing networks [Kalafut *et al.*, 2006], where shared files might sometimes be infected with malware, and it would be impossible to prevent this without some form of centralized signing and certification mechanism, or consensus forming by a majority of peers in the network.

3.8 Related Work

When comparing similar applications, file-sharing applications like Gnutella [Gnutella, 2004] and BitTorrent [Cohen, Bram, 2004] are the first two applications that come to mind. However, these protocols are designed to work with always-connected clients. Further, the base protocols for peer-to-peer file sharing applications are very inefficient and use a lot of packets to communicate and exchange files. These involve too much overhead for a mobile network.

JXTA [Sun Microsystems, 2001] is a library that enables development of XML-based P2P protocols to allow peers in a network to interact with each other. However, just like the Gnutella and BitTorrent networks, JXTA is suitable for devices that are rarely disconnected from the Internet or large-scale networks.

Hayes and Wilson [Hayes, Anna and Wilson, David, 2004] have built a platform based on Gnutella for sharing files on a peer-to-peer mobile ad-hoc network. However, they use the Gnutella protocol which includes routing capabilities that are not needed in the 7DS system, which is meant to be a one-hop system. The 7DS protocol is much lighter and requires very little data to be exchanged. Further, by virtue of being an application level service, it is abstracted from the underlying network. Hence, in contrast to Hayes' work that runs only on Bluetooth, 7DS is capable of running on any IP-based network, be it Bluetooth, Ethernet, Wi-Fi or other networks.

Klemm, Lindemann and Waldhorst [Klemm, Alexander and Lindemann, Christoph and Waldhorst, Oliver, 2003] have built a P2P file-sharing system called ORION (Optimized Routing Independent Overlay Network) for mobile ad-hoc networks. It uses an overlay network that combines application level query processing with network layer route discovery for file sharing. 7DS' multicast system works similarly, but without requiring a routing system. Further, 7DS enables a whole set of network applications, not just file sharing.

iClouds [Heinemann *et al.*, 2003] is another P2P application that enables information sharing in mobile environments. iClouds is built on the J2ME platform. iClouds uses a UDP "ping/pong" mechanism (to term to describe packets that require acknowledgement) to discover nearby services. In contrast, 7DS uses ZeroConf for service discovery. The iClouds' "virtual notice board" concept using information exchange of iHave and iWant lists is similar to 7DS' community extensions, even though they are implemented differently. The 7DS community extensions allow a user to define and build their own extensions of content and application sharing.

Proem [Kortuem *et al.*, 2001] is a platform similar to the 7DS system. Like 7DS, it is meant for P2P sharing on disconnected mobile ad-hoc networks. Proem is a protocol stack that allows other developers to build on top of it, but is not an application itself that can be deployed like 7DS.

Earlier versions of the 7DS system were developed several years ago [Papadopouli, Maria and Schulzrinne, Henning, 2001] [Papadopouli, Maria and Schulzrinne, Henning, 2000], but they were written in Java. Our current implementation was built from ground-up in C, and it is hence smaller and faster than the previous version.

TribeHive [TribeHive, 2016] is a company founded in 2013 that builds custom apps for large sports stadiums, and also provides custom connectivity as an alternative for what they call expensive Wi-Fi setup at the same venue. TribeHive mobile apps make use of the WiFi Direct feature found in modern smartphones to connect with other smartphones in the neighboring vicinity, and allow the apps to share data and update information from the local network rather than having to connect to the Internet to get the latest data [The Engineer, 2014] [Engineering and Technology Magazine, 2014]. Previous versions of the app apparently used Apples GameKit library to offer the link-local data sharing functionality,

but that has been replaced by WiFi Direct communication in more recent versions of the software. In this case, the functionality is very similar to how 7DS operates in that it uses link-local networks to share and exchange information.

3.9 Conclusion

The 7DS system fulfills its role in serving as a platform for exchanging information in a opportunistic network. The components we have built so far enable webpages and e-mails to be exchanged within the opportunistic network.

In the absence of ubiquitous connectivity, the 7DS system presents a good solution for implementing transparent data exchange in an opportunistic network. As devices join and leave the network, they bring in new information or propagate internal information that needs to be sent to the outside network.

Chapter 4

BonAHA: Service Discovery Framework for Opportunistic Network Applications

4.1 Introduction

In today's mobile networks, devices often move from one wireless or cellular network to the next, forming transitory associations without a standard client-server infrastructure. Devices constantly transition from one network to another, meeting new peers and exchanging information. In this scenario, traditional models for writing networking software, such as the client-server model or even the peer-to-peer model, turn out to be unsuitable for writing mobile applications. A new framework needs to be developed for this class of applications to be aware of device transitions as well as metadata (properties) that the devices possess. In this chapter, we present a framework we have built for this class of applications. Our library, called BonAHA (Bonjour for opportunistic network applications) aims to be easy and intuitive, abstract the opportunistic networking details and at the same time provide flexibility to the developer to develop powerful and rich opportunistic network applications.

As I showed in Chapter 3, we need a new class of software applications to function in opportunistic networks. For software applications to work properly in highly mobile

networks, it is necessary to maintain some sort of awareness of network state and discover devices entering and leaving the network. This requires multicast queries with unicast responses in order to keep track of devices in the network.

Service discovery protocols provide a simple framework to match our requirements. However, raw service discovery APIs, while suitable for writing applications that announce and browse for services, are not inherently suitable for writing mobile applications that run in opportunistic networks. This is because service discovery protocols only address service announcement and discovery, and require the developer to implement the details of monitoring network transitions, which could quickly become very tedious.

With the BonAHA framework we have developed, we provide a framework for easy development and deployment of such opportunistic network applications. BonAHA is built on top of a popular set of service discovery protocols called Zero Configuration networking (ZeroConf) [Zeroconf Working Group, 2008], the most popular implementation of which is a library by Apple Computer called Bonjour [Bonjour, 2005].

In this chapter, we will present our motivations for developing the BonAHA API in Section 4.2. In Section 4.3, we introduce service discovery, its features and its shortcomings when applied to highly mobile network nodes. We focus on Bonjour.

In Section 4.4, we introduce our BonAHA framework, compare its API to that of service discovery and show how BonAHA is much simpler and more intuitive for developing opportunistic network applications. Several technical details, particularly the comparison of Bonjour and BonAHA, as well as details about the Bonjour protocol, are also presented. In Section 4.5, we discuss sample applications that we have written using the BonAHA API and present sample code to show how it can be used.

4.2 Motivation

An overview of the BonAHA framework we have developed is shown in Figure 4.1. BonAHA exposes the devices in the network as objects that can be accessed using object-oriented function calls. It also enables the developer to handle network events such as devices entering and leaving the network, as described later in the chapter.

BonAHA – OO Network Events

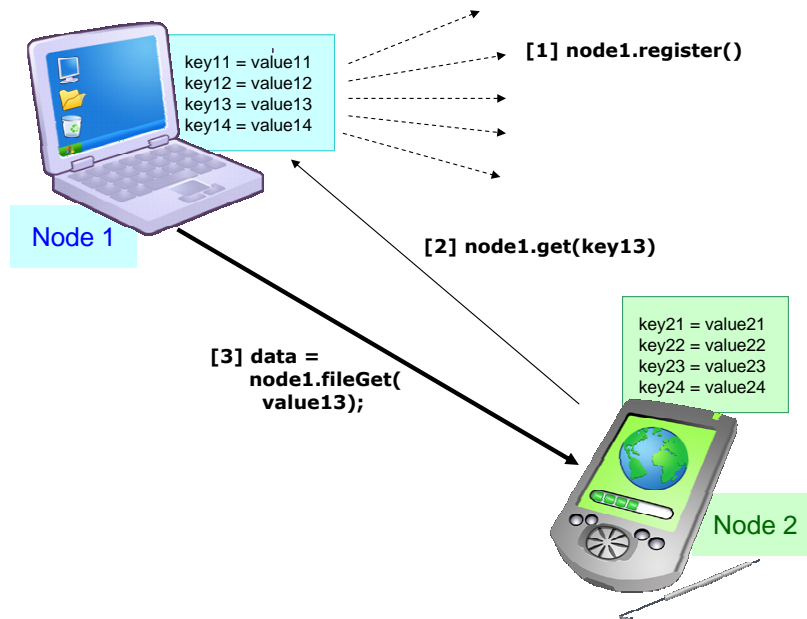


Figure 4.1: The overall architecture of BonAHA networking from the developer's perspective. The API allows the developer to treat the network as a set of objects with associated metadata. Network events such as devices entering and leaving the network can be handled through simple event handling functions. The diagram shows how events are announced and browsed (1), how metadata is set and obtained (2), and how network communication is performed based on the above operations (3).

For networked applications to function consistently in opportunistic networks, they need to be aware of changes in the network, such as devices entering or leaving the network, and changes in properties associated with that device.

In order to develop applications that support opportunistic networks, an application developer will have to write code that implements and handles several networking events. In fact, when we implemented our first version of an application to allow distributed web access and e-mail delivery [Srinivasan *et al.*, 2007], we coded multicast packets for announcement and discovery manually. However, it becomes tedious to rewrite the same code and maintain all internal network state when developing applications for opportunistic network applications.

We found that the service discovery most suitable for our purpose is an implementation of multicast DNS in the form of Bonjour, an open-source technology from Apple Computer, which is implemented and runs on Mac OS, Windows, Linux, Unix variants and several other platforms. Based on our use of Bonjour for building mobile opportunistic network applications [Srinivasan *et al.*, 2007], we find that for a truly mobile application to be completely implemented and functional, it has to implement all the callback functions in the Bonjour API in order to work well in an opportunistic network. This is difficult unless the developer completely understands Bonjour.

Our BonAHA framework runs on top of Bonjour and is suitable for writing opportunistic network applications. By handling the details of the service discovery protocol, BonAHA allows the developer to focus on developing opportunistic network applications by allowing him or her to easily keep track of network state, the devices in the network, as well as the metadata associated with each device.

4.3 Service Discovery

Service discovery refers to protocols which enable automatic detection of devices and services on a computer network. Service discovery is fairly mature technology and has been around for over a decade. Service discovery protocols range from lightweight protocols such as DHCP [Droms, 1997][Lemon and Sommerfeld, 2006][Aboba *et al.*, 2006] (which can include

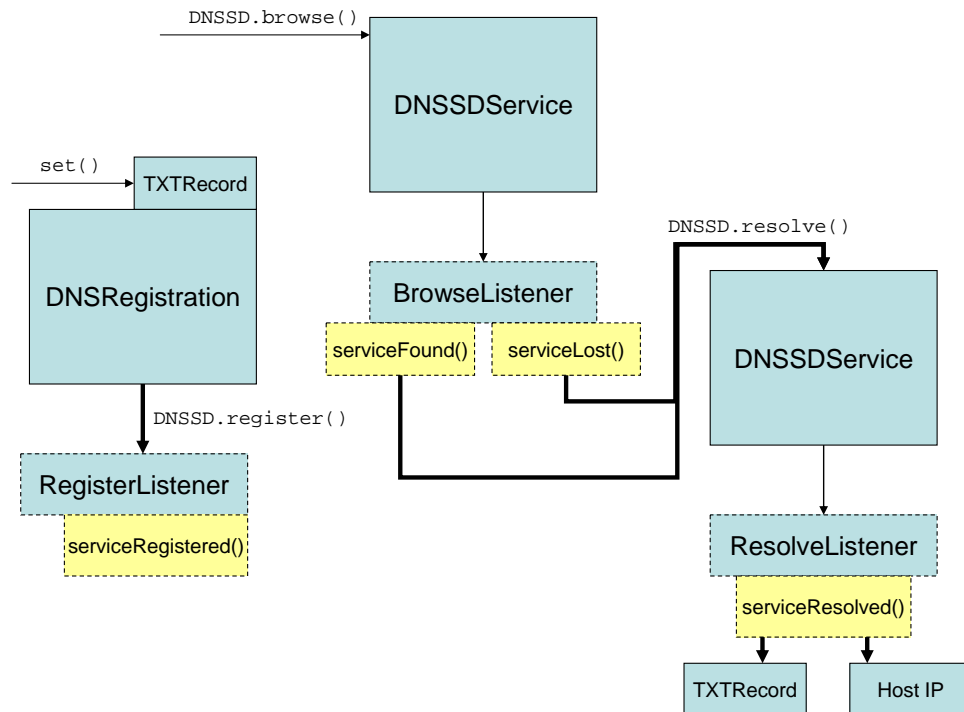


Figure 4.2: The state diagram for the Bonjour API.

service discovery information on top of the basic DHCP) to heavyweight protocols like JXTA [Sun Microsystems, 2001].

However, service discovery protocols that can be applied to writing applications that run in opportunistic networks are very few in number. Examples of these are Peer2Me [Wang *et al.*, 2007] and LightPeers [Bent Guldberg Christensen, 2007], both described in more detail in Section 4.8.

Service discovery includes classes of distributed technologies such as distributed hash tables (DHTs). However, such technologies are actually too heavyweight for use in a limited-device, intermittent network scenario. Further, most of these protocols require some sort of bootstrap device which is not practical in intermittent and especially highly mobile network scenarios.

Among the limited set of service discovery protocols that are suitable for opportunistic network applications, the Bonjour implementation seems to be the most mature and stable.

4.3.1 Bonjour

However, the Bonjour API, even though simple, requires a learning curve and has some shortcomings that make it not completely applicable to opportunistic networks.

The fundamental reason why Bonjour, in its native form, is not suitable for opportunistic network applications is that it is primarily concerned with changes in the network, such as services entering and leaving the network. It makes no attempt at capturing and maintaining a view of nodes currently in the network, which is important for easily developing applications that work in opportunistic networks.

The following is a list of shortcomings in the Bonjour API when applied to opportunistic networks:

- Bonjour has three interfaces and five event-handling functions, all of which have to be completely implemented by an application developer who is attempting to implement an application that runs in opportunistic networks.
- Bonjour makes no attempt to capture and store the metadata of nodes currently in the network. Even though the Bonjour API enables the developer to handle network

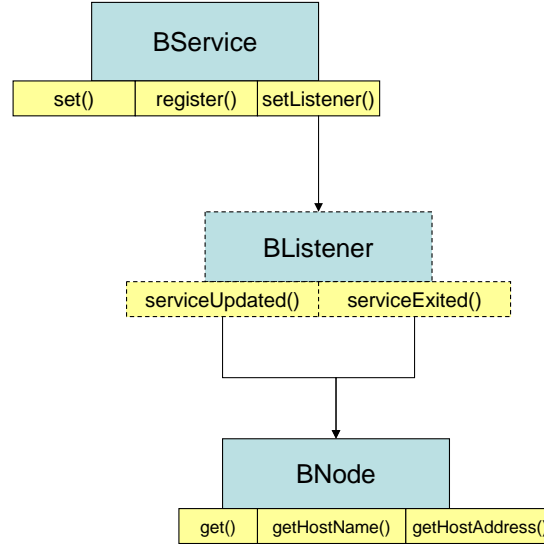


Figure 4.3: The state diagram for the BonAHA API.

events, it does not internally store this state of the network, such as devices present and associated metadata. The developer has to maintain this state himself.

- Bonjour requires a two-step process for detecting a device entry or exit. In order to access device details and metadata, device entry needs to be chained with another function that gets metadata from the device.

4.4 BonAHA

We have released BonAHA under the GPL license on Sourceforge [SourceForge, 2009]. As part of our earlier work, we completed a GUI library called BonSwing [Srinivasan and Schulzrinne, 2007] which provides developers a GUI library to build applications for oppor-

Listing 1 The details of the BonAHA API.

Classes:

BService: Description of a DNS-SD service

Bdevice: A device that offers a BService. Note that while a *Bdevice* maps to a physical device, there may be a many-to-one relationship as each physical device can offer multiple services, and hence correspond to multiple *Bdevices*.

Interfaces:

BListener: Handling network events such as services entering or exiting the network

Callback functions for events

BListener

serviceUpdated() when services appear or are updated

serviceExited() when services leave

tunistic networks. The library and sample applications are also available for download from SourceForge [Suman Srinivasan, 2009].

4.4.1 Architecture

BonAHA uses a simple concept of service. Applications can register or listen to a particular service on a network. This service is instantiated as a string name. The name is the same as the DNS-like names used in DNS-SD's service names. As an example, a device announcing a HTTP server service would have its DNS service name set to *_http.tcp*.

Metadata for the device is set using suitable object-oriented function calls. Services on the network can be discovered by instantiating a service object and registering it to respond to network events, such as a device update or devices arriving or exiting the network. Metadata for neighboring devices in the network can be obtained by using making object-oriented function calls using the underlying mDNS protocol.

The API used for handling network events in BonAHA is given in Listing 1. The state diagram is outlined in Figure 4.3.

An application that is announcing a service will usually follow the following steps:

- Create a service object with the name of the service;
- Set any metadata associated with that service;
- Register it.

An application that is listening for service announcements on the network will usually follow the following steps:

- Create a service object with the name of the service;
- Set an event handler object for this service;
- The class handling events for the service will handle events corresponding to device updates (which includes devices arriving in a network) and device exits;
- Metadata associated with that device can be retrieved from the devices.

Using the BonAHA library, an application developer would be able to completely have network device and metadata events in a few lines of code. Without this framework, he would need to understand Bonjour or other service discovery protocols thoroughly and implement at least several dozen lines of code to listen to network events and device arrival and departure.

4.4.2 BonAHA API

The BonAHA API aims to present the network to the developer as a set of objects with metadata which enter and leave the network, thus triggering events. An outline of the API can be seen in Figure 4.3 and in Listing 1.

The *BService* class allows one to construct a service instance which corresponds to a DNS-SD service in the network. Its constructor allows for easy creation of such service instances without needing to know the full DNS name of the service.

The *BListener* interface handles two types of events in the network: device entry (or update) and device exit. Device arrivals are treated the same as device updates, since this

simplifies the event handling without sacrificing any functionality. The two functions that are handled by this event are *BListener.serviceUpdated()* and *BListener.serviceExited()*, both of which return a *Bdevice* object.

The *Bdevice* object corresponds to a device in the network offering the service requested, and exposes properties of the device, such as its host name, host address, service name offered as well as the metadata (key-value pairs) of the device.

While the *Bdevice* maps to a physical device, it actually corresponds to a particular service type that is advertised by the physical device. For instance, a physical device would return two *Bdevice* objects if it was offering two services of the same type. Similarly, a physical device would return a unique instance of a *Bdevice* for each service type it was offering.

A developer who needs to offer an instance of, or view instances of, a service type in the network, would create a *BService* object. After creating this object, he would do either or both of the following operations:

- Register a service with *BService.register()*
- Listen to network events for the service type with *BService.setListener*

The *BService.setListener*, which takes an object that implements a *BListener* interface, associates the network events with the implementation's function calls.

Using the BonAHA API, we have developed software applications such as an automatic location finder, and a simple networked Tic-Tac-Toe game that uses only BonAHA to update game status. These sample applications are described in the next section.

4.5 Sample Applications

In this section, I present two simple sample applications that showcase the functionality present in the BonAHA API.

4.5.1 Location Finder

The LocationFinder application is a simple command line application. The presumed scenario for this sort of application is where a device does not have access to GPS data and only

has access to the location information of nearby nodes. Using this information, it updates its own location by averaging the location of all the other nodes in the neighborhood.

We have written the Location Finder application using the Bonjour API, and for equivalent functionality, the Bonjour code required twice as many lines as code as the BonAHA code.

Some of the basic code for running the Location Finder application can be found in Program 1.

4.5.2 Tic Tac Toe

While multiplayer games like Tic-Tac-Toe are rather easy to write using sockets and other forms of network programming, our BonAHA framework exposes an entirely new way of writing such multiplayer games.

We will briefly explain how the multiplayer functionality of the game works. First, a *BService* object is created and *register()* is called to announce its availability on the network. A listener interface is also attached to listen to network events.

Upon receiving a *serviceUpdated()* event, the event handler code first processes which node the event update is from. Next, it processes the metadata from the incoming node and maps it to an internal data structure representing the location of the players' moves, which is used to update the game.

Some of the networking code used to create the networked Tic-Tac-Toe application can be found in Program 2. A screenshot of this program is shown in 4.4.

A traditional multiplayer Tic-Tac-Toe or other networked game would require the developer to write client and server sockets and process data packets. With the BonAHA framework, the developer is able to implement the networking functionality in four lines for the Tic-Tac-Toe game.

The BonAHA framework enables the developer to handle nodes entering and leaving the network. Our Tic-Tac-Toe game is able to automatically terminate a game when a user leaves, and wait for and detect when another user wants to join. This feature was added with just three lines of code.

Program 1 The program for implementing location updates using the BonAHA API. Only the code using the BonAHA API for service announcement and updates is shown.

```
public LocationFinder() {
    // Create the BonAHA service and set metadata
    service = new BService('7ds.location', 'tcp');
    // Set my location
    service.set('Latitude', lat);
    service.set('Longitude', lon);
    service.register(); // Register myself
    // Listen for new nodes on the network
    service.setListener(this);
}

/* Another node enters or updates its location */
public void serviceUpdated(BNode n) {
    System.out.print('Updates from:' + n.getHostName());
    // Get the node's location metadata
    String nodeLat = n.get('Latitude');
    String nodeLong = n.get('Longitude');
    // Process peer's information
}

/* When a node leaves the network */
public void serviceExited(BNode n) {
    System.out.println(n.getHostName() + ' (' +
        n.getHostAddress() + ') ' + 'left the system');
}
```

Program 2 The code for getting the list of values from neighboring nodes for the Tic-Tac-Toe game.

```
/* When I make a move */
public void mouseReleased(MouseEvent e) {
    // Get the location where user wants to move
    int col = (e.getX() * 3) / getSize().width;
    int row = (e.getY() * 3) / getSize().height;

    // Update my node's metadata to reflect state.
    // This is announced on the network
    service.set(col+"",row, col+"",row);
}

/* New player; or other player made a move */
public void serviceUpdated(BNode n) {
    // Check which player made a move.
    // Then get node values (player positions)
    String[] values = n.getValues();

    // Update internal data structure

    // Repaint the game
    this.repaint();
}

/* Player has left */
public void serviceExited(BNode n) {
    // Display message; wait for new player
}
```

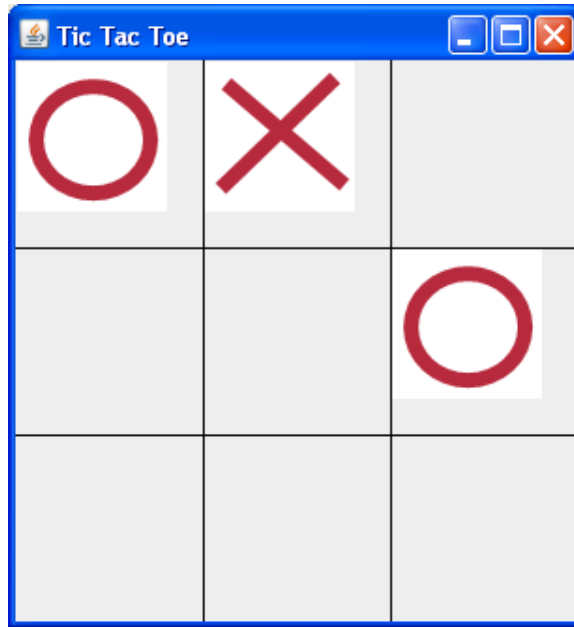


Figure 4.4: A screenshot of the Tic-Tac-Toe game developed using BonAHA. While it looks similar to any regular networked two-player Tic-Tac-Toe game, it is much simpler to develop due to the BonAHA framework.

4.6 Scalability

While building an application that runs in opportunistic networks, software architects need to consider scalability. For example, such applications may be used in the middle of a transportation hub such as Penn Station in central Manhattan, with thousands of users in close proximity at rush hour, and quite possibly, hundreds within the physical proximity of the wireless opportunistic network close to one user. As described in Chapter 5, the underlying Bonjour service discovery library is capable of handling discovery packets from hundreds of users simultaneously, and it is quite unlikely that even in spaces with dense user populations, more than hundreds of users are on the same opportunistic network it is more likely that there will be several network islands identified by their own unique ad-hoc wireless network names, each with their own opportunistic network that users can join and leave.

4.7 Security and Privacy

BonAHA aims to provide functionality that allows discovering nearby peers and exchanging metadata about the state of the node that would allow devices to further communicate with each other. BonAHA itself does not provide any cryptographic functionality, and it is up to the application to provide end-to-end or authentication encryption for data transmission if necessary.

BonAHA doesn't share any private information other than network connectivity information (such as IP addresses) that devices need to communicate with each other. All other metadata is determined by the application, and the application can choose which data is public and can be seen by all peers, and which data is private and needs to be encrypted before sharing.

4.8 Related Work

JXTA [Sun Microsystems, 2001] is a peer-to-peer framework implementation made by Sun Microsystems for the Java platform. JXTA is a very powerful framework and has been implemented in J2ME for Java-based mobile platforms. However, JXTA does not have the necessary framework to handle network events, such as devices joining or leaving the network, which is necessary for our class of applications. In addition, the JXTA protocols appear to be more suited to maintaining distributed metadata in always-connected networks and hence more suitable for heavyweight applications such as file-sharing programs.

Peer2Me [Wang *et al.*, 2007] is an implementation of a mobile ad-hoc framework using JXTA. It appears to overcome the problem of network discovery using a platform-specific network library for detecting Bluetooth network changes. However, it is currently limited to J2ME devices and devices that use Bluetooth.

The only framework that is so far closely comparable to our work is the LightPeers framework [Bent Guldberg Christensen, 2007]. LightPeers is a library framework written in Mono, and runs on Windows Mobile, Windows and Mac OS platforms. LightPeers enables development of mobile peer-to-peer and opportunistic network applications. There are some minor differences, in that LightPeers' discovery mechanism uses an announcement packet

every second to check for nearby devices, while our mDNS-based platform uses exponential backoff to reduce the number of packets over time. Further, the BonAHA API allows developers to get and set metadata associated with each device, which allows the developer to treat the entire network as a collection of objects and network events. LightPeers does not have such a feature. Further, LightPeers is built on its own custom network stack, rather than using service discovery standards such as mDNS.

AllJoyn [Alliance, 2016] is a framework developed by the AllSeen Alliance that allows for discovery, attachment and data sharing among devices in close proximity. This framework was first announced at the Mobile World Congress in 2011 by Qualcomm. It runs on the most popular desktop and mobile operating systems, and supports Wi-Fi, wired Ethernet and Bluetooth. Instead of using existing service discovery protocols, AllJoyn uses a slightly modified version of the D-Bus messaging bus service that is used for inter-process communication in Linux through sockets, and extends this to function across multiple devices for service discovery [Center, 2011]. AllJoyn allows nodes to announce the services they provide, and connect and share data through sessions, which could be point-to-point (node-to-node) or multi-point (group of nodes). Its API tutorial [Alliance, 2015] for a sample Hello World application shows that the general API structure is very similar to BonAHAs concept of discovery and metadata and information sharing. Using AllJoyn, Lokhandwala et al. [Lokhandwala *et al.*, 2015] developed an app to allow for users in a local network to edit documents through a Min-O-Mee (minutes of meeting) app. The Min-O-Mee application was specifically developed for sharing and writing meeting minutes over a shared connection, but other applications can be built on the AllJoyn framework.

OpenPeer [OpenPeer, 2014], developed in 2013, is an open P2P signaling protocol that is open and allows for peer communication. It allows applications to be built as web applications that run on web browsers, or as standalone applications. In addition to the discovery mechanisms, it also supports WebRTC [W3C, 2011], which is a protocol for sharing multimedia content (mostly for voice and video chat) in a peer-to-peer manner without requiring a central server. Based on the overview, protocol specification and sample code [GitHub, 2014], OpenPeer is built as an enabling mechanism specifically to support WebRTC and audio and video calls across a network, rather than enabling generic P2P applications to

exchange data for other purposes like BonAHA is meant to be.

Apple introduced a library called GameKit [Apple, 2009] for iOS in 2009 that allows developers to create their own social games. It allows for a mode of connectivity called peer-to-peer connectivity, which allows your game to create an ad hoc Bluetooth or wireless network between multiple iPhones in the same local area. The documentation also states that even though meant for games, the functionality can also be used for other forms of data transfer. However, this functionality was limited only to running on iOS devices and is closed source. An additional impediment is lack of published protocol specifications or other documentation for GameKit, which would mean that we would have to reverse engineer and look into network packets to reconstruct the protocol, which would make it difficult to port and run on other platforms. The GameKit library was removed in iOS version 7 (2013) [Stackoverflow, 2013].

Funai et al [Funai *et al.*, 2016] explore connecting devices through the Wi-Fi Direct standard mentioned earlier. While Wi-Fi Direct was designed following a client-server hierarchical architecture, where a single device manages all the communications within a group of devices, Funai et al. propose other solutions for supporting the communications between multiple Wi-Fi Direct groups and create multi-hop ad hoc networks. While this paper explores how to extend WiFi Direct beyond the initial range that it was meant for, it does not provide for an additional or easier-to-use interface on top of WiFi Direct for building applications that run in such networks.

4.9 Conclusion

We believe that the BonAHA framework provides a promising start for easing development of highly mobile opportunistic network applications. The appendix listing some of the BonAHA applications we have developed, such as chat, bulletin board system, etc shows that it is possible to build real, functional applications on top of the BonAHA framework.

Chapter 5

Measuring and Improving Service Discovery Protocols on Wireless Networks

5.1 Measurements of Service Discovery Performance in Wireless Networks

In this section, I describe my joint work on the measurement and analysis of the popular Zero Configuration Networking protocol [Zeroconf Working Group, 2008] in wireless networks. We found that ZeroConf fails to work properly in wireless networks due to its exponential backoff feature and inability to handle wireless node transitions. We came up with an algorithm that allows ZeroConf to function properly in wireless networks.

Since service discovery protocols, particularly those based on multicast, could easily create a large amount of background traffic in wireless networks, it is necessary to evaluate the impact of these protocols in such networks. I describe our measurement of the use and prevalence of multicast service discovery protocols on a large campus network, Columbia University's wireless network.

In the previous two chapters, we presented applications and frameworks that run on opportunistic networks. In this chapter, we present our analysis of and improvement on the

ZeroConf protocol implementation in wireless networks, but this analysis and improvement applies to opportunistic networks as well, as the transitions of devices in 802.11 wireless networks is the same as transitions in opportunistic networks.

The service discovery work was joint work done with my colleague Se Gi Hong.

5.2 Improving Service Discovery in Wireless Networks

Service discovery is a vital part of applications that run on wireless and opportunistic networks, as it allows those applications to automatically discover services as well as announce their services. When mobile devices are intermittently connected to the Internet, information can be shared with peers using ad-hoc networking. These wireless mobile ad-hoc networks do not have any infrastructure, such as a DNS server and a DHCP server. For these reasons, IP-based mobile ad-hoc networks require protocols such as Zero Configuration Networking (Zeroconf) [Zeroconf Working Group, 2008]. Zeroconf provides for the assignment of IP addresses, host naming, and service discovery without any central servers or human administration. Therefore, Zeroconf plays an important role in order for such applications to work properly.

We analyze the service loss rate, which we define as the percentage loss rate of network service announcements (using ZeroConf packets).

Figure 5.1 show our analysis of the service loss rate for different intervals and various device residence times in a wireless network [Hong *et al.*, 2007]. From the figure, we can see that Zeroconf suffers from a large service loss rate in wireless networks, where the frequency of devices joining and leaving a local wireless network is very high. Network changes are not announced to other devices, and there is no algorithm in Zeroconf to detect these frequent network membership changes as nodes enter and exit the network.

In our work, we analyzed the relationship between the interval of service browsing, average residence time of devices in a local wireless network, and the probability that new services announced by new joining peers are not discovered. We then proposed a new algorithm that improves the service discovery protocol and allows devices to discover network changes and new services in real time while minimizing network overhead. In our

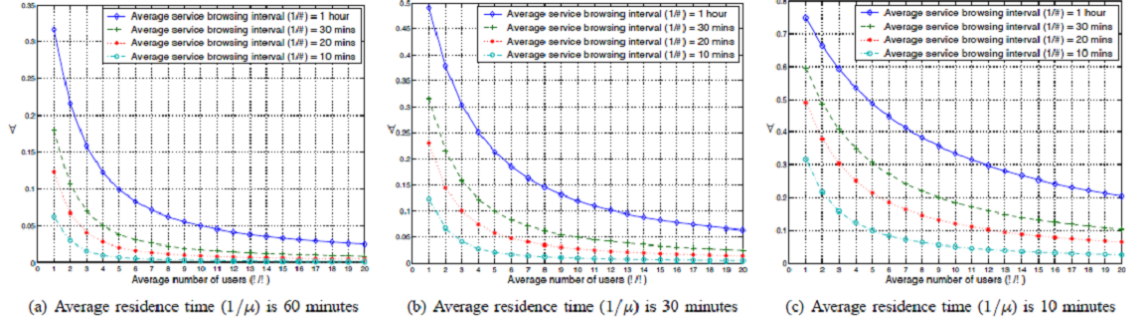


Figure 5.1: Service loss rate for different residence times and browsing intervals for ZeroConf in opportunistic networks.

algorithm, each device can detect whether it has joined a new network group by monitoring changes in its wireless network ID as well as beacon frames. If a node detects that it has joined a new network, it resolves possible conflicts of IP address and host names, and announces or browses for services.

Our implementation [Hong *et al.*, 2007] monitors network changes that occur when a wireless device joins a new wireless network. This enables our networking stack to notify ZeroConf of node changes that it may not otherwise be aware of.

5.3 Multicast Service Discovery in a Campus-Wide Wireless Network

iTunes, a highly popular multimedia application, uses the DNS-based service discovery (DNS-SD) [Cheshire and Krochmal, 2013a] and multicast DNS [Cheshire and Krochmal, 2013b] protocols to allow users to browse playlists of other iTunes users in the same subnet. mDNS generates significant traffic load. Such load is especially seen on college campuses, where wireless networks are pervasive and a large number of wireless users are working on the same subnet.

Since we could not find any previous measurement and analysis of the overhead of multicast service discovery traffic in a campus wireless network, we measured and analyzed the traffic overhead of mDNS traffic [Hong *et al.*, 2009] in a typical college wireless network.

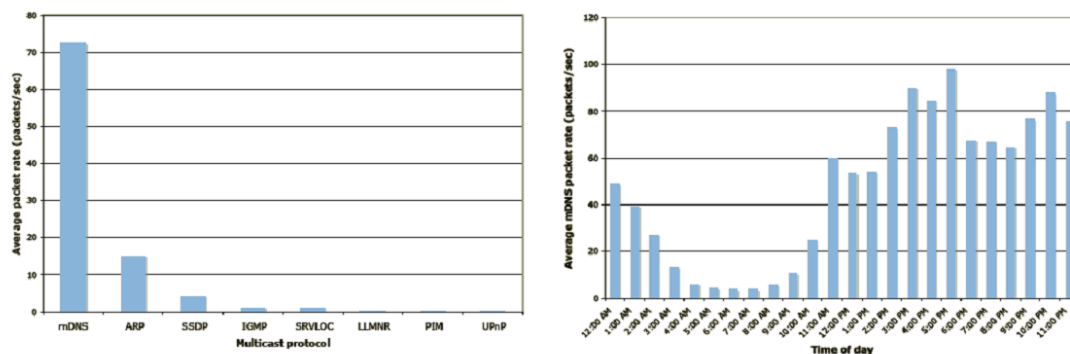


Figure 5.2: Left: the number of packets per second for the various multicast and broadcast packets in the campus-wide wireless network. Right: the average hourly rate of packets per second sent over a period of 24 hours.

In Figure 5.2, we show the bandwidth usage of mDNS packets and the effect of multiple APs on multicast packets in the wireless network of Columbia University. The left graph of Figure 5.2 shows the number of packets per second for the various multicast packets, separated by type. The graph on the right shows the number of mDNS packets per second throughout the day. The graphs show that mDNS is the multicast protocol that is most widely used, and that the traffic patterns for mDNS are consistent with usage patterns for college students (usage peaks at afternoons and evenings and drops late at night).

In addition to our measurement of these protocols in a campus network, we also modeled the behavior of this protocol. We defined three service discovery models which correspond most closely to the DNS-SD/mDNS traffic behavior. We analyzed the performance of these three service discovery models in terms of traffic overhead and service discovery delay for different network sizes and lifetimes.

We did this work primarily because we noticed an increasing number of service discovery messages in computer networks, particularly mDNS messages due to the increasing popularity of iTunes and similar services. Our work, done in 2008, corroborated that service discovery and announcement packets are indeed growing in number (particularly compared to earlier publications on the same subject) particularly on college campuses.

Part II

Delivering Video Content—CDNs and Beyond

Chapter 6

Introduction: Content Delivery Networks and Content Networking

Internet video has become a cornerpiece of content consumed on the Internet today. By 2012, Youtube was serving up 4 billion streams a day [Reuters, 2012]. In April 2015, Facebook said its social media site was serving 4 billion views every day [Fortune, 2015]. yet accounts for only 10% of the Internet traffic today [PCMagazine, 2011]. In contrast, NetFlix traffic constitutes about 33% of the Internet traffic [PCMagazine, 2011], and is in the midst of developing its own custom content delivery network (CDN) [Netflix, 2012]. Cisco’s 2015 Visual Networking Index predicts that Internet traffic is expected to reach 2 zettabytes per year by 2019 (up from 1 zettabyte at the end of 2016), of which 80% will be Internet video, up from 67 percent in 2014 [Cisco, 2015].

It has never been more important to study the trends in Internet video, in particular how content delivery networks (CDNs) such as Akamai and others operate, as video and video delivery platforms have now become the largest drivers of Internet traffic. In this light, this thesis includes research performed in relation to CDNs and improving the networking and delivery of content on such networks.

Chapter 7 explores on-path CDNs, a mechanism to terminate content requests in-network and serve content more efficiently through a caching node in the network.

Chapter 8 briefly touches upon the NetServ service virtualization framework, which

allows us to build in-network application modules such as ActiveCDN. Chapter 9 details ActiveCDN, a CDN platform built on top of the NetServ framework that allows for dynamic CDN modules to be installed on the fly at edge nodes to provide not only caching, but also dynamically generated and localized content.

Chapter 10 describes the analysis of video traffic in the real world which I performed while working at Longtail Video, which drives 500 TB of video traffic per month across various online properties. I believe this analysis will be helpful in evaluating trends in networked video traffic as well as efficient caching methods for the same.

Chapter 7

Unveiling the Content-Centric Features of TCP

7.1 Introduction

Content-centric networks [Jacobson *et al.*, 2009b] have been proposed as a new network paradigm that is centered around the distribution of content, which we can define as any reproducible object that end users would like to download in order to view, hear or execute. A key idea of content-centric networks is to query content by content name rather than by host name and to enable any node inside the network to respond to content requests rather than just a few endpoints. Most proposals for content-centric networks require a “clean slate” approach and a replacement of today’s TCP/IP protocol stack, which raises questions about a feasible deployment path.

In this chapter, we ask the question to which extent the ideas of content-centric networks can be realized on top of today’s IP protocol suite, and propose an approach for name-based addressing that extends today’s TCP/IP protocols in a fully standard compliant way. We implement our new method in order to demonstrate its feasibility and evaluate the performance of the system using both latency and processing overhead as measures. The obtained results, detailed in Section 7.5 demonstrate that name-based addressing on IP is feasible.

In content-centric networks, any network node can be enabled to respond to a request if

the node holds the requested content item. Current proposals for content-centric networks require a departure from today's IP protocol stack. However, replacing today's IP protocol stack with something new requires substantial investment in network infrastructure and end systems. Given the very slow uptake of the next version of the IP protocol, IPv6, it is questionable when and if at all a radically new protocol stack can see widespread deployment.

In this chapter, we question the common belief that content-centric networks require a radical departure from the current IP protocol suite. We explore to which extent content-centric networks can be realized on top of IPv6. We propose an approach for a name-based content network, On-Path CDN, that extends TCP/IP. Our approach enables any node on-route between the end user and the content provider to serve requested content.

We have implemented our new method in order to demonstrate its feasibility and evaluate the performance of the system using both latency and processing overhead as measures. The obtained results demonstrate that it is possible to realize a name-based addressing mechanism on TCP/IP. We also show that On-Path CDN can enhance end-user experience for watching audio or video over the Internet.

We detail the operations of traditional and on-path CDNs in Section 7.2. We explain our mechanism and the implementation of a prototype system in Section 7.3 and Section 7.4, respectively. In Section 7.5, we evaluate the performance of our system using network latency and processing overhead as metrics. We discuss related work in Section 7.7.

7.2 CDNs for Content-Centric Networking

7.2.1 Traditional Approach

We first contrast our work on On-Path CDNs to existing methods of content delivery by highlighting two issues.

The first limitation that we address relates to the method of content delivery itself. Today, content providers either host their high-bandwidth content themselves, or more commonly pay content delivery network (CDN) providers such as Akamai and Limelight Networks for the delivery of their content. When content is hosted on a CDN, a user

request for it is usually redirected to a server closer to the user that is operated by the CDN service provider. Though this fundamental task appears to naturally fall into the space of the networking layer, the host-centric Internet architecture was not designed with such a service.

This has led to the development of application-specific and non-interoperable mechanisms, the most common of which are: redirection through domain name resolution using DNS [Mockapetris, 1987a][Mockapetris, 1987c]; request redirection using HTTP [Fielding *et al.*, 1999]; and other application-level mechanism, e.g., based on HTML rewriting or distributed hash tables (DHTs) (specific implementations of these are cited in more detail in Section 7.7). The most popular of these implementations in the real world is the HTTP and DNS based approach that was popularized by Akamai [Tom Leighton, 2009] and now used by other CDN vendors.

The above redirection work-arounds require some form of command-and-control mechanism and impede cooperation of CDNs operated by different parties; e.g., to allow a national or international backbone CDN to reach into a metro network and make use of the CDN resources of the local operator. Once the redirection is set up, a local CDN will be unable to serve the content from another node even if it is closer to the end user than the node that the user has been redirected to. As a consequence, CDN deployment is rather static and scaling them to adapt to sudden changes, such as unexpected flash crowds, is difficult with the current Internet architecture. Even using services such as Amazon Cloudfront [Amazon, a] or its Amazon Web Services cloud architecture, it is only possible to stand up nodes in the handful of geographical regions that Amazon operates data centers at.

The second issue is in regards to the networking architecture and naming perspective. Currently, requests for content are usually routed based on the Internet address (IP address) of the node that has the content. While this is in keeping with the current Internet architecture, it does not offer a direct way of addressing content, which is independent of the location of the content itself. Methods of addressing naming issues, such as Content-Centric Networking [Jacobson *et al.*, 2009b] as well as systems like DONA [Koponen *et al.*, 2007] and i3 [Kannan *et al.*, 2004] require a clean-slate redesign of the Internet architecture in order to be useful. While it is possible to run some of the solutions on top of existing

protocols such as IP, such a solution would defeat its purpose since it would overlay a pure content-centric solution on top of something that relies on host names.

Our work presents an implementation of content-centric networking that runs on today's Internet technologies and protocols. In this chapter, we raise and answer the question of how far we can go in the direction of CCN based on today's IP protocol suite, with our additions being standards compliant. We propose a design for an IP compliant CCN architecture, present a prototype implementation, discuss limitations and performance as well as unexpected road-blocks in the implementation.

7.2.2 On-path Content Delivery

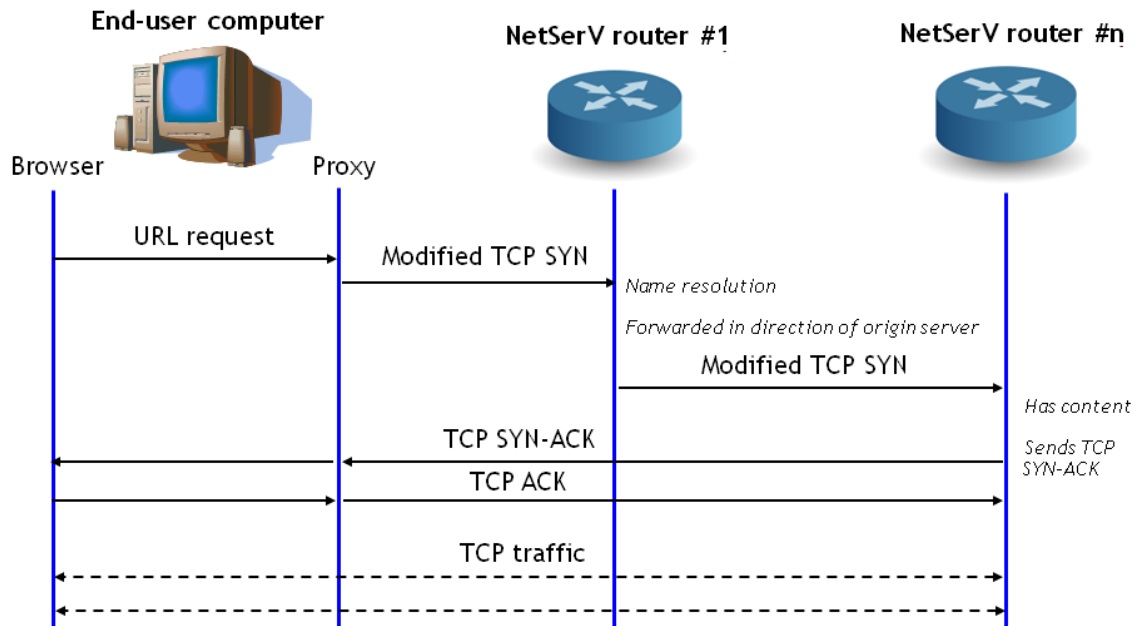


Figure 7.1: The handshakes and networking messages used in the TCP-interception method of on-path CDNs.

We propose an alternative method for the delivery of multimedia content that enables delivering multimedia content from any node that is on the route from the end user to the content provider. In our approach, any intermediate node can respond to content requests and serve content if it has a copy of the content cached. This avoids explicit redirection to another server and reduces latency, thereby improving the end user's multimedia experience.

In addition, our implementation and solution also provides a method of serving content to the users without redirecting them to a particular node. Our on-path content delivery mechanism enables true content-based delivery without the network having to worry about which nodes the content resides on. It makes use of signaling messages that piggyback on existing TCP handshake mechanisms, as described in the next section.

7.2.3 Advantages of On-Path Content Delivery

There are many advantages in allowing multimedia content to be served using intermediate nodes on the path from an end user to a content provider, in contrast to using a statically deployed CDN network, or worse yet, serving content from just one central location. Some of the advantages are:

1. Network latency is reduced and there is less congestion on the Internet. Especially as video traffic is predicted to grow and account for 90% of the Internet traffic in 2013 [John Markoff, 2010], having video served from nodes that are closer to the end user will dramatically reduce congestion in the network.
2. Regional networks and the Internet dynamically adjust to high loads of unanticipated traffic, e.g., such as in the case of “flash crowds” or highly viral content.
3. Service providers can reduce redundant traffic on expensive transit links.
4. CDN service providers can establish business relations with each other to provide better end-to-end multimedia experience to a larger user population.

We describe the design decision and details of the core mechanism of our solution, which enables these advantages, in the following section.

7.3 Mechanism Details

Non-realtime video on the Internet today is delivered almost exclusively using HTTP, with TCP as its underlying transport protocol. Since the goal of our work is to enable a solution over today’s Internet without any modifications to the end user and networking stack, we focus on a solution that is compliant with and requires no changes to the TCP specification [Postel, 1981].

At the beginning of a HTTP session the user application establishes a TCP connection to either the host indicated in the URL [Berners-Lee *et al.*, 1994] that identifies the requested content or to a proxy if configured. Once the TCP session has been established, the user application then uses the URL to issue a HTTP request identifying the content. Thus, intercepting a content request on path for inspection and content routing decision requires proxying the TCP session at the intercepting node. Since an established TCP connection cannot be renegotiated and transferred, however, on-path routing without additional mechanisms could easily result in daisy-chaining TCP connections for a single user session and result in additional hops and delays for the end user.

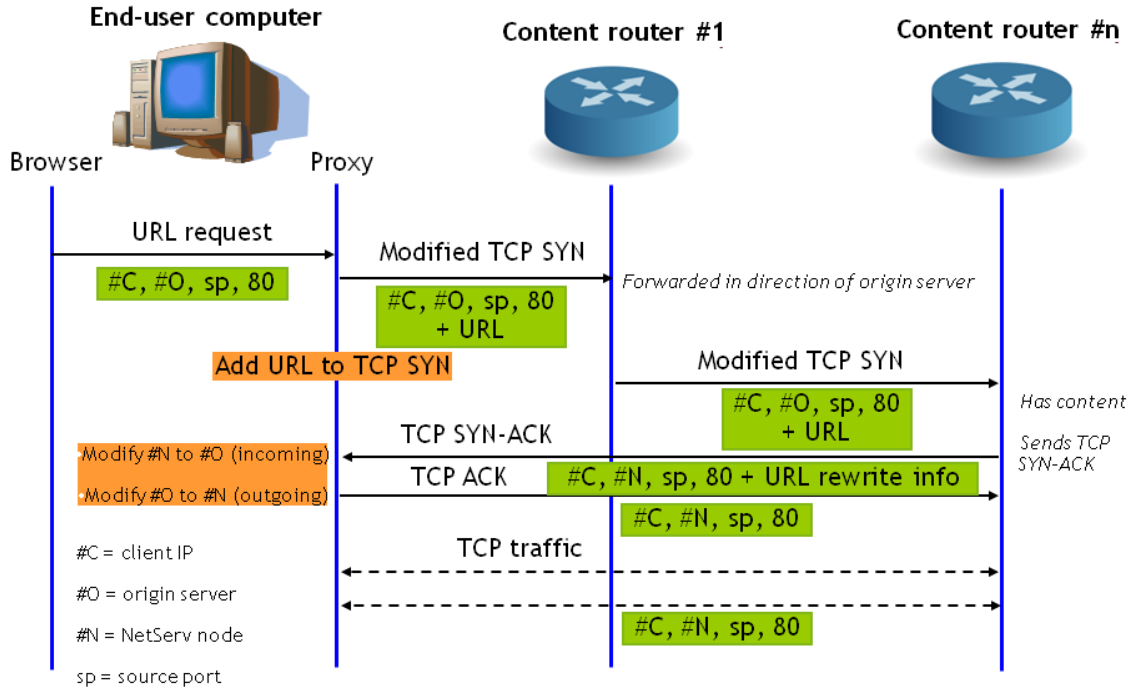


Figure 7.2: The details of implementing the handshakes and networking messages in the TCP-interception method.

Session establishment in TCP is signaled using the same packet format as for the actual transfer of the application data; i.e., a TCP packet consists of a header and a payload data. Our method of on-path content delivery exploits the fact that the TCP packets for the initial handshake of a session don't include any payload. We piggyback some key

information in two of three TCP handshake messages, particularly the TCP SYN and the TCP SYN-ACK packets. Though this would be possible by modifying the protocol stack at the user end, we enable this through use of a modified HTTP proxy that rewrites the TCP handshake messages as outlined in Section 7.2. In the proxy scenario, the application (e.g., the browser) establishes a session with the proxy server, which rewrites the TCP handshake messages to enable the on-path delivery mechanism. Thus, there is no need to rewrite the networking stack at the end user’s node, and at the same time, the user is able to benefit from an “opt-in” option to make use of this feature.

Our method uses the first TCP SYN message from the client to the content provider for carrying the pointer to the requested content (such as the URL) in its payload. This message moves through the network in the direction of the content provider. It passes through all intermediate nodes that are not able to serve the content (or are ignoring TCP SYN payloads). However, once an intermediate node detects it has the content referenced in this packet, it terminates the request forwarding process and replies to the TCP SYN packet with a TCP SYN-ACK packet.

The TCP SYN-ACK packet sent by the intermediate node contains a payload consisting of the initial content pointer (the URL), along with a delimiter and an identifier of the node that has intercepted the request. This allows the networking stack in the proxy server that is serving the end user to realize that an intermediate node is responding to the initial content request.

When the client proxy receives the TCP SYN-ACK message, it replies with an ACK to the intermediate node, thus completing the TCP handshake. At this point, a TCP session is set up between the proxy and the intermediate node that can serve the content directly, and all future content requests are served directly from the intermediate node.

We built a prototype of this system, which is described in the next section.

7.4 Prototype Implementation

Our prototype implements the previously described the TCP handshake interception. Furthermore, a module in the prototype evaluates each handshake message to determine

whether it is able to handle and serve the content that is referenced in the TCP SYN message, and only then does it decide whether to respond to it with a payload-added SYN-ACK message.

For our implementation we used iptables [iptables, 2008] and netfilter [nfqueue, 2008] to set up rules to intercept our required packets. We also use the library libnetfilter_queue (nfqueue) [nfqueue, 2008] to allow us to programatically set up event callbacks to network events. We intercepted requests for packets on both port 80 for HTTP and port 3128 for Squid proxy caching.

We implemented our prototype using the Python programming language and relevant network libraries for intercepting and modifying packets. We used the nfqueue-bindings [nfqueue, 2008] and the Scapy [scapy, 2009] libraries for advanced networking and packet functionality. Our Python scripts implemented the functionality required for the protocol as described in the previous section.

We ran our tests on Alcatel-Lucent (ALU) networks, and hence ALU is considered the local network. We use a HTTP proxy to communicate with nodes outside the Alcatel-Lucent network.

Our setup consisted of:

- 1) A client with a browser set up to use our on-path CDN proxy to access the network
- 2) Our CDN proxy, which handles the initial TCP SYN packet and also performs the modification of network addresses in a manner similar to a NAT.
- 3) Our CDN caching node, which intercepts the TCP SYN signaling message, responds with a TCP SYN-ACK with the node information payload, and serves the cached content from its local storage.
- 4) The origin server, which is the original source for the requested content.

Of the above, we implemented the CDN proxy and the CDN caching node. Both of these were implemented in the Python programming language using the networking libraries listed.

We encountered an interesting problem while working with the implementation of the network stacks that bears mentioning. In order to intercept and serve content based on request, we added payloads to the TCP SYN and SYN-ACK packets, and correctly changed

the size and the checksum on these packets. To our surprise, the TCP SYN and SYN-ACK packets were received and processed correctly by the network stack at the sending and receiving nodes, but the subsequent data packets were not handled properly, and we saw TCP RESET messages terminating the session.

We found that the reason this was happening was that the network stacks were assuming (incorrectly) that the TCP handshake messages had no payload and were expecting corresponding SEQ and ACK counters. In other words, the operating systems' network stack implementations completely neglected the data size and SEQ/ACK numbers set in the TCP handshake messages and started the SEQ/ACK numbers for the data packets assuming a zero payload. This caused some unexpected problems in the running of our protocol.

In order to resolve this SEQ/ACK problem, our packet processing had to change the SEQ and ACK numbers in the TCP packets, reducing or incrementing them as necessary to allow the network stack to recognize them as legitimate packets.

7.5 Performance Evaluation

For our performance measurements, we measured the network latency using the round-trip-time delay from one control node to four content servers that we worked with during the course of our experiments.

7.5.1 Experimental Setup

We used one node in the Alcatel-Lucent network as the control node. We measured the round-trip time of TCP packets to the following nodes:

1. A node on the same subnet;
2. A node on the edge of the network: Alcatel-Lucent's main web server (www.alcatel-lucent.com);
3. Cable New Network (CNN's) main web server (www.cnn.com); and
4. Akamai's content server for CNN (ht.cdn.turner.com).

7.5.2 Round-trip Time and Latency

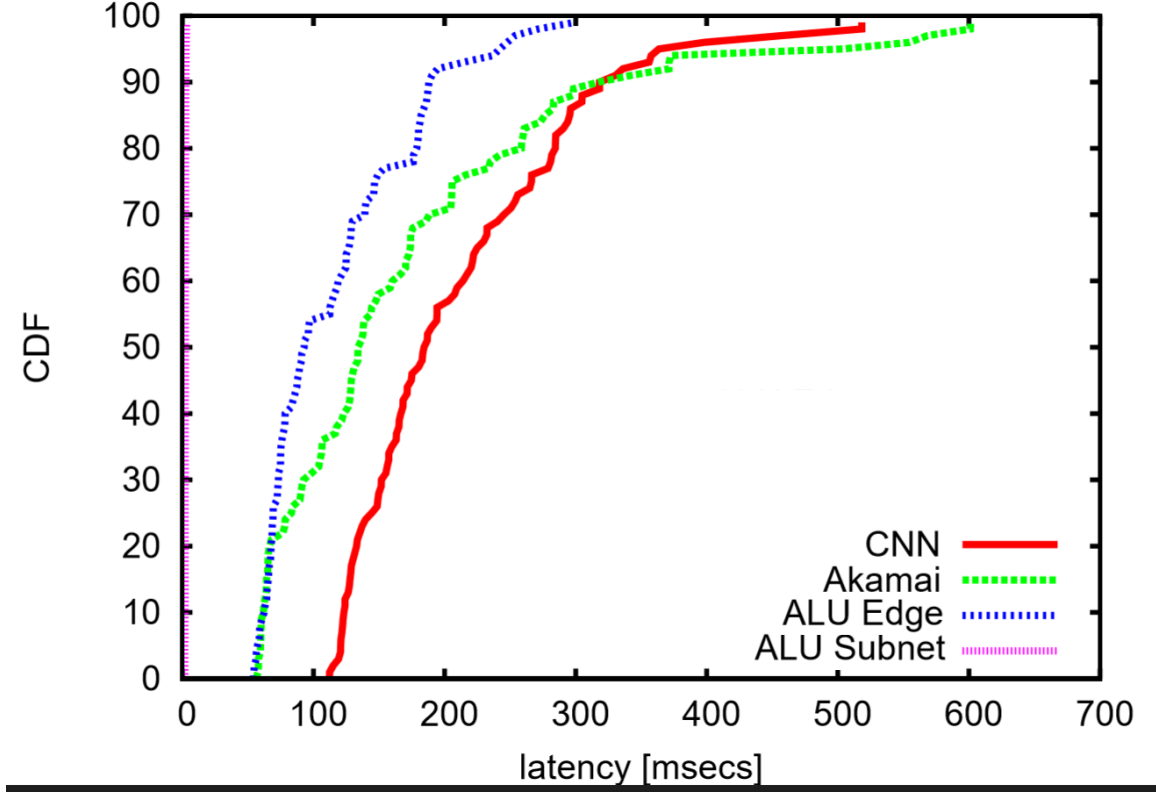


Figure 7.3: TCP latency of the four test nodes.

We used `apachebench` [ApacheBench, 2008] to test the delays. The `ab` binary (part of the `apachebench` package) allows us to measure round-trip time delays for TCP signaling messages (SYN and SYN-ACK) which we use to measure the delay in sending and receiving TCP messages. We used Python’s `scapy` [scapy, 2009] network package for generating the network graph, as well as validating the results from `apachebench`.

The results presented in Figure 7.3 show that the round-trip times for the CNN and Akamai servers are very close to each other. We believe that this may be because our tests were conducted in the U.S., and both servers were sufficiently close enough in terms of network topology to respond to requests with latency close to one another. The interesting result is the difference in the delay between the edge node (the Alcatel-Lucent web server) and the CNN and Akamai servers. Our measurements indicate that latency to the edge

node is only around 50% of the latency for CNN and Akamai; and latency for the edge node is also far more consistent than the latency to the origin servers, whether CNN or Akamai's servers.

The above results show that an on-path CDN with deployed nodes at the edge of the network can indeed substantially reduce network latency. Furthermore, we conclude that it can reduce network congestion and other networking problems associated with large volumes of multimedia traffic on the Internet.

7.5.3 Processing Overhead

Another important performance metric is the processing overhead that our mechanism adds to the network nodes. As the results of our performance measurements indicate, this overhead is very small. The typical overhead for simply intercepting and checking the content of the TCP handshake messages is on the order of 400 to 500 microseconds, and the overhead of interception and modification of the TCP messages is on the order of 600 to 700 microseconds. Hence, the overhead added to the interception is in the range of 40% to 50% of the base delay. We could expect far lower overhead if we had used C/C++ instead of Python.

It is important to note that the interception is performed only on TCP handshake messages, particularly only on TCP SYN messages passing through the content router. Furthermore, the payload addition is done only for content that the router is able to handle. And finally, our prototype was built using the Python scripting language and an iptables/netfilter implementation that sent packets from kernel space to user space. With an implementation in a kernel module, the speed of processing packets could be significantly improved.

7.6 Implementation Alternatives

There are several alternate ways of implementing our on-path method for serving content, such as by using UDP signaling or NetServ (described in Chapter 8 of this thesis). We believe that being able to intercept and serve content in response to TCP message signalling

is the best way to handle content delivery on network flows that are being set up. In addition, such methods work on an “off-path” channel or as control messages, and we believe the TCP intercept mechanism allows for true content-centric networking to be implemented on today’s Internet architecture.

7.7 Related Work

There has been work on intercepting content requests to serve multimedia content from nodes on path, such as redirecting requests based on predetermined criteria [John D. W. Brothers / Nortel Networks, 2002]. Layer 4/7 switches [Web Switch, 2011], also known as web or content switches perform similar functionality and are used primarily for caching repeatedly requested content. In contrast to our solution, however, they need to keep track of every TCP connection that they are routing. This is impractical in a large network.

Sarolahti, Ott and others [Sarolahti *et al.*, 2011] explore providing content-centric networking (CCNs) on top of TCP, with some modification of TCP required to work. In particular, they state that “Our solution, Multi-Receiver TCP (MRTCP), is based on TCP as the primary transport protocol for our target application class. It requires a modest amount of modifications to sending (usually server) TCP/IP stack, but it works with unmodified off-the-self TCP receivers (usually clients).” Their implementation is very similar to ours, in that they maintain awareness of content and flow in their modified TCP (MRTCP) implementation, and also provide for improved performance when intermediate routers are MRTCP aware. However, one major difference is that they require modification of the TCP stack on one side of the connection (server side), while our solution does not require any changes to the TCP or IP layer at the end points, but only on routers at the network core.

TCP interception has been enabled by Cisco on its routers. While this seems to have been initially introduced to allow systems to reduce TCP SYN-flooding attacks, such as NetBouncer [Thomas *et al.*, 2003], it has also been used for a variety of other purposes, such as speeding up content delivery on wireless networks [Housel *et al.*, 2004] [Kopparty *et al.*, 2002]. While we were not able to find related work using this feature for content delivery, we believe this feature allows for our method to be implemented on many commercial, off-

the-shelf routers as long as content storage is available on these routers.

There is also some work done on QoS routing for multimedia applications [Wang and Crowcroft, 1996] and multipath routing for unicast video [Chen and Chan, 2001], but these papers on multimedia routing do not involve intercepting content packets.

There has been a body of work in the fields of naming and redirection. For example, i3 [Kannan *et al.*, 2004] and OCALA [Joseph *et al.*, 2006] are naming overlays that work on top of the existing Internet architectures. The Data-Oriented Network Architecture (DONA) [Koponen *et al.*, 2007] attempts to address the issue of Internet naming and name resolution by allowing a client to request content by name, rather than using a host address.

In contrast to these naming and addressing approaches, Content-Centric Networking (CCN) [Jacobson *et al.*, 2009b] aims at treating content as a primitive for routing requests to the destination. However, most of the robust work on naming, addressing and content based routing require a clean-slate redesign of the Internet. We believe ours is the first example of work that enables use of signaling for delivery of multimedia content without requiring a complete overhaul of Internet protocols. Our solution avoids the routing problem that CCNx suffers from, while allowing for the content location flexibility that it aims to provide.

7.8 Conclusion

This chapter introduced the concept of on-path content delivery networks which are able to serve multimedia content to the end user through TCP handshake interception mechanisms. We described our motivation for developing this new architecture, and detailed how this architecture helps in improving latency for the end user, while reducing networking problems such as congestion and latency. We have detailed the protocol call-flows for enabling an on-path CDN, and have also described the implementation of our prototype on existing Internet infrastructure. We also presented performance measurements that show how the on-path CDN is an improvement over existing statically deployed CDNs in terms of reducing latency and round-trip time. Our approach enables a new class of content networking architectures with a large number of dynamically deployed content nodes.

Chapter 8

NetServ: Dynamically Deploying Software Defined Networking

8.1 Introduction

We present NetServ, an extensible architecture for core network services for the Internet of today and the future. NetServ aims to provide a layer to allow for executable functionality at the network core, on Internet routers, similar to software-defined networking [Kirkpatrick, 2013]. The functions and resources available on a network node are broken up into small and reusable building blocks. A new core network service is implemented by combining the building blocks, and hosted in a sandboxed execution environment that provides security, portability, resource control, and the ability to deploy modules dynamically.

We describe our first prototype, a novel combination of the Click router and the Java-based OSGi module system. Our measurement results indicate that the processing overhead incurred by the Java layer is a reasonable trade-off for the level of modularity we achieve in our system.

Despite the tremendous success of the Internet in the past decade, a number of shortcomings of the current Internet architecture have become apparent. The *ossification* of the Internet, often suggested as the main problem of the current architecture, refers to the fact that it is nearly impossible to add new functionality and services to the network core. This is clearly shown by the dismal rate of adoption of new Internet protocols such as multicast

routing and QoS, even when the need for them is widely recognized. Many have naturally turned to implementing network services on the application layer using overlay networks formed by end hosts, since providing services through overlay networks eliminates the need to update the network core. However, such solutions tend to be ad hoc, often duplicating the effort of other overlay networks, and inefficient because certain basic functions can be achieved much more effectively at the network core.

We present NetServ [NetServ Website, 2011], our on-going research effort to design an extensible architecture for core network services for the next generation Internet. The key idea of NetServ is *service modularization*. The functions and resources available on a network node are broken up into small and reusable *building blocks*. A new core network service can then be implemented by combining the functionality of building blocks available on multiple network nodes. We use the term *service modules* to refer to the building blocks or the composite components that use multiple building blocks.

Another piece of the NetServ architecture is the *virtual services framework*, which refers to the architecture of the network nodes that host service modules. The virtual services framework provides a sandbox-like execution environment for the service modules, offering security, portability across hardware platforms, and the ability to control resource allocation among modules. In addition, the framework supports adding and removing service modules at runtime, by network administrators or even by content providers and end users, enabling on-demand and per-flow services in the network core.

In this chapter, we describe our first prototype implementation of the NetServ architecture. We used the Click modular router [Kohler *et al.*, 2000; Click Website, 2009] as a base router platform, and augmented it with a Java-based dynamic module system called OSGi [OSGi Alliance, 2011], which provides the ability to load and unload service modules at runtime. The prototype implemented a component inside the Click router that intercepts incoming packets and sends them to Java service modules that can be installed and uninstalled at runtime. The Java technology provides portability across hardware platforms, and a comprehensive security framework on which to build our security and resource control mechanisms in the future.

The rest of this chapter is organized as follows. Section 8.2 gives a brief overview of the

two technologies that we have used to implement our prototype: the Click modular router and the OSGi framework. Section 8.3 describes our prototype implementation in detail. Section 8.4 presents our measurement methods and results. Section 8.5 summarizes the related work. Finally, we conclude and discuss future work in Section 8.6.

8.2 Technology Overview

8.2.1 Click Modular Router

Click [Kohler *et al.*, 2000; Click Website, 2009] is a modular software architecture for Linux and other UNIX-like platforms that allows for the creation of easily reconfigurable routers and switches. Click functionality is manipulated using a text file that specifies how modules, called *elements*, are arranged in a directed graph. The graph structure allows for numerous possibilities. One such possibility is shown in Figure 8.1. This example of a very simple Click configuration receives packets from network interface eth0, counts them, and discards them. Click includes hundreds of predefined elements so it is easy to reconfigure a graph to implement many types of network devices. In addition, custom elements can be written to further extend functionality.

The Click router can run in two modes: user-mode or kernel-mode. User-mode runs as a user-level process. This means it does not replace the routing performed by the underlying Linux kernel. In contrast, kernel-mode runs as a module inside the Linux kernel and can replace the routing functionality of Linux.

The performance of the Click router is much higher in kernel-mode, and it can be further enhanced by replacing the standard Linux Ethernet drivers with polling drivers. Polling drivers turn off Linux’s interrupt structure and device handling, and allow the network card to poll for packets.

Kernel-mode uses the `proc` file system to access data from a running element or to change the element’s settings. If more extensive changes are required, Click offers the ability to replace the running configuration with an entirely new one, called *hot-swapping*. Compared to NetServ, Click’s hot-swap feature is limited in three ways. First, Click elements are written in C++, thus the elements in binary form can be installed only on a particular

```
FromDevice(eth0) -> Counter -> Discard;
```

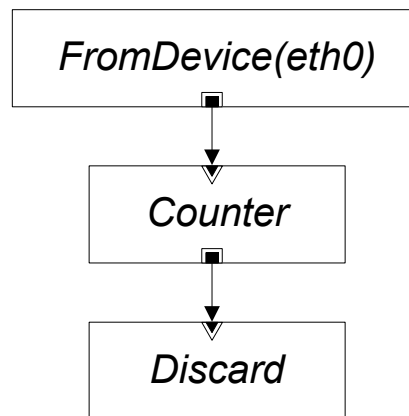


Figure 8.1: A minimal Click configuration.

hardware platform. Second, Click elements run inside the kernel so there is little to no security or access control. Third, the ability to hot-swap a particular element into a running Click router is dependent on the element having been compiled into the Click kernel module. The router must be restarted to insert a newly developed element.

8.2.2 OSGi Framework

OSGi™ [OSGi Alliance, 2011] is a component framework for Java. In the OSGi framework, an application is organized as a set of modules, called *bundles*, which are Java Archive (JAR) files [Sun Microsystems, 2004a] that conform to the structure specified by the OSGi framework. The bundles can be loaded and unloaded at runtime. This enables installing a new feature into a running application or upgrading a part of it with newly written code, without having to shutdown and restart the application. There are a number of implementations of the OSGi framework available today, including open source software such as Apache Felix [Apache Foundation, 2008] and Eclipse Equinox [Eclipse Foundation, 2007].

In a normal Java application, a class can usually access any other public class in the

same application, i.e., a class can create an instance of another public class and invoke a method on it. In OSGi, the scope of such an unrestricted access is limited to the enclosing bundle. In other words, the classes that belong to a bundle are *not* visible to the other classes that belong to other bundles. The only method of inter-bundle communication is for a bundle to explicitly *export a service* by listing a package containing the interfaces in the manifest file of its JAR file, and for another bundle to explicitly *import* the service, also by using its manifest file. The OSGi framework achieves this isolation of bundles by using a custom class loader.

8.3 NetServ Implementation

We implemented a prototype of NetServ using the Click modular router as the base platform. The Click router provides a high performance router platform that can be easily extended because of its modular approach. Extending the Click router is a matter of writing a new C++ class—an *element* in Click terminology—that extends a simple base class with a few member functions. This enabled us to develop our prototype concentrating on the NetServ functionality without having to worry about the basic router functionality. The current version of NetServ is based on the user-mode Click.

On top of the Click router platform, we used the OSGi framework. OSGi provides an ideal foundation on which we can realize our vision of a secure and portable services framework that supports dynamic distribution of services. Since OSGi is based on Java, it naturally inherits the portability across hardware platforms and the comprehensive Java security architecture [Sun Microsystems, 2003]. OSGi’s ability to load and unload bundles at runtime satisfies the fundamental requirement of dynamic distribution of services. The strict separation of OSGi bundles provides a solid starting point to address the security concerns associated with dynamic distribution of services.

Figure 8.2 depicts the overall architecture of the prototype implementation. The shaded boxes represent different components of NetServ, and the thick arrow represents the flow of a packet being forwarded by the router, taking a detour into the NetServ components.

We wrote a Click element in C++, called NetServ, and configured a Click router to

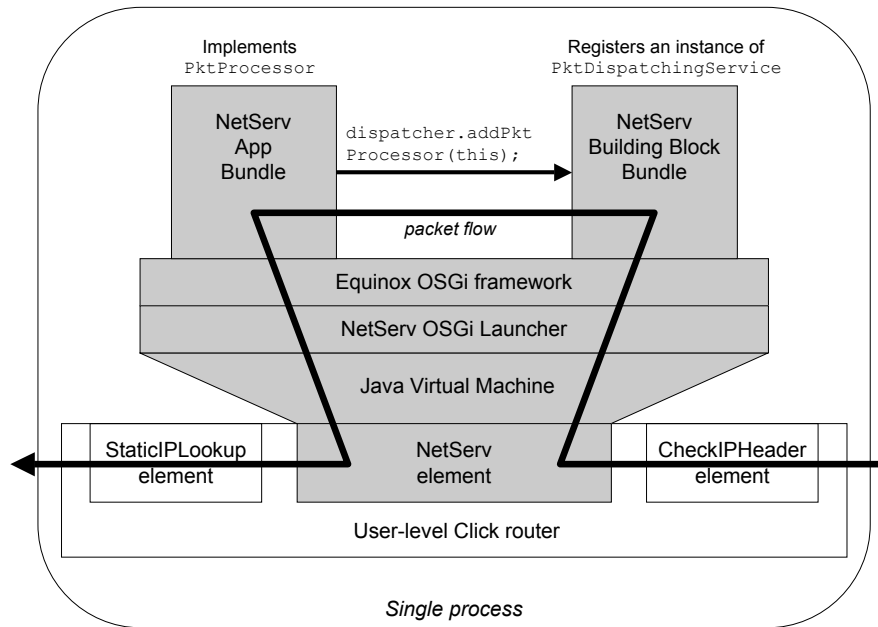


Figure 8.2: NetServ prototype architecture.

place the NetServ element on the path of the packet flow. Our test configuration was based on the basic IP router configuration that came with the Click software package. In that configuration, the NetServ element was placed between the CheckIPHeader and StaticIPLookup elements. When a Click router is started, it calls the `initialize()` member function of each element. NetServ's `initialize()` creates a Java Virtual Machine (JVM), launches the OSGi framework, and loads the configured bundles.

The NetServ element creates a JVM using the Invocation API, which is a part of the Java Native Interface (JNI) [Sun Microsystems, 2004b]. The JNI specification provides various ways for Java code and C/C++ code to call each other. The Invocation API, in particular, makes it possible for an application written in C/C++ to embed a whole JVM in the same process. After creating a JVM, the NetServ element invokes a Java function to run inside the JVM. That Java function is an entry point into the `NetServ.launch` package, represented as a box labeled NetServ OSGi Launcher in Figure 8.2.

The NetServ OSGi Launcher serves two purposes. First, it launches the OSGi framework, which in turn will load the NetServ Building Block Bundle and all configured application bundles. Figure 8.2 shows only one application bundle loaded—labeled NetServ App Bundle—but multiple application bundles can be loaded as well, in which case the packet will travel through each bundle in the order they were loaded. An application bundle implements the `PktProcessor` interface, and registers itself with the global packet dispatcher in order to receive the incoming packets. The global packet dispatcher is a singleton object which is exported as a service by the NetServ Building Block Bundle.

Second, the NetServ OSGi Launcher provides a Java class called `PktConduit`, which is visible from the Building Block Bundle and also accessible from the C++ code in the NetServ element. The `PktConduit` class therefore acts as a bridge between the Java and C++ regions. Such a bridge is necessary because an OSGi bundle is loaded using a custom class loader, making it invisible to other bundles or any other code outside the OSGi framework.

The NetServ element's `initialize()` function, before it returns, also finds and saves a handle to the `injectPkt` method of the `PktConduit` class using JNI. After the initialization is completed, the NetServ element diverts every incoming packet to the Java components by calling `PktConduit.injectPkt()`, which in turn will hand over the packet to the Building

Block Bundle, which in turn will invoke all registered packet processors.

We avoid copying a packet when it is passed from C++ to Java. We construct a direct byte buffer object that simply points to the memory address containing the packet using the `NewDirectByteBuffer()` JNI call. The reference to this object is then passed to the Java components.

8.4 Evaluation

We want to ensure that our goal of increased modularity for network services does not come with unacceptable trade-offs. There is a performance penalty associated with the detour that packets take into the Java layer of NetServ. We show that the performance penalty is a reasonable trade-off. We measure NetServ's maximum loss free forwarding rate (MLFFR), which is defined to be the maximum number of packets that a router can forward without incurring any packet loss. We compare NetServ's performance to a plain Click router running in user-mode (Plain Click) and also to a Linux kernel acting as a router (Bare Linux).

The overall test for MLFFR involves having node 1 send and count packets, node 3 receive and count packets, and node 2 forward packets between node 1 and 3. We compare the counts from nodes 1 and 3 to determine when and how many packets are dropped. We identify MLFFR as the highest packet rate for which the packet count at node 1 is the same as the packet count at node 3. Our measurements show that the MLFFR for Bare Linux is 115,000 packets/sec, Plain Click is 36,500 packets/sec, and Click with NetServ is 27,900 packets/sec.

When looking at the 1,500 byte packet test, Bare Linux, Plain Click, and Click with NetServ are all capable of forwarding rates that are comparable: just over 8,200 packets/sec. Each router levels off at the same maximum rate because they reach the bandwidth limit of our setup, which is 100 Mb/s. This result is favorable, as real world use of NetServ is more likely to involve manipulating larger packets instead of sending minimum sized packets as fast as possible.

Since the MLFFR of 115,000 packets/sec for bare Linux was sufficiently close to the

theoretical MLFFR of 148,800 packets/sec for 100 Mb/s Ethernet connection [Kohler *et al.*, 2000], we tested to make sure that the number represented the limit of the bare Linux routing performance rather than the line limit. We replaced node 2 with a 10/100 Mb/s Ethernet switch. A direct connection between the two nodes causes the connection to run at 1 Gb/s. The switch forced a 100 Mb/s connection. This ensures comparability with our other test cases which use 100 Mb/s connections. Performing our MLFFR test in this scenario resulted in a rate of about 142,200 packets/sec. In order to ensure that this is the line limit for 100 Mb/s and not some other limit, we also directly connected node 1 and 3 allowing the connection to run at 1 Gb/s. This resulted in a forwarding rate that approached 500,000 packets/sec. These checks demonstrate that we are reaching the limit of Bare Linux and not another barrier elsewhere (for example, in the CPU or networking hardware).

8.5 Related Work

Our work is fundamentally different from the active networking proposals such as ANTS [Wetherall *et al.*, 1998], JanOS [Tullmann *et al.*, 2001], NetScript [Yemini and Silva, 1996] and Switchware [Alex *et al.*, 1998]. In contrast to active networking, NetServ provides for virtualized services on current, passive networks by installing modules on the router control plane.

A service-centric view of the network core is not new. Tilman Wolf proposes a new abstraction for information transfer in the next generation Internet [Wolf, 2006]. NetServ complements the idea, as it can provide the technology platform on which to implement the abstraction.

Much work has been done on virtualizing different parts of the Internet architecture. Their focus is sharing network resources such as bandwidth. NetServ's focus is providing a uniform hosting architecture for network services. Three of these virtualization projects are described below.

The DaVinci project [He *et al.*, 2008] presents the design of a system that allows one physical network to support multiple classes of traffic. Major commercial routing hard-

ware vendors, such as Cisco and Juniper, are also offering increasingly fine-grained network virtualization services for their customers [Cisco, 2009a; Cisco, 2009b; Juniper, 2010]. Vyatta, an open-source routing platform vendor, also offers similar networking virtualization services [Vyatta, 2009].

The OpenSolaris Crossbow [Sun Microsystems, 2009] project aims to enable network virtualization and resource control for each service or protocol such as HTTP or FTP. It does so by virtualizing the protocol stack and the NIC for each service.

The VROOM router project [Wang *et al.*, 2008] presents “virtual routers” that can be moved from one physical node to another and controlled using network primitives. Egi *et al.* [Egi *et al.*, 2007] evaluate the implementation issues of designing a virtual router using the Xen virtual machine framework.

There are several other architectures that decouple control from the data plane in routers. The OpenFlow Switch [Stanford University, 2010b] aims to allow a standard interface to routers to enable researchers to run experimental protocols on their campus networks. Ethane [Stanford University, 2010a] provides a management model that aims to allow simple management and security in enterprise networks.

It is also interesting to compare NetServ with some of the newer approaches of service and network virtualization.

Software Defined Networking (SDN) allows network administrators to handle network services and packets like applications, allowing custom functionality to be applied to a set of packets or data based on specific conditions. [Wikipedia, 2016] SDN allows for this by de-coupling the control plane from the data plane at the network layer. OpenFlow [Open Networking Foundation, 2016] is one of the more popular implementations of SDN which allows network administrators to instruct switches to route packets based on specific header information to decoupled nodes that can process the information in specific packets. SDN has grown in popularity due to the rise of cloud services, big data and a massive increase in data consumed by end users and consumers. [Wikipedia, 2016] NetServ provides a full stack solution to the network virtualization problem. While SDNs separate the data plane and the control plane and allow the control plane to send specific data or packets to other locations to be processed separately, NetServ provides a network and service virtualization

framework in the router itself, so it is able to intercept and process the data and packets as it serves them to the end user or consumer of that data. It is possible for NetServ and SDN technologies like OpenFlow to co-exist; in fact, a NetServ implementation has been built on top of OpenFlow. [Maccherani *et al.*, 2012]

Docker [Docker, 2016] is an application virtualization platform that allows for automation of application deployment inside software containers without requiring operating system level virtualization. Docker allows for services and applications to be deployed across multiple servers or data centers through configuration files, and uses resource isolation features of the Linux kernel to create application specific containers without requiring the overhead of starting up virtual machines with their own operating system. While Docker has increased in popularity in recent years [ZDNet, 2014], primarily due to the ease with which it is possible to deploy applications and services through Docker, Docker is limited to application and application isolation, and does not support network virtualization on routers.

Fog computing [Wall Street Journal, 2014] refers to the use of end-user clients or servers near the edge of the network to provide services that are usually associated with cloud computing, such as Software as a Service (SaaS) and Platform as a Service (PaaS). While Amazon, Google, Microsoft and others provide cloud services through their data centers, there is a growing need for these sorts of applications to be running and installed at the edge of a local network, whether in home routers or cellular base stations. Fog computing services allow for such services. While there appear to be no major vendors of fog computing services, service providers such as GitHub [GitHub, 2016] allow for locally installed enterprise versions of their software to run in local enterprise networks. Fog computing provides for Software as a Service (SaaS) and Platform as a Service (PaaS) service classes, while NetServ provides for network virtualization, but it is possible to envision fog computing services to be distributed through NetServ to the end user or edge node, allowing for fog computing services to run on computing devices in the local network instead of at the data center of the service provider.

8.6 Conclusion

We described a prototype implementation of NetServ, an extensible architecture for core network services for the next generation Internet. We augmented the Click router with the Java-based OSGi framework to provide security, portability, resource control, and the ability to dynamically deploy service modules. Our measurement results indicate that the performance difference between our prototype and the Click router (essentially the processing overhead of the Java layer) is much smaller than the rate difference between the Click router and the Linux kernel. The difference between Click and Linux comes from the fact that the Click router runs as a user process and thus every packet incurs a transition from kernel to user mode. In a system where a module can be installed dynamically, such transitions are likely unavoidable, because running a service module inside a kernel would pose an unacceptable security risk.

Chapter 9

ActiveCDN: Cloud Computing Meets Content Delivery Networks

9.1 Introduction

In Chapter 8, I described joint work on the NetServ network virtualization project. In order to demonstrate the usefulness of such virtualization layers in the network, we need to build useful and working applications that show not only the functionality and purpose of such a layer, but also the application improvements that could be gained from using such a layer. In this chapter, I describe one such application, called ActiveCDN, which leverages the NetServ virtualization framework to set up and run a dynamic topology of CDN nodes in the network that adapts to changes in network traffic.

Content delivery networks play a crucial role in today's Internet. They serve a large portion of the multimedia content on the Internet and solve problems of scalability and indirectly network congestion. Most content delivery networks today rely on a statically deployed configuration of nodes and network topology that makes it hard to grow and scale dynamically. We present ActiveCDN, a novel CDN architecture that allows a content publisher to dynamically scale their content delivery services using network virtualization and cloud computing techniques.

Content delivery networks (CDNs) have proved to be crucial for the Internet to scale with the current increase in multimedia-rich content consumption [Tom Leighton, 2009]. CDNs

allow for content to be distributed to nodes distributed around the world, and retrieved from nodes that are nearest to the user requesting content. This allows CDNs to serve content with less latency to the end user, and at the same time, alleviate network load by pushing content closer to the edges. Today’s CDNs, however, are statically deployed and central redirection mechanisms which impede dynamic deployability and result in performance penalties.

Despite this dramatic growth in CDN technology and use, most CDN operators and networks still rely on static and pre-configured node deployments to operate and serve content. This could result in over- or under-provisioning of hardware and networking, and can result in too many CDN nodes in an area that sees few request for content, or too few CDN nodes in an area that has a lot of demand for content. When such scenarios occur, it is hard for CDN service operators to dynamically migrate and instantiate new nodes, resulting in a lack of quick responsiveness to user demand and viral content.

We present ActiveCDN, a novel solution for this problem that utilizes cloud computing and network virtualization techniques to enhance CDN instantiation. ActiveCDN enables content publishers to dynamically deploy and instantiate CDN modules on participating nodes, such as core and edge routers with programmable functionality, based on user requests, and to position and balance the location of these CDN nodes on an as-needed basis. Content publishers can thus choose to serve the content directly themselves, or instantiate new CDN nodes at locations closest to the users as more and more requests come in. In addition, the CDN modules can be set up to “time out” after a certain period of time if no new requests are seen at an ActiveCDN node, thus freeing up computing and space resources on that node.

We describe our motivation in more detail in Section 9.2. In Sections 9.3 and 9.4, we describe our current implementations, one of which was highlighted and chosen as an alpha project for the National Science Foundation’s (NSF) GENI next-generation networking project [NSF, 2009].

9.2 Motivation

The Internet has seen an explosive growth in traffic carrying multimedia content in recent years. Today, content providers either host their high-bandwidth multimedia content themselves, or more commonly host their content on content delivery network (CDN) providers such as Akamai and Limelight Networks which then deliver the content [Tom Leighton, 2009]. When content is hosted on a CDN, a user request is usually redirected to a server closer to the user. Application-specific mechanisms are used for this, such as domain name resolution using DNS [Mockapetris, 1987b] or request redirection using HTTP [Fielding *et al.*, 1999]. CDNs are statically deployed and scaling them to adapt to sudden changes, such as unexpected flash crowds, is currently difficult to do without manual intervention and over-provisioning in advance. An example of such a flash crowd event was the inauguration of President Barack Obama in January 2009, which “generated massive Web traffic”, leading to site slowdowns [Perez, 2009].

CDN technology results in reduced traffic for the provider and the network. Pallis and Vakali [Pallis and Vakali, 2006] show that CDNs can bypass “traffic jams on the Web, since data is closer to user and there is no need to traverse all of the congested pipes and peering points.” Pallis states that CDN costs are high (citing costs from 2004), but CDN pricing has decreased recently. For example, Amazon’s cloud services, such as Amazon S3 [Amazon, b] (for storing content) and Amazon CloudFront [Amazon, a] (its CDN offering which serves content hosted on S3), have been estimated to reduce hosting costs for low-bandwidth sites up to 75% [Pandey *et al.*, 2009]. Hosangar et al [Hosanagar *et al.*, 2004] present an analysis of how as CPU, infrastructure and traffic costs reduce, the cost of CDN services will reduce further over time. The website cdnpricing.com [CDNPricing, 2011] contains updated information about the costs of CDN pricing and is a good reference in this regard.

While cloud computing is delivering on the promise of elastic and flexible compute cycles in the cloud, we have yet to see it fulfill its promise and potential in the CDN space. For instance, while the combination of Amazon’s S3 and CloudFront services allow for CDN functionality, it does not allow for dynamic deployment of nodes, and also relies on a largely static and pre-configured network topology which is located in a limited set of data centers around the world.

Further, while CDNs are able to serve up static content efficiently, they are not able to process and transform content in a manner required by today’s networking stack. For instance, a video may need to be accompanied by local weather or finance information, which today’s CDNs are not able to deliver on due to their niche being on efficiently serving up static content.

We can see there is a need for being able to dynamically instantiate and deploy nodes at required places in the network, as well as actively process and serve localized versions of content. We believe that ActiveCDN solves this problem and provides the features needed.

In the following two sections (sections 9.3 and 9.4), we describe our current implementations of ActiveCDN, one of which processes and handles content on the edge node, and one of which processes metadata at the edge node while deferring content processing to the client itself.

9.3 ActiveCDN on the Edge Node

ActiveCDN enables a next-generation content-delivery mechanism using CDNs, and also allows for “pop-up content store” nodes that appear on an as-needed basis on the Internet without having to be pre-deployed. Such a CDN architecture would also be able to serve content more efficiently in disconnected, opportunistic networks. For instance, consider a highly popular video that is in high demand at a certain location. ActiveCDN can automatically pop-up a CDN content store at nodes on some of the routes that the content traverses as it is delivered to the end users (described in Chapter 7 of this thesis), and that node is able to cache the video on its local storage and serve the video directly to the end user.

Our current implementation of ActiveCDN operates on top of the NetServ network virtualization and cloud computing framework [Srinivasan *et al.*, 2009a] [Lee *et al.*, 2011a], which is described in detail in Chapter 8 of this thesis. NetServ allows for participating nodes to dynamically instantiate and run modules signed by content publishers.

Figure 9.1 shows how ActiveCDN works in a simple scenario where users are downloading a video file from a content provider. The content provider develops a ActiveCDN NetServ

module and makes it available for download. The content provider's server machines respond to the content requests as usual.

When a content provider sees a request, it can - based on a number of factors such as content popularity - choose to use NetServ's signaling capabilities to install a module on a particular NetServ node along the path to the end user or users.

When the rate of requests from a network location goes above a certain threshold, the server triggers on-path signaling to tell the NetServ nodes in the path that they should download and install the provider's ActiveCDN module. The signaling message contains the URL and other information about the module, so that the NetServ nodes can determine if the module is compatible with the node's policy and capability, and where to download it from.

Once the module is installed and its binary is verified by the NetServ installer (through an MD5 checksum), the content provider can signal the ActiveCDN module to process the video if necessary, and redirect requests for content to the NetServ node that now holds the cached and processed video. After a NetServ node successfully downloads the module, it signals back to the provider's server to register itself as one of the caching nodes. Now, the content requests originating from the vicinity of that caching node can be redirected to the node. The caching node will then fetch the requested content, sending it to the user as it is being fetched, and cache it for future requests.

As part of our implementation, our ActiveCDN service processes the original video, adding a watermark image on top of the video frames. The ActiveCDN module functionality was implemented using Java, Java Servlets API and OSGi. A Java library called Xuggler [Xuggler, 2009] was used for processing video. The ActiveCDN module itself processes the video and adds a watermark video to the original video, and also serves the processed video to the end user. (In our NSF GENI demos, we added a watermark and local weather information as a text overlay on top of the video.)

In our current implementation, the content server keeps track of all the NetServ nodes that it has instantiated in a database, and upon getting a request from a client, calculates the closest NetServ node based on Euclidian distance between the two nodes, based on IP address and geo-location. The content server then redirects the request to the suitable

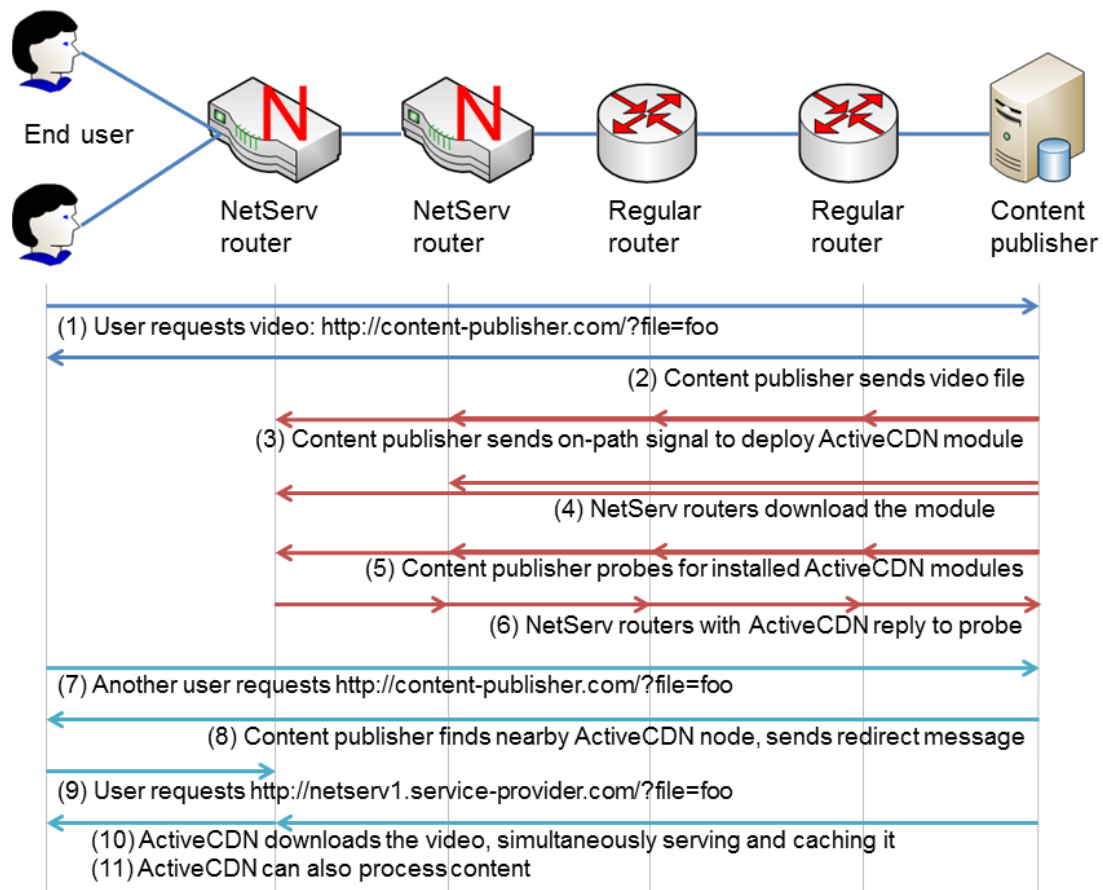


Figure 9.1: How on-demand content caching using ActiveCDN works.

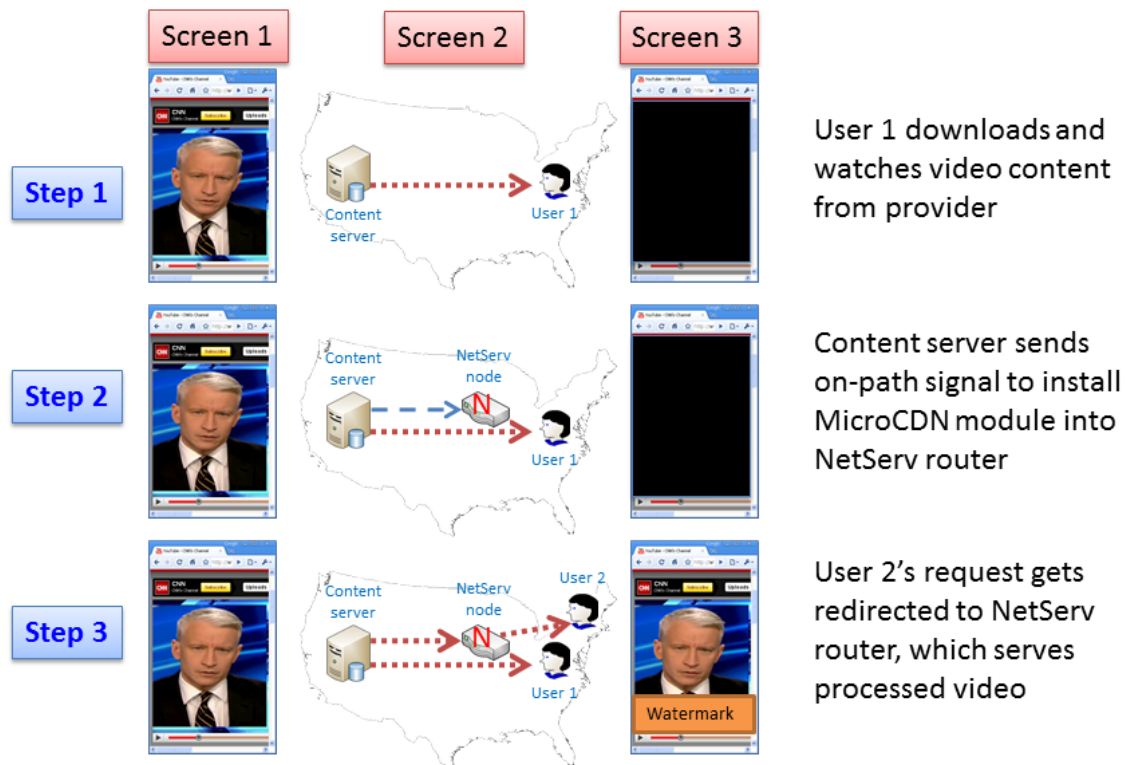


Figure 9.2: A visual depiction of what happens in our ActiveCDN implementation. Our implementation allows multiple users or clients to request video from the content server. The content server is able to control which nodes ActiveCDN is instantiated on, and redirect users to the node of its choice.

NetServ node, and if no such NetServ node exists, it triggers an on-path installation of an ActiveCDN module that will install ActiveCDN on a NetServ node closest to the user.

The ActiveCDN module caches the content based on the request and serves the content, but also performs background processing on the cached content. The processing is determined by the functionality programmed by the content provider, and could include anything from ad insertion to news and weather watermarking on top of the video. Subsequent requests for the content result in ActiveCDN serving the localized, cached content.

The ActiveCDN module will uninstall itself after a period of inactivity. And, of course, it can be reinstalled when new demands arise. The content provider controls the tunable parameters such as the inactivity time before module expiration. In fact, since the ActiveCDN module is written by the content provider specifically for its own use, the provider controls every aspect of the module's behavior, from the cache replacement policy to the algorithm to locate the nearest caching node. This is indeed the biggest advantage of ActiveCDN compared to the traditional content distribution network. Using ActiveCDN, content providers can employ any distribution strategy that satisfies their need, rather than being locked in by the mostly static infrastructures of traditional CDN providers.

Figures 9.3 and 9.4 show screenshots of the content as seen by the user, for a request that is served from the origin server and by an intermediary ActiveCDN node respectively. As seen in the second screenshot, the ActiveCDN node can also perform content processing in addition to actively caching the content. In this particular example, the ActiveCDN node retrieves the weather information for that location and creates a text overlay on top of the video.

Our current implementation gets the weather information from NOAA's web-based weather API service (weather.gov). This uses latitude and longitude information, which in turn is obtained using MaxMind's GeoIP library that resolves IP addresses to a location.

9.4 ActiveCDN on the Client

In addition to processing the content on the nodes, we have also worked on a mechanism to have content processed on the client, thus relieving the core or edge router from doing most

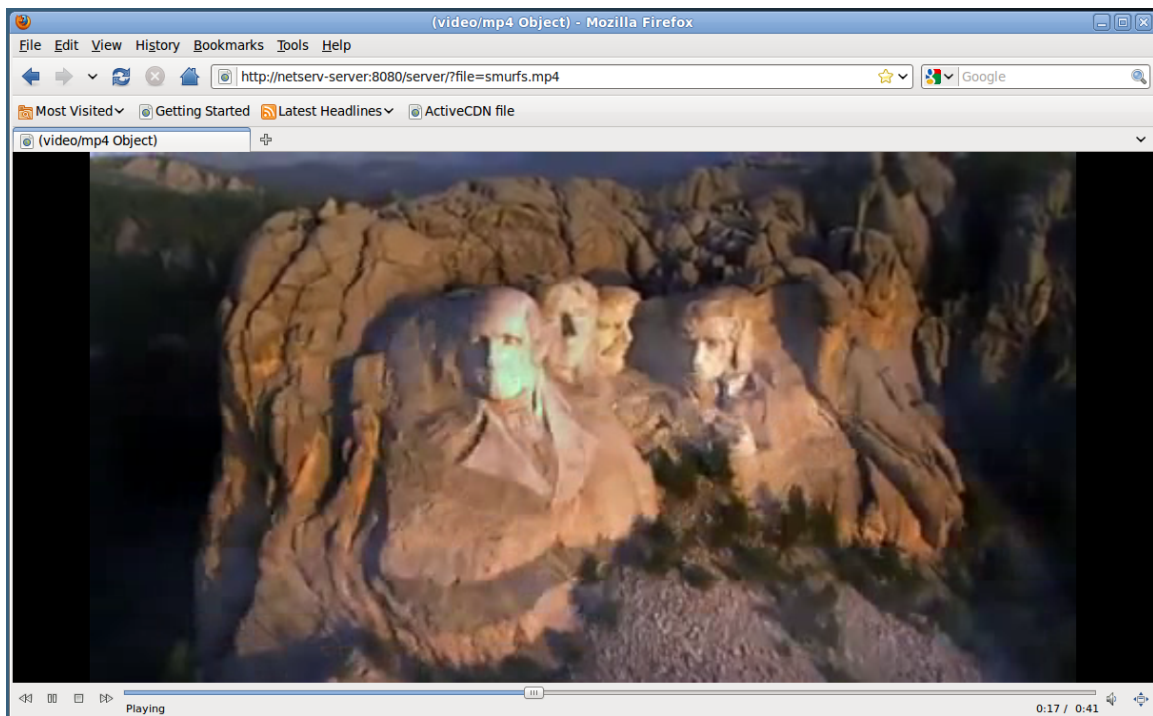


Figure 9.3: A screenshot of the original video playing on a user's browser. This is the video that is directly served by the origin server.

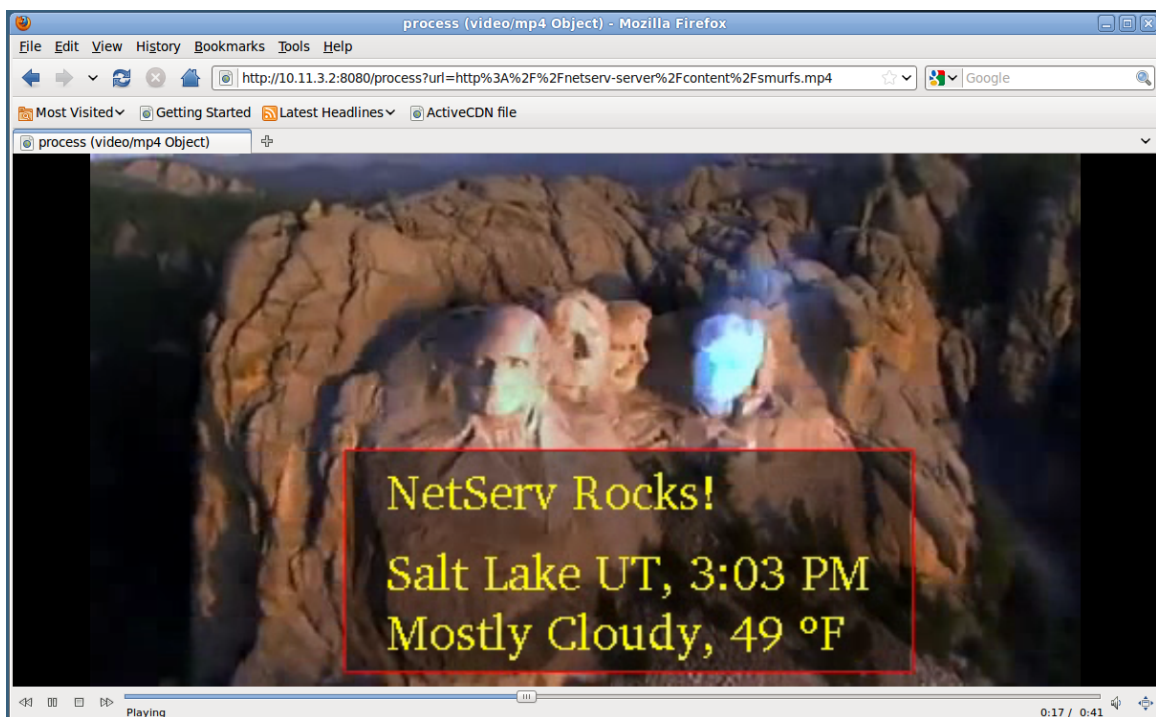


Figure 9.4: A screenshot of the processed video playing from one of the ActiveCDN nodes. The video, in addition to being cached and served from the ActiveCDN node, has been processed, adding the local weather information.

of the processing. In this model, the ActiveCDN module would still do some processing, such as adding additional metadata and dynamic content information, but would only be adding textual or markup information, with all the heavyweight multimedia processing, such as watermarks or text and image overlays, done on the client. This would free up resources on an edge or core router, thus allowing it to handle a larger workload, while at the same time being indistinguishable to the client who is able to see the same processed content. Finally, client-side processing using a markup language (like XML or SMIL [W3C, 2005]) at an intermediate node allows true localization of content and information that is specific to the end user, while at the same time allowing the local edge node to tag the content with markup language that can be localized to a particular region.

For our current implementation, we used the SMIL markup language [W3C, 2005]. SMIL is a HTML-like language used for multimedia presentations which integrate audio and video with images, text and other media types. In the previously described implementation Xuggler, a Java library, was used for content processing and data overlay, and the processing was done on a Netserv node or router in the network. In this alternative implementation using SMIL, we instead only generate a SMIL presentation file at the intermediate node, and this file is sent to the end user along with the rest of the dependencies.

The module which serves the SMIL file is implemented as a Java servlet process, similar to the previous implementation. It receives the video requests and serve a SMIL file, and also fetches the video from the content server and caches it.

The process for serving SMIL enhanced multimedia content is very similar to that described in the previous section, using Figures 13.1 and 9.1, with the only difference being that instead of processing the original video, the ActiveCDN module simply adds an additional SMIL file with the modified content meta-data along with the original video.

The end user experiences the following steps in this implementation: original video with localized weather information as a text overlay; advertisement video plays in middle of the original video; rest of the original video has news ticker overlay on top of the video. The operations that allow this to happen are described below.

The main components for the client-side ActiveCDN module are a content server, a NetServ node running the servlet process, and the Ambulant plugin for the Firefox browser

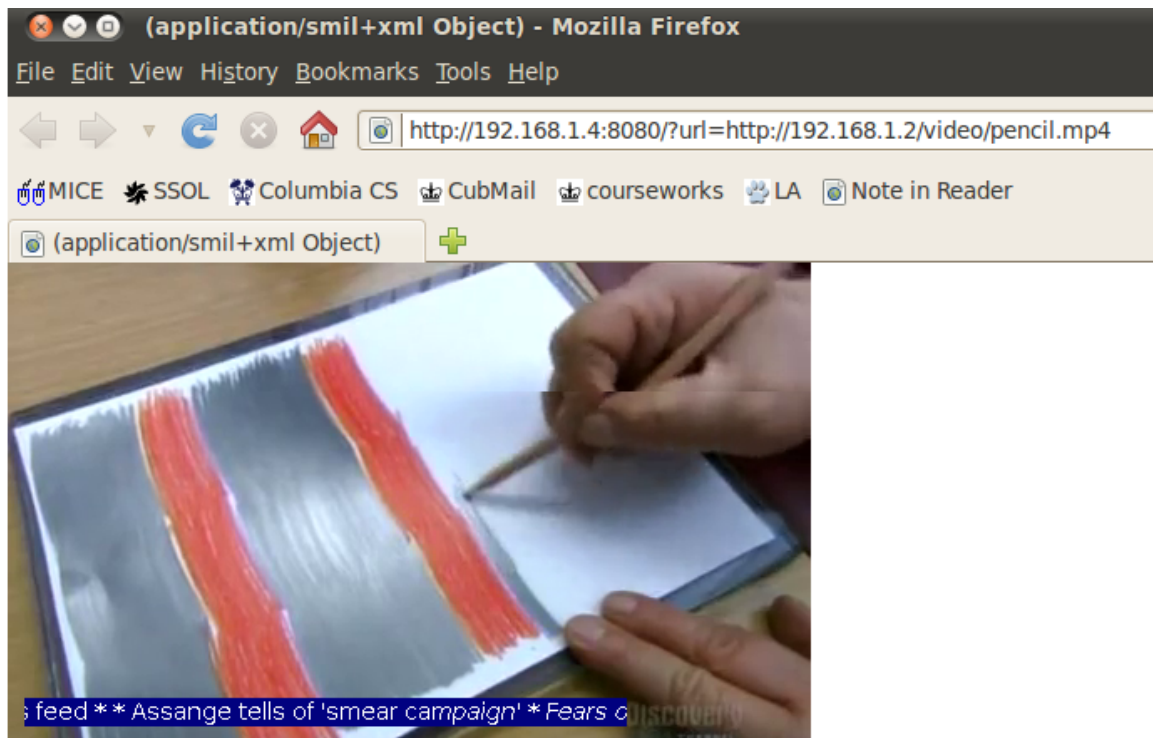


Figure 9.5: Client processing: A screenshot of the video with a scrolling news ticker overlaid on the video. The video is unmodified, and the news ticker information comes from the SMIL file that was sent to the client.

which displays the SMIL presentation file to the user. Ambulant is an open source SMIL player which supports the SMIL 3.0 standard. The NetServ node stores a repository of text and video advertisements. For each video request in our demo, it fetches the local weather information as well as the latest news feed from a news source (BBC). A weather lookup (similar to that described in the previous section) is also carried out. We also include a video advertisement, which in our current implementation is simply a random video chosen from all the videos in a specified directory. All XML data are parsed with the SAXBuilder class of JDOM Java library.

When the first request for the video comes in, this video is not in the NetServ node's cache. The servlet process in this case generates a SMIL file which points to the video on the content server and responds to the user with the generated SMIL file. The ActiveCDN module also downloads the video from the content server and caches it at the node.

For subsequent requests for the video, if the requested video is found in the cache, the servlet process generates a SMIL file pointing to the video found in cache. Along with this, it also fetches weather data, a video advertisement and latest news feed and adds them to the SMIL presentation. The video advertisement is randomly selected from a set of videos and inserted at a particular position in the original video. The NetServ node runs a lighttpd web server to serve the cached video when the request comes from Ambulant plugin.

9.5 Implementation

For the ActiveCDN server implemenation, we implemented and tested across seven virtual machines, which were Fedora or Ubuntu Linux images running on VMWare. The seven machines consisted of one content server, two NetServ edge nodes and four clients. The networking and routing tables were set up so that the clients always connected through the NetServ nodes (the gateway) to the content server. We used Fedora for the content server and the NetServ nodes since we also ran the same experiment setup on the NSF GENI Alpha testbed, which all ran the Fedora Linux distribution. The implementation described in the previous section ran on the virtual machine topology, as well as the real-world GENI topology that spanned several states and timezones across the United States,

namely, Massachusetts, Kansas, South Carolina and Utah.

We implemented ActiveCDN client (the “SMIL” implementation) on one physical machine running Ubuntu 10.04 and two virtual machines (on VirtualBox) running the same version of Ubuntu. The physical machine was used as the user machine running Ambulant plugin on Firefox browser. One of the virtual machines hosted the content server and the other virtual machine hosted the NetServ node running the servlet process.

The screenshots in the previous section were taken from running these implementations on top of the GENI topology and our local testbed. This experiment took place in 2010 as part of the NetServ project, which itself was part of the National Science Foundation’s GENI project [NSF, 2009].

9.6 Security and Privacy

ActiveCDN caching and content delivery is publisher driven, in the sense that the publisher decides when to deploy ActiveCDN modules and where in the network to deploy them. As such, this CDN deployment mechanism is fully controlled by publishers, as opposed to interception and end-user caching techniques such as On-path CDN (Chapter 7) and Coral-CDN [Freedman *et al.*, 2004], respectively. The publisher publishes the ActiveCDN module, which gets installed on the NetServ router devices, and those modules can cache content based on publisher directives. In addition to the module being removed when unnecessary, the ActiveCDN module could also delete cached files when they are not required anymore, requiring them to be downloaded from the server.

If any data that is associated with the users information is created in the ActiveCDN node (such as advertisements specific to the user or location where the content is cached), it can be stored and removed in that particular node, without reaching the central server. So the privacy of individual users and users in that location can be preserved.

9.7 Related Work

Traditional CDNs, such as Akamai [Tom Leighton, 2009] and Limelight Networks, started gaining traction in the late 1990s as large content providers resorted to using CDN services

in lieu of their own hosting to save costs and increase content delivery efficiency. In the early 2000s, large-scale ISPs (such as AT&T and Level 3) started to build their own CDN functionality. While most of the early CDNs served large content providers (due to the costs involved in using a CDN), in recent years, services such as Amazon S3 [Amazon, b] and Amazon CloudFront [Amazon, a] have introduced the concept of pay-as-you-go CDN services which allow smaller websites to use CDN services for their content. There have been several studies of how commercial content-delivery networks operate, mostly through reverse-engineering [Huang *et al.*, 2008], as well as research-oriented CDN services, particularly those such as CoDeeN [Wang *et al.*, 2004] that run on PlanetLab. Static CDNs are mostly made up of pre-deployed nodes placed at optimal locations; the deployment and location of these nodes is usually calculated based on network traffic and flow. ActiveCDN introduces a new CDN paradigm by allowing the content publisher to create “pop-up” CDN nodes on-the-fly, dynamically.

Content-centric networking (CCN) [Jacobson *et al.*, 2009a] has gotten a lot of attention recently, and for good reason. Content-centric networking, which envisions computer networking based on content names rather than host names, allows for multiple, signed copies of content that can be fetched or placed at any location on the network, with requests for content being served from the closest location. However, while CCN discovery services such as flooding and broadcasting work well in local networks, it is hard to see how they scale to wide-area networks. None of the existing CCN implementations have yet been able to address this issue. CCNx [Jacobson *et al.*, 2009b] is one of the most complete architectures and implementations of content-centric networking today. There are other projects aimed at developing similar content-centric networking models, such as Nebula [Smith, 2010] and eXpressive Internet Architecture (XIA) [Anand *et al.*, 2011], but they are still in their infancy and it remains to be seen how they fundamentally differ from the seminal CCNx work and what additional features they would have in comparison to CCNx.

ActiveCDN significantly differs from CCNx. In ActiveCDN, the publisher is able to instantiate nodes, as well as revoke node privileges, as necessary, while in CCNx, once content is published, it can be disseminated to any node. Naming in ActiveCDN is accomplished by the user typing in the regular HTTP URL to the content on their existing Internet

	<u>ActiveCDN</u>	Content-Centric Networking	CDN (Akamai-style)
Naming	HTTP URLs	CCN-specific naming	HTTP URLs (hostnames mapped to CDNs)
Discovery	Content provider instantiates appropriate CDN node and redirects user.	Part of underlying network naming and routing.	DNS resolution resolves to closest, pre-deployed CDN node.
Routing	User redirected to regular CDN node.	“Flooding” (multicast) in local network, left to router implementation for wide-area Internet.	Content is routed to and from the CDN node that was resolved in discovery.
Delivery	Regular network traffic from source (either content provider or CDN) to user.	Content blocks are copied to requesting host.	Regular network traffic from CDN node to user.
Load balancing	Able to achieve dynamic and highly localized load balancing through real-time deployment.	Perfect load-balancing assuming infinite node supply, or that all nodes in network are CCN enabled (since all nodes on route will cache copy of content).	Limited by CDN node deployment by CDN vendor.

browser (which may be redirected as necessary). CCNx has a tree-based and hierarchical naming structure similar to HTTP, but cannot be used without the full stack being present on the end user’s node. Routing in ActiveCDN is performed through redirection from the central content publisher to the node of the publisher’s choice (usually determined by network location and geographic proximity), whereas in CCNx, routing in the local network is performed through broadcast announcements for interest packets till a corresponding data packet is received in response. Routing across a wide-area network in CCNx is not clearly described yet.

CoralCDN [Freedman *et al.*, 2004] is probably the most prominent peer-to-peer, decentralized CDN, where content is served from P2P nodes that join and leave the network. Users, upon requesting content, have their DNS requests resolved by a Coral DNS server which checks for DNS and HTTP proxies near the client’s resolver. The user is then redirected to a Coral node near the user that contains the content. The content publisher has no control over the location of the content, and in fact, no control over the quality of the network since the P2P nodes join and leave the network at unpredictable intervals. CoralCDN

relies on a peer-to-peer network for content delivery, which may be unreliable as shown in the recent massive failure of Skype due to a bug that led to cascading failures [Skype official blog, 2010]. ActiveCDN, in contrast, allows the content provider to dynamically control the instantiation of nodes as well as the positioning and location of nodes. ActiveCDN allows for localization and processing of content. Finally, it runs on NetServ-enabled nodes at edge routers run by Internet Service Providers, which are likely to be more reliable and backed by service guarantees and SLAs.

Among the traditional CDN approaches that come closest to solving the CDN problem through a dynamic method are those that are optimized for handling flash crowds. A recent study [Yoshida, 2009] compares some of the state-of-the-art solutions, and I will contrast ActiveCDN with three of them. DotSlash [Zhao and Schulzrinne, 2006] and FCAN [Pan *et al.*, 2006] allow websites to deal with flash crowds through a pool of servers that are designed to handle flash crowds for websites. DotSlash allows several web sites (and their servers) to work with each other, using spare capacity to offset flash crowds to any of the participating sites. FCAN uses an overlay of caching proxies to store files and deliver them to users, and invokes this overlay when there are flash crowds. Globule [Pierre and Steen, 2006], works in a manner similar to DotSlash and FCAN, and allows any web server to join the pool of available servers through the installation of an Apache module. These solutions require servers to be pre-configured to handle large increases in traffic volume.

CoopNet [Padmanabhan *et al.*, 2002], DCDN (Distributed Content Delivery Network) [Mulerikkal and Khalil, 2007] and SCAN (Scalable Content Access Network) [Chen *et al.*, 2002] are hybrid CDN frameworks that combine infrastructure CDNs with peer-to-peer end nodes. DCDN uses the concept of “surrogates”, Internet users with high bandwidth, similar to the “supernode” concept in Skype, which handle content requests redirected from master or local DCDN nodes. CoopNet uses cached data at the clients to offload the central server, with a central server handling the redirects. These solutions require modification on the client-side in order to support the peer-to-peer mode of operation. ActiveCDN requires NetServ functionality at the edge routers, and does not require any modification on the client side.

MetaCDN [Broberg *et al.*, 2009] offers a “mashup” service, allowing content providers

to use cloud services such as Amazon CloudFront and others, through a single interface. MetaCDN takes care of replicating content and looking up the best location for each request. While MetaCDN is able to dynamically use cloud services APIs to store content in the “cloud”, it is still restricted by the server locations of these cloud services since it does not actually provide any server functionality or control the location of the servers. ActiveCDN allows the content provider to install nodes at precise locations and points along the path.

Perhaps the feature that is most unique about ActiveCDN is its ability to do local processing, with the processing functionality completely controlled by the content publisher. While there is some existing work on video and content processing for networking applications, most of the work relates to the processing of the video format itself [Magalhaes and Pereira, 2004] [Liu *et al.*, 2006] or of pure text-based content (no video or audio content) [Yuan *et al.*, 2004], and not in regards to the content delivery network. Some recent patents [Tidwell *et al.*, 2011] do discuss customized advertisement mechanisms, but assume the functionality is already installed, and focus more on the specifics of delivering customized ads, not on module placement and dynamic installation. ActiveCDN appears to be the first content delivery framework that allows the content provider to push custom content processing power into the delivery or service network.

The Akamai NetSession Interface is a “secure application that may be installed on your computer to improve the speed, reliability, and efficiency for application, data and media downloads and video streams from the Internet. It is used by many software and media publishers to deliver files or streams to” the end user [Akamai, 2015]. NetSession, as an Akamai application, downloads and caches content to the end users computer, and also uses idle client bandwidth to serve content to another requesting user, thus effectively serving as a peer-to-peer last mile caching solution for Akamai. As pointed out in one Microsoft support forum post [Microsoft Forums, 2012], “Akamai does not come right out and say it, but the reason NetSession is installed on your computer is to allow them to use your computer to upstream content to other users. By installing NetSession, you are allowing Akamai to use your idle bandwidth to upload files to other Akamai users.”

Thus, like CoralCDN [Freedman *et al.*, 2004], Akamai NetSession appears to use the local users storage and bandwidth to act as a peer-to-peer last mile caching solution, in addition

to its own network of central servers distributed around the world. While this does provide the benefits of caching closer to the end user, it requires the user to proactively download and install NetSession software on their computers. Given that the user has to download this unknown application, and since NetSession does not offer any particular benefit to the user himself, it is difficult to imagine end-users installing NetSession on their computers, although large companies might install NetSession on their managed desktops to reduce their WAN bandwidth needs.

Anycast [Abley and Lindqvist, 2006] is a addressing and routing mechanism where data packets from a sender are sent to the nearest node in the network out of a group of receivers. This could potentially be used by CDN networks to route packets to the servers closest to the users. While implementation details are sparse, it appears that some popular CDNs such as MaxCDN [MaxCDN, 2013] and CloudFlare [CloudFlare, 2011] use anycast on their network to respond to CDN content requests. Since each anycast address adds one routing entry to the global BGP routing tables, it does not scale well to large numbers of CDNs.

9.8 Conclusion

In this chapter, we described ActiveCDN, a novel approach to dynamic content distribution networks that can be deployed using network virtualization and cloud computing techniques. We believe that ActiveCDN is one of the first pieces of work to truly bridge the two disparate worlds of content delivery networks (CDNs) and cloud computing models. In addition to the dynamic edge router deployability features, we have also presented how ActiveCDN can perform processing and media transformation as required by the publisher, thus adding true CDN service virtualization and presenting it as an in-network functionality.

We argue that deploying ActiveCDN can dramatically increase user satisfaction using a related study. Golrezaei et al [Golrezaei *et al.*, 2011] describe how the use of femtocells to cache popular video content helped increase satisfaction for end users. They define and compare a metric called user satisfaction, where a user is satisfied “when his/her average download delay is below a given QOS (in seconds) threshold.” In their work, they study the effect of deploying femtocell base station “helpers” with large storage capacity. These

helpers form a wireless distributed caching network, quite similar in setup to ActiveCDN, all of which cache popular content and deliver the content locally. Their work compares the quality improvement based on real-world campus traces of video content, and they argue that through their setup, they can demonstrate “performance improvements on the order of 400% to 500% more users at reasonable QoS levels.” We believe that their analysis corresponds closely to performance improvements that would be seen if we deployed ActiveCDN in a large network environment.

Chapter 10

Trends in Online Internet Video Usage

10.1 Introduction

Video consumption on the Internet has risen rapidly in the past few years [Cisco, 2015]. It is desirable to evaluate the consumption of video on the Internet, in order to get a better idea of how we can improve networking protocols and architectures to handle the growing consumption of multimedia. This chapter presents my analysis of trends in Internet video consumption based on video data analyzed during my work at JW Player (previously named LongTail Video).

JW Player is an Internet startup head-quartered in New York, and named after its flagship product, an online video player called JW Player. The company provides online video services, such as video hosting, video advertising and transcoding, alongside player hosting and setup. The flagship product, JW Player, is a very popular hybrid Flash/HTML5 video player that allows for video to be embedded on websites and played on the Internet along with plugins for advanced playback functionality. Advanced playback functionality includes live streaming and support for Internet video ad formats such as VAST and VPAID. Video Player Ad-Serving Interface Definition (VPAID) establishes a standard request format for video players to deal with ad units, enabling a rich interactive in-stream ad experience. Video Ad-Serving Template (VAST), provides a common ad response file format for video

players to enable video ads to be served across all compliant video players. Layering VPAID onto VAST allows for well-defined ad delivery alongside content to viewers, and also allows for a consistent framework for collecting ad playback and interaction details for the video producer.

Commercial versions of JW player are found on popular websites such as the movie database website IMDb [Internet Movie Database, 2014], The Guardian newspaper [The Guardian, 2015], popular crowd-funding website Kickstarter [Kickstarter, 2016] and the White House websites. In 2013, JW Player was delivering an estimated 5 billion streams monthly and was used by or embedded in over 2 million domains (A domain is defined as a web site or a set of web sites under the control of a particular organization or individual). Hence it would qualify as a large sample set for my analysis. However, it is not possible to run analysis on the entire sample set for most of our analysis, nor is it necessary in most cases, and for each of my analysis below, I will note which subset (if any) of the data set I used. I refer to the data as being from the “JW Player network”, that is, content from all websites or domains that have the JW Player embedded with analytics enabled.

I note that the findings are only an evaluation of this data set and may not generalize to Internet video as a whole. It is likely to be reflective of short form video rather than feature-length video content.

10.2 Summary of Results

The results derived from this chapter can be summarized into two sets. The first set deals with video streams. Streaming video is video content sent (mostly in compressed form) over the Internet and displayed by the viewer software in real time. With streaming video, the web user (viewer) does not wait to download the entire video before playing it. Instead, the media is sent in a continuous stream of data and is played as it arrives. A video stream is defined as a single video file being viewed over the internet in this form.

In the first set, I find out if there is any correlation between video streams and length of video; video streams, video length and the type of device playing the video; the length of video and country; video streams and the domains containing the video. I find that

shorter videos are much more popular than longer videos across all devices. When it comes to devices, mobile users prefer shorter videos over longer videos. The top 10 and top 100 domains (domains are ranked by the number of visits in descending order and the top 10 and 100 are picked) follow an exponential distribution with respect to the number of videos streamed from them.

In the second set, various relationships are studied such as the number of video streams between 2010 and 2012, with weekly buckets; the relationship between the video viewing time (time of day) and country; the relationship between the day of week and views per country. Sunday is the day with the highest number of views. For time of day, afternoon and evenings seem to be the time when most number of videos are viewed.

10.3 Distribution of Video Popularity

I first analyze the distribution of video popularity, measured in terms of “streams” (number of plays). How does the popularity distribution of videos change over time and across countries?

For the analysis in this section, I look at the 100 most popular videos in the JW Player network (measured by total number of times they were played by end users) across a specific metric, whether it be time range or country.

10.3.1 Popularity Distribution Based On Time

For the popularity distribution based on time, I look at the distribution of the top 100 videos for August 2013, broken down by weeks. We want to investigate whether the popularity distribution of videos over time has a similar pattern, and if so, what sort of regression model we can apply and whether the curve is similar. Do video distributions follow a Zipf pattern? Or some other curve?

Unfortunately, this is one analysis where the answer is not so clear cut. The comparative graphs for just the top 20 (out of top 100 videos) are shown in Figure 10.1 and it appears that there is no clear-cut pattern to the curve. While a couple of the curves may fit an exponential or Zipf pattern, it is clear that distribution of the most popular videos over

time varies.

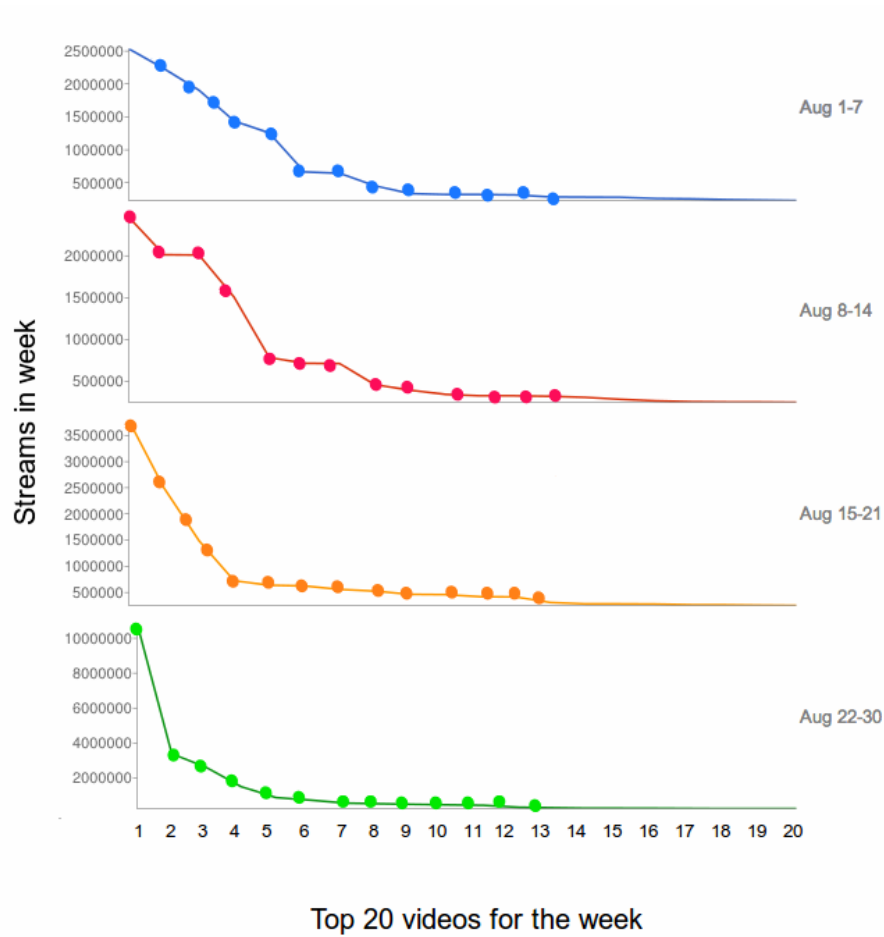


Figure 10.1: Popularity distribution of the top 20 videos for August 2013, broken down by week.

10.4 Video Length and Popularity

The second set of analysis relates to how the length of a video relates to its online popularity. I use a similar sample set to the one used in the earlier section on video popularity, and use this to analyse the relationship between video length and popularity of videos.

For this analysis, I took the top 1000 most popular videos (again measured in terms of total plays by end users) across the entire JW Player network for November 2012.

Bucket	Video length	# of plays
1	0 - 30 secs	52,323,431
2	30 - 60 secs	12,262,353
3	1 - 2 mins	3,683,704
4	2 - 5 mins	1,238,470
5	5 - 10 mins	519,437
6	10 - 30 mins	238,274
7	30 - 60 mins	313,817
8	>60 mins	393,484

Table 10.1: Video length and number of plays

10.4.1 Are shorter videos more popular?

As seen in Table 10.1, shorter videos indeed seem more popular. When we look at the top videos by number of plays, shorter videos outnumber longer videos. The previous analysis does not take into account the number of shorter videos; that is, since its likely that there are many more shorter videos than longer ones, the results may be skewed. Hence, we need to measure the average number of views and the total number of videos in each of those categories. But even when we take into account average number of views divided by total number of videos, so as to normalize the the distribution of videos in the top 1000, we can see from Table 10.2 that shorter videos are still relatively more popular than longer videos.

The average number of views in this analysis follows a U-shaped relationship with respect to the video length. However, we cannot conclude the same given the limitations of this data. Further, the bucket size for video length may skew the results. For example, if we had bucketed video length along, say, buckets of 1-10 minutes, 10-60 minutes and over 60 minutes, the distribution may be changed slightly. Hence we can only say that for our chosen distribution of data on the JW Player network, the average number of views based on video length seems to decline proportional to the length of the video, but does show a U-shaped curve for some long-form video (likely because of the popularity of those videos.)

Video length	# of videos	Number of plays	Average number of views per video
0 - 30 secs	547	52,323,431	95,655
30 - 60 secs	213	12,262,353	57,570
1 - 2 mins	104	3,683,704	35,420
2 - 5 mins	43	1,238,470	28,802
5 - 10 mins	36	519,437	14,429
10 - 30 mins	27	238,274	8,825
30 - 60 mins	18	313,817	17,434
>60 mins	12	393,484	32,790

Table 10.2: Video length and average number of views

10.4.2 Are Shorter Videos More Popular On Mobile?

A related question is whether shorter videos are more popular on mobile. In other words, do mobile users have a stronger preference for shorter videos, while desktop users may have a countenance for longer videos?

We analyze the data set used in the above video length comparison for mobile vs desktop device consumption. Note that by the definition of “mobile” in this context refers to hand-held devices like smartphones, and not tablets (which users may use for watching long-form content.)

As seen in Table 10.3, video viewing on mobile devices does drop off steeply as the length of the video increases. This is likely due to the fact that as the video length gets longer, users who are aware of the fact have either keyed up to watch it on their desktops or laptops, and avoid using their mobile devices (smartphones) to view long-form content. This is particularly significant for videos that are longer than 60 minutes in length; there, the video watching on mobiles drops to less than one percent of the total audience. In contrast, for videos that are between zero to 30 seconds in length and 30 to 60 seconds in length, video watching on mobiles hovers between 8.2% and 9.3%.

	# of videos	Desktop plays	Mobile plays	Desktop %	Mobile %
0 - 30 secs	547	48,012,933	4,310,498	91.8	8.2
30 - 60 secs	213	11,123,531	1,138,822	90.7	9.3
1 - 2 mins	104	3,469,383	214,321	94.2	5.8
2 - 5 mins	43	1,153,813	84,657	93.2	6.8
5 - 10 mins	36	493,049	26,388	94.9	5.1
10 - 30 mins	27	232,019	6,255	97.4	2.6
30 - 60 mins	18	308,923	4,894	98.4	1.6
>60 mins	12	391,234	2,250	99.4	0.6

Table 10.3: Comparing desktop and mobile plays

10.4.3 Video Length Viewed Across Countries

Yet another question that relates to video length across countries is how average video length relates to video consumption across different countries. Do countries with higher bandwidth have an appetite for consuming longer videos, while those that are bandwidth-constrained consume shorter videos?

In order to establish this, we need to distinguish between countries that are high bandwidth and low bandwidth countries. I was unable to find any clear definition of what would qualify as a high-bandwidth or low-bandwidth country, and hence the measure used for this study will be as follows: high-bandwidth countries are those countries that have an average Internet bandwidth of 10 Mbps or higher, while low-bandwidth countries have an average Internet bandwidth of lower than 10 Mbps. According to NetIndex's findings [NetIndex 2012, 2012], 72 countries fall into this thesis' definition of high-bandwidth, and 114 (out of 186 total countries in the study) fall under low-bandwidth.

Using this distinction and the above data-set, we will distinguish video viewership by high-bandwidth and low-bandwidth and total video length.

As we can tell from Table 10.4, the bandwidth of the country heavily determines the length of the video that people watch. It is also possible that content providers for those countries restrict themselves to short-form video content. As we can see from the table,

	# of videos	High-bandwidth	Low-bandwidth	High %	Low %
0 - 30 secs	547	41,893,823	10,429,608	80.1	19.9
30 - 60 secs	213	10,738,243	1,524,110	87.6	12.4
1 - 2 mins	104	3,193,842	489,862	86.7	13.3
2 - 5 mins	43	1,133,834	10,4636	91.6	8.4
5 - 10 mins	36	482,043	37,394	92.8	7.2
10 - 30 mins	27	234,039	4,235	98.2	1.8
30 - 60 mins	18	302,843	10,974	96.5	3.5
>60 mins	12	387,029	6,455	98.4	1.6

Table 10.4: Comparing high and low bandwidth countries

videos that are about 0-2 minutes long have more than 10% of their audience in low-bandwidth countries (as high as 20% for videos less than 30 seconds), but for videos that are longer than 10 minutes, that viewership rate drops to lower than 3.5%. This implies that countries with lower bandwidth do indeed view shorter-form content, whether they are restricted due to their content providers, or whether they experience a poor quality of video watching over the Internet in those countries.

10.5 Popularity Distribution Across Domains

One of the most interesting questions in the study of video data is the popularity distribution of videos. While we were not able to investigate the popularity distribution of individual video URLs, we were able to plot a popularity distribution of streams (or plays) aggregated to the domain they were streamed on (here domain refers to the full domain name of the hosting site - so `www.columbia.edu`, `columbia.edu` and `cs.columbia.edu` would register as three different domains.) This analysis was performed across all domains that had the latest version of JWPlayer (version 6.0 and above) and at least one stream played from the domain as an origin for the month of May 2013.

We studied two data sets:

- One is log files containing the access requests to the player's watermark logo. These

logos are loaded when the free version of JW Player (which displays the watermark logo) is initiated on a website and starts playing a video. Due to the popularity of this player, the access logs total about 4.5 TB of data, with more than 15 billion total impressions recorded. Through looking at the whole data—or a subset containing more recent data—we are able to obtain a macroscopic view of general video trends.

- The second data set are log files for access requests to their popular video hosting platform called “Bits on the Run”. This set of log files is interesting not only in terms of size—over 40 million streams—but particularly in terms of video engagement (which we define as how long a user has stayed watching a particular video) and related metrics. Bits on the Run collects video engagement data that enables customers (customers are the people who host their video on the platform) to see pingback data on video, which allows the evaluation of play, pause, stop, progress and skip actions on video.

Furthermore, I will state what we do and cannot do with the data at our disposal.

- Many of the results shown involve some sort of date and time. In this scenario, how we adjust for time zones around the world? For hour-of-the-day calculations, we normalize the time of day from our server logs by adjusting the hour to the user’s timezone, which we determine by using the user’s IP address for geo-location.
- We are unable to evaluate any data related to meta-data or classification of video data. This is because even though we might have access to the page URL or video URL of the data, there is little data that is actually set in the data directly accessible by the player. In order to actually evaluate any data related to classification of data, we would have to crawl all the URLs and obtain meta-data on the content related to that video. This would require us to build our own search engine.
- We are unable to evaluate any metrics related to total length of the video. This is because (like video classification data) this is metadata, that is not passed to the player. We find that though there are Javascript-based mechanisms that are supposed to allow for finding the total length of the video, most of the browser implementations

do not fully support this. In particular, this is a problem for streaming data which does not have a length set at all! For this reason, we are able to measure events (such as play, stop or progress) but not length of the video.

For the data set in the popularity study, we saw a total of 134,214 domains that qualified. There were 175,000 domains in total that had JWPlayer 6.x embedded, but the rest did not have a single play on them. These domains had the following characteristics in terms of data:

- A total of 1.04 billion streams (1,042,070,768 to be precise) were played across all domains.
- The top domain had 111,860,150 million streams (10.73% of total streams) across the month.
- The top 10 domains accounted for 406,485,036 streams (39.01% of total streams).
- The top 100 domains accounted for 797,945,601 streams (76.57% of total streams).

We plot the total number of streams per domain for the top 10 domains (Figure 10.2) and top 100 domains (Figure 10.3) in the following two graphs (a graph mapping all 134,000 domains looks empty due to the long-tail of domains with low streams.) It is clear from the graphs that the popularity distribution of video plays across domains is an exponential one.

10.6 Desktop, Mobile and Tablet Usage

One of our first—and one could say most obvious findings—is the steady growth of mobile and tablet consumption of video. Figure 10.4 shows the super-linear growth of smartphones and tablets as a percentage of devices used in consuming online video content. In this graph, we show only the breakdown of percentage of video consumption across mobile devices, particularly smartphones and videos.

While the growth seems to relate closely to the growth of mobile vs desktop traffic as observed by other website analytics providers [StatCounter, 2012], we note that usage

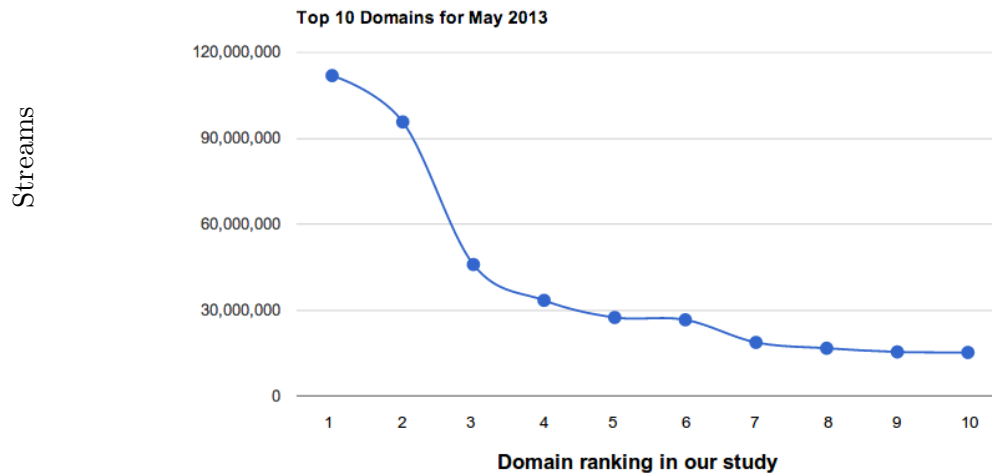


Figure 10.2: Popularity distribution of the top 10 domains serving video in May 2013 based on streams.

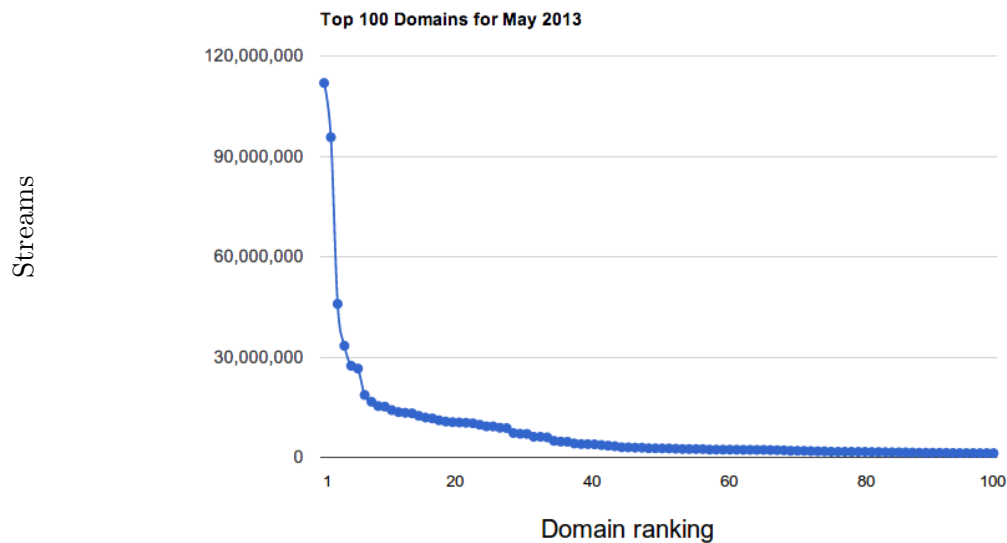


Figure 10.3: Popularity distribution of the top 100 domains serving videos in May 2013 based on streams.

of tablets for video consumption seems higher than the adoption numbers seen on website popularity tracking services. We believe this may be because tablet users—with their larger screens and easy mobility—to be more interested in visiting and playing video than users with smaller screens.

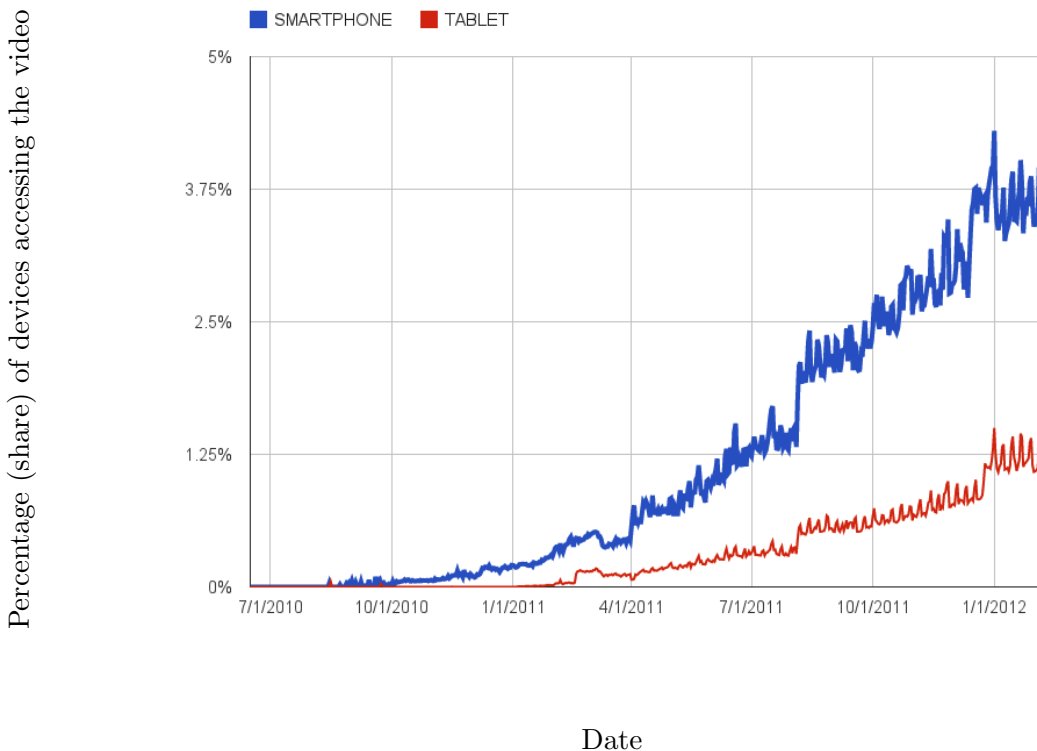


Figure 10.4: Changes in percentage terms of mobile devices (compared to overall device usage) used to access video on websites over a period of two years.

We note also the growth of the various mobile operating systems and platforms in Figure 10.5. This is along expected lines. We believe that the growth of Android phones compared to iOS phones might be due to the nature of JW Player traffic, which sees a lot of impressions from Europe and Asia and might skew the traffic in favor of Android for mobile data (and Windows for desktop data.)

We can also drill down to see specifically the growth of mobile traffic over the past couple of years in Figure 10.5. This figure shows the growth of the mobile operating systems, particularly Android and iOS, start to dominate the online video viewership.

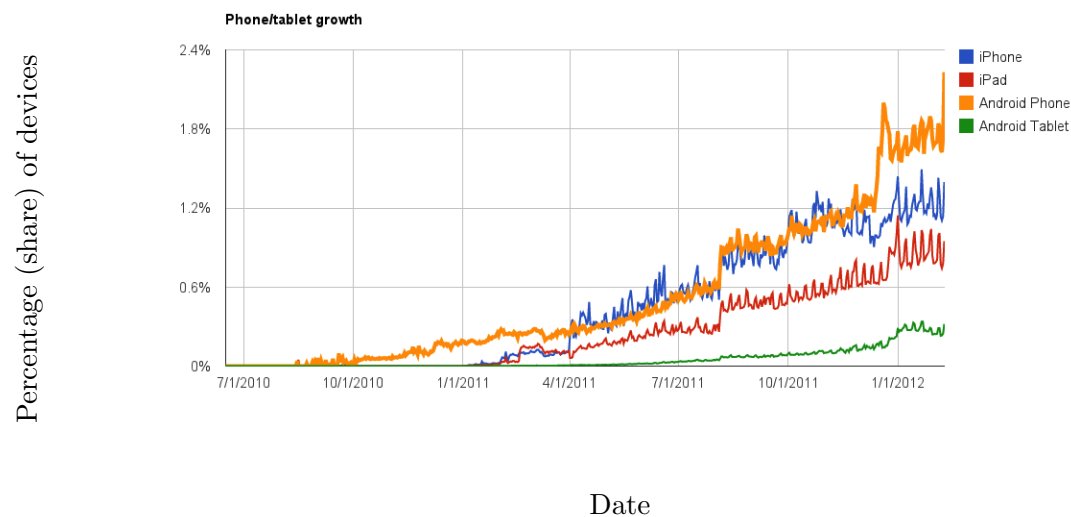


Figure 10.5: Percentage of video accesses for four device types (two each of smartphone and tablet) shows a growth trend, from June 2010 to January 2012.

Finally, in relationship to the above two numbers, we also present the growth the player numbers itself in Figure 10.6. The player is showing more than 30 million daily impressions in early 2012, showing the growth of video consumption on the Internet. Regarding the peak in late September 2011, we are not clear what drove the increase in video consumption on that one day, but it is possible that a video (or group of videos) went viral.

10.7 Country Viewing by Day and Hour

Another interesting statistic is the viewership of videos by country, and by the hour. I believe that these statistics give us some very interesting insight into the browsing and online video consumption habits of different countries. The breakdown of video watching patterns for four countries by day are shown in Figure 10.7. As we can see, the video watching patterns are similar across countries, decreasing during the course of a week and then peaking on Sundays.

A more interesting pattern begins to emerge in the breakdown of viewership by hour, as shown in Figures 10.8 and 10.9. For this analysis, I have taken the top 10 countries measured by number of plays or streams from that country, and broken those down into

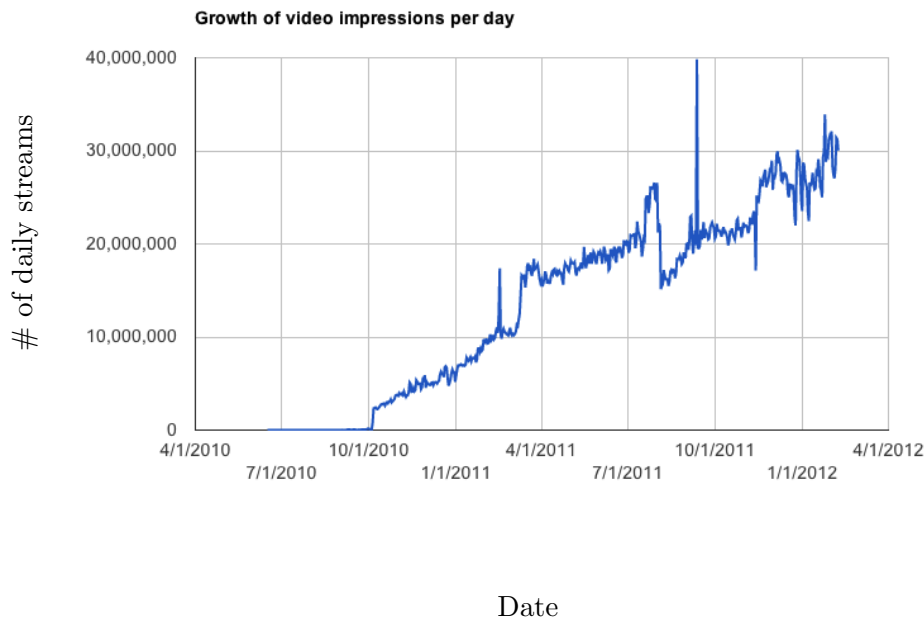


Figure 10.6: The number of daily video streams served using just the free version of the popular JW Player. These statistics include only player versions that are HTML5 compatible or higher (version 5.3 and above).

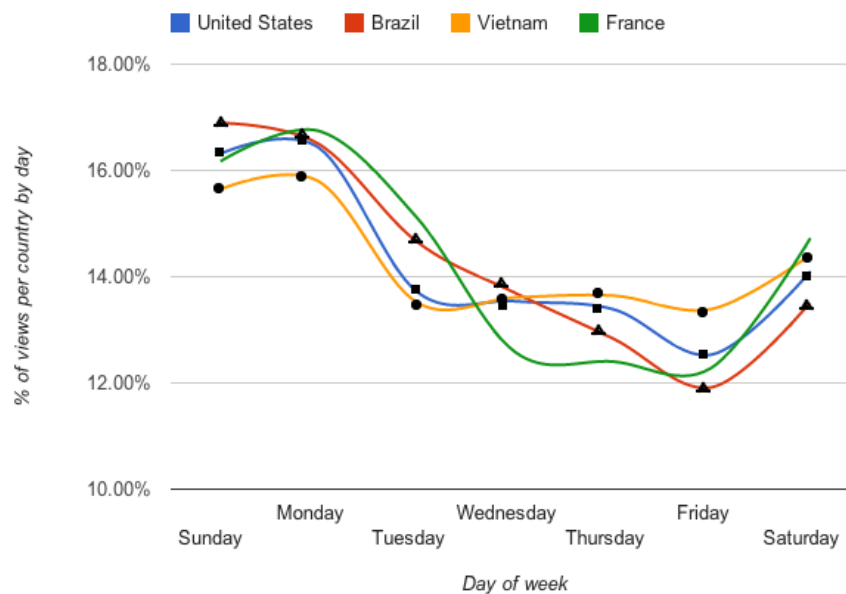


Figure 10.7: Breakdown of viewership by country, by day of week.

two buckets, with the top four countries in one graph, and the next six countries in another. The graphs are plotted with the number of streams as a percentage of total plays, bucketed into the hour they were played at, and smoothened to display on a curve.

In these two graphs, the viewership by country and hour is seen to differ slightly between countries.

One instance of viewership difference is that in our analysis, Russia and the U.S. both show patterns of usage that only slowly dip over time, even late at night. In contrast, other countries' video viewing tends to decrease steeply towards the night, showing that the usage pattern at late nights is much lower in those countries than in Russia and the U.S.

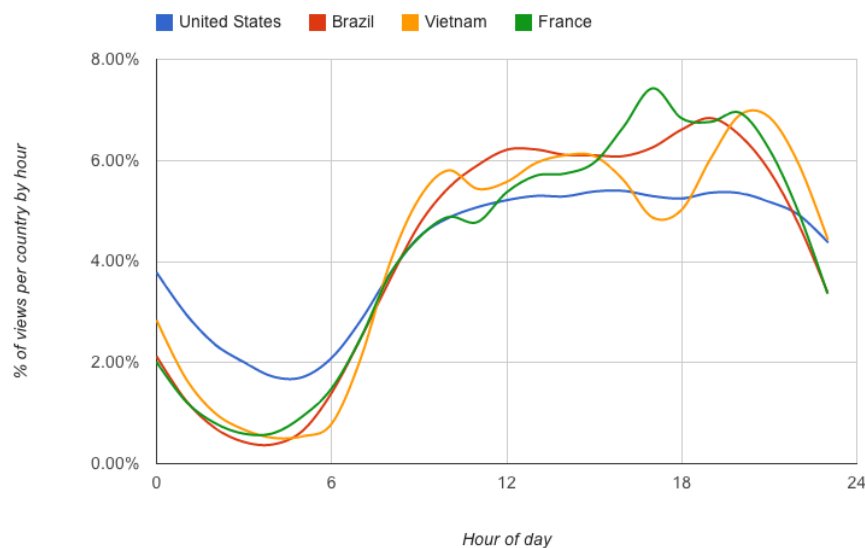


Figure 10.8: Breakdown of video viewership by country set #1, by hour, for the United States (with time zones adjusted), Brazil, Vietnam and France.

10.8 OS Viewership by Day

We analyzed how people use their tablets and smartphones to watch video, over time of the day and day of the week. This data was measured over a period of two months, January and February 2012.

As seen in Figure 10.10, over the course of a day, tablet users seem to predominantly use

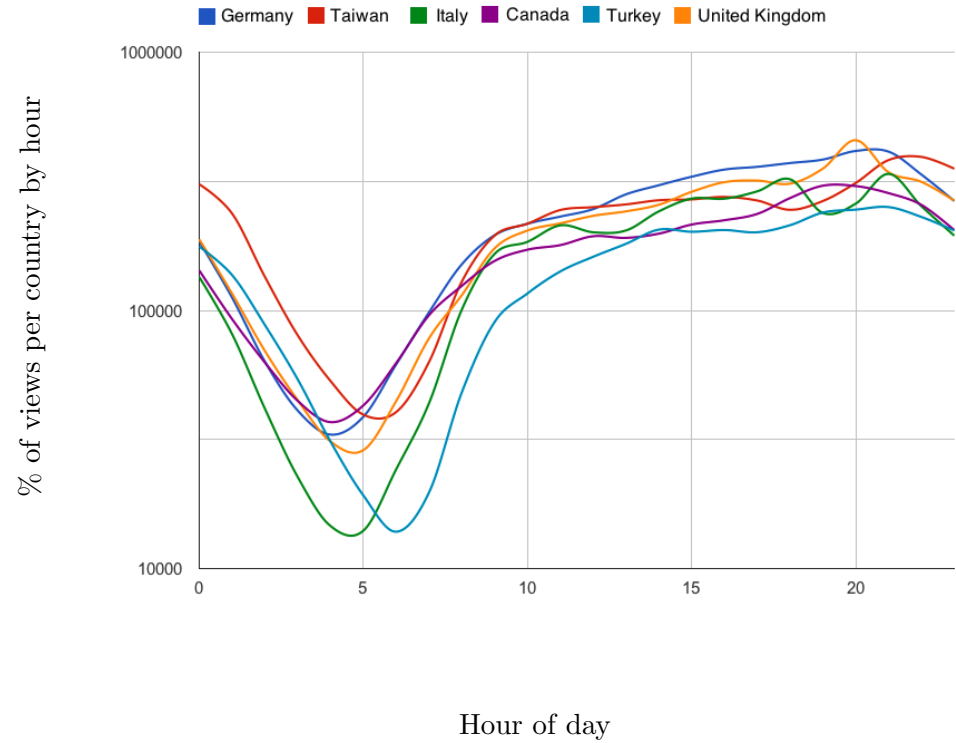


Figure 10.9: Breakdown of video viewership by country set #2, by hour, for Germany, Taiwan, United Kingdom, Italy, Canada and Turkey.

their tablets to watch video during the evening hours (after 6 pm). In contrast, connected device users (e.g., PlayStation) seem to prefer to use their device in the early hours of the morning, and so do smartphone users to some extent. Desktop users predominantly use their devices during the afternoon.

But over the course of the week (Figure 10.11), the variance of video consumption by various devices is smaller. That being said, phone and tablet usage are slightly higher over the weekends compared with desktop users.

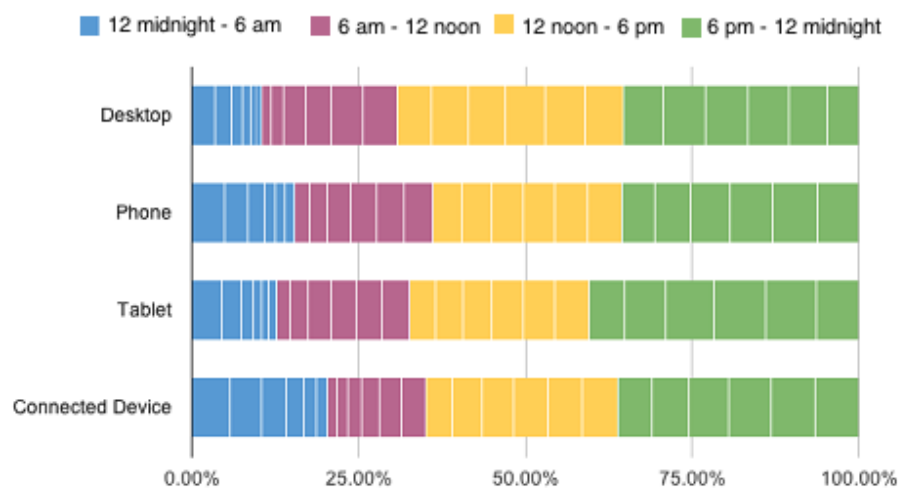


Figure 10.10: Device usage by time of day. The numbers indicate the hours of day according to a 24-hour clock, so 0 is midnight, 1 is 1 AM, etc. Blue indicates late night / early morning hours, pink indicates morning hours, yellow is afternoon / early evening, and green is late evening / night.

10.9 Play and Open Rate by Day of Week

For this part of the thesis, we define “plays” as when a video play actually happened (either when the webmaster enables auto-start, thus automatically starting the video when the page is loaded, or the user clicks the play button). “Opens” are page views or video embed actions, meaning that the user has opened a website with a video on it, but not necessarily played it.

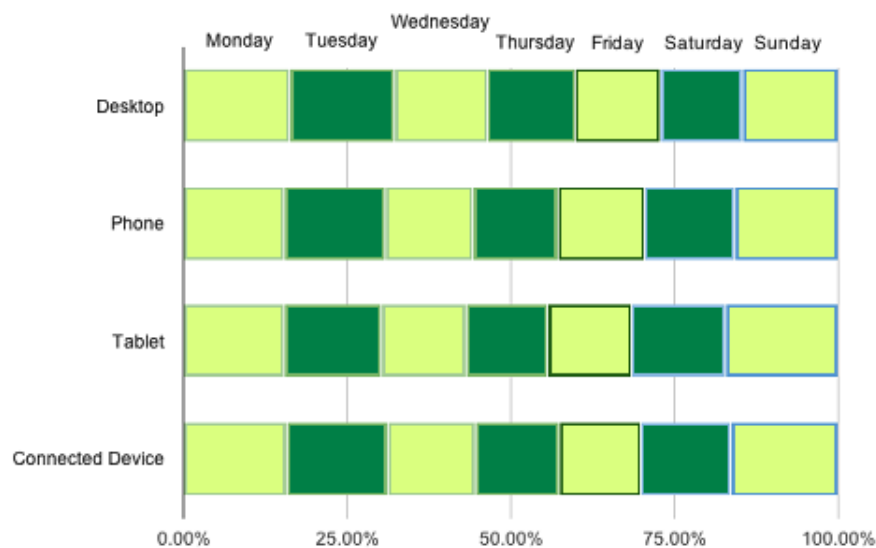


Figure 10.11: Device usage by day of week. Light green is Monday to Thursday, dark green is Friday, and blue is weekends. (Connected devices, as explained earlier, refer to living room devices such as the PlayStation.)

We look at the play and open rates of video by day. We use the “Bits on the Run” data for evaluating it. The results are shown in Figure 10.12.

As can be seen, the ratio of plays to opens are very similar throughout the week.

10.10 State Transitions in Video Playing

Another interesting measurement is the number and ratio of state transitions between various player states. This is shown in Figure 10.13.

The state transitions measured by “Bits on the Run” analytics are the following: ready, play, progress, stop, skip, and complete. The states are described briefly here:

“Ready” when a page is loaded with the video player.

“Play” when the play button is pressed and a video starts playing.

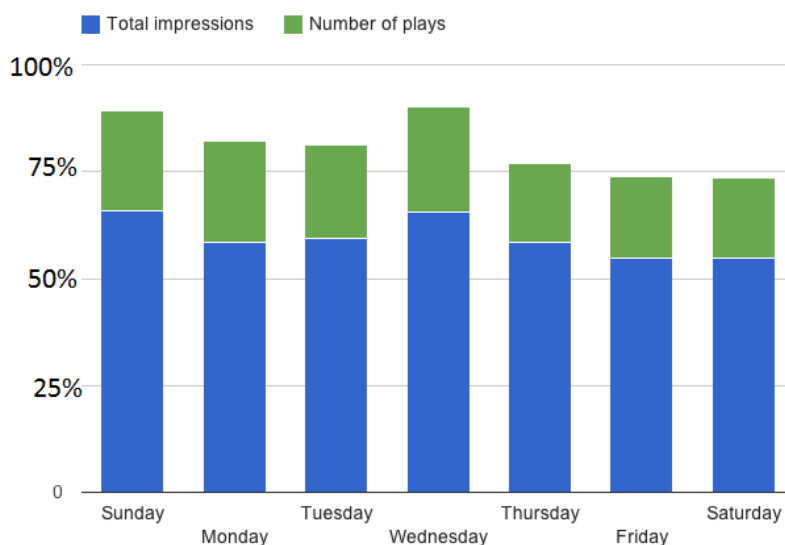


Figure 10.12: A breakdown of open and play rates across the days of the week.

“Progress” when a user has allowed the video to progress for a certain length of time (in our case, 30 seconds) without taking any action.

“Stop” when the video is stopped.

“Skip” when the user skips to another section of the video.

“Complete” when the video plays to completion (the stop event is additionally triggered in this case).

We see that there are quite a few state transitions covered in this diagram, and that viewers cross all state transitions, though some are more popular than the others. For instance, we can see that only 30% of the people who land on a page with video actually press play to watch the video. Out of those who play a video, about 40% of them keep viewing it (progress), 30% stop it and another 15% proactively seek a place in the video they are interested in (before any other event is triggered).

One thing to note is that even though the number of entry items is normalized to 100 (for easier viewing of transition data), the total number of transitions over the graph do not add up to 100. The reason for this is that a user could take multiple actions at any

state. For example, a user, once he starts playing a video, could allow it to progress for 5 intervals, stop it twice, resume it twice and then allow it to progress to completion. So each entry point could have multiple events associated with it.

Out of the people who skip viewing some length of the video, about 45% of them are likely to seek again (skip some other section of the video), another 45% just allow the video to progress and 10% of them stop the video.

Meanwhile, out of the people who are watching the video without frequently skipping, people are twice as likely to stop the video as to seek to a specific place in the video.

Note that while it would be interesting to estimate the total number or the length of time that someone allows a video to progress, we are unable to do so using our current measurements since: (a) the videos we have are of different lengths (b) each progress event only indicates that someone had watched that video for 30 seconds prior, but does not indicate where they were in the video when the event fired.

In addition, it would also be interesting to study abandonment ratio, but we cannot count this number because this behavior depends on the window close event in a browser. Tracking this event is difficult across various browsers, particularly the Safari browser and certain versions of Internet Explorer.

10.11 Related Work

Several studies have focussed on the view of video traffic data from the edge of the network. Gill et al [Gill *et al.*, 2007] measured Youtube traffic at the edge of the network on a campus, and provide network characterization and caching analysis of video traffic based on the Youtube video popularity seen at the edge of the network. We believe that some of our findings about country and regional localization corroborate their findings about regional popularity, but at the same time, our advantage of being at the “center” or the origin as a provider gives us greater insight into global traffic patterns across the world. From our analysis, the findings in [Gill *et al.*, 2007] apply not only to one edge network, but are pretty similar to those seen across the world.

Rao et al [Rao *et al.*, 2011] evaluate Netflix and Youtube, which are the primary sources

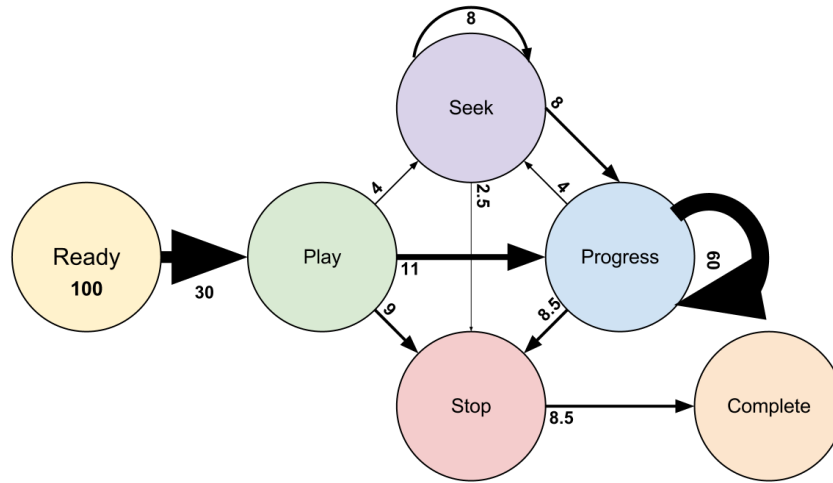


Figure 10.13: Video state transitions across the multiple states that we measure (load, play, seek, stop, progress). The numbers indicated have been normalized to the number of people who are in the starting (“ready”) state which is set to 100.

of video traffic on the Internet. They evaluate primarily the TCP traffic characteristics of this traffic, and find that Netflix and Youtube use three streaming strategies ranging from non-ack on-off to bulk TCP for delivering video efficiently to the end user. Adhikari et al [Adhikari *et al.*, 2011] reverse engineer of YouTube network and find that it has a “Flat” video id space, multiple DNS namespaces and a 3-tier physical cache. Finamore et al [Finamore *et al.*, 2011] find that users abort video playback quite frequently and that 60% of videos are watched for less than 20% of duration. They also find that users access independent of device used, that Youtube adopts different mechanisms based on device, and that the amount of video downloaded but not played is large (25%-39%), and larger for mobile devices.

In contrast to client-side measurements, YouTube statistics have also been measured at an ISP [Adhikari *et al.*, 2010]. The authors find that YouTube employs a location-agnostic, load-balanced method for delivering video content. They also build a routing traffic matrix and analyze the early-exit routing of YouTube traffic in ISP network, and find that YouTube traffic has what they call a “locality bias” which refers to videos showing popularity mostly within a region. The paper also explores the potential size of Youtube data centers. This

paper again differs from ours in that it measures video traffic at the networking layer from an ISP's perspective, and the work is complementary to our work.

Yu et al [Yu *et al.*, 2006] study weekly access patterns of usage and infer arrival rates of video viewers as well as metrics such as session length, popularity and popularity distribution of popular content. Their findings are also done at the content provider. Their findings are similar to ours for most metrics such as weekly access patterns, popularity distribution, etc, with the only difference being in total number of data points. Also, their study was done in early 2006, and ours is more current (2012) and we believe at a time when Internet video is more mature. Finally, due to video engagement data, in addition to raw video loads, we are able to measure much more interesting trends such as user engagement (such as transition between play, stop, seek and progress states), which gives us more insight into the interplay between the viewer, the provider and the network.

Costa et al [Costa *et al.*, 2004] study usage patterns of video viewership. But the data is several years old (it was done in 2004). Our work is more recent, and is being performed as the Internet video market is becoming rapidly mature and more mainstream.

10.12 Conclusion

This chapter presents my analysis of video consumption based on data seen at JW Player. I've put together several results related to trends in video consumption, such as: the growth of alternate devices (mobile and tablets); video consumption across countries, time of day, and day of week; video engagement as measured by event/state transitions. In contrast to some of the related work (most of which deals with video traffic views from the edge), the work presented in this chapter presents video consumption as measured from the center of the service provider.

Part III

Information Centric Networks

Chapter 11

Introduction: Information-Centric Networks

As the volume of video content on the Internet keeps rising, network and service providers are scrambling to add more hardware and caching to improve the delivery of content, but a new genre of networking research aims to solve the problem of caching at the networking layer.

ICN (Information Centric Networking), also known as CCN (Content-Centric Networking), is a networking framework in which content, and not hosts, are a first-class citizen of the network. In particular, where the Internet Protocol was about host-to-host communication, ICN/CCN is about content communication: requests are made for pieces of content, with no knowledge of the underlying network architecture or even end hosts necessary.

While the implementation of ICN is well put together in the CCNx platform [Jacobson *et al.*, 2009b], ICN is still a fairly nascent research topic, and there are still many real world problems that ICN faces. For example, do we have to replace our entire current networking stack - on IP - with a new stack for ICNs? How would services, remote procedures and dynamic content work?

In addition to the research that on CDNs (presented earlier in this thesis), this thesis also presents research topics in ICNs that relate to real-world use of networks, such as services and dynamic processing, as well as using existing networking architectures for ICNs.

Chapter 12 describes CCNxServ, a platform built on top of CCNx that allows for dynamic content to be generated on the fly in CCNx networks, with or without explicit requests from the end user. The CCNxServ platform does not require any modifications to the underlying CCNx platform, and runs a service layer, thus enabling migration of services and processes on an ICN.

Chapter 13 introduces IPv6 for ICN, using a subset of IPv6 addresses as unique content names/identifiers. Through this approach, we could run a pure ICN or CCN network on top of today's IPv6 protocols and current infrastructure without having to re-invest or rebuild an ICN networking architecture from scratch. Several challenges involved in running a pure ICN network on top of an IP network are also described, as are ways of how we can overcome those challenges.

Chapter 12

CCNxServ: Dynamic Service Scalability in Content-Centric Networks

12.1 Introduction

Content-centric networks promise to address content networking issues in a better way than today’s host-based networking architecture. But content-centric networking does not inherently address the issue of “content services” (which we define as an action that can be performed on a piece of content to transform that content), particularly service scalability and mobility. We present our work on CCNxServ, a system that allows for dynamic service deployment and scalability in a pure content-centric networking implementation (CCNx) through an intuitive use of the content naming scheme.

Content-centric networking projects, such as CCNx [Jacobson *et al.*, 2009b], XIA [Anand *et al.*, 2011] and Nebula [Smith, 2010], aim to improve content networking by providing an entire network stack centered around content and handling content requests. Content-centric networking typically focuses on static content objects, but many content services do not deliver the same bits to every receiver, but transform and personalize them. Examples include personalized advertisements and watermarking for content protection.

Our previous work, ActiveCDN 9, also deals with distributing content elements to users and providing content services. The naming of data elements differ due to the nature of the underlying network layer, but both provide closely-related services, namely access to content, including the possibility of manipulating the content “on the fly”.

We believe that a complete content networking architecture will focus not only on content-centric networking, but will involve content services as well. In this chapter, we present our architecture and implementation of CCNxServ which allows for dynamic services and service scalability on top of the CCNx content-centric networking framework. We present how we use the CCNx naming scheme to add service functionality to the purely content-centric CCNx architecture, thereby leveraging the existing content-centric features of CCNx and allowing for service scalability and mobility in CCNx. We add composable media transformation services as an integral component of the CCNx framework.

In our current service implementation, we are mainly concerned with media transformation services which provide transformative processing services on media without having to maintain state for each request or user. We acknowledge that there are a wide variety of services that require some maintenance of state, databases and personal information, such as social media and banking applications, to name just two such examples. A complete treatment of such services in a content-centric network is beyond the scope of this work.

Running services over CCNx and similar content-centric networks will allow for services to be deployed and to scale dynamically, as they can be distributed and duplicated dynamically, similar to how content would be replicated in such networks. In contrast, today’s host-based networks require precise information of the network topology as well as knowledge of node location in order to be able to deploy services to those locations. For example, the popular content delivery network Akamai offers some value-added services on top of its content delivery network, such as its “Advertising Decision Solutions”, which allows for companies to “seamlessly incorporate real-time anonymous Web browsing information with anonymous online purchasing data from advertisers’ websites to present the most relevant ad” [Akamai, 2008]. However, such services provided by Akamai and other large vendors are still restricted to predefined services and statically located data centers and do not allow for dynamic deployment of services in the network, while our services framework running

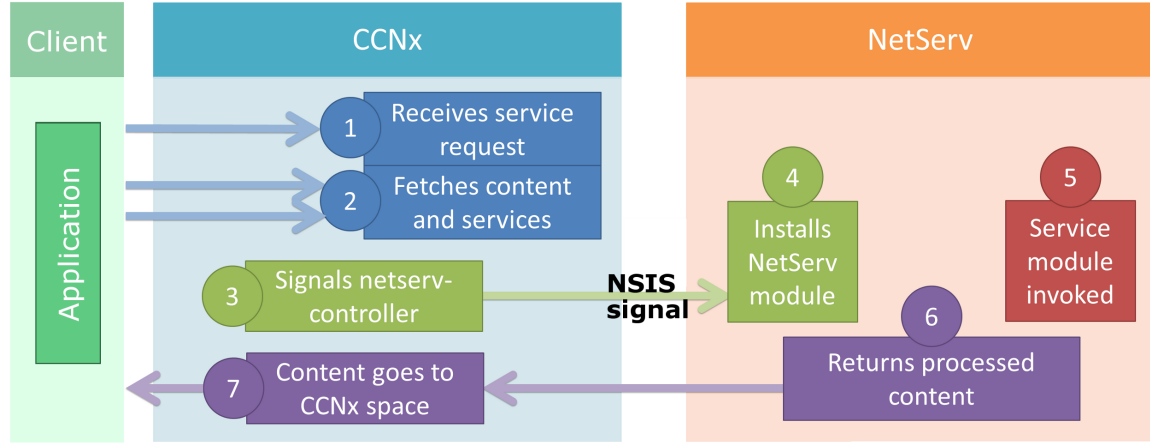


Figure 12.1: Overall CCNxEserv architecture.

on top of a content-centric network can provide such services and dynamic deployability. In addition, by being able to replicate services on demand, CCNxEserv can scale dynamically and be replicated closer to hotspots where there is a lot of demand for similar services.

There are a class of services that run based on information from the end user at the edge of the network. For instance, when a user requests a YouTube page or a website from a content provider, it is not uncommon for an advertisement to show up alongside the video or webpage. However, these advertisement requests involve a separate request for the advertisement content which is still served from an origin server. This is a separate request, and as such, could be potentially blocked by the end user by ad blocking software. If instead the advertisement was served to the end user after being processed at an edge node that had a replica of the service, it would be one piece of content that would be customized to the user but indistinguishable from the original content request. (We are not trying to argue for or against advertisement on video or other pieces of content; this is just an example of how in-network service processing would differ from current service requests.)

12.2 Scaling Services Dynamically in CCNx

In this chapter, we present our implementation of service-centric networking on top of CCNx [Jacobson *et al.*, 2009b]. We believe that CCNx and its current implementation (available as open-source on the CCNx website [Xerox PARC, 2010]) are a solid starting point for

building a service networking architecture. The version of CCNx that we used was the one that Xerox PARC released in November 2010, which was CCNx 0.3.0 [Michael Plass, 2010].

The architecture for our current implementation is shown in Figure 12.1. Our current CCNxServ implementation allows for a single service to be invoked alongside the content name (our naming scheme is *contentname+servicename*). When such a name is seen by a content router, the content router is able to parse the name into the content name and the corresponding service that is to be invoked on it. If it does not yet have a copy of the requested content, it retrieves the file via CCNx. Also, if it does not yet have a copy of the service module corresponding to the requested service, it fetches the corresponding service module via CCNx. Then, the content router invokes the service on the content and, after the processing is complete, it serves the processed content to the requesting client. In addition, it places the processed content back into the CCNx namespace so that future requests for this combination of *contentname+servicename* will directly be able to fetch the processed content. We allow these processed content objects to be cached for a default time interval (say, 15 minutes) for faster response time while enabling the application to change the caching interval or even disable caching completely.

Today’s data center infrastructures usually require data to be moved to central locations where they are processed and transformed. CCNxServ allows for services to be moved right to where the data are situated, thereby enabling service mobility to locations where they are most needed, and allowing for transfer of lightweight executable modules rather than requiring large data sets to be moved around.

In CCNx, we can also define such services implicitly in the content request. Because CCNx allows for the concept of “prefix matching” (where it matches prefixes at any level of the content name) any content router that implements our architecture could add a service name to the end of the name. Our current implementation will work in both centralized and distributed infrastructures that are supported by content-centric networking.

12.3 Architecture

For our current implementation, we focused on creating and running three commonly-used services on top of the framework we describe below. The “weather” service takes a video file as input and generates output with the latest weather information from www.weather.gov as an overlay on top of the video. The “ads” service inserts a random video advertisement (from a given list of ad files in a certain directory) in the middle of the video. The “news” service generates an output with the latest newsfeed (from BBC) as a marquee on top of the video. This application is similar to the application presented in the previous chapter on ActiveCDN 9, but the underlying request mechanism and architecture are different, as described below.

As shown in Figure 12.1, an application running on a client device or a network node makes a request for a content name and a service it wishes to invoke on the content (e.g., *ccnx://video.mp4+ad*). The request is converted to a CCNx interest packet and forwarded to CCNx. The request is intercepted by any of the nodes that operate as a Content Router (step 1 in the figure), and if the (processed) content “video.mp4+ad” does not exist, the Content Router looks for a service corresponding to the service name. If it does not find a service in its local cache, it fetches it by issuing a CCNx interest packet for that service module (step 2). The same is true for the content; if the file (video.mp4) is not in the local cache, it is fetched from the content-centric network (step 3). Once both content and the service module are located and downloaded on the Content Router, the service module is installed (step 4) and executed on the content (step 5), thus producing a processed version of the content, which is returned to the client (step 6). In addition, the processed content is put back into the CCNx namespace for future requests (step 7).

In order to demonstrate that we are able to provide “plugin” functionality for a fully robust service platform, we integrated the CCNx services implementation with the NetServ service virtualization framework [Lee *et al.*, 2011b]. NetServ is described in detail in Chapter 8. With NetServ, an edge router can become a platform for publishers’ content and services, allowing content publishers to dynamically deploy within NetServ their services on these edge routers.

We created a CCNx controller that signals the NetServ service stack and runs service

functionality through NetServ and its OSGi stack [OSGi Alliance, 2011], while still using the content networking functionality provided by CCNx. The NSIS signalling protocols [Hancock *et al.*, 2005] [Schulzrinne and Hancock, 2010] help to instantiate modules. The NSIS protocol is used for the routing and transport of per-flow signaling messages along the path taken by that flow through the network. The CCNx controller complements the IP-based signaling controllers with CCNx-based signaling.

In addition, as a result of our network-based signaling, one CCNx controller will be able to signal and control multiple NetServ nodes. This mode of operation could be useful in large data centers, where some of the nodes could be CCNx-enabled and others only IP-enabled, with some running a service framework and others a content-networking framework. In addition, such a topology would allow for services and content transformations to be scaled and moved to appropriate nodes depending on load factors.

12.4 Implementation

We implemented our solution on top of the open-source implementation of CCNx 0.3.0 provided by PARC [Xerox PARC, 2010]. The reference implementation provides the core CCNx protocol stack implementation along with a few sample applications and utilities. We use some of the existing utilities in the CCNx implementation, and have modified and implemented our own functionality on top of it.

One of the utilities that we use is the `ccnfileproxy`, a proxy for the file system that makes files on the local file system available over CCNx. It takes a directory from which to serve files, which it treats as the root of its content tree, and an optional CCNx URI to serve as the prefix for that file content as represented in CCNx. For example, if there is a directory `/foo` in the file system and the CCNx URI is defined as `ccnx://testprefix`, `ccnfileproxy` is called with the arguments `-/foo ccnx://testprefix`, and a request for `ccnx://testprefix/file.txt` would return `file.txt`.

For our implementation, we modified the `ccnfileproxy` as follows: we intercept the interest packet and scan the content name to see if any service names are included. If no service name is included, we allow the `ccnfileproxy` to continue its normal mode of execution.

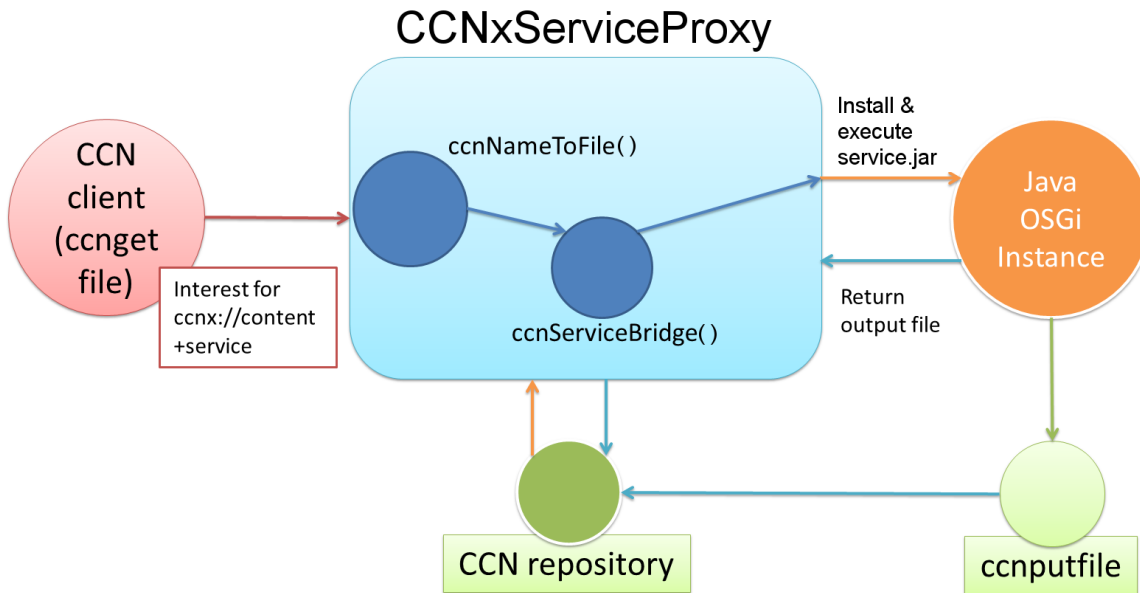


Figure 12.2: The architecture of our CCNxServiceProxy implementation and how it interacts with the various CCNx utilities.

If there is a service name specified, we map it to an internal service module file name, issue another interest for the corresponding service file (a JAR file in our implementation) using `ccngetfile`, and dynamically load it. Then we find the appropriate class in the JAR file, and call the appropriate service method in the class on the content file. We call this modified proxy `CCNxServiceProxy`.

The pseudocode for our implementation, in pseudo-Java, is shown below:

```
ccnName = "ccnx://content+service";
array(service, file) = parse(ccnName);
bundleFile = download("ccnx://service" + ".jar");
content = download("ccnx://content/");
controller = initializeOSGi();
serviceBundle = controller.installBundle(bundleFile);
processedFile = serviceBundle.execute(content);
putFileIntoCCNx(processedFile);
```

To add a new service to our implementation, a new JAR file with the service has to

be created. The most important file in the JAR file is the service class, which has to implement our *CCNxService* interface for the service implementation. The *execute(Object param)* method has to be overridden in this service class to provide service functionality specific to the class. When creating an OSGi bundle, the CCN-Service attribute is used in the manifest file of the JAR, and this points to the class which implements the *CCNxService* interface. An Activator class (which activates and enables the module for OSGi) can be included in the service bundle as well. All the service related class files are packaged into a `service.jar` file. The JAR files are then loaded into CCNx namespace through `ccnxfileproxy` or a similar utility.

The overall architecture of our CCNxServiceProxy is shown in Figure 12.2. When the CCNx interest reaches a CCNxServiceProxy, we invoke the *ccnNameToFile()* method on the interest object. This function call translates the CCNx content name into the corresponding file name while parsing it. If any service is found in the content name, the *ccnServiceBridge()* method is invoked. This method handles the loading of the service module, invoking it on the content, and returning the processed content into the CCNx space.

When the *ccnServiceBridge* method is invoked, we check whether the output file, of the form `CONTENT_NAME%SERVICE_NAME`, is present in the file repository. If the file is present, we check for any further service invocations. If the file is not present, we check whether the OSGi bundle corresponding to the service name is already installed. If the OSGi bundle is installed, we invoke *executeModule()* method on the OSGi controller with the bundleID and content name as the parameters. The *executeModule* call returns the file name of the processed file which is used as the input content for the next service in the chain. This way we can execute multiple services on a single content through chaining, for, e.g, *ccnx://content+service1+service2*; this call will result in two service invocations, service1 on “content” and service2 on the “content+service1” content file.

When the *executeModule* method is called, we load the CCN Service class using the bundle header, `CCN-Service`. After creating an instance of the CCNxService type class, we can invoke the execute method of CCNxService interface directly on this class instance. The invocation returns the file name of the processed file.

Videos are requested as shown below:

```
ant -DCCNX_PREFIX=ccnx:/ -DFILE_PREFIX=/home/dhruva/Videos -DLOGGING=--loggingoff run-fileproxy
Buildfile: build.xml
compile:
jar:
run-fileproxy:
[java] Starting file proxy for /home/dhruva/Videos on CCNx namespace /...
[java] Requested service along with the video: Weather2
[java] Overwriting file: weather2.jar
[java] ccngetfile took: 381ms
[java] Calling dynamic loader
[java] Loading: weather2.jar
[java] Dynamic processing of the video done. File will be served shortly.
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2
[java] Requested service along with the video: Weather2
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2
[java] Requested service along with the video: Weather2
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather2/%FD%04%DC2%D20%00/_meta_/.header
[java] Requested service along with the video: Weather
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather
[java] Requested service along with the video: Weather
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bweather/%FD%04%DC2~%E0%00/_meta_/.header
[java] Requested service along with the video: Advertisement
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bads
[java] Requested service along with the video: Advertisement
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bads
[java] Resulting path name: /home/dhruva/Videos/animal.mp4%2Bads/%FD%04%DC%25%BD%C0%00/_meta_/.header
```

Figure 12.3: A screenshot showing the processing of CCNx content after the Content Router interprets the content request.

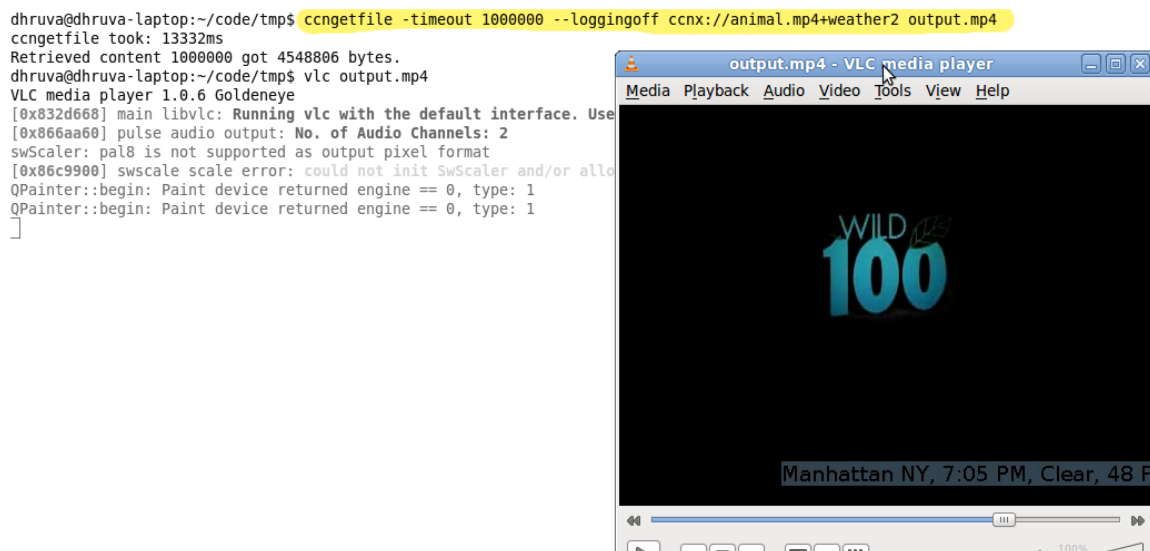


Figure 12.4: The transformed content, with an overlay containing weather information at the bottom right of the video, being played in VLC player.

```
ccngetfile -timeout <value> --loggingoff ccnx://video.mp4+weather <output file>
```

Figures 12.3 and 12.4 show screenshots from the CCNxServ testbed. Figure 12.3 shows the processing of CCNx content after the Content Router interprets the above content request for the weather service invoked on a video file (*ccnx://animal.mp4+weather2*). After intercepting the interest packet, the content name is scanned and the service name “weather2” is detected. An interest for the corresponding service file *weather2.jar* is issued using `ccngetfile`. After successfully downloading the JAR file, the service is dynamically loaded and the content is processed. Finally, the file is sent back to the CCNx layer, which passes the processed content to the requesting user. Figure 12.4 shows the processed content being played at the client device. The invoked services added an overlay containing the local weather information at the bottom right of the video.

12.5 Challenges and Incentives

In this section, we present the economic incentives for the CCNxServ work as well as the technical challenges that we faced while implementing it.

12.5.1 Economic Incentives

Video and content consumption on the Internet are rapidly growing and require enormous amounts of bandwidth. At the network core, the increasing consumption of multimedia content on the existing wired networks and in mobile systems is putting a strain on the network core.[Cisco, 2015]. Content distribution networks (CDNs) are becoming more and more popular as a means of efficiently distributing multimedia content to end-users on the Internet. Pallis and Vakali [Pallis and Vakali, 2006] show that CDNs can reduce “traffic jams” for web traffic, since data is closer to user and there is no need to traverse all of the congested pipes and peering points. Content-centric networks (CCN) take the CDN argument and implementation further with the concept that content, not hosts, are the cornerstone of the Internet today.

CCN proponents [Jacobson *et al.*, 2009b] have argued that CCNs can decrease the cost of content delivery on the Internet. Park et al [Park *et al.*, 2012] argue that a backbone

network provider could see as much as 30 percent traffic reduction using content-centric networks. In this light, the CCNxServ work that we are doing will enable a new class of services to run on top of the emerging CCNx architecture at reduced cost. In addition, as mentioned in Section 12.1, CCNxServ enables lightweight executables to be migrated to locations where there is a large quantity of content or data requests, achieving service mobility to cover large datasets, further reducing networking costs.

12.5.2 Technical Challenges

We will briefly describe some of the technical challenges that we faced while building CCNxServ.

The first challenge was making CCNxServ work on a completely content-centric network infrastructure. Because the entire foundation of CCN networks rests on the concept of content, CCNxServ needs to share services by expressing them as a form of content, and at the same time, use the naming mechanism to indicate their use as executable services and not static content. This can be thought of as introducing active networking into a content-centric network. We believe our mechanism of addressing this solves the services problem while constraining it to work in a content-centric network.

The second challenge was integrating systems that combined the best features of content-centric networking and the traditional networking model, particularly for services. We chose to integrate our CCNxServ implementation with NetServ [Lee *et al.*, 2011b]. We were helped by the fact that while NetServ was initially built for IP networks and for host-based communication, its core service modularity allowed its networking layer to be reworked to support content-centric networking. Through refactoring CCNxServ, we were able to expose its core functionality as a controller to NetServ, and thus leverage NetServ’s service APIs while being able to use CCNxServ’s naming and service delivery mechanism on top of a pure content-centric network. However, this may prove a challenge with frameworks and libraries that have been hard-coded to the IP networking stack, such as OpenFlow [McKeown *et al.*, 2008] and others.

12.6 Related Work

While there has been a lot of work done in the field of information-centric networking (ICN) the field of service-centric networking, especially as applied on ICNs, is fairly new.

SCAFFOLD [Freedman *et al.*, 2010] allows for multiple service instances to be represented by one common name, but with the specific service represented through `serviceIDs` and selected through anycast routing through service routers. MILNGENI (million-node GENI) [University of Washington, 2011] allows for services to be deployed and run on top of many end-systems that are connected via the experimental GENI testbed. Even though these projects attempt to deal with the problem of addressing content and making content requests efficient, they operate on top of the IP layer and require host-to-host communication. Hence, their APIs and middleware are built for host-based networking.

Service-centric networking (SCN) [Braun *et al.*, 2011] is a project aiming to build a network that runs with services as the primary construct, rather than content. It aims to supersede CCNx and thus involves building a superset of CCNx. In contrast, we are building a service platform on top of only CCNx, providing a service stack on top of CCNx, and we have been able to implement a fully service-oriented networking architecture on top of a pure ICN stack. We believe this demonstrates that it is possible to build services on ICN networks without requiring a superset of the features offered in ICN implementations.

SoCCeR [Shanbhag *et al.*, 2011] is an attempt to build a service layer on top of CCNx, and it works as a control layer to manipulate the underlying Forwarding Information Base (FIB), thereby performing distributed best-service selection using an ant colony optimization (ACO) approach. In contrast to our work, it requires modifying the CCNx interest and data packets to enable the control layer to work. In addition, our work deals with service deployment and dynamic scaling, and thus relates to the problem of effective service placement, which would complement SoCCeR's best service selection algorithm. Further, there appears to be no published API or working code for service-centric networking, and hence no working prototypes or services to test.

Another area that is somewhat related to our CCNx services work is in the field of data migration to data centers closer to the services. This work deals with large data sets and data centers, including tackling design issues to address data migration and latency. Tiwana

et al [Tiwana *et al.*, 2010] propose exposing the network location of data to the service, so that the service is able to optimize processing of data based on location. Volley [Agarwal *et al.*, 2010] attempts to automate application data placement across data centers efficiently. PADS [Belaramani *et al.*, 2009] provides a data plane mechanism for transmitting data and maintaining consistency in large distributed applications and data centers. Time-shifted TV [Li and Simon, 2011] uses CCNx to improve localized and cooperative caching using content routers. It takes advantage of proximity features in CCNx (whereby a node that is closer and has the data is automatically chosen over a node that is further away) to do cooperative caching of content. However, all of these approaches only deal with data latency and moving data across locations to bring them closer to the application. Our approach, in contrast, is about moving services themselves to where the data is located, and thus being able to move small executable modules to places with large data, allowing the data to be processing more efficiently.

Research on service placement is also related to our work. Service placement algorithms dynamically try to find the optimal number and locations of service instances given a certain service demand and network topology. A good overview of related work in the area of service placement is given in [Wittenburg, 2010]. In contrast to pure caching solutions, service placement tries to increase the performance of a service based on application-specific quality metrics, while at the same time minimizing the overall network load and service costs. Therefore, services are replicated in the network and the service locations are dynamically adapted to the changing network conditions and service demands.

An architecture supporting a dynamic, distributed service provisioning for mobile users is described in [Lundqvist *et al.*, 2011]. It supports both optimized service placement within an operators network and placement of service components in a foreign network. This leads to both more efficient resource usage and better quality for the users.

12.7 Conclusion

We believe that services (as defined in the introduction) are central to network operations. CCNxServ shows that it is possible to build service functionality including dynamic service

invocation, scalability, and mobility on top of content-centric networking, thus extending its features towards a service-centric network. We have a working implementation of scalable, dynamic service architecture implemented on top of the CCNx protocol stack. Our implementation enables dynamic invocation of services and true service mobility and scaling in a purely content-centric network based on need. By exposing services in a content-centric networking framework and the intuitive use of CCNx's content naming scheme, we are able to provide true content-based service functionality in a future Internet platform that is focussed on providing both content and service functionality.

Chapter 13

IPv6 Addresses For Naming Content

13.1 Introduction

Multimedia content is quickly becoming the dominant Internet application. But the problem of addressing and naming content is one of the most central issues to a practical content-centric networking approach, since a robust and naming-centric networking strategy will enable the building of next-generation Internet architectures that can easily scale and adapt and cache content correctly. In this chapter, we propose using IPv6 addresses for naming content, and we argue that using IPv6 addresses for naming content will allow us to solve the problems of routing and directory services associated with naming.

Several research projects attempt to address this using a variety of means. Naming schemes (such as DONA [Koponen *et al.*, 2007], OCALA [Joseph *et al.*, 2006] and i3 [Kannan *et al.*, 2004]) aim to solve the problem by looking at the aspect of naming, in particular through replacing or complementing the DNS for name-based lookups. Content-centric networking (such as CCNx [Jacobson *et al.*, 2009b] and XIA [Anand *et al.*, 2011]) aim to replace the IP-based Internet stack with one based on content and content names. (These pieces of related work are described in more detail in the related work section.)

We propose solving the content issue through a remarkably simply but counter-intuitive proposal: using IPv6 addresses for content names. Using IPv6 addresses for content names

solves several problems with content networking in a clean way, and at the same time, IPv6 provides an architecture for handling routing and security in a coherent way. In other words, we propose solving the content networking problem by mapping content names to a resource that addresses network problems comprehensively: IPv6 addresses.

In this chapter, we describe our initial proposal for content networking using IPv6 addresses for content names. We describe the motivation and current problems in more detail in Section 13.2. In Section 13.3, we look at some of the fundamental properties of IPv6 that make it so appealing to content-centric networking. In Section 13.5, we specifically address how we map IPv6 addresses to content names, and vice versa. In Section 13.7, we describe an initial prototype implementation and initial performance measurements of the implementation.

13.2 Motivation

Since the beginnings of the Internet as a DARPA project in the 1970s, the Internet has evolved from a small network used in military research to a planetary network connecting billions of people. But at the core of the Internet, the protocols used - such as IP, TCP and UDP - are still point-to-point or host-based and host-centric, and the Internet has, despite its massive growth, remained largely host-centric and application-agnostic.

But as content consumption rapidly increases on the Internet [Cisco, 2015] [PCMagazine, 2011], particularly due to the popularity of audio and video sites and services such as Youtube, Netflix, Pandora, Spotify and iTunes. With the amount of content estimated to be reaching into zetabytes, with a large amount of duplication [Jacobson *et al.*, 2009b], there have been several research projects that have proposed an alternative - and in most cases, a clean slate - approach to solving this problem through a content-centric approach, called content-centric networking.

While most of the existing work focusses on replacing existing Internet protocols with a new set of protocols based on content, one key issue stands out: would content routers scale for large volumes of content?

Our proposal is to use IPv6 addresses as content names, whereby we use part of the

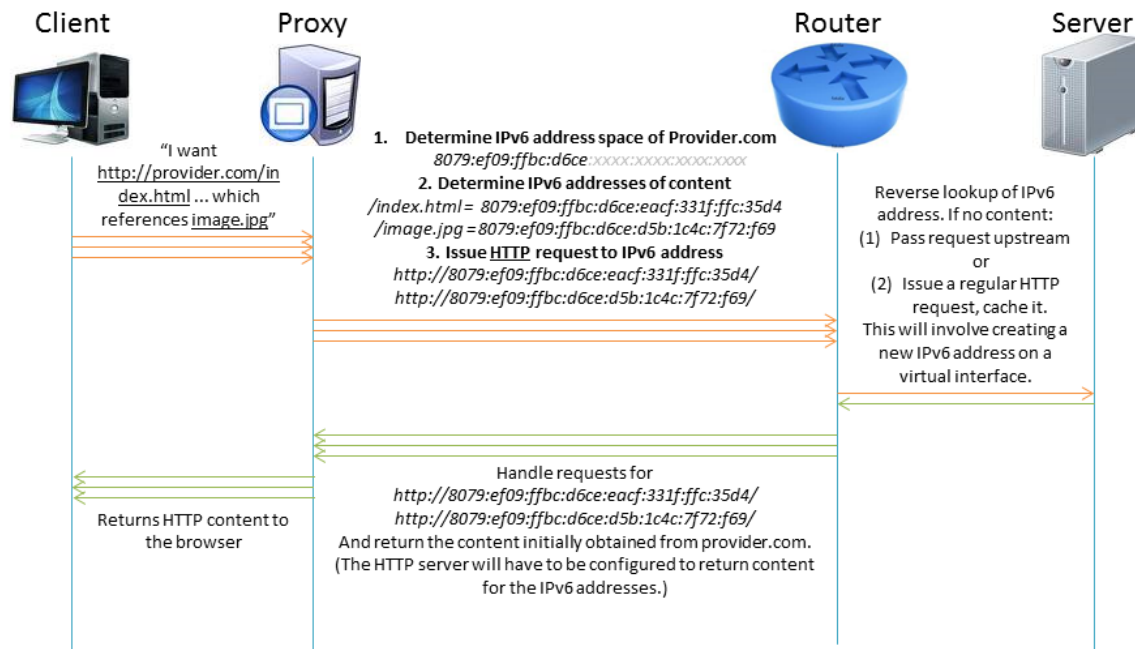


Figure 13.1: The architecture diagram of our IPv6 content addressing system. In our system, the regular browser makes a HTTP request through a proxy, which translates HTTP requests to an IPv6 content addressing system. The request is sent out over the network, until a router on path that has the content responds to the request. The proxy then translates the retrieved content back into a HTTP response to the user's browser.

IPv6 address space as content names, thus allowing IPv6 to function as a content-centric networking layer.

By using this approach, we no longer have to replace the Internet stack, and can instead use the existing Internet architecture and move towards a full-fledged information-centric architecture that uses IPv6 addresses as content identifiers. This solves the problem of the clean-slate approach, which would involve a loss of existing massive investment of networking infrastructure in order to retrofit a new networking architecture, and allows us to re-use the existing network stack for content networking.

13.3 IPv6 to the Rescue

In our approach, we favor using IPv6 unicast addresses. The IPv6 standard [Deering and Hinden, 1998] specifies using 64 bits for routing and 64 bits for the interface ID, with a recommendation to use 48 bits for the routing prefix, 16 bits for the subnet ID and 64 bits for the interface ID. The routing prefix is allocated globally similar to IPv6. The subnet ID is used to allow local network administrators to partition their networks into subnets based on need. The 64-bit interface ID is usually generated automatically from the device's MAC address, or assigned through a DHCP server.

Our content-to-IPv6 proposal requires no changes to the initial 64 bits - the 48 bit routing prefix or the 16 bit subnet ID. The first 64 bits are used to identify the content publisher. We only focus on the 64-bits for the interface ID and use that as a unique content identifier.

Having established this, we would be able to map IPv6 addresses to content names and vice versa. For example, an IPv6 address such as 2001:0db8:85a3:0000:0000:8a2e:0370:7334 could refer to a content identifier "publisher.com/example/test.html". Further, the name doesn't have to be a HTTP or other URL, it could be a CCNx or other name as well (though in our implementation, we do focus on HTTP URLs for reasons described in Section 13.3.)

There are several features that we get for "free" from the IPv6 networking stack once we have this implementation. The following are only some examples.

Routing Aggregation or "supernetting" in IPv6 is a useful feature for our content mapping. It aggregates routes to smaller networks, and thus increases convergence speed of routing messages across routers. Assuming that requested content is popular or viral, this setup enables speedy recovery of content from a nearby cache.

IPv6 also specifies IPsec as part of its security framework. It allows for the payload to be encrypted while the header is not. This could be used to encrypt the contents of the data packet inherently in the network, and would be similar to how HTTPS allows for secure HTTP traffic over SSL. It may further be possible to use authentication header to protect IP address itself (naming security).

There are several other small features that are very useful for content networking. Multicasting is guaranteed to be always enabled in IPv6, and this is very useful for multicast

content such as popular TV shows or sports games.

In regards to router processing of IPv6 packets, there is no fragmentation of packets, and hence transfer of packets is much more efficient. We can specify the use of jumbograms, which allow efficient transfer of large amounts of content through large data packets. IPv6 avoids triangular routing, and hence it works well with mobile nodes and mobile networks, alleviating the need for a shim layer for mobile IP and similar problems.

Finally, we can transfer IPv6 over IPv4 networks through the use of the many so-called "6on4" proposals (named as such because they identify how to run IPv6 addressing and networking on top of existing IPv4 networks) and hardware that exist, thus ensuring that content routing and delivery works even on IPv4 networks.

13.4 Implementation

In the next section, we describe our method of mapping content names to IPv6 addresses, and vice-versa. But before we delve into that, we would like to highlight the issue of naming security. Recent literature show that there is interest in securing the name in addition to the content. We would like to highlight that this would be possible in the IPv6 content naming architecture.

In our current architecture, we use a central lookup mechanism to map content names to IPv6 addresses and resolve them (we envision that this may be replaced by DNS hierarchy mechanisms, such as NAPTR [Mealling and Daniel, 2000]). Our current implementation thus keeps the content name "open", meaning that anyone along the path would be able to do a reverse-lookup and find the human-readable name of the content.

Our current implementation uses HTTP on top of the IPv6 stack. In order for this to work, we assume the presence of several nodes along the route that are able to detect these IPv6 packets and respond to them (an example of this would be NetServ nodes, described in Chapter 8, that have an IPv6 content identifier module.) The client-side browser is configured to use our IPv6 proxy, which receives all the HTTP requests for content and translates them into IPv6 addresses, which it obtains from our central lookup server. New IPv6 packets are sent out to the network, which contain the content name, with the IPv6

packet itself having the sender IP as the source address, and the IPv6 content name as the destination address. When the IPv6 packet hits a router that recognizes this IPv6 content packet, the router checks its internal cache to see whether it has the corresponding content. If it does, it performs a TCP termination for that connection, and sends a response with the content. If it does not, the router allows the packet to pass through, where in the worst-case scenario, it reaches the origin server which is able to properly handle the request. We use Cassandra [Apache Foundation, 2012] as the central lookup store.

13.5 Mapping Content Names to IPv6, and Vice Versa

Even though we use IPv6, which can theoretically support up to 2^{128} content names, we face some challenges. First, there is an existing hierarchy for IPv6 address allocations, and these are performed by the IANA. Hence we are not able to arbitrarily assign IPv6 addresses for our content. Hence, we will need some sort of prefix identifier to identify that the IPv6 addresses are actually referring to content names, so that they do not conflict with pre-existing and assigned IPv6 addresses.

In addition, we need to be able to segment the IPv6 address so as to be able to differentiate between the protocol, publisher and the content name (corresponding to that publisher). This will allow for uniformity in names, as well as separation of functionality and naming.

Finally, we require a centralized (perhaps hierarchical) mechanism for storing and looking up IPv6 addresses corresponding to content names. The reason is that even though the address space for IPv6 is 2^{128} and hence virtually unlimited, we need to be able to map the publisher and content names to an IPv6 address using hashing or a similar mechanism. In this situation, the hash cannot be reversed, and hence we need a lookup or mapping mechanism similar to DNS for IPv6 content names. Note that while we mentioned using a Cassandra database for the mapping earlier, that is just at one local node, so we need some way to make sure that this name-to-IPv6 two-way lookup can be done globally.

Our current mapping of content names to IPv6 addresses is as follows:

- 16 bits = protocol (such as HTTP, etc)

- 48 bits = publisher name
- 64 bits = content name

Note that we allocate 16 bits for the protocol to allow the publisher to run more sophisticated networking than just plain HTTP. We believe that $2^{16} = 65,536$ possible protocol listings will suffice for identifying a specific protocol. We also believe that there are less than 2^{48} publishers in the known universe. Finally, we believe that 2^{64} possible permutations suffice for the individual content names corresponding to each publisher.

We implement the mapping of content names to IPv6 addresses as follows: we parse the content name into protocol, publisher name and content name. An MD5 hash of each of these strings is then performed. The first 16, 48 and 64 bits (respectively) of each MD5 hash is taken and used for the corresponding parts of the IPv6 address.

Using this implementation, we obtain the following sample IPv6 addresses for the following content names:

d6de:490c:6a8e:b10d:8c7d:d922:ad47:494f	ccnx:// <i>parc.com/file</i>
d6de:490c:6a8e:b10d:d41d:8cd9:8f00:b204	ccnx:// <i>parc.com/</i>
8079:1d59:20f4:b44b:d41d:8cd9:8f00:b204	http:// <i>google.com/</i>
8079:9ee:a68a:92d6:d41d:8cd9:8f00:b204	http:// <i>epochtim.es/</i>
8079:1b37:2650:3af8:1d78:a723:dee0:2522	http:// <i>nyt.ms/video</i>
8079:1b37:2650:3af8:eacf:331f:ffc:35d4	http:// <i>nyt.ms/</i>

13.6 Potential Improvements

Our current implementation allows for a decentralized lookup mechanism for content names corresponding to IPv6 addresses.

We envision most IPv6 content name addresses as being public, and we foresee that there might be a need to hide certain content names for security and other purposes. In these scenarios, we believe that it is possible to have public and private lookup mechanisms for the content address. One way of doing this could be through using portions of the publisher or content name be assigned privately (similar to how IPv6 allows a subset of its

address to be used for subnet IDs), and those references could work only within a private network.

In addition, we could secure certain parts of the IPv6 address namespace. For instance, the protocol and the publisher names could be looked up and made public, while the content name part of the address could be made private and lookup possible only after one is authorized to be able to do so.

13.7 Implementation

We have implemented a prototype of the system in the Python programming language. The implementation uses libraries for packet capturing and packet spoofing. The implementation has two main components, a content proxy and a content router. The content proxy acts as an HTTP proxy that converts HTTP URL requests into IPv6 addresses. Upon getting the request from the browser, we call a central lookup service to get the IPv6 address corresponding to the content name. If the IPv6 address is retrievable, we create an IPv6 request packet to get the content. In addition, if we are unable to retrieve the IPv6 address, the content name is converted into IPv6 address and added to the central lookup service as an `ipv6-to-content_name` mapping and `content_name-to-ipv6` mapping for content router lookup.

ContentProxy:

```
while(True)
    client = socket.accept()
    content_name = get_content(client)
    ipv6 = central_lookup.get(content_name)
    if(ipv6 not present)
        ipv6 = convert_content_to_ipv6(content_name)
        central_lookup.put(ipv6, content_name)
        central_lookup.put(content_name, ipv6)
    data = getContent(ipv6)
    client.send(data)
```

The content router uses the NetFilter API to capture IPv6 packets. We examine the packets using the Python Scapy library. First, we initialize the NetFilter queue with a callback function to capture all IPv6 packets. Once the packet is captured in the callback function, the destination of the IPv6 packet is compared with the corresponding file name in the router's local storage. If a match is found, the packet is dropped and the file is returned to the client. If the file is not found in the local store, we then forward that packet further or we fetch the content directly from the publisher and store it locally for future requests.

ContentRouter :

```

queue = nfqueue.queue()
queue.bind(socket.AF_INET6)
queue.set_callback(process_packet,6)
queue.create_queue(0)
while(True)
    queue.try_run()

process_packet(payload)
    packet = payload.get_data()
    destination = get_destination(packet)
    source = get_source(packet)
    file = check_local_storage(destination)
    if (file)
        return source.send(file)
    else if(forward)
        forward(payload)
    else
        file = fetch_content(central_lookup.get(
            destination))
        store_locally(file)
        return source.send(file)

```

Our test network topology consisted of eight nodes connected in a local network. We used eight nodes with two of the nodes being the end nodes, and some of the nodes being simple routers, and one or two possessing IPv6 content handling functionality. We were able to get the requests to work over this network.

13.8 Related Work

The biggest differences between our work and the existing work in information-centric networking (ICN) is that our IPv6 implementation allows for running a pure ICN network on top of today's Internet architecture, using existing hardware and protocols. In contrast, almost all the work done so far in ICN require a clean-slate approach to networking, which would translate to billions or trillions of dollars in infrastructure upgrades to replace our current Internet.

For instance, naming schemes (such as DONA [Koponen *et al.*, 2007], OCALA [Joseph *et al.*, 2006] and i3 [Kannan *et al.*, 2004]) attempt to solve the content problem by looking at the aspect of naming. They introduce a level of indirection in the naming scheme to get around the content and service addressability aspect of the Internet. Content-centric networking (such as CCNx [Jacobson *et al.*, 2009b] and XIA [Anand *et al.*, 2011]) aim to replace the IP-based Internet stack with one based on content and content names. While some of these protocols - such as CCNx - can run on top of IP networks, they tend to require their own network infrastructure to be able to truly deliver content delivery benefits.

One other area of related work is in the area of naming security. In current protocols such as HTTP, security (such as SSL/TLS) secure the content but not the name. Work such as [Atkinson *et al.*, 2010] and [Dannewitz *et al.*, 2010] aim to preserve privacy of content names in the network. Almost all of these would require some sort of credential validation to be done on the network.

There are other research projects that attempt to split the concept of ID and location. LISP (Location/ID Separation Protocol) [Farinacci, 2013] and [LISP, 2014] is an IETF proposal that deals with a naming system that would configure host-names and machine network address identifiers and map them to locators. Virtual ID [So-In *et al.*, 2010] is

an extension to IPv6 that allows ID/location split techniques to be used in mobile IPv6 networking. These deal with removing the coupling between the identifier and location, but don't specifically address content naming, though they could be extended for similar purposes.

13.9 Conclusion

In this chapter, we described our proposal for using IPv6 addresses for content naming, and describe how IPv6 can solve a host of issues traditionally associated with this information-centric networking. We also presented an initial implementation of the same. We believe that using IPv6 addresses for naming content will allow for a robust naming scheme for the future Internet, while addressing the fundamental issues of caching and routing that are an anathema to current naming-based schemes.

Part IV

Conclusions

Chapter 14

Conclusions

The Internet, which came into being in the 1970s and 1980s, was initially designed to carry small volumes of information, such as e-mails and networking packets. But with the introduction of the World Wide Web and HTTP in the 1990s, the Internet transformed into a platform for transferring richly formatted content through HTML. Today, almost 30 to 40 years after the Internet first came into existence, it is a global consumer network carrying zetabytes of information, primarily video and multimedia traffic, to billions of consumers over wireless, wired and cellular networks. Traditional communication platforms, such as telephone networks and cable, which initially enabled the Internet Protocol and the Internet itself, are now simply becoming services deployed on top of the IP network.

As the Internet and networking change, there are two specific forms of changes that I believe are most important, and those two fields form the core of the research topics in my thesis.

First, there is a rapid growth of powerful handheld devices such as smartphones and tablets, and use of cellular and wireless networks is becoming pervasive. Telecommunications players are aiming to bring their cellular services to every corner of the world, but there are still areas that they cannot yet reach and may be economically unfeasible for them. This opens up interesting application scenarios for opportunistic networks. In my thesis, I have explored several ways to enable real-world applications on top of such mobile opportunistic networks.

Second, multimedia consumption over the Internet is experiencing massive growth,

driven by online services such as Youtube, Facebook, Pandora and iTunes. With larger and larger data volumes being pushed over the Internet, advances are necessary in the fields of Content Delivery Networks (CDNs) and Content-Centric Networking (CCNs) to make sure that multimedia and large volumes of data are being efficiently transferred over the Internet. In this thesis, I've presented several methods of improving efficient delivery of multimedia content over the Internet, either using existing IP protocols, content-centric networks, or a hybrid of both.

14.1 Opportunistic Networks

To enable efficient delivery of content on opportunistic networks, we have to solve the problem at three layers: the networking protocol, the application layer and by providing a library or framework that allows for easy development of applications in this network.

Our analysis of Zero Configuration Networking (ZeroConf) protocol on wireless networks (Chapter 5) enabled us to implement a method to enhance the ZeroConf framework and allow it to function correctly in opportunistic networks. We also saw an increasing use of service discovery and content delivery protocols, particularly multicast DNS (mDNS), in real-world networks (in our case, our university campus), driven by popular applications such as iTunes.

The Seven Degrees of Separation (7DS) suite of applications (3) provides end-user application services in mobile opportunistic networks where data packets could get delayed during transit. 7DS provides the necessary transport and application layer functionality for mobile nodes to exchange information using store-carry-forward communication. Using this application suite, it is possible for users on handheld devices and in opportunistic networks to communicate meaningfully.

Developing mobile applications that function properly in opportunistic networks is a difficult process, since opportunistic networks do not follow the client-server model of operation. Even peer-to-peer models of networking do not work in such a network due to the high churn rate of mobile nodes in such networks. I designed and implemented a software library called BonAHA (Bonjour for Ad-Hoc Applications) (Chapter 4) which provides an

API for building applications that run in such opportunistic networks. BonAHA is easy and intuitive, provides a level of abstraction to opportunistic networking and at the same time provides flexibility to the developer to develop applications that run in these networks. The thesis also describes several real-world applications (outside of 7DS) built on top of the BonAHA library, and describes how it is possible for a developer to build their own applications using the BonAHA library.

14.2 Content delivery

In my thesis, content delivery improvements at the Internet core are addressed in a few different ways. First, to enable dynamic deployment of nodes necessary to allow for optimal delivery and caching of content in response to real-time traffic changes, we look into service virtualization using NetServ, and building a content-delivery network (ActiveCDN) on top of this service virtualization framework. Then, I explore a method of enabling dynamic deployment without an additional service-virtualization layer, through On-Path CDNs.

The thesis introduces the first prototype for NetServ (Chapter 8), a research effort to design an extensible architecture for core network services for the next generation Internet. NetServ enables service virtualization at the Internet core, thus allowing efficient use of applications such as content delivery networks (CDNs). This NetServ prototype demonstrates the possibilities that active networking and service functionality at the network core can open up.

Combining the NetServ service virtualization framework and in-network, dynamic content delivery, ActiveCDN (Chapter 9), a NetServ module, can dynamically deploy CDN modules and serve content from participating NetServ nodes that are near the edge of the network. This work was selected and demonstrated as part of the NetServ modules at the National Science Foundation's 8th and 9th GENI Engineering Conferences (GEC8 and GEC9). ActiveCDN allows for content providers to dynamically deploy CDN nodes across the Internet based on demand, thus alleviating traffic load in core networks.

It is also possible to innovate at the current network core, without using an additional service virtualization layer. I implemented on-path content delivery networks (Chapter 7),

which can dynamically intercept and redirect requests for content as well as serve content from a local cache using existing TCP traffic. The measurements and analysis of this implementation using popular content providers indicate that it is possible to deploy on-path CDNs while incurring only a relatively small overhead.

14.3 Content Centric Networks

Revisiting the question of in-network service virtualization, it is possible to perform such virtualization on top of a new model of content networking, called content-centric networking. While content-centric networks allow for efficient distribution of content and make content the center of the networking stack, they do not properly or correctly handle the issues of services. In my thesis, I have addressed the topic of running dynamic content-oriented services on content-centric networking (Chapter 12). The architecture and implementation of a prototype (CCNxServ) were presented, and our prototype allows for building services on top of a pure content-centric networking stack.

While looking at content-centric networks, we need to also be aware of the fact that most forms of content-centric networking envision some sort of clean-slate Internet architecture that invalidates years of investment and development on the existing host-based IP networks. But by using our proposed IPv6-based content networking (Chapter 13), we could use the address space of the existing host-based networking as content identifiers, and thus allow content-centric networks to co-exist with the existing host-based and IP networking stack.

14.4 Real world video traffic patterns

Finally, I presented insight into trends in video traffic on the Internet (Chapter 10) using real-world data that I gathered and analyzed while working at LongTail Video, which makes the popular JWPlayer video player used on a large number of popular websites. This evaluation helps us understand the nature of real-world video traffic patterns and can allow us to optimize content delivery strategies as Internet content grows.

Part V

Bibliography

Bibliography

- [7DS Homepage, 2005] 7DS project home page. <http://www.cs.columbia.edu/IRT/>, 2005.
- [Abley and Lindqvist, 2006] J. Abley and K. Lindqvist. Operation of anycast services. BCP 126, RFC Editor, December 2006.
- [Aboba *et al.*, 2006] B. Aboba, J. Carlson, and Stuart Cheshire. Detecting network attachment in ipv4 (dnav4). RFC 4436, RFC Editor, March 2006.
- [Adhikari *et al.*, 2010] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. Youtube traffic dynamics and its interplay with a tier-1 isp: an isp perspective. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, 2010.
- [Adhikari *et al.*, 2011] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. How do you 'tube'. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, 2011.
- [Agarwal *et al.*, 2010] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Harbinder Bhogan. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *Proceedings of the 7th USENIX symposium on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, April 2010.
- [Akamai, 2008] Akamai. Akamai introduces advertising decision solutions. http://www.akamai.com/html/about/press/releases/2008/press_102108.html, October 2008. [Online; accessed 26-March-2016].

- [Akamai, 2015] Akamai. NetSession. <https://www.akamai.com/us/en/solutions/products/media-delivery/netsession-interface-faq.jsp>, 2015. [Online; accessed 20-March-2016].
- [Akkus *et al.*, 2006] Istemi Ekin Akkus, Oznur Ozkasap, and M. Reha Civanlar. Secure transmission of video on an end system multicast using public key cryptography. In *Lecture Notes in Computer Science*, pp. 603-610, Volume 4105, 2006.
- [Alex *et al.*, 1998] D. Scott Alex, William A. Arbaugh, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles, and Jonathan M. Smith. The SwitchWare Active Network Architecture. *IEEE Network*, May 1998.
- [Alliance, 2015] Allseen Alliance. AllJoyn API. <https://allseenalliance.org/framework/documentation/develop/tutorial/core>, 2015. [Online; accessed 20-March-2016].
- [Alliance, 2016] Allseen Alliance. Alljoyn. <https://allseenalliance.org/framework>, 2016. [Online; accessed 20-March-2016].
- [Almeida, Virgílio and Bestavros, Azer and Crovella, Mark and Oliveira, Adriana de, 1996] Almeida, Virgílio and Bestavros, Azer and Crovella, Mark and Oliveira, Adriana de. Characterizing reference locality in the WWW. In *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.
- [Amazon, a] Amazon. Amazon cloudfront. <http://aws.amazon.com/cloudfront/>. [Online; accessed 26-March-2016].
- [Amazon, b] Amazon. Amazon simple storage service (s3). <http://aws.amazon.com/s3/>. [Online; accessed 26-March-2016].
- [Anand *et al.*, 2011] Ashok Anand, Fahad Dogar, Dongsu Han, Boyan Li, Hyeontaek Lim, Michel Machado, Wenfei Wu, Aditya Akella, David Andersen, John Byers, Srinivasan Seshan, and Peter Steenkiste. XIA: An Architecture for an Evolvable and Trustworthy

- Internet. In *Tenth ACM Workshop on Hot Topics in Networks (HotNets-X)*, November 2011.
- [AOLMessenger, 2006] Aol instant messenger. <http://www.AIM.com/>, 2006. [Online; accessed 26-March-2016].
- [Apache Foundation, 2008] Apache Foundation. Apache Felix. <http://felix.apache.org/>, 2008. [Online; accessed 26-March-2016].
- [Apache Foundation, 2012] Apache Foundation. Apache Cassandra. <http://cassandra.apache.org/>, 2012. [Online; accessed 26-March-2016].
- [ApacheBench, 2008] ApacheBench. ApacheBench. <http://httpd.apache.org/docs/2.0/programs/ab.html>, 2008. [Online; accessed 26-March-2016].
- [Apple, 2009] Apple. GameKit Framework Reference. https://developer.apple.com/library/ios/documentation/GameKit/Reference/GameKit_Collection/, 2009. [Online; accessed 20-March-2016].
- [Apple, 2012] Apple. Apple airdrop. <https://en.wikipedia.org/wiki/AirDrop>, 2012. [Online; accessed 26-March-2016].
- [Atkinson *et al.*, 2010] R. Atkinson, S. Bhatti, and S. Hailes. Evolving the internet architecture through naming. *IEEE Journal on Selected Areas in Communications*, October 2010.
- [Belaramani *et al.*, 2009] Nalini Belaramani, Jiandan Zheng, Amol Nayate, Robert Soulé, Mike Dahlin, and Robert Grimm. PADS: a policy architecture for distributed storage systems. In *Proceedings of the 6th USENIX symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, April 2009.
- [Bent Guldbjerg Christensen, 2007] Bent Guldbjerg Christensen. LightPeers: A Lightweight Mobile P2P Platform. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007.
- [Berners-Lee *et al.*, 1994] Tim Berners-Lee, Larry Masinter, and Mark McCahill. Uniform resource locators (url). RFC 1738, RFC Editor, December 1994.

- [Bonjour, 2005] Apple Computer's Bonjour. <http://developer.apple.com/networking/bonjour/>, 2005. [Online; accessed 26-March-2016].
- [Braun *et al.*, 2011] Torsten Braun, Volker Hilt, Markus Hofmann, Ivica Rimac, Moritz Steiner, and Matteo Varvello. Service-centric networking. In *2011 IEEE International Conference on Communications Workshops (ICC)*, June 2011.
- [Broberg *et al.*, 2009] J. Broberg, R. Buyya, and Z. Tari. MetaCDN: Harnessing "storage clouds" for high performance content delivery. In *Journal of Network and Computer Applications, Volume 32, Issue 5*, September 2009.
- [CDNPricing, 2011] Cdnpricing.com, 2011. [Online; accessed 26-March-2016].
- [Center, 2011] Qualcomm Innovation Center. Peer-to-Peer Technology: Driving Innovative User Experiences in Mobile. http://cdn.oreillystatic.com/en/assets/1/event/61/Peer-to-Peer%20Technology_%20Driving%20Innovative%20User%20Experiences%20in%20Mobile%20Presentation.pdf, 2011. [Online; accessed 20-March-2016].
- [Chen and Chan, 2001] J Chen and SHG Chan. Multipath routing for video unicast over bandwidth-limited networks. In *Proc. of GLOBECOM'01*, 2001.
- [Chen *et al.*, 2002] Y. Chen, R. Katz, H. Randy, and J. Kubiawicz. Scan: A dynamic, scalable, and efficient content distribution network. In *Proceedings of the First International Conference on Pervasive Computing (Pervasive '02)*, 2002.
- [Cheshire and Krochmal, 2013a] Stuart Cheshire and Marc Krochmal. Dns-based service discovery. RFC 6763, RFC Editor, February 2013.
- [Cheshire and Krochmal, 2013b] Stuart Cheshire and Marc Krochmal. Multicast DNS. RFC 6762, RFC Editor, February 2013.
- [Cisco, 2009a] Cisco. Cisco network virtualization. http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns431/ns658/net_qanda0900aecd804a16ae.html, 2009.

- [Cisco, 2009b] Cisco. Service-oriented network architecture. http://www.cisco.com/en/US/netsol/ns629/networking_solutions_packages_list.html, 2009. [Online; accessed 26-March-2016].
- [Cisco, 2015] Cisco. Cisco visual networking index: Forecast and methodology, 2014-2019. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html, 2015. [Online; accessed 26-March-2016].
- [Civicspacelabs, 2008] Civicspacelabs. civicspacelabs.org. http://civicspacelabs.org/services_main, 2008. [Online; accessed 26-March-2016].
- [Click Website, 2009] The Click Modular Router Project. <http://read.cs.ucla.edu/click/>, 2009. [Online; accessed 26-March-2016].
- [CloudFlare, 2011] CloudFlare. A Brief Primer on Anycast. <https://blog.cloudflare.com/a-brief-anycast-primer/>, 2011. [Online; accessed 20-March-2016].
- [Cohen, Bram, 2004] Cohen, Bram. BitTorrent protocol. <http://wiki.theory.org/BitTorrentSpecification>, 2004. [Online; accessed 26-March-2016].
- [Costa *et al.*, 2004] Cristiano P. Costa, Italo S. Cunha, Alex Borges, Claudiney V. Ramos, Marcus M. Rocha, Jussara M. Almeida, and Berthier Ribeiro-Neto. Analyzing client interactivity in streaming media. In *Proceedings of the 13th international conference on World Wide Web, WWW '04*, 2004.
- [Crocker, 2009] D. Crocker. Internet mail architecture. RFC 5598, RFC Editor, July 2009.
- [Cygwin, 2007] Cygwin toolkit, 2007. [Online; accessed 26-March-2016].
- [Dannewitz *et al.*, 2010] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren. Secure naming for a network of information. In *IEEE INFOCOM 2010*, March 2010.
- [DarwinPorts, 2007] DarwinPorts, 2007. [Online; accessed 26-March-2016].
- [Deering and Hinden, 1998] Stephen Deering and Robert M. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460, RFC Editor, December 1998.

- [Docker, 2016] Docker. Docker. <https://www.docker.com/>, 2016. [Online; accessed 20-March-2016].
- [Droms, 1997] Ralph Droms. Dynamic host configuration protocol. RFC 2131, RFC Editor, March 1997.
- [Eclipse Foundation, 2007] Eclipse Foundation. Eclipse Equinox. <http://www.eclipse.org/equinox/>, 2007. [Online; accessed 26-March-2016].
- [Egi *et al.*, 2007] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerd, Laurent Mathy, and Tim Schooley. Evaluating Xen for Router Virtualization. In *Computer Communications and Networks (ICCCN)*, pages 1256–1261, 2007.
- [Engineering and Technology Magazine, 2014] Engineering and Technology Magazine. Poor connectivity at football matches solved. <http://eandt.theiet.org/news/2014/aug/football-tribehive.cfm>, 2014. [Online; accessed 20-March-2016].
- [Facebook, 2004] Facebook. Facebook. <http://www.facebook.com/>, 2004. [Online; accessed 26-March-2016].
- [Farinacci, 2013] D. Farinacci. The locator/id separation protocol (lisp). RFC 6830, RFC Editor, January 2013.
- [Fielding *et al.*, 1999] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999.
- [Fielding, 2000] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, 2000.
- [Finamore *et al.*, 2011] Alessandro Finamore, Marco Mellia, Maurizio M. Munafò, Ruben Torres, and Sanjay G. Rao. Youtube everywhere: impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, IMC '11*, 2011.
- [FireChat1, 2015] Firechat on google play, 2015. [Online; accessed 26-March-2016].

- [FireChat2, 2015] Firechat demo, 2015. [Online; accessed 26-March-2016].
- [Fortune, 2015] Fortune. How Facebook’s video-traffic explosion is shaking up the advertising world. <http://fortune.com/2015/06/03/facebook-video-traffic/>, June 2015. [Online; accessed 26-March-2016].
- [Freedman *et al.*, 2004] M. J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coralCDN. In *USENIX 1st symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
- [Freedman *et al.*, 2010] Michael J. Freedman, Matvey Arye, Prem Gopalan, Steven Y. Ko, Erik Nordstrom, Jennifer Rexford, and David Shue. Service-Centric Networking with SCAFFOLD. Technical report, Princeton University, September 2010.
- [Funai *et al.*, 2016] Colin Funai, Cristiano Tapparello, and Wendi B. Heinzelman. Supporting multi-hop device-to-device networks through wifi direct multi-group networking. *CoRR*, abs/1601.00028, 2016.
- [Gill *et al.*, 2007] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: A view from the edge. In *ACM SIGCOMM IMC (Internet Measurement Conference) 2007*, 2007.
- [GitHub, 2014] GitHub. Openpeer Documentation. <https://github.com/openpeer/op-docs>, 2014. [Online; accessed 20-March-2016].
- [GitHub, 2016] GitHub. GitHub Enterprise. <https://enterprise.github.com/home>, 2016. [Online; accessed 20-March-2016].
- [Gnutella, 2004] Gnutella. Gnutella. <http://www.the-gdf.org/>, 2004.
- [Goland *et al.*, 1999] Y. Goland, Ting Cai, Paul Leach, and Ye Gu. Simple service discovery protocol. Technical report, IETF, 1999.
- [Golrezaei *et al.*, 2011] Negin Golrezaei, Karthikeyan Shanmugam, Alexandros G. Dimakis, Andreas F. Molisch, and Giuseppe Caire. Femtocaching: Wireless video content delivery through distributed caching helpers. <http://arxiv.org/abs/1109.4179>, September 2011. [Online; accessed 26-March-2016].

- [GoogleTalk, 2006] Google talk. <http://www.google.com/talk/>, 2006. [Online; accessed 26-March-2016].
- [Gosling *et al.*, 2005] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.
- [Guttman *et al.*, 1999] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. Service location protocol, version 2. RFC 2608, RFC Editor, June 1999. [Online; accessed 26-March-2016].
- [Halepovic and Deters, 2002] Emir Halepovic and Ralph Deters. Building a P2P forum system with JXTA. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 41. IEEE Computer Society, 2002.
- [Hancock *et al.*, 2005] R. Hancock, G. Karagiannis, J. Loughney, and S. Van den Bosch. Next steps in signaling (nsis): Framework. RFC 4080, RFC Editor, June 2005.
- [Hayes, Anna and Wilson, David, 2004] Hayes, Anna and Wilson, David. Peer-to-Peer Information Sharing in a Mobile Ad Hoc Environment. In *Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications*, 2004.
- [He *et al.*, 2008] Jiayue He, Rui Zhang-shen, Ying Li, Cheng Yen Lee, Jennifer Rexford, and Mung Chiang. DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In *Proceedings of CoNEXT*, 2008.
- [Heinemann *et al.*, 2003] Andreas Heinemann, Jussi Kangasharju, Fernando Lyardet, and Max Muhlhauser. iClouds – Peer-to-Peer Information Sharing in Mobile Environments. In Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors, *Euro-Par 2003. Parallel Processing, 9th International Euro-Par Conference*, volume 2790 of *Lecture Notes in Computer Science*, pages 1038–1045. Springer, 2003.
- [Hellstrom and Jones, 2005] G. Hellstrom and P. Jones. RTP Payload for Text Conversation. RFC 4103, RFC Editor, June 2005.

- [Hong *et al.*, 2007] Se Gi Hong, Suman Srinivasan, and Henning Schulzrinne. Accelerating Service Discovery in Ad-hoc Zero Configuration Networking. In *IEEE Conference on Global Communications (GLOBECOM)*, Nov 2007.
- [Hong *et al.*, 2009] Se Gi Hong, Suman Srinivasan, and Henning Schulzrinne. Measurements of Multicast Service Discovery in a Campus Wireless Network. In *IEEE Globecom 2009*, December 2009.
- [Horton and Adams, 1987] M.R. Horton and R. Adams. Standard for interchange of usenet messages. RFC 1036, RFC Editor, December 1987.
- [Hosanagar *et al.*, 2004] K. Hosanagar, R. Krishnan, M. Smith, and J. Chuang. Optimal pricing of content delivery network (CDN) services. In *International Conference on System Sciences*, 2004.
- [Housel *et al.*, 2004] Barron C. Housel, George Samaras, and David B. Lindquist. Webexpress: A client/intercept based system for optimizing web browsing in a wireless environment. *Mobile Networks and Applications*, 2004.
- [Howl, 2003] Porchdog Software’s Howl. <http://www.porchdogsoft.com/products/howl/>, 2003. [Online; accessed 26-March-2016].
- [Huang *et al.*, 2008] C. Huang, A. Wang, J. Li, and K.W. Ross. Measuring and evaluating large-scale CDNs. In *Internet Measurement Conference (IMC)*, November 2008.
- [Ibing and Boche, 2012] A. Ibing and H. Boche. Ad hoc mobile devices and ad hoc networks, May 2012. US Patent App. 12/941,861.
- [iChat, 2006] Apple ichat. <http://www.apple.com/macosx/features/ichat.html>, 2006. [Online; accessed 26-March-2016].
- [icq, 2006] Icq. <http://www.icq.com/>, 2006. [Online; accessed 26-March-2016].
- [Internet Movie Database, 2014] Internet Movie Database. Internet Movie Database. <https://www.imdb.com/>, 2014. [Online; accessed 14-November-2015].

- [iptables, 2008] iptables. <http://www.netfilter.org/>, 2008. [Online; accessed 26-March-2016].
- [Jacobson *et al.*, 2009a] Van Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. L. Braynard. Networking named content. In *Proc. of CoNEXT '09*, December 2009.
- [Jacobson *et al.*, 2009b] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proc. of CoNEXT '09*. ACM, December 2009.
- [Jelenkovic, Predrag and Radovanovic, Ana, 2003] Jelenkovic, Predrag and Radovanovic, Ana. Asymptotic Insensitivity of Least-Recently-Used Caching to Statistical Dependency. citeseer.ist.psu.edu/jelenkovic03asymptotic.html, 2003. [Online; accessed 26-March-2016].
- [John D. W. Brothers / Nortel Networks, 2002] John D. W. Brothers / Nortel Networks. Method and system for redirecting web page requests on a tcp/ip network, 2002.
- [John Markoff, 2010] John Markoff. Scientists Strive to Map the Shape-Shifting Net. In *New York Times*, March 2010.
- [Joseph *et al.*, 2006] Dilip Joseph, Jayanth Kannan, Ayumu Kubota, Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle. Ocala: an architecture for supporting legacy applications over overlays. In *Proc. of NSDI'06*, pages 20–20. USENIX Association, 2006.
- [Juniper, 2010] Juniper. Juniper Networks Partner Solution Development Platform. <http://www.juniper.net/us/en/products-services/nos/junos/psdp/>, 2010. [Online; accessed 26-March-2016].
- [Kalafut *et al.*, 2006] Andrew Kalafut, Abhinav Acharya, and Minaxi Gupta. A study of malware in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 327–332, New York, NY, USA, 2006. ACM.
- [Kang and Ruland, 2005] Namhi Kang and Christoph Ruland. MDS: Multiplexed Digital Signature for Real-Time Streaming over Multi-sessions. In *Lecture Notes in Computer Science*, pp. 824–834, Volume 3391, 2005.

- [Kannan *et al.*, 2004] Jayanth Kannan, AKK Lakshminarayanan, Ion Stoica, and Klaus Wehrle. Supporting Legacy Applications Over i3. *IEEE Computer*, 2004.
- [Kickstarter, 2016] Kickstarter. Kickstarter. <https://www.kickstarter.com/>, 2016. [Online; accessed 14-November-2015].
- [Kirkpatrick, 2013] Keith Kirkpatrick. Software-defined networking. *Commun. ACM*, 56(9):16–19, September 2013.
- [Klemm, Alexander and Lindemann, Christoph and Waldhorst, Oliver, 2003] Klemm, Alexander and Lindemann, Christoph and Waldhorst, Oliver. A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks. In *Proc. IEEE Semiannual Vehicular Technology Conference (VTC2003-Fall)*. IEEE, October 2003.
- [Klensin, 2001] J. Klensin. Simple mail transfer protocol. RFC 2821, RFC Editor, April 2001.
- [Kohler *et al.*, 2000] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.
- [Koponen *et al.*, 2007] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proc. of SIGCOMM’07*, pages 181–192. ACM, 2007.
- [Kopparty *et al.*, 2002] Swastik Kopparty, Srikanth V. Krishnamurthy, Michalis Faloutsos, and Satish K. Tripathi. Split tcp for mobile ad hoc networks. In *Proc. of GLOBECOM’02*, pages 138–142, 2002.
- [Kortuem *et al.*, 2001] Gerd Kortuem, Jay Schneider, Dustin Preuitt, Thaddeus Thompson, Stephen Fickas, and Zary Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *P2P ’01: Proceedings of the First International Conference on Peer-to-Peer Computing (P2P’01)*, page 75. IEEE Computer Society, 2001.

- [LEAF, 2006] Linux Embedded Application Firewall (LEAF). <http://www.leaf-project.org/>, 2006. [Online; accessed 26-March-2016].
- [Lee *et al.*, 2011a] Jae Woo Lee, Roberto Francescangeli, Jan Janak, Suman Srinivasan, Salman Abdul Baset, Henning Schulzrinne, Zoran Despotovic, and Wolfgang Kellerer. NetServ: Active Networking 2.0. In *IEEE FutureNet workshop*, June 2011.
- [Lee *et al.*, 2011b] Jae Woo Lee, Roberto Francescangeli, Wonsang Song, Jan Janak, Suman Srinivasan, Michael Kester, Salman Abdul Baset, Eric Liu, Henning Schulzrinne, Volker Hilt, Zoran Despotovic, and Wolfgang Kellerer. NetServ Framework Design and Implementation 1.0. Technical report, Columbia University, May 2011.
- [Lemon and Sommerfeld, 2006] T. Lemon and B. Sommerfeld. Node-specific client identifiers for dynamic host configuration protocol version four (dhcpv4). RFC 4361, RFC Editor, February 2006.
- [Li and Simon, 2011] Zhe Li and Gwendal Simon. Time-Shifted TV in Content Centric Networks: The Case for Cooperative In-Network Caching. In *2011 IEEE International Conference on Communications (ICC)*, June 2011.
- [libcurl, 2005] libcurl. <http://curl.haxx.se/>, 2005. [Online; accessed 26-March-2016].
- [libesmtp, 2006] libESMTP. <http://www.stafford.uklinux.net/libesmtp>, 2006. [Online; accessed 26-March-2016].
- [lighttpd, 2006] lighttpd, a small-footprint web server. <http://www.lighttpd.net/>, 2006. [Online; accessed 26-March-2016].
- [LISP, 2014] LISP. LISP. <https://datatracker.ietf.org/wg/lisp/documents/>, 2014. [Online; accessed 14-November-2015].
- [Liu *et al.*, 2006] Z. Liu, D.C. Gibbon, and B. Shahraray. Multimedia content acquisition and processing in the miracle system. In *IEEE CCNC 2006*, January 2006.
- [Lokhandwala *et al.*, 2015] Hatim Lokhandwala, Srikant Manas Kala, and Bheemarjuna Reddy Tamma. Min-o-mee: A proximity based network application leveraging the alljoyn framework. *CoRR*, abs/1511.06061, 2015.

- [Lundqvist *et al.*, 2011] H. Lundqvist, Z. Despotovic, G. Kunzmann, J. Frtunikj, and W. Kellerer. Service Program Mobility. In *Proceedings of IEEE Globecom 2011, Mobile Computing and Emerging Communication Networks Workshop (MCECN)*, December 2011.
- [Maccherani *et al.*, 2012] E. Maccherani, M. Femminella, J. W. Lee, R. Francescangeli, J. Janak, G. Reali, and H. Schulzrinne. Extending the netserv autonomic management capabilities using openflow. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 582–585, April 2012.
- [Magalhaes and Pereira, 2004] J.J. Magalhaes and F. Pereira. Using mpeg standards for multimedia customization. In *Journal of Signal Processing: Image Communication (Elsevier)*, May 2004.
- [MaxCDN, 2013] MaxCDN. How Anycast IP Routing Is Used at MaxCDN. <https://www.maxcdn.com/blog/anycast-ip-routing-used-maxcdn/>, 2013. [Online; accessed 20-March-2016].
- [McKeown *et al.*, 2008] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Computer Communications Review*, March 2008.
- [Mealling and Daniel, 2000] M. Mealling and R. Daniel. The naming authority pointer (naptr) dns resource record. RFC 2915, RFC Editor, September 2000.
- [Metzger and Chuah, 2008] Ryan Metzger and Mooi Choo Chuah. Opportunistic information distribution in challenged networks. In *Proceedings of the Third ACM Workshop on Challenged Networks*, 2008.
- [Michael Plass, 2010] Michael Plass. ccnx-0.3.0 is released. <https://www.ccnx.org/pipermail/ccnx-dev/2010-November/000266.html>, 2010. [Online; accessed 26-March-2016].

- [Microsoft Forums, 2012] Microsoft Forums. What is Akamai NetSession Client. http://answers.microsoft.com/en-us/windows/forum/windows_other-security/what-is-akamai-netsession-client/6c85ea38-e236-42b4-8c02-ea425d5658dc?auth=1, 2012. [Online; accessed 20-March-2016].
- [Mockapetris, 1987a] P. Mockapetris. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987.
- [Mockapetris, 1987b] P. Mockapetris. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987.
- [Mockapetris, 1987c] P. Mockapetris. Domain names - implementation and specification. STD 13, RFC Editor, November 1987.
- [Moghadam *et al.*, 2008] Arezu Moghadam, Suman Srinivasan, and Henning Schulzrinne. 7ds-a modular platform to develop mobile disruption-tolerant applications. In *Second IEEE Conference and Exhibition on Next Generation Mobile Applications, Services, and Technologies (NGMAST 2008)*, September 2008.
- [MSN Messenger, 2006] MSN Messenger. <http://messenger.msn.com/>, 2006. [Online; accessed 26-March-2016].
- [Mulerikkal and Khalil, 2007] J.P. Mulerikkal and I. Khalil. An architecture for distributed content delivery network. In *IEEE Conference on Networks*, November 2007.
- [Netflix, 2012] Netflix. Announcing the Netflix Open Connect Network. <http://blog.netflix.com/2012/06/announcing-netflix-open-connect-network.html>, June 2012. [Online; accessed 26-March-2016].
- [NetIndex 2012, 2012] NetIndex Bandwidth by Country. <http://www.netindex.com/download/allcountries/>, 2012. [Online; accessed 26-March-2016].
- [NetServ Website, 2011] The NetServ Project. <http://www.cs.columbia.edu/irt/project/netserv/>, 2011. [Online; accessed 26-March-2016].

- [Networks *et al.*, 2014] Bright House Networks, Cox Communications, Optimum, Time Warner Cable, and XFINITY. Cable WiFi. <http://www.cablewifi.com/>, 2014. [Online; accessed 26-March-2016].
- [nfqueue, 2008] nfqueue bindings for netfilter. <http://software.inl.fr/trac/wiki/nfqueue-bindings>, 2008. [Online; accessed 26-March-2016].
- [Nottingham and Sayre, 2005] Mark Nottingham and Robert Sayre. The atom syndication format. RFC 4287, RFC Editor, December 2005.
- [NSF, 2009] NSF. NSF GENI project. <http://www.geni.net/>, 2009. [Online; accessed 26-March-2016].
- [ONEChatWebsite, 2006] Onechat home page, 2006. [Online; accessed 26-March-2016].
- [Open Networking Foundation, 2016] Open Networking Foundation. Open Networking Foundation. <https://www.opennetworking.org/>, 2016. [Online; accessed 20-March-2016].
- [OpenPeer, 2014] OpenPeer. Hookflash. <http://openpeer.org/>, 2014. [Online; accessed 20-March-2016].
- [OSGi Alliance, 2011] OSGi Alliance. OSGi Alliance. <http://www.osgi.org/>, 2011. [Online; accessed 26-March-2016].
- [Padmanabhan *et al.*, 2002] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *IEEE NOSS-DAV02*, May 2002.
- [Pallis and Vakali, 2006] G. Pallis and A. Vakali. Insight and Perspectives for Content Delivery Networks. In *Communications of the ACM*, January 2006.
- [Pan *et al.*, 2006] C. Pan, M. Atajanov, M. B. Hossain, T. Shimokawa, and N. Yoshida. FCAN: Flash Crowds Alleviation Network. In *2006 ACM Symposium on Applied Computing (SAC 2006)*, 2006.

- [Pandey *et al.*, 2009] S. Pandey, K.K. Gupta, A. Barker, and R. Buyya. Minimizing cost when using globally distributed cloud services: A case study in analysis of intrusion detection workflow application. In *CloudCom 2009*, 2009.
- [Papadopouli, Maria and Schulzrinne, Henning, 2000] Papadopouli, Maria and Schulzrinne, Henning. Seven Degrees of Separation in Mobile Ad Hoc Networks. In *IEEE GLOBECOM, San Fransisco*, November 2000.
- [Papadopouli, Maria and Schulzrinne, Henning, 2001] Papadopouli, Maria and Schulzrinne, Henning. Design and Implementation of a Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users. In *First NY Metro Area Networking Workshop, IBM TJ Watson Research Center, Hawthorne, New York*, March 2001.
- [PARC, 2015] Xerox PARC. What is CCN? <http://www.ccnx.org/what-is-ccn/>, 2015. [Online; accessed 26-March-2016].
- [Park *et al.*, 2012] Choongul Park, Yeongil Seo, Kunyoul Park, and Youngseok Lee. The concept and realization of context-based content delivery of ngson. In *IEEE Communications Magazine*, January 2012.
- [PC Engines, 2007] PC Engines. WRAP board single board computer. <http://www.pcengines.ch/wrap.htm>, 2007. [Online; accessed 26-March-2016].
- [PCMagazine, 2011] PCMagazine. Netflix Eats Up 32 Percent of U.S. Bandwidth During Peak Times . <http://www.pcmag.com/article2/0,2817,2395372,00.asp>, October 2011. [Online; accessed 26-March-2016].
- [Perez, 2009] Juan Carlos Perez. Obama inauguration drives record web usage. In *PC World*, January 2009.
- [Perrig *et al.*, 2000] A. Perrig, R. Canetti, D. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of IEEE Security and Privacy Symposium*, 2000.
- [Pierre and Steen, 2006] G. Pierre and M. Steen. Globule: A collaborative content delivery network. In *IEEE Communications Magazine*, August 2006.

- [Postel, 1981] Jon Postel. Transmission control protocol. STD 7, RFC Editor, September 1981.
- [Public Internet, 2002] Public Internet Project - Wireless AP Density in Manhattan. http://publicinternetproject.org/research/research_details.html, 2002. [Online; accessed 26-March-2016].
- [Rackspace, 2015] Rackspace. What is a CDN? http://www.rackspace.com/knowledge_center/article/what-is-a-cdn, May 2015. [Online; accessed 14-November-2015].
- [Rao *et al.*, 2011] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, CoNEXT '11, 2011.
- [Reuters, 2012] Reuters. Exclusive: YouTube hits 4 billion daily video views. <http://www.reuters.com/article/2012/01/23/us-google-youtube-idUSTRE80M0TS20120123>, January 2012.
- [RSS, 2006] The RSS 2.0 Specification. <http://blogs.law.harvard.edu/tech/rss>, 2006. [Online; accessed 26-March-2016].
- [Sarolahti *et al.*, 2011] Pasi Sarolahti, Jrg Ott, Karthik Budigere, and Colin Perkins. Poor Man's Content Centric Networking (with TCP). Technical report, Aalto University, March 2011.
- [scapy, 2009] Scapy. <http://www.secdev.org/projects/scapy/>, 2009. [Online; accessed 26-March-2016].
- [Schull *et al.*, 2006] J Schull, M Axelrod, and L Quinsland. Multichat: Persistent, text-as-you-type messaging in a web browser for fluid multi-person interaction and collaboration. In *Proceedings of the 39th Annual Hawaii International booktitle on System Sciences, Hawaii, USA, 2006.*, 2006.
- [Schulzrinne and Hancock, 2010] Henning Schulzrinne and R. Hancock. Gist: General internet signalling transport. RFC 5971, RFC Editor, October 2010.

- [Shanbhag *et al.*, 2011] Shashank Shanbhag, Nico Schwa, Ivica Rimac, and Matteo Varvello. SoCCeR: Services over Content-Centric Routing. *ACM Workshop on Information-Centric Networking (ICN)*, August 2011.
- [Shen *et al.*, 2010] Xuemin Sherman Shen, Heather Yu, John Buford, and Mursalin Akon. *Handbook of peer-to-peer networking*, volume 34. Springer Science & Business Media, 2010.
- [Skype official blog, 2010] Skype official blog. CIO update: Post-mortem on the Skype outage. http://blogs.skype.com/en/2010/12/cio_update.html, 2010. [Online; accessed 26-March-2016].
- [skype, 2006] Skype. <http://www.Skype.com/>, 2006. [Online; accessed 26-March-2016].
- [Smith, 2010] Jonathan Smith. Nebula. <http://nebula.cis.upenn.edu/about.html>, 2010. [Online; accessed 26-March-2016].
- [So-In *et al.*, 2010] Chakchai So-In, R. Jain, S. Paul, and J. Pan. Virtual id: A technique for mobility, multi-homing, and location privacy in next generation wireless networks. In *IEEE CCNC (Consumer Communications and Networking Conference)*, January 2010.
- [Socialized.NET, 2006] Socialized.net. <http://www.socialized.net/>, 2006. [Online; accessed 26-March-2016].
- [Sorce *et al.*, 2007] Salvatore Sorce, Francesco Cinquegrani, Salvatore Anzalone, Dario Cacc, and Antonio Gentile. A dynamic system for personal communications: the opportunistic chat. In *International booktitle on Intelligent Pervasive Computing*, 2007.
- [SourceForge, 2009] SourceForge. BonAHA Download Page. <http://bonaha.sourceforge.net/>, 2009. [Online; accessed 26-March-2016].
- [sqlite, 2006] SQLite database. <http://www.sqlite.org/>, 2006. [Online; accessed 26-March-2016].
- [Srinivasan and Schulzrinne, 2007] Suman Srinivasan and Henning Schulzrinne. BonSwing: A GUI Framework for Ad-Hoc Applications Using Service Discovery. In *ACM Interna-*

- tional Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2007.
- [Srinivasan *et al.*, 2007] Suman Srinivasan, Arezu Moghadam, SeGi Hong, and Henning G Schulzrinne. 7DS - Node cooperation and information exchange in mostly disconnected networks. In *IEEE International Conference on Communications (ICC)*, 2007.
- [Srinivasan *et al.*, 2009a] Suman Srinivasan, Jae Woo Lee, Eric Liu, Michael Kester, Henning Schulzrinne, Volker Hilt, Sriram Seetharaman, and A. Khan. NetServ: Dynamically Deploying In-network Services. In *ACM ReArch '09 (CoNEXT workshop)*, December 2009.
- [Srinivasan *et al.*, 2009b] Suman Srinivasan, Arezu Moghadam, and Henning Schulzrinne. BonAHA: Service Discovery Framework for Mobile Ad-Hoc Applications. In *IEEE Consumer Communications & Networking Conference 2009 (CCNC'09)*, 2009.
- [Stackoverflow, 2013] Stackoverflow. GKSession is deprecated in iOS 7, what should i use now? <http://stackoverflow.com/questions/18939472/gksession-is-deprecated-in-ios7-what-should-i-use-now>, 2013. [Online; accessed 20-March-2016].
- [Stanford University, 2010a] Stanford University. Ethane. <http://yuba.stanford.edu/ethane/>, 2010. [Online; accessed 26-March-2016].
- [Stanford University, 2010b] Stanford University. The OpenFlow switch. <http://www.openflowswitch.org/>, 2010. [Online; accessed 26-March-2016].
- [StatCounter, 2012] Mobile vs Desktop from Dec 2008 to March 2012. <http://bit.ly/I9jclr>, 2012. [Online; accessed 26-March-2016].
- [Suman Srinivasan, 2009] Suman Srinivasan. BonSwing Download Page. <http://bonswing.sourceforge.net/>, 2009. [Online; accessed 26-March-2016].
- [Sun Microsystems, 2001] Sun Microsystems. Project JXTA, 2001.

- [Sun Microsystems, 2003] Sun Microsystems. Java 2 Platform Security Architecture. <http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html>, 2003.
- [Sun Microsystems, 2004a] Sun Microsystems. JAR File Specification. <http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html>, 2004.
- [Sun Microsystems, 2004b] Sun Microsystems. Java Native Interface Specification. <http://java.sun.com/j2se/1.5.0/docs/guide/jni/spec/jniTOC.html>, 2004.
- [Sun Microsystems, 2009] Sun Microsystems. OpenSolaris project: Crossbow: Network virtualization and resource control. <http://opensolaris.org/os/project/crossbow/>, 2009. [Online; accessed 26-March-2016].
- [swish-e, 2006] Swish-e search and indexing library. <http://www.swish-e.org/>, 2006. [Online; accessed 26-March-2016].
- [t140, 2008] T140 library. <http://sourceforge.net/projects/rtp-text-t140/>, 2008.
- [The Engineer, 2014] The Engineer. App Boosts Football Fans' Mobile Internet Signal in Crowds. <http://www.theengineer.co.uk/app-boosts-football-fans-mobile-internet-signal-in-crowds/>, 2014. [Online; accessed 20-March-2016].
- [The Guardian, 2015] The Guardian. The Guardian. <https://www.theguardian.com/>, 2015. [Online; accessed 14-November-2015].
- [Thomas *et al.*, 2003] Roshan Thomas, Brian Mark, Tommy Johnson, and James Croall. NetBouncer: Client-legitimacy-based High-performance DDoS Filtering. In *Proc. of DIS-CEX III*, pages 14–25, 2003.
- [Thompson *et al.*, 2001] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. *W3C Recommendation*, 2, 2001.
- [thttpd, 2006] thttpd, a small-footprint web server. <http://www.acme.com/software/thttpd/>, 2006. [Online; accessed 26-March-2016].

- [Tidwell *et al.*, 2011] J. Tidwell, E. Samame, and B. Santangelo. Methods and apparatus for evaluating an audience in a content-based network. In *U.S. Patent Application Publication #US20110016482*, January 2011.
- [Times, 2008] Los Angeles Times. iPhone becomes top handset in U.S., passing RAZR. <http://latimesblogs.latimes.com/technology/2008/11/the-iphone-beco.html?sr=hotnews>, 2008. [Online; accessed 26-March-2016].
- [Tiwana *et al.*, 2010] Birjodh Tiwana, Mahesh Balakrishnan, Marcos K. Aguilera, Hitesh Ballani, and Z. Morley Mao. Location, location, location!: modeling data proximity in the cloud. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, October 2010.
- [Tom Leighton, 2009] Tom Leighton. Improving Performance on the Internet. In *Communications of the ACM*, February 2009.
- [TribeHive, 2016] TribeHive. TribeHive. <http://tribehive.co.uk/>, 2016. [Online; accessed 20-March-2016].
- [Tullmann *et al.*, 2001] Patrick Tullmann, Mike Hibler, and Jay Lepreau. Janos: A Java-oriented OS for Active Network Nodes. *IEEE Journal on Selected Areas in Communications*, 19:501–510, 2001.
- [University of Washington, 2011] University of Washington. A Prototype of a Million Node GENI. <http://groups.geni.net/geni/wiki/MillionNodeGENI>, 2011. [Online; accessed 26-March-2016].
- [Vyatta, 2009] Vyatta. Vyatta network virtualization. <http://www.vyatta.com/products/virtualized.php>, 2009. [Online; accessed 26-March-2016].
- [W3C, 2005] W3C. Synchronized multimedia integration language 2.0. <http://www.w3.org/TR/2005/REC-SMIL2-20050107/>, 2005. [Online; accessed 26-March-2016].
- [W3C, 2011] W3C. Web Real-Time Communications Working Group. <https://www.w3.org/2011/04/webrtc/>, 2011. [Online; accessed 20-March-2016].

- [Wall Street Journal, 2014] Wall Street Journal. Forget 'the Cloud'; 'the Fog' Is Tech's Future. <http://www.wsj.com/articles/SB10001424052702304908304579566662320279406>, 2014. [Online; accessed 20-March-2016].
- [Wang and Crowcroft, 1996] Zheng Wang and Jon Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE JSAC*, 1996.
- [Wang *et al.*, 2004] L. Wang, K. S. Park, R. Pang, V. Pai, and L. Peterson. Reliability and security in the codeen content distribution network. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2004.
- [Wang *et al.*, 2007] Alf Inge Wang, Tommy Bjornsgard, and Kim Saxlund. Peer2me - rapid application framework for mobile peer-to-peer applications. In *International Symposium on Collaborative Technologies and Systems 2007*, 2007.
- [Wang *et al.*, 2008] Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 231–242. ACM, 2008.
- [Web Switch, 2011] Layer 4/7 switch (web switch). http://en.wikipedia.org/wiki/Multilayer_switch, 2011. [Online; accessed 26-March-2016].
- [Wetherall *et al.*, 1998] David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE OPENARCH*, 1998.
- [Wikipedia, 2015a] Wikipedia. Host model — Wikipedia, The Free Encyclopedia. <http://en.wikipedia.org/w/index.php?title=Host%20model&oldid=580821550>, 2015. [Online; accessed 14-November-2015].
- [Wikipedia, 2015b] Wikipedia. Internet backbone — Wikipedia, The Free Encyclopedia. <http://en.wikipedia.org/w/index.php?title=Internet%20backbone&oldid=687897550>, 2015. [Online; accessed 14-November-2015].

- [Wikipedia, 2016] Wikipedia. Software-defined networking — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Software-defined_networking, 2016. [Online; accessed 20-March-2016].
- [Wittenburg, 2010] Georg Wittenburg. *Service Placement in Ad Hoc Networks*. PhD thesis, Department of Mathematics and Computer Science, Freie Universität Berlin, October 2010.
- [Wolf, 2006] Tilman Wolf. Service-centric end-to-end abstractions in next-generation networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN)*, pages 79–86, October 2006.
- [Wong and Lam, 1998] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *In Proc. IEEE ICNP’98*, 1998.
- [Xerox PARC, 2010] Xerox PARC. Project CCNx. <http://www.ccnx.org/>, 2010. [Online; accessed 26-March-2016].
- [Xuggler, 2009] Xuggler java library. <http://www.xuggler.com/xuggler/>, 2009. [Online; accessed 26-March-2016].
- [Yemini and Silva, 1996] Yechiam Yemini and Sushil Da Silva. Towards programmable networks. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, 1996.
- [Yoshida, 2009] N. Yoshida. Dynamic CDN against flash crowds. <http://www.springerlink.com/content/q2j5130831r84365/>, 2009. [Online; accessed 26-March-2016].
- [Yu *et al.*, 2006] Hongliang Yu, Dongdong Zheng, Ben Y. Zhao, and Weimin Zheng. Understanding user behavior in large-scale video-on-demand systems. *SIGOPS Oper. Syst. Rev.*, April 2006.
- [Yuan *et al.*, 2004] C. Yuan, Y. Chen, and Z. Zhang. Evaluation of edge caching/off loading for dynamic content delivery. In *IEEE Transactions on Knowledge and Data Engineering*, October 2004.

- [Yuen and Schulzrinne, 2006] Wing Yuen and Henning Schulzrinne. Performance evaluation of time-based and hop-based TTL schemes in partially connected ad hoc networks. In *Proc. IEEE ICC '06*, June 2006.
- [ZDNet, 2014] ZDNet. What is Docker and why is it so darn popular? <http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/>, 2014. [Online; accessed 20-March-2016].
- [Zeroconf Working Group, 2008] Zeroconf Working Group. Zeroconf Working Group. <http://www.zeroconf.org/>, 2008. [Online; accessed 26-March-2016].
- [Zhao and Schulzrinne, 2006] Wen Zhao and Henning Schulzrinne. Enabling On-demand Query Result Caching in DotSlash for Handling Web Hotspots Effectively. In *IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb'06)*, November 2006.

Part VI

Appendices

Appendix A

BonAHA Applications

A.1 ONEChat: Enabling Group Chat and Messaging in Opportunistic Networks

A.1.1 Introduction

Text-based chat applications can enhance collaboration and augment oral communication in such networks. Hence, a group chat or instant messaging application for opportunistic networks would be very useful.

A group chat application is a collaborative software (also referred to as groupware) which is designed to help people communicate with each other in real time. IM (Instant Messaging) is another tool which allows real-time text-based communication application among people, either with one another or in a group.

Unfortunately, most existing group-chat and instant messaging applications today work in a client-server manner. Most of the popular solutions also rely on the use of proprietary protocols and servers. Hence they cannot be used in opportunistic networks. Recently, a few IM applications have been proposed for opportunistic networks, but to the best of our knowledge, they can not support group chat efficiently. More details about related work will be introduced in Section A.1.2.

We have implemented an efficient group chat application for opportunistic networks called ONEChat (Opportunistic Network Chat). ONEChat was built in 2009 using the

Java programming language [Gosling *et al.*, 2005]. Even though we mostly focus on single-hop local opportunistic networks, ONEChat works in any network that supports multicast communication.

ONEChat is a group chat and instant messaging program that works in opportunistic networks. ONEChat uses message multicasting on top of service discovery protocols in order to support group chat and reduce bandwidth consumption in opportunistic networks. ONEChat does not require any pre-configuration, a fixed network infrastructure or a client-server architecture in order to operate. In addition, it supports features such as group chat, private rooms, line-by-line or character-by-character messaging and file transfer.

ONEChat does not need to be manually configured, nor is a fixed infrastructure required for it to work properly. ONEChat works very well even in the presence of transient nodes that enter and leave the network quickly. The implementation of ONEChat is greatly simplified by building our program on top of the BonAHA framework [Srinivasan *et al.*, 2009b] and the real-time text protocol [Hellstrom and Jones, 2005].

In addition to supporting simple group chat, ONEChat also has several additional and useful features that make it a fully featured application. For instance, it allows for users to create their own private groups, where messages are encrypted and which only users with the knowledge of a shared key can join. ONEChat also supports exchange of small files as well as buddy icon updates from other users.

ONEChat leverages the properties of real-time text to allow transmission of messages in line-by-line mode, with the user indicating the completing a message through a signal such as pressing the Enter key, or in character-by-character mode, where a character is transmitted as soon as it is entered.

The rest of the section is organized as follows. Section A.1.2 introduces related work. Section A.1.3 covers the implementation details of ONEChat.

A.1.2 Related Work

There are a plethora of IM applications today. ICQ [icq, 2006], GTalk [GoogleTalk, 2006], MSN [MSNMessenger, 2006], AOL [AOLMessenger, 2006], Skype [skype, 2006], and MultiChat [Schull *et al.*, 2006] are among the most popular and widely known ones. However,

none of these popular IM programs or their clones can be used in opportunistic networks because they require connection to the Internet, in particular, connection to the servers that are run by the companies that host these IM programs.

Recently, some chat applications for opportunistic networks have appeared. iChat [iChat, 2006], Socialized.NET [Socialized.NET, 2006], Opportunistic Chat [Sorce *et al.*, 2007], and DTN (Disruption Tolerant Networks) Jabber Proxy [Metzger and Chuah, 2008], and most recently FireChat [FireChat1, 2015] [FireChat2, 2015] are representative of these class of applications. However, they can not support group chat for opportunistic networks due to the drawbacks listed below.

Both iChat and Socialized.NET work in a P2P manner. However, neither of them support message multicast, so a message has to be sent multiple times in order to reach all users within a group. This would consume a lot of bandwidth.

Opportunistic Chat [Sorce *et al.*, 2007] introduces a Bluetooth-TCP/IP hybrid approach: if two users next to each other want to talk directly, they can set up a Bluetooth link; if they are too far away to be able to use Bluetooth, or if they want to chat within a group, then they should setup a client-server communication link via a TCP/IP network. Therefore, Opportunistic Chat cannot handle group chat for opportunistic networks either.

DTN Jabber Proxy can work in opportunistic networks, but it requires a complicated server configuration and the server proxy needs to be installed and available to the network.

In order to fully support group chat for opportunistic networks, we have implemented a pure-Java, lightweight and configuration-free application called ONEChat. As soon as ONEChat applications start up, they can discover each other without querying any central servers, and they can work without requiring any pre-configuration.

Once a ONEChat application enters or leaves the network, all the other nodes will be notified automatically. For each message, there is only one multicast transmission to all the other group members to save bandwidth. The following section will introduce the implementation details of ONEChat.

A.1.3 Implementation Of ONEChat

In this section, we will explain how ONEChat works with the BonAHA library and the real-time text protocol. We will also explain how the different types of groups and messages are defined in an ONEChat application. We will also detail how some of its essential features of ONEChat, such as creating a group, joining and leaving a group, notification of network entry, and notification of leaving a network, work.

A.1.4 Introduction to ONEChat

BonAHA [Srinivasan *et al.*, 2009b] is a framework for opportunistic networks based on the multicast DNS and Zero Configuration [Zeroconf Working Group, 2008] networking suite of service discovery protocols. The BonAHA library allows for easy development of applications that work in link-local, opportunistic networks. BonAHA can work in local (single-hop) opportunistic networks as well as in regular wired or wireless connections.

For each user, we define his or her *network* as all users within his or her transmission range.

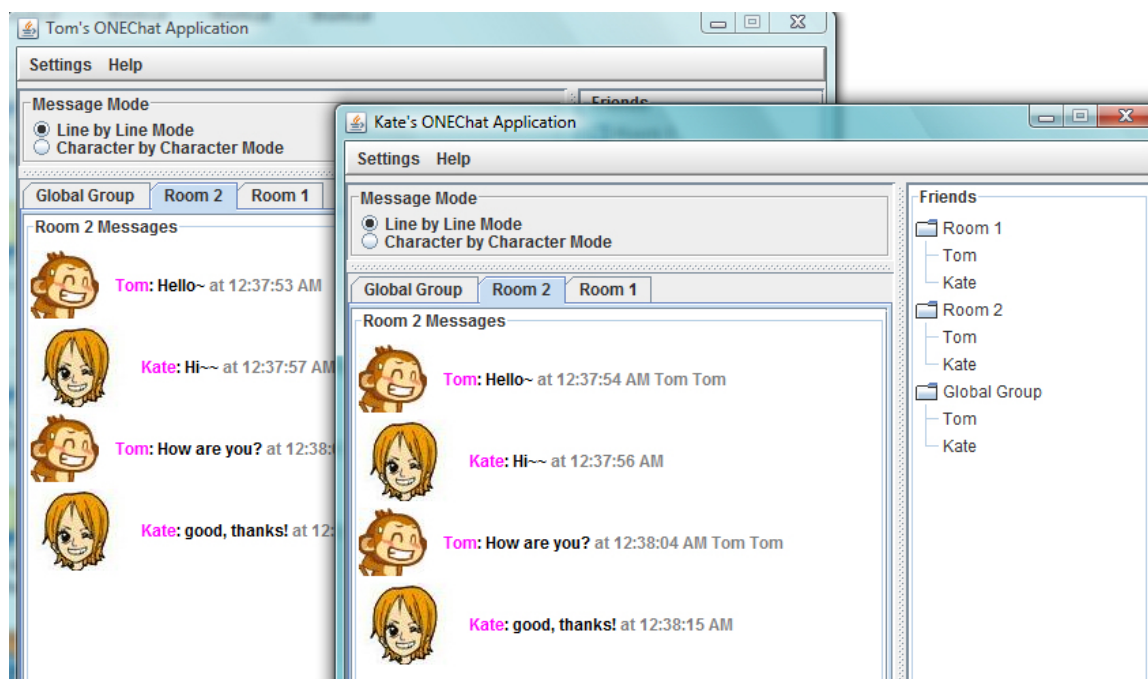
Figure A.1 shows a screenshot of ONEChat’s user interface. The left part of the user interface contains a list of instant messages between users, while the right side of the UI contains a list of the user’s ”friends” who are in the same network.

For example, in this screenshot, the user’s name is *Kate*, and there are already two groups, *Global Group* and *Room 1*, in Kate’s ONEChat application. (The Global Group is always present in the ONEChat applications, while other local rooms can be created as necessary.)

Correspondingly, there are two tabs on the left part of the UI with these two group names, and there are two tree components on the right side which display the groups. The chat messages in a group are shown in the main window under a tab, while a tree component on the right side of the UI lists the users in that group.

A.1.5 User and Message Discovery Using BonAHA

There are two kinds of messages in ONEChat: *system messages* and *user messages*.

Figure A.1: Tom and Kate chat within group *Room 2*.

We use BonAHA to send and receive system messages in order to perform the “behind the scenes” work to allow ONEChat applications and users to signal each other, as well as entry and exit in the network. We use the Real Time Text (RTT) protocol to send and receive user messages (The RTP and RTT protocols are described in more detail in the next section.)

In addition to chat, ONEChat also supports file transfer and the buddy icon update features, which are also implemented as user messages. These features will be introduced later in this section.

The message publishing and signalling mechanism for ONEChat is built on top of BonAHA, which uses mDNS (multicast DNS) service discovery protocol [Zeroconf Working Group, 2008].

ONEChat is meant to work in opportunistic networks which are highly transient. So it is necessary to keep track of the state of users in the network, such as users entering and leaving the network. ONEChat recognizes these events as system events and uses BonAHA to publish them as system messages.

ONEChat uses two functions from the BonAHA framework to be notified when users enter and leave the network - *serviceUpdated(BNode n)* and *serviceExited(BNode n)*.

ONEChat uses the *set(String key, String text)* function call in the BonAHA framework to publish a system message. This sets the global properties of the user's key, and all the other users within its transmission range receive this message.

Table A.1.5 summarizes how ONEChat uses the BonAHA functions to handle the low-level network events in the opportunistic network.

Table A.1: Usage of the four BonAHA functions in ONEChat

Function	Usage
<i>serviceUpdated</i>	Triggered when a new ONEChat user enters the network.
<i>serviceExited</i>	Triggered when a ONEChat user leaves the network.
<i>set</i>	Called when ONEChat publishes a system message.
<i>get</i>	Called when ONEChat retrieves the information published on the network.

A.1.6 Messaging using RTP and RTT

The user message publishing mechanism in ONEChat is built on top of the real-time text (RTT) protocol, and implemented using the T140 library [t140, 2008].

The real-time text protocol uses RTP (Real-time Transport Protocol) and defines an RTP payload type [Hellstrom and Jones, 2005] for text conversation. In the real-time text protocol, as soon as a character is typed, it is sent and displayed immediately to the recipient. This allows text to be used in the same conversational mode as voice and video.

In ONEChat, we use the real-time text protocol to provide two transmission modes for user messages: line-by-line mode and character-by-character mode. In line-by-line mode,

we buffer a group of characters that a user has typed and then transmit them; in character-by-character mode, text is sent and received character by character in a real-time manner.

As shown in Figure A.1, users can choose a message mode in the *Message Mode* field. Once they choose the character-by-character mode, the *Send* button is disabled since they do not need it (they just need to type the message and the typed characters will automatically be sent). After the user switches back to the line-by-line mode, this button is enabled.

A.1.7 Private and Secure Messaging Using ONEChat Groups

We provide a primitive security feature for ONEChat. There are two kinds of groups in ONEChat: the *Global Group* and *protected groups*. All public messages published within one's network are displayed in his or her *Global Group*, which is always present in any ONEChat session and available to all users currently on that opportunistic network.

By default, each ONEChat user always stays in the *Global Group*. All the other groups created by users are protected groups. A protected group must be created with a password, and others who want to join this group must know this password. We assume that this password is distributed using some out-of-band method. ONEChat does not send any password to the network.

All messages published within a protected group will be encrypted with its group password using AES in Counter Mode.

Currently, we do not consider malicious users bent on disrupting service. In multicast scenarios where keys are shared between members, it is easy to authenticate the source and prove that a member of the group has sent a message, but difficult to prevent one member from impersonating another.

This problem is called DOA (Data Origin Authentication). There are already some promising proposals in this area [Perrig *et al.*, 2000] [Akkus *et al.*, 2006] [Kang and Ruland, 2005] [Wong and Lam, 1998].

We have not implemented DOA for this current version of our application, but we will quickly summarize its features. DOA can be done using signatures. There are two kinds of approaches. The first approach involves signing each RTP packet [16] [17]. This approach

provides good source authentication but suffers from high computation overhead in signing and verifying the signature for each packet. The second approach involves amortizing a single signature over multiple packets or sessions [18] [19]. This reduces the overhead but it is not satisfactory when transmission is lossy. However, in some scenarios like small group chat through wireless links, the computation overhead is not a bottleneck, and the first approach is more acceptable.

A.1.8 Messages in ONEChat

There are two kinds of messages in ONEChat, *system messages* and *user messages*.

The first kind is *system message*, which is sent and received by the *set* and *get* functions provided by the BonAHA framework. Table A.1.8 illustrates the usage of all the five kinds of ONEChat system messages.

The other kind is *user message*, which is published by users. In addition, the file transfer and buddy icon update mechanisms also work on top of user message transmissions.

Table A.2: Usage of the five ONEChat system messages

Type	Usage
<i>sys_create</i>	Notify others that a new group was created.
<i>sys_join</i>	Notify group members that I joined this group.
<i>sys_reject</i>	Notify someone that the password he typed was wrong.
<i>sys_enter</i>	Notify others that I entered the network.
<i>sys_exit</i>	Notify others that I left the network.

The user message transmission mechanism is based on the real-time text protocol, and this protocol works on top of UDP multicast.

Retransmission of lost packets may result in re-sending a complete file several times. It is necessary to look into RTP-compatible retransmission mechanisms to mitigate packet loss. RFC 2354 [20] proposes several techniques, such as FEC (Forward Error Correction), retransmission, and interleaving, which may be considered to increase packet loss resiliency. In addition, RFC 4588 [21] proposes a comprehensive RTP retransmission payload format for both unicast and small multicast groups. This format is defined in the AVPF profile (RFC 4585 [22]), and is used by receivers to send retransmission requests. There are already some open source multicast file transfer program like UFTP [23], but they are not based on RTP. Thus, the RTP-compatible retransmission mechanism mentioned in RFC 4588 may be more appropriate to mitigate the UDP fragmentation problem.

Both system and user messages have three fields: *message type*, *destination group* and *message content*. The value of the *message type* field can be any one of the values in the first column in Table A.1.8, or *user_lbl*, (a user message under line-by-line mode), or *user_cbc*, (a user message under character-by-character mode).

Since ONEChat is a local chat application, there is only one multicast RTP group for all user messages among different chat groups, so we need a *destination group* field to indicate which group a message belongs to. A message would only be displayed in a group whose name is the same as that of the *destination group*. The *message content* field stores the real content of a message.

A.1.9 Creating, Joining and Leaving a Group

When a user creates a new group, he or she is required to enter a password (the group key), and others who want to join the group need to get this password information from the group creator.

Take an interaction between two users, Kate and Tom, as an example. As shown in Figure A.2, suppose Kate creates a group named *Room 2*. This event will be encapsulated into a *sys_create* (Table A.1.8) message and broadcast automatically to all users within Kate's network. Tom receives this message and a message is displayed on his ONEChat UI, as shown in Figure A.1.

If Tom is interested in joining *Room 2*, he double-clicks this and attempts to join the

group. This results in a popup window which asks Tom to enter a password. After Tom enters a password, the new group *Room 2* will appear in Tom's ONEChat window, and a *sys_join* (Table A.1.8) message will be sent to all users in Group *Room 2* notifying them that Tom has joined this group.

After Kate receives this message, she uses the group key (her password) to decrypt the message and tries to get the tag. If this succeeds, then Kate knows that Tom has typed the correct password, otherwise a *sys_reject* message will be sent to Tom to expel him from the session. Tom's ONEChat window will then close the *Room 2* tab.

If Tom is able to successfully authenticate and join *Room 2*, he can talk to Kate and others who are members of this group. Figure A.1 shows a screenshot of a established connection and group chat.

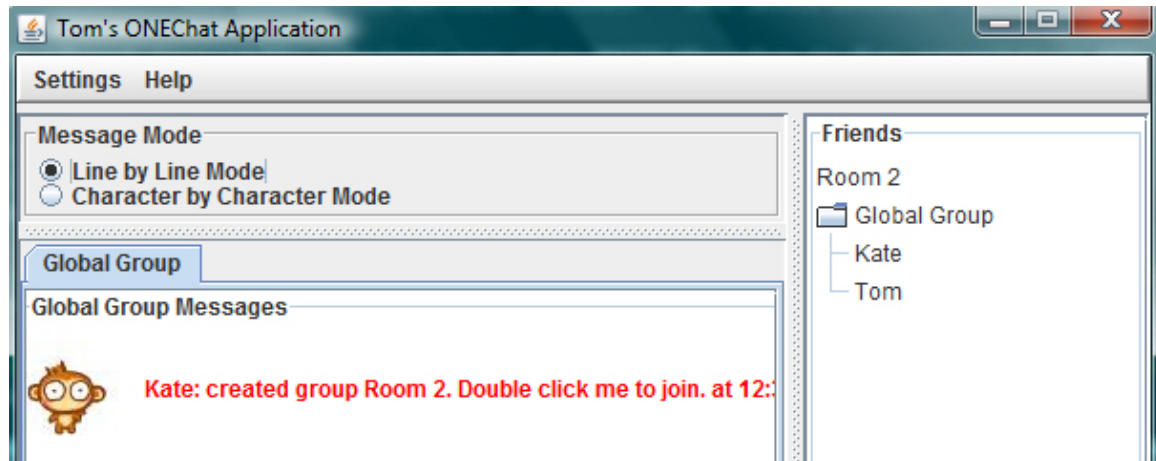


Figure A.2: Tom is informed that Kate created group *Room 2*.

In Figure A.1, Tom uses an icon (a monkey) and Kate another icon (a girl). The icons in the two users' applications are synchronized with the buddy icon update mechanism: once a user updates his icon, this icon image will be transmitted through multicast to all users within the network, and the other users can view this updated buddy icon. The buddy icon update mechanism is encapsulated as a public user message.

A user can leave a group at any time. Suppose Tom wants to leave *Room 2*, he closes the tab named *Room 2* in his application. A *sys_leave* (Table A.1.8) message will be automatically multicast to all users within this group.

This message will appear on the message window in Kate's *Room 2*, and Kate knows that Tom has left. Figure A.3 depicts the interactive sequence of the creating, joining and leaving group procedures between two users.

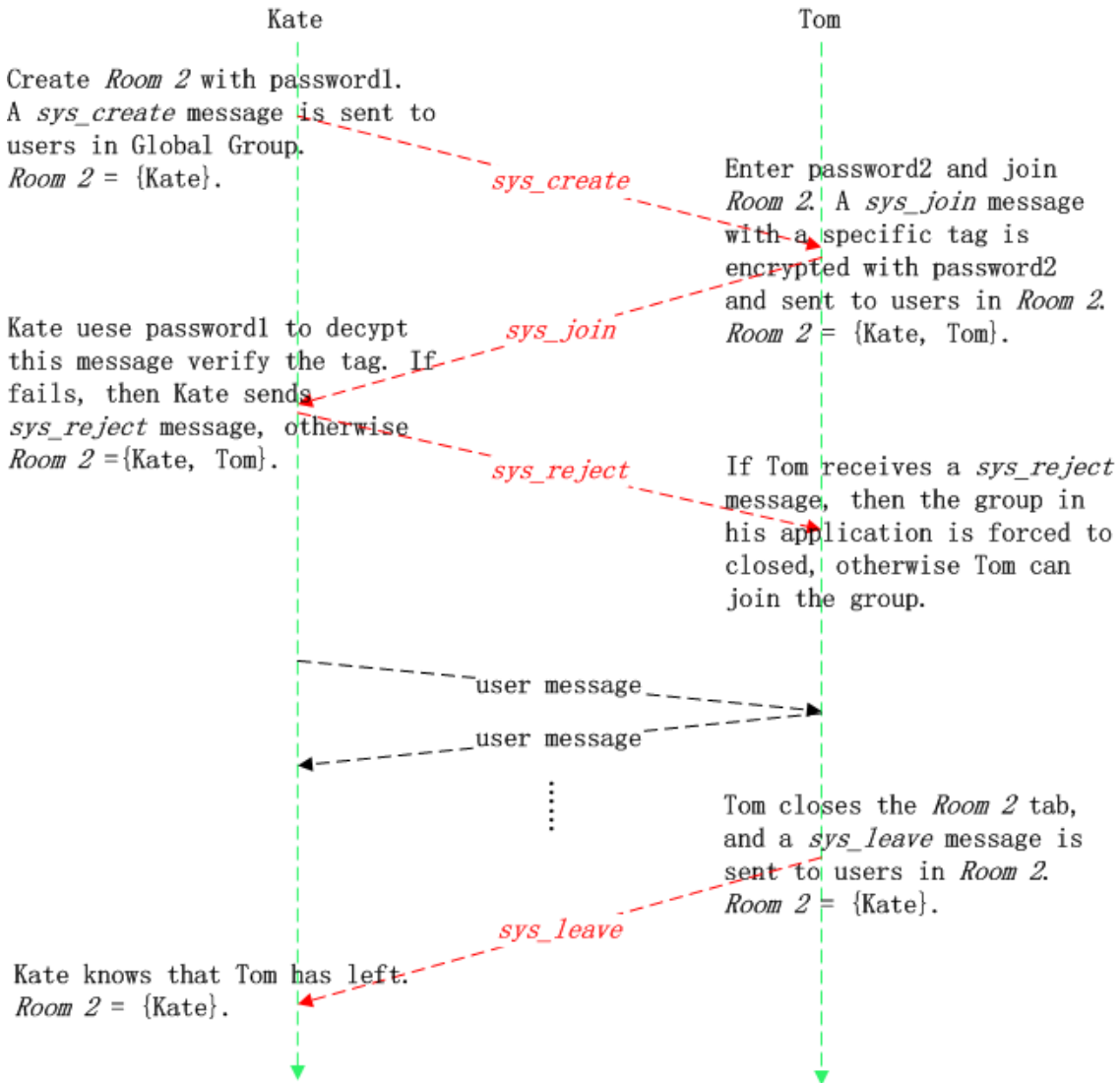


Figure A.3: The sequence of the creating, joining and leaving a group.

A.1.10 Enter-Network Notification

Once a user enters the network, all the other users in his or her network will be notified by BonAHA's *serviceUpdated* mechanism. Figure 5 shows how a enter-network notification

works.

Take Tom and Kate as an example. Once Tom launches a ONEChat application, a *sys_enter* message (Table A.1.8) is generated automatically and published by the *set* (Table A.1.5) function provided by BonAHA. All the other users in Tom's network receive this message immediately. Kate's ONEChat makes sure that this is a *sys_enter* message and displays it on the message list of her Global Group, and Tom is listed in Kate's friend list.

A.1.11 Leave-Network Notification

ONEChat can capture a leave-network event as well as an enter-network event. We have developed the procedure of leave-network notification by implementing the *serviceExited* (Table A.1.5) interface function provided by BonAHA.

Take Tom and Kate as an example. Suppose Tom closes his ONEChat application. This action triggers a *sys_exit* event down to his BonAHA framework, and this event is multicast to all the others in Tom's network. At Kate's end, a system message with the message *Tom left the network* is displayed on the message list of her Global Group and the name *Tom* is deleted from her friend list.

A.1.12 Performance Evaluation

We compare the performance of ONEChat's messaging system in opportunistic networks by comparing it to peer-to-peer chat and instant messaging clients, since we were not able to find similar IM clients that operate in a manner similar to ONEChat. We also believe that this evaluation validates our design of the ONEChat messaging system as efficient in the opportunistic network scenario.

The performance metric is the total amount of bytes sent to the network (bandwidth consumption) in the below scenario.

Assume that we have a group G_0 with m users: $\{N_1, \dots, N_m\}$. Each user N_i is within the single-hop communication range of all other users, and he or she is going to send k_i messages to all the others within this group. Each message has a size of L bytes. Let $B_{ONEChat}$ denote the total amount of bytes sent by ONEChat and B_{P2P} by P2P.

In P2P applications, each message has to be sent to everyone else within the group once (message retransmission is ignored), so the total amount of bytes sent to the network is:

$$B_{P2P} = \sum_{i=1}^m L \times k_i \times (m - 1)$$

In BonAHA, a multicast message will be sent a few times using the redundant transmission mechanism defined by Zeroconf [Zeroconf Working Group, 2008].

Using Wireshark [14], we found that each system message is transmitted at most three times, and each user message is transmitted once. Therefore, the *average transmission times* (in short, *avt*), which is defined as the total number of transmitted messages divided by the total number of distinct transmitted messages. We will consider the worst case scenario for the *avt* value, which is three.

Since ONEChat applications perform message multicasting for group chat, the total amount of bytes sent in the network is:

$$B_{ONEChat} = \sum_{i=1}^m L \times avt \times k_i$$

From the two equations above, we can see that when $(m - 1) > avt$, the amount of bytes sent to the network in P2P applications is more than that in ONEChat. Given $avt \leq 3$, if there are more than four users in a group, ONEChat consumes less bandwidth than P2P.

Hence, we can confirm that ONEChat's operation for opportunistic networks is more efficient than regular P2P clients which use unicast for messaging.

A.1.13 Future Work

Our future work for this project includes improving its security against malicious users, and solving the packet fragmentation problem in large file transfers.

A.1.14 Conclusion

This section describes a group chat application, ONEChat, for opportunistic networks. There are two main contributions. First, ONEChat eliminates configuration and the necessity of a fixed network infrastructure, making it easily deployable in opportunistic networks. Second, it uses multicast techniques to reduce bandwidth consumption in group chat scenarios. Quantitative performance analysis shows that ONEChat outperforms P2P-based applications like iChat and Socialized.NET in bandwidth consumption as long as there are

more than four members in a group.

ONEChat also has several interesting and useful features, such as private groups which are secure and require a shared key to join, as well as line-by-line and character-by-character modes of communication using Real-Time Text (RTT), that make it quite useful and usable as a full-featured application in real opportunistic networks today.

Our implementation of ONEChat (including additional documentation and details of data transfer protocols and RTP) is available for download at [ONEChatWebsite, 2006].

A.1.15 Acknowledgment

This work was supported by NSF Grant No. 04-54288 and No. 04-12025.

A.2 BBS-ONE: Bulletin Board and Forum System for Mobile Opportunistic Networks

A.2.1 Introduction

Electronic bulletin boards and forum systems are commonly used to exchange opinions, news, event notifications, documents and other media on the Internet. However, such systems usually require a central server hosting the content. Such servers cannot be installed in ad-hoc opportunistic wireless networks, which are created when mobile devices congregate to form a localized and short-lived network without Internet connectivity.

We present BBS-ONE, a bulletin board system for opportunistic networks, and describe its service model and implementation. BBS-ONE works in highly mobile opportunistic networks, considers the mobility of nodes, and allows nodes to operate even when churn is high when nodes join and leave the network. It transparently disseminates public data and posts and persists desired data by operating in a peer-to-peer fashion and using a store-carry-forward model of communication. It maintains the data consistency needed for a BBS and forum system. We have implemented the application on generic desktop OS platforms (Windows, Linux, Mac) as well as a mobile platform (iPhone/iPod).

Bulletin board systems are an important tool for collaboration and information exchange among peers. The real-world bulletin board systems on college campuses, apartment com-

plexes and other social areas provide a way for students, neighbors and peers to interact with each other and allow others to be mutually aware of events going on as well as facts that may interest others.

In recent years, with the rise of the Internet, the BBS has come to refer to a central, online repository or forum where users can post messages and files to exchange with other members of the board. The online forums or BBSes are often associated with a specific subject, topic or neighborhood, and provide similar online community features for its members.

However, with the growing use of opportunistic networks, where mobile nodes join together to form network islands with ad-hoc wireless connectivity, the traditional BBS model begins to fail. In the previously described scenarios, the BBS model only works because of the presence of a central server that handles the forum and community information. In opportunistic networks, there is no such central server, and to add to the problem, there is high mobility and churn rate, with nodes leaving and joining the network frequently.

We explore how we successfully built a real-world BBS application for opportunistic networks, called BBS for Opportunistic Networks (BBS-ONE). BBS-ONE allows users of mobile devices, such as iPhones, to share and distribute content with their local community connected by an opportunistic network, such as an 802.11 ad-hoc network.

By operating in a peer-to-peer manner, BBS-ONE provides BBS and forum functionality without requiring a central server. Our service model also enables us to keep data persistent, even when mobile nodes move across networks, thus fulfilling the basic requirement of a forum which allows for information to be spread in the local community.

We present our design of the system model of BBS-ONE to handle opportunistic networks and the state transitions that occur when nodes move in and out of the network.

In Section A.2.2, we introduce the concept of BBSes and forum software. In Section A.3, we describe the service model on which this system is based, in order to keep, disseminate and carry data for BBS. In Section A.4, we show how the system was designed and implemented, with details of the specific technologies involved. Screenshots of the BBS-ONE application on these different platforms are also included.

In Section A.4.3, on related work, we compare this system to various services that provide

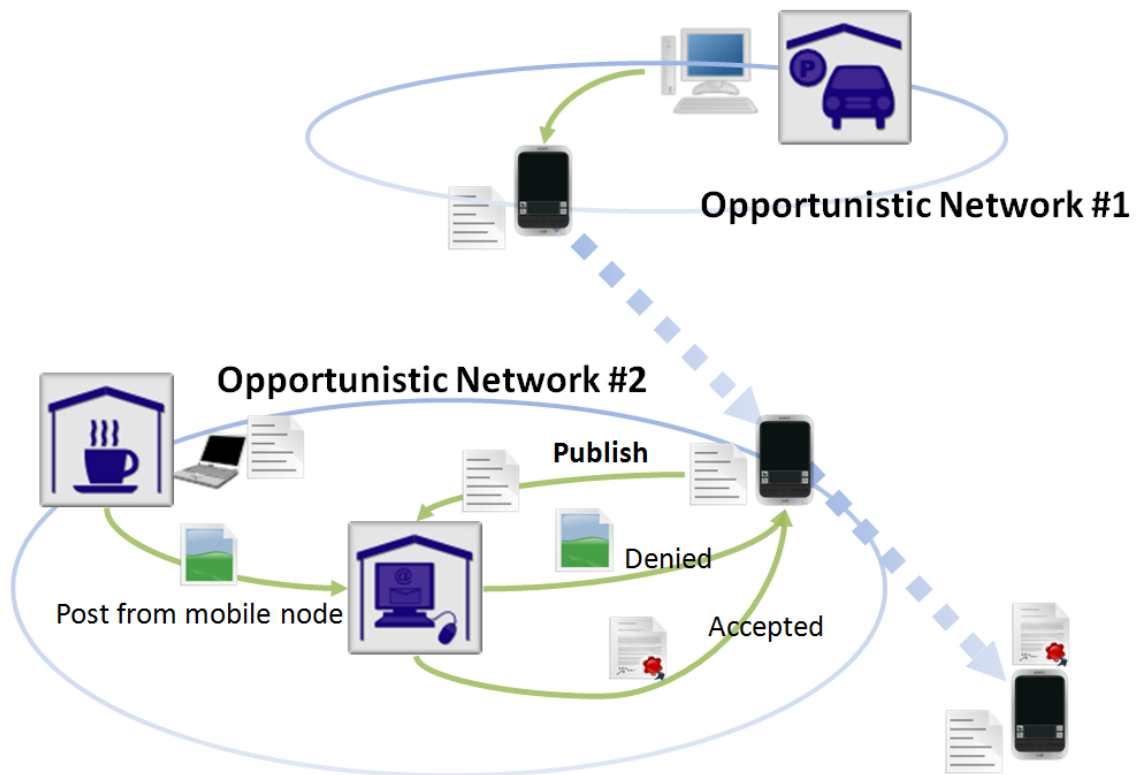


Figure A.4: Usage Scenario for the BBS-ONE. A mobile node moves from one isolated opportunistic network to another one, carrying desired information to a location where no connection to the infrastructure. Dotted lines indicate the direction of the movement of the node in and out of an area with wireless ad-hoc connectivity.

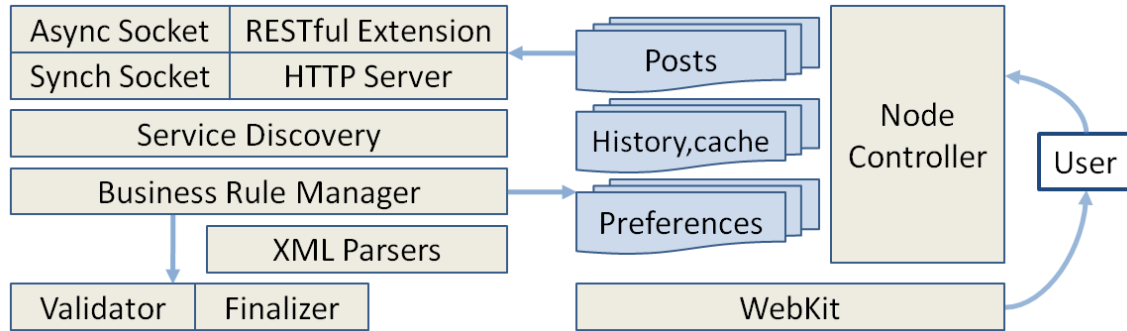


Figure A.5: System Architecture of the BBS-ONE system. The diagram shows the networking, service discovery and rule validators, the post and node managers and how the components interact.

information sharing, online forums, traditional BBSes, and synchronization applications for portable devices such as PDA and mobile phones.

A.2.2 Forums and BBS Software

Newsgroups (list servers) and web-based forums became quickly popular on the Internet, providing a forum for online communities to form and discuss topics and events, based on either interests or locality.

Generic BBS systems utilize a centralized system hosted on one or more physical servers which is controlled by system administrators and forum administrators. However, often, users are allowed to generate, post and exchange content to their heart's content, unless such content violates the terms set by the forum administrators.

BBSes can be expanded one step further to become social networking sites used to build online communities. The most prominent use of these online communities recently has been seen during political events, such as DeanSpace [Civicspacelabs, 2008] and Facebook Statuses for Obama [Facebook, 2004].

A.2.3 BBS in Opportunistic Networks

BBS-ONE enables us to share information in opportunistic networks. Unlike the traditional BBS, our system does not require a central server machine to store all data from users.

Our implementation needs to work on devices running on opportunistic networks with high mobility and churn rates and in the absence of a central server. Hence, we need a unique way of solving the problem of running a BBS or forum. We need to solve the issue of sharing data in a disconnected network. The nodes need to be aware of such mobility, and be able to recover gracefully in case of any abrupt cessation of communication.

A.3 Service Model

The fundamental requirements of bulletin board system are providing a virtual space where people can interact with others by sharing information of interests. BBS-ONE fulfills this requirement. Throughout the system, data management and networking schemes are used to provide bulletin board and forum features.

Figure A.4 shows one possible scenario in which a mobile user whose device is the part of the system enters and leaves opportunistic networks enroute to a location where no wireless service is available. BBS-ONE works in both networked area (where the node is connected to an infrastructure such as the Internet) as well as a disconnected area (where the only connections are local connections). These scenarios show an example of the store-carry-forward scheme scenario, and the system can be utilized for the different use cases. User A stores information acquired based on a user's preferences consisting of keywords.

A.3.1 Data Management

A.3.1.1 Post Handling

A post is a user submitted message in traditional BBSes, and we define it the same way for BBS-ONE. A post is the conceptual elementary entity of information that users want to share or to acquire through this system. With a post, user can describe their idea and information as text with attached resources such as images, music, and possibly other types of files. We regard a post as the elementary form of information in this system, in that all operations, such as creating and sharing a post, are applied to posts based on privileges which can be determined by examining the author of a certain post and other permissions set on that particular post.

It also has a life cycle that a post must follow. According to the current state of a post in the life cycle, BBS-ONE determines if a specific post can be shared, edited, or limited in terms of dissemination.

Each post has a field to keep information about the nodes which the post traversed and other trace information. It is possible for several nodes to acquire the same post containing the same content at a certain time. However, it is possible for several copies of an identical post to have different trace information in it, because it is quite common for a post to traverse different routes.

A.3.2 Deployment

Our BBS-ONE can operate in two modes: a client-server mode in the presence of a stationary node, and a true P2P mode in the absence of an stationary node.

A.3.2.1 Operation in presence of stationary node

The BBS-ONE stationary node refers to a node that is placed in a static location and does not move. Such a stationary node could be deployed at locations with high traffic, such as subway stations. The stationary node stores data from incoming nodes and forwards data to newcomers. When a node comes in a network area, it finds the information of a neighboring stationary node through service discovery. When it finds an stationary node, the node sends all posts to that stationary node, including metadata to avoid duplication.

A.3.2.2 P2P operation

In the absence of an stationary node, there is no facility for storing data, so all operations share data among the mobile nodes and they work in a totally peer-to-peer fashion. Every node needs to find connection information using multicast service discovery, and it directly connects to the peer node to exchange posts. When a node wants to acquire a post containing the desired content, it first sends search keywords to neighboring nodes found during the discovery phase, and then one of nodes that has the related posts sends an offer to it in order to get an acceptance response.

A.3.3 Networking

The various networking operations in BBS-ONE are made to correspond to existing HTTP and REST [Fielding, 2000] features. This way, our application uses existing standards, and it is also possible for other developers to build compatible implementations that can interact with existing networking and web forum applications using XML-RPC and other web service methods.

A.3.3.1 Pull-based transfer

The transfer of data in BBS-ONE applications is primarily through a pull-based mechanism. The major advantage of this approach is that clients can access information when they need it. The one major disadvantage is that the nodes will not be easily aware of any network connection below the application layer. In order to check if there are any updates on posts in nearby peers, clients will need to constantly poll for updates, which might reduce the battery life. Traffic will increase based on the number of existing nodes, no matter how many updates nodes have.

A.3.3.2 Push-based transfer

The main advantage of push-based exchanging is that updates will arrive when available, not when needed and hence continuous polling with all other nodes is not needed. Being notified of updates is an efficient way for being made aware of when communication has to be established. However, in contrast to the previous pull-based approach, the traffic to inform a node of updates will increase as a function of the number of nodes holding updates.

Furthermore, one of the characteristics that this system has, is that providing information (which is pushed to the reader) is determined according to the preference of the reader. The preferences are supposed to be multicast first, before the information is delivered.

A.3.3.3 Publish/Subscribe

The BBS-ONE system does not use the traditional publish/subscribe messaging paradigm; rather subscribers in the system multicast their keywords of interest. Publishers get to

know which nodes are interested in certain contents or data, and initiate exchange of posts by offering them data which are supposed to be pushed to the subscribers.

Also, a stationary node can act as a stationary or mobile repository. This node is able to perform a store-and forward function to deliver data from original authors to readers. Thus, even if nodes that make contact with this access point are disconnected with each other, they are still able to receive information from nodes that came earlier.

A.3.3.4 Epidemic Dissemination

In epidemic dissemination, a peer node finds information about how it can communicate with another node using connection information obtained during the discovery phase, and is able to initiate a connection with a specific node.

When a node has a post to be published but there no access point in the network that the node has just found, it picks up another node randomly to send posts for advertisements and public posts in order.

However, epidemic dissemination cannot be applied to every type of data in the system, since the right to choose which posts can be delivered is on the receiver, not on the data holder. Every node is restricted by how many posts it can disseminate, and this is based on how much it contributes when it comes to publishing articles that are of interest to the user, based on the keywords they have expressed interest in.

A.4 Design and Implementation

In this section we explain our design and implementation of BBS-ONE for two platforms, Java virtual machine and iPhone, which allow the given service model described in the previous section, to work.

A.4.1 Architecture Overview

Service discovery and the 7DS [Moghadam *et al.*, 2008] and BonAHA [Srinivasan *et al.*, 2009b] frameworks are the architectural components that support the system's working in an opportunistic networking environment, while abstraction layers and the model-view-

controller pattern describe how the system has been written from the perspective of actual implementation.

Figure A.5 shows the overall architecture of BBS-ONE.

A.4.2 Implementation

To implement BBS-ONE, we developed a command-line version of the system in Java that runs on Windows, Linux and Mac OS platforms. This Java version is built using our BonAHA framework and allows a user to create, edit and share his or her posts with others.

The iPhone implementation of the BBS-ONE includes a GUI that is very similar to other iPhone applications. The implementation allows a user to see other posts on the network and be able to view them through a scroll screen based user interface. The iPhone implementation was written in Objective-C. (These applications were written in 2008-2009, shortly after Apple first introduced the iOS developer kit.)

Because the iPhone platform does not support Java, we rewrote portions of the BonAHA library in Objective-C and integrated it into our application in order to develop the discovery mechanism of the BBS-ONE application.

The basic components of the implementation are Zero Configuration (Zeroconf) using Apple's Bonjour implementation, an internal HTTP proxy (a HTTP server) and its RESTful service extension, a cache manager as well as data service, and user interfaces, implemented both on Java virtual machine using JRE1.5 and iPhone OS 2.1 or later.

A.4.2.1 Data Format and Identification

A post is identified based on fields such as author information, the title and the description of the post. These fields contain hashed values. Some fields in a post such as trace information cannot be subjected to hashing in order to identify a post. We used MD5 as the hash function to identify a post.

We have attempted to make BBS-ONE meet the spirit of Web 2.0 and be as easy to reverse engineer so that alternative implementations compatible with BBS-ONE can be built. The default format of files presenting a post is based on XML. The schema is described by XML Schema definition language (XSD) [Thompson *et al.*, 2001] and the publication of

the post information contained in each node is based on the Atom feed format [Nottingham and Sayre, 2005].

A.4.2.2 Versioning and Expiration

Once a post leaves its original node, it is not easy for the author of the post to do any work on it, especially when the Post arrives on another network and it has been completely isolated from the connected environments. In many cases, disseminated data cannot be reached by any other nodes that work in physically separated networks.

Accordingly, it is more reasonable that, if a post has been pushed to another node, it is regarded as an independent data set that should identify itself. In such a case, any modification generates a new version of the original post.

A.4.2.3 Platforms

We wanted to implement the BBS-ONE application on a variety of platforms in order to prove its usefulness, while at the same time, meet reasonable goals and deadlines. Hence, we built the application using Java, which we have tested on Windows, Linux and Mac OS. It is also quite likely that this application runs on other operating systems and platforms that are Java compatible.

In choosing a mobile platform, we chose the iPhone/iPod platforms because it is widely deployed [Times, 2008].

One major difference in the HTTP proxies on the iPhone and the command-line Java version of BBS-ONE is the socket layer underlying the application layer protocol. The Java version can use multiple threads and blocking sockets, but in the iPhone version, the implementation of the HTTP server relies on an asynchronous style or non-blocking sockets. However, this does not have any major impact on our implementation.

A.4.2.4 Mobile Nodes

For the iPhone OS, BBS-ONE implements only the mobile reader version, allowing a user to publish, advertise and gather post information from either stationary nodes or other mobile

nodes. The GUI for this version is based on the native user interface on iPhone as well as a web browser page displaying posts.

A.4.3 Related Work

The original BBSes were virtual communities created when individuals or service providers allowed people in the local communities to dial-in and connect to their systems, download, and upload files and other data, as well as share information with others in the community.

USENET [Horton and Adams, 1987], which evolved from UUCP, allowed users to read and post messages to categories known as newsgroups. It supported threaded discussions and could be accessed by any compatible newsreader software. USENET is widely regarded as a precursor to today’s web forums.

However, with the advent and popularity of the Internet, Internet-based BBS systems became more and more popular. A huge number of forums (such as Google Groups and others on Facebook) now exist around the world, spanning almost every conceivable topic. Due to the open nature of these forums, several have implemented some form of monitoring and checking.

Among forum use on disconnected, ad-hoc or opportunistic networks, there is very little work. The primary reason for this seems to be that forums are naturally oriented towards online, long-term communities.

The only available work similar to ours in this field seems to be the JXTA forum software [Halepovic and Deters, 2002]. This forum software, built on top of Sun Microsystems’s JXTA protocol and framework, implements a forum software by converting an existing client-server system into use in a P2P system. However, even this system requires the use of a connected network. The authors state, “total decentralization is often neither necessary nor desirable.” However, opportunistic networks are completely decentralized and mobile. Further, the system requires a naming service, which requires some long-term or super nodes, as well as setting up peer groups to avoid a flat search structure. This is not possible in opportunistic networks either.

A.4.4 Discussion and Future Work

There are some possible improvements to our current BBS system. For example: exchanging personal contacts in places such as conference halls or public transportations might be one way to utilize the peer-to-peer exchanging information in an isolated opportunistic network, if permissions are considered.

We can also consider the functionality of supporting concurrent transmission of posts, and limiting the number of times a post moves from/into other nodes. Preventing spamming using identical posts and prohibited words also needs to be considered. Also, a method for handling partially transmitted posts needs to be devised.

A.4.5 Conclusion

In this section, we presented BBS-ONE, a system that enables forum and BBS functionality in opportunistic networks that are highly mobile and disconnected from the network. BBS-ONE allows users to exchange information and posts even in the absence of central servers or connection to wide-area networks.

We also presented our implementation of BBS-ONE on desktop platforms using our Java implementation, as well as on the iPhone and iPod platforms.

We believe that the BBS-ONE application provides a unique and novel implementation of BBS and forum functionality in a new networking scenario that is rapidly evolving yet has very few functional applications. BBS-ONE, as with our other applications for opportunistic networks, will provide users in disconnected and opportunistic networks to be able to share data with relative ease without needing some physical form of data transmission or file copying.

We have attempted to build BBS-ONE on as many standard technologies as possible, such as the Bonjour protocol stack, W3C standards (HTTP, XML and related technologies), and common concepts of architecture such as REST style and MVC pattern so that compatible implementations for the same or other platforms can be easily built.

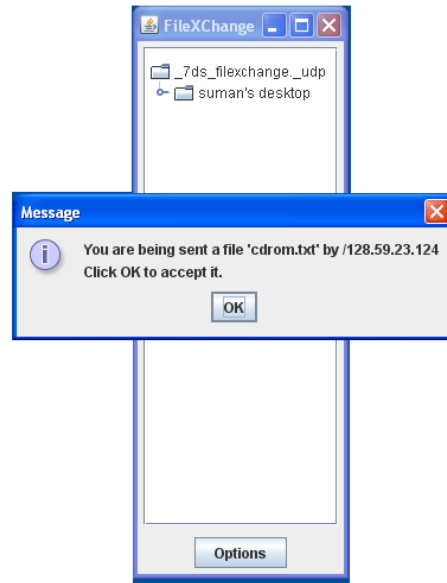


Figure A.6: A screenshot of the FileXChange system.

A.5 FileXChange: Drag-and-Drop File Sharing in Opportunistic Networks

Because most file sharing applications and protocols assume the use of always-connected networks with central servers, there are few suitable applications for file sharing systems on opportunistic networks, and users have to resort to using archaic methods such as using USB flash drives to copy data. (The one exception is Apple’s AirDrop [Apple, 2012] which was introduced in 2011, but that only works on Apple devices and not on other platforms.)

In order to alleviate this problem, we developed FileXChange, a drag-and-drop file sharing system, to allow users to share files in various network environments, including opportunistic and transient wireless networks. FileXChange is built on top of our BonAHA platform, which allows users and applications to automatically detect each other on the opportunistic networks without any manual configuration or central servers. FileXChange is designed to be disruption-tolerant to allow file exchange over opportunistic networks and handle failure gracefully. It can run on any operating system with a Java Virtual Machine.

FileXChange allows users to transfer files through an intuitive drag-and-drop GUI while

automatically finding peers in opportunistic networks. FileXChange is able to handle peers entering and leaving the network at any time and displays a list of all peer nodes to the user. The user interface for FileXChange is shown in Figure A.6.

We use service discovery using our BonAHA library [Srinivasan *et al.*, 2009b] to implement FileXChange. The service discovery mechanism allows devices to detect each other through their unique member names, and users can view a list of peers through the GUI.

TCP sockets are used to provide the file transfer service. Two kinds of data are exchanged for each file: control and information data including the file's name, length, sender's information, recipient's response and file data. The control and information data are sent before the file data, so the recipient can choose to accept or decline the file based on that information.

There are five kinds of disruptions we handle through the control messages: (1) sender cancels the transfer; (2) sender leaves the network during the transfer; (3) receiver accepted the transfer but fails to choose the target file within a certain threshold time (before data transfer); (4) receiver cancels the transfer during data transfer; (5) receiver leaves the network during file transfer.

A.5.1 Acknowledgment

This work was supported by NSF Grant No. 04-54288 and No. 04-12025.