

ABSTRACT

PAPAPANAGIOTOU, IOANNIS. Design and Optimization of Mobile Aggregation Networks. (Under the direction of Michael Devetsikiotis.)

As more and more users access the Web through mobile devices, their traffic patterns need to be better understood. This is important both to wireless network carriers and to network administrators, as they provision their networks for the expected growth of smartphones and tablets.

This thesis exposes some of the differences in terms of the networking behavior of smartphones, with a control group, namely laptops. In order to achieve this goal we introduce a device identification technique that is based on data mining. We show that the proposed technique can identify more than 97% of the devices. We perform a thorough study of the networking traffic generated by these devices. We focus on traffic protocols in which smartphones and laptops have distinct differences. Hence, we first look into the DHCP traffic behavior, and the implications of each operating system (e.g. iOS, Android, Windows, Mac OS X, Linux) to the network address space utilization. We propose a new DHCP leasing strategy which improves the network address utilization. Secondly, we examine Web traffic, and identify the differences in the multimedia content delivery, and the implications to caching. For instance, we see that smartphone browser caches are not very effective, and propose policies that may provide maximum savings with only a 10MB browser cache. Similarly, we evaluate proxy caches, and show how important is to handle partial downloads.

We extend our caching work in techniques that can identify redundant bytes, in a finer granularity, for example the byte level. We show the extra savings that can be attained from the implementation of a byte cache in a wireless network and indicate the limitations that these systems introduce. We propose a hybrid system, that not only resolves these limitations, but may also provide extra savings. We evaluate these techniques with actual traces collected from an educational institution and from a corporate environment.

Finally, we propose further optimizations from the architectural point of view. We show through a Network Design Problem that ISPs may consider re-engineering their infrastructure to accommodate the increasing demand for video and P2P traffic. Our results suggest that ISPs should distribute some of the functionalities closer to the subscriber, as well as invest intelligent systems with higher line card/interface availability and caching systems.

© Copyright 2012 by Ioannis Papapanagiotou

All Rights Reserved

Design and Optimization of Mobile Aggregation Networks

by
Ioannis Papapanagiotou

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering and Operations Research

Raleigh, North Carolina

2012

APPROVED BY:

Michael Devetsikiotis
Co-chair of Advisory Committee

Harilaos Perros
Co-chair of Advisory Committee

Weney Wang

Do Young Eun

Robert D Callaway

DEDICATION

To my mother Vasiliki, whose memory will always be alive.

BIOGRAPHY

Ioannis Papapanagiotou received the Dipl. Ing. degree in Electrical and Computer Engineering from the University of Patras, Greece in 2006, and the M.Sc. in Computer Engineering from North Carolina State University in 2009. He is currently a candidate for the Doctor of Philosophy degree in Computer Engineering and in Operations Research. He has been awarded the best paper awards in IEEE GLOBECOM 2007 and IEEE CAMAD 2010, and he is also the recipient of the IBM PhD Fellowship, Academy of Athens and A. Metzelopoulos scholarships. He is also an IBM Partner, and upon completion of his doctoral degree, he will join the Emerging Technology Institute of IBM Software Group as an Advisory Software Engineer. His interests lie in the areas of data deduplication, mobile traffic analysis, data mining and network design.

ACKNOWLEDGEMENTS

I wish to express my gratitude to my advisor, Michael Devetsikiotis. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own, but at the same time provided me the guidance, encouragement and support to complete my work. I am also deeply indebted to Bob Callaway whose assistance and stimulating suggestions have been thought provoking during the research process. Moreover, I would like to thank Harry Perros, the co-chair of my committee, for his advices and for being a great queueing theory instructor. Finally, I would like to thank Weney Wang and Do Young Eun for their feedback and comments on my research work.

I was also fortunate to have worked with Vasileios Pappas and Erich Nahum, from IBM Research, who provided me with excellent feedback, and helped me reach a higher standard in my research. I am also grateful to Matthias Falkner, from Cisco, for his technical advices in the network optimization model. In my office, Network Performance Research Group, I was surrounded by friendly people with whom I spent countless and happy hours. In my department, I am grateful for the support of numerous people; more specifically to Marhn Fullmer, manager of the NetLabs, who has checking everyday on my projects; given my unique dual-major situation, I would also like to thank Ms. Elaine Hardin and Ms. Kendall Del Rio for making sure that I am following the graduate program rules properly.

I am also grateful to the benefactors of my research, Ms. Vasiliki Bekiari-Vekri for the Academy of Athens scholarship, and Mr. Andreas Metzelopoulos for the University of Patras alumni scholarship. Finally, I would like to thank IBM for offering me the prestigious IBM PhD Fellowship.

Very special thanks goes to my family, especially to my mother who bore me, raised me, taught me and loved me. Finally, I would like to thank the cradle of my life, my wife Sofia Kotsiri. This dissertation would have been impossible without your unflagging love, support, smile, and songs.

TABLE OF CONTENTS

List of Tables	vii
List of Figures	viii
Chapter 1 Introduction	1
1.1 Contributions of this Dissertation	4
1.2 Outline of this Dissertation	5
Chapter 2 Smartphones vs Laptops: A Traffic Study	7
2.1 OS Fingerprinting	7
2.1.1 Device Identification Background	7
2.1.2 Device Classification	8
2.1.3 Trace Collection	10
2.1.4 Association Rule Development	10
2.2 DHCP Lease Time Allocation in the Smartphone Era	14
2.2.1 Lease Time Analysis	14
2.2.2 DHCP Lease Policies	19
2.3 Web study and caching implications	22
2.3.1 Trace Collection	22
2.3.2 Web Traffic	22
2.3.3 Browser Cache Emulation	33
2.3.4 Proxy Cache Emulation	35
2.4 Related Work	39
2.5 Conclusion	42
Chapter 3 Data Redundancy Elimination in Wireless Networks	43
3.1 Byte Caching in Wireless Networks	45
3.1.1 Methodology	45
3.1.2 Results	50
3.2 Trace Characteristics	50
3.2.1 Proxy Cache	52
3.2.2 Byte Cache Match-Match	53
3.2.3 Byte Cache Chunk-Match	56
3.2.4 Comparison Analysis	59
3.3 Object and Byte Caching: A Hybrid Approach	63
3.3.1 System Design	64
3.3.2 Module Design Principles	65
3.3.3 Implementation and Data Set	68
3.3.4 Results	69
3.4 Related Work	75
3.5 Conclusion	77
Chapter 4 Network Design of Aggregation Architectures	78

4.1	Architectural Comparison	79
4.1.1	Network Elements in an Aggregation Network	80
4.1.2	Centralized Single-Edge Architecture	80
4.1.3	Centralized Multi-Edge Architecture	81
4.1.4	Distributed Single-Edge Design	81
4.2	Methodology and Assumptions	83
4.2.1	Overview	83
4.2.2	Definitions	84
4.2.3	Variables	85
4.3	Modeling the Traffic Flows	88
4.3.1	Internet (non local) Traffic Class	89
4.3.2	P2P (Local) Traffic Class	90
4.3.3	Multicast Traffic Class	91
4.4	Edge Design Model	92
4.4.1	Constraints	93
4.5	Model Implementation	101
4.6	Evaluation and Results	103
4.7	Discussion	109
4.8	Related Work	110
4.9	Conclusion	111
	Chapter 5 Conclusions	113
5.1	Summary of this Dissertation	113
5.2	Future Work	115
5.2.1	Dynamic Scheduler for the selection of caching modules	115
5.2.2	Quality of Service in Byte Caching	116
5.2.3	Distributed Byte Caching for Mobile Networks	116
	References	117
	Appendices	124
	Appendix A Dynamic Host Control Protocol	125
	A.1 DHCP Functionality	125
	A.2 DHCP Request Header	126
	A.3 DHCP Behavior and Sleeping Policies	127
	Appendix B Proxy Cache Details	129
	B.1 Caching Metrics	129
	B.2 Proxy Cache Rules	129
	Appendix C Other Research	131
	C.1 Research on Performance Analysis of Wireless Networks	131

LIST OF TABLES

Table 2.1	Dataset Properties	10
Table 2.2	Common Association Rules for Device Classification	12
Table 2.3	Distribution of Devices in the Trace	13
Table 2.4	DHCP Requests	15
Table 2.5	Relative (%) of DHCP Request Types	15
Table 2.6	TCP/IP Application Breakdown (%)	23
Table 2.7	HTTP Request Method (%)	23
Table 2.8	Conditional GET Requests (%)	24
Table 2.9	Zipf Law fit for Request Popularity	31
Table 2.10	Top 20 Web Sites (Requests)	32
Table 2.11	Top 20 Web Sites (Bytes)	32
Table 2.12	Median browser cache hit rates.	33
Table 2.13	Cacheable Bytes and Requests per OS	34
Table 2.14	Hit rate for infinite sized proxy space with and without handling Range Offsets	37
Table 2.15	Smartphone Related References	41
Table 3.1	Trace File Details	50
Table 3.2	Average chunk size	57
Table 3.3	Hash Memory Requirements (GB)	61
Table 3.4	RE cache module constants and variables definition	67
Table 3.5	Trace File Details	69
Table 4.1	Modeling Input Parameters	86
Table 4.2	Types of Edge Functionalities t	86
Table 4.3	Volumes of traffic	88
Table 4.4	Input Parameters for Aggregation Network Characteristics and Interface Properties	101
Table 4.5	Network Element Features	103
Table 4.6	Cost Breakdown in thousands of dollars for IPTV=6Mbps and HSI=1Mbps108	108

LIST OF FIGURES

Figure 2.1	CDF of the number DHCP <i>Request</i> messages in the Corporate and Educational network	16
Figure 2.2	CDF of the interarrival time of <i>Request</i> messages with different <i>Lease Time</i> setting	17
Figure 2.3	Calculating Active and Inactive times.	18
Figure 2.4	CDFs of Active and Inactive durations.	19
Figure 2.5	(a) Address space utilization (b) Broadcast messages (c) Server load, versus lease time, per device type	20
Figure 2.6	(a) Address space utilization (b) Broadcast messages (c) Server load, for various policies	21
Figure 2.7	Response Code per Operating System	24
Figure 2.8	Content Types per Operating System	25
Figure 2.9	CDF of downloaded bytes per content type	26
Figure 2.10	Response Codes for Multimedia Traffic across all types of devices	27
Figure 2.11	Video Content Type across all the devices	28
Figure 2.12	Applications that generate multimedia traffic in iOS vs MAC OS X (bytes,responses)	29
Figure 2.13	Cumulative Distribution of the iOS multimedia traffic per object size	30
Figure 2.14	HTTP downloads per device type, iOS content type, and iOS application	30
Figure 2.15	Comparing browser cache hit rates by varying (a-b) the size, (c-d) the storing policies.	33
Figure 2.16	Campus Network with a Web Proxy Cache	36
Figure 2.17	Proxy cache hit rate with several revalidation strategies and persistent storage sizes.	37
Figure 2.18	Proxy cache savings for infinite sized cache per content type for each operating system	38
Figure 2.19	(a) Effect of Prefetching on Savings and (b) Conditional Caching for Partial Downloads	38
Figure 3.1	Byte Cache Architecture	48
Figure 3.2	Protocol and Application Distribution of bytes in the traces	50
Figure 3.3	HTTP Content Type Byte and Object Distribution	51
Figure 3.4	HTTP Byte and Object Hit Rate for a Web Object Cache per Content Type	52
Figure 3.5	HTTP Relative Savings and Cacheable Content for a Web Object Cache per Content Type	53
Figure 3.6	Optimal Fingerprint Size for each Protocol (Max Match byte cache)	54
Figure 3.7	Optimal Fingerprint Size for each Application (Max Match byte cache)	55
Figure 3.8	Protocol Decomposition and Savings (Max Match byte cache)	55
Figure 3.9	Application Decomposition and Savings (Max Match byte cache)	56
Figure 3.10	HTTP Content type decomposition and savings (Max Match byte cache)	57
Figure 3.11	Total Savings as a function of Y	58
Figure 3.12	TCP and UDP Relative Savings as a function of the modulo operator Y	58

Figure 3.13	Byte Cache Max-Match and Chunk-Match vs Web Cache (identifiable HTTP content)	60
Figure 3.14	Byte Cache Max-Match and Chunk-Match vs Object-level Cache (all traffic)	60
Figure 3.15	Max-Match Byte Cache vs Web Object Cache (all traffic)	61
Figure 3.16	Proposed Architectural Approach of a Hybrid Cache	64
Figure 3.17	Protocol and application distribution of bytes in the traces	69
Figure 3.18	Savings and memory overhead as a function of the input parameters. . . .	70
Figure 3.19	Savings for proxy cache standalone, RE standalone and hybrid system. . .	71
Figure 3.20	Redundancy for proxy cache standalone, RE standalone and hybrid system.	72
Figure 3.21	Memory requirement to represent data for proxy cache standalone, RE standalone and hybrid system.	73
Figure 3.22	CPU throughput for an RE standalone and the hybrid system.	73
Figure 3.23	Disk Capacity Allocation for each module for an 1GB (a) and 10GB (b) hybrid system (RE module size - Proxy module size)	75
Figure 4.1	Centralized Single Edge Overlay Architecture	81
Figure 4.2	Centralized Multi Edge Overlay Architecture	82
Figure 4.3	Distributed Single Edge Overlay Architecture	83
Figure 4.4	Cost comparison for the big service provider per aggregation network for 0.5Mbps and 4Mbps of Local & Non Local traffic (Internet)	104
Figure 4.5	Cost comparison for the small service provider per aggregation network for 0.5Mbps and 4Mbps of Local & Non Local traffic (Internet)	105
Figure 4.6	Used ports vs available ports in an aggregation location with variable internet and constant IPTV=6Mbps traffic	106
Figure 4.7	Traffic capacity and bandwidth of edge locations plotted with variable internet traffic and constant IPTV=6Mbps traffic	107
Figure 4.8	Number of subscribers that are terminated per location with variable internet traffic and constant IPTV=6Mbps traffic	108
Figure A.1	Exemplar DHCP Request message and the important information for the OS Fingerprinting	126

Chapter 1

Introduction

As more and more users access the Web through mobile devices, their traffic patterns need to be better understood. This is important both to wireless network carriers and to enterprise network administrators, as they provision their networks for the expected growth of smartphones and tablets. In fact, as smartphones reach the 1 billion threshold, they are already contributing more than 10% of Internet traffic [20]. Recently a number of studies have been conducted on smartphone usage [39, 40, 42, 47, 52, 63, 93, 96]. Yet many aspects of smartphone traffic behavior have not been fully apprehended. This may have implications for network design and provisioning, particularly if smartphone traffic is significantly different from other devices, such as PCs. This thesis exposes some of the differences by comparing the network access behavior of smartphones with a control group, namely laptops.

One of the key issues with the inclusion of smartphone devices is the configuration of the Dynamic Host Control Protocol (DHCP) [38]. DHCP enables devices to attach to networks without manual configuration. It does, however, require manual configuration of access policies at the DHCP servers. One of the most critical parameter of the DHCP server configuration is the lease duration, indicating how long a device can use an IP address. Setting up proper DHCP lease values has been an art rather than a science. Long lease times can lead to exhaustion of the network address pool assigned for DHCP, while short ones can result in increased broadcast traffic and unnecessary activation of wireless interfaces by power limited devices. There have been few studies on the DHCP lease times [32, 59], both of which were done before the onslaught of smartphones in local area networks. Smartphones present a challenge in correctly configuring DHCP leases. A single device may acquire multiple IP addresses during a day due to its continuous attachment, in either asleep or active mode, with the campus-wide wireless networks. For example, as a student moves from one side of the campus to another, her devices can re-associate with various campus subnets, acquiring a different address each time. In this scenario, setting DHCP lease times even as low as one hour may not necessarily be enough to reduce network address utilization.

Another important aspect is the Web traffic that the mobile devices are introducing to the network and the corresponding implications on caching designs. We look into the HTTP traffic, as it represents the majority of the bytes in the network. We examine standard metrics

such as Web requests and responses, popularity, content type, and multimedia downloads. We focus mainly on those areas where the traffic of the two device categories differ. We study the differences of how video is delivered over HTTP in mobile devices. We also examine the effectiveness of caching in detail, both at the browser and at a proxy. Nonetheless, proxy caches perform only object level deduplication. Hence, they cannot identify redundancy across objects that are slightly different, or objects that are the same but have a different URL. Moreover, we identified a significant amount of similar web data that are not cacheable [17], and there is also a portion of the redundant traffic that is not HTTP.

Hence techniques that can remove redundant bytes at a smaller granularity may be more efficient than proxy caches. Protocol-independent redundancy elimination techniques that perform deduplication at the byte level have quickly emerged as commercial products in WAN and enterprise networks [9, 35, 11]. They are powerful techniques that help to bridge the gap by making WAN communication more efficient through elimination of redundancy in traffic. Their main advantages are that they a) perform redundancy elimination across a wider range of protocols and applications than just HTTP; b) perform deduplication at a finer granularity (bytes rather than objects) and thus can identify redundant content in partially modified files; c) can partially or fully cache Web content that would be have been classified as uncacheable by Web cacheability rules and/or HTTP header fields.

These techniques are known by different names: network data de-duplication, network redundancy elimination, etc. For simplicity and consistency, in this thesis we refer to these techniques as *byte caching*, both to stress the granularity on which redundancy is determined (sequences of bytes) and to contrast with the earlier approach of Web object caching. Indeed, Web object caching can be seen as a form of redundancy elimination, where full Web objects that have been already retrieved across the network are not retrieved again.

While byte caching has been examined in the context of enterprise networks and data centers, there has been little insight on its applicability in the wireless context. Our research both examines the characteristics of traffic being generated by mobile devices and evaluates the achieved bandwidth savings by exploiting redundancy elimination techniques. Our analysis shows that the amount of HTTP traffic being generated from mobile devices is higher than that reported from the backbone of an enterprise or a university network [26]. The latter would possibly include traffic from servers or data centers, which is highly redundant. In contrast, we perform a trace driven analysis on the effectiveness of deduplication techniques in the wireless context.

Byte caches also have some limitations. A sliding window moves byte and byte, generating the fingerprints, using the Rabin fingerprinting algorithm [81]. Each one of them is compared with a global constant to derive the boundaries of each chunk. However, performing these steps over all of the data may create a bottleneck in higher bandwidth links [65]. Moreover, the

hash overhead per object in proxy caches is much smaller than the hash overhead per chunk in byte caches. Hence byte caches require a significant amount of memory to store the hashes. In this thesis, we propose a hybrid redundancy elimination technique. The system consists of a scheduler, a byte cache module and a proxy cache module. The decision on which module the byte stream should flow through is determined by the scheduler. The benefits of such an approach can be summarized as follows:

- Fewer hash computations are performed, therefore allowing our hybrid system to be deployed within higher bandwidth links.
- An application layer compression scheme, instead of the standard IP packet-based RE, should lead to better savings and storage overhead.
- A reduction in the memory overhead compared to a standalone RE system. As some byte streams flow through the proxy cache module, they do not need to be broken in chunks and unnecessary hash generation and storage can be avoided.
- The proposed approach can be implemented on the encoding box of a WAN optimization system without significant architectural modifications.

The last chapter of this thesis focuses on a network design & dimensioning problem. This work has been motivated by the lack of established principles on the design of next generation aggregation architectures. More specifically, most of the design problems address the issue of deploying new infrastructure. However, the cost to build out new infrastructure may be prohibitively expensive. In addition, vendors have introduced multipurpose edge “systems” (backplanes) that incorporate different functionalities in modular “sub-systems” (line cards). Finally, Internet Service Providers (ISPs) have to deal with the explosion of video traffic in different formats, e.g. multicast, peer-to-peer, mobile etc.

Hence, it is of vital importance for the ISPs to re-engineer their infrastructure to meet these challenges. Moreover, Network Design Problems (NDPs) have to be revisited to include those multipurpose edge systems, instead of single-functionality devices (such as L2 switches or L3 routers). Considering the above challenges, we set out to:

- Define the possible aggregation architectures: 1. *Centralized Single-Edge*, 2. *Centralized Multi-Edge*, and 3. *Distributed Single-Edge*. We further differentiate based on whether the edge system is clustered or maintains its functionalities in a single box.
- Develop a network design model that goes beyond the well investigated *location* and *dimensioning* problems. Our modeling approach is applied to both designing an aggregation architecture, as well as upgrading the current infrastructure.

- Propose models based on edge “systems”, rather than network elements. Edge systems may support different types of intelligence, either as part of the backplane or as part of the modular “sub-systems”, i.e. high-end line cards.

Finally, we have evaluated the model with two close-to-real-life scenarios and multiple traffic profiles, based on data provided by EU ISPs. Our results yield that ISPs will benefit from distributing the functionalities closer to the subscriber. Furthermore, sensitivity analysis suggests that ISPs should invest on systems that support more line cards and interfaces, rather devices with higher bandwidth.

1.1 Contributions of this Dissertation

- **OS Fingerprinting:** We introduce a data-mining technique for device identification based on DHCP header information. We show that earlier approaches to device identification, which rely on the User-Agent header, are not nearly as accurate. Our approach identifies $> 97\%$ of the devices in two campus networks, as compared to 84% for approaches using the User-Agent and MAC layer information.
- **DHCP Traffic Analysis and Lease Time Policies:** We show that DHCP message exchanges vary both across device types (e.g., laptops, smartphones) and across operating systems (e.g., iOS, Android, Windows, Mac OS X, Linux), with each device type contributing a different amount of DHCP related traffic and having a varying effect on the network address utilization. We propose a new DHCP leasing strategy that does not require any protocol changes, and which takes advantage of the varying usage patterns per device type. Using simulation results, driven by our traces, we show that the new strategy, compared to current approaches, improves the network address utilization sixfold without considerably increasing the DHCP overhead.
- **Web Traffic:** We examine Web traffic across both smartphones and laptops. We show that smartphone browsers make less use of HTTP caching mechanisms, as compared to laptops. Multimedia content (audio and video) is a large fraction of traffic across all device types, and iOS users in particular consume a large amount of video in terms of bytes. Partial downloads occur frequently, and larger transfers are more likely to be aborted, especially if the content is streaming video. Most video is retrieved using progressive downloads with range requests.
- **Browser Caching:** We examine browser caching for each device type and show that cache savings vary widely across users, with benefits from 0 to 80% in terms of both request hit rates and byte savings. We demonstrate that smartphone browser caches are not as effective as laptop caches. Adding a small (10 MB) browser cache to each device is sufficient to capture most of the achievable savings, thus the limitation of smartphone

browser caches is not due to size. Instead, the reasons are because smartphone Web objects tend to have lower expiration times, which hinders caching, and because browsers do not cache partial content as well as laptops. We propose and evaluate several strategies for caching partial content.

- **Proxy Caching:** We evaluate the effectiveness of a Web proxy cache for our environment. We show request hit rates from 8 to 40%, and bandwidth savings from 5 to 25%. We break out caching results per content type, and show that video is not very amenable to caching. We show it is important to handle partial downloads properly to obtain the highest caching benefits, otherwise a cache can actually increase bandwidth consumption by a factor of 3.
- **Byte Caching:** Our analysis shows that a byte-level data deduplication technique provides savings up to twice as high as a Web cache. An additional 4% of redundant content from non-Web applications is also removed by caching content using a byte cache. We compare the benefits of different fingerprint sizes to determine the optimal one, as well as the appropriate technique to identify matching regions. We also devise an encoding technique on which a match is encoded by a pair of numbers called location-length pair.
- **Hybrid Redundancy Elimination:** We propose a system that consists of a scheduler, an RE module and a prox cache module. The decision on which module the byte stream should flow through is determined by the scheduler. The benefits of such an approach are (a) fewer hash computations, therefore the hybrid system can be deployed within higher bandwidth links; (b) a reduction in the memory overhead compared to a standalone byte cache; (c) application level compression, leading to better savings and storage overhead.
- **Aggregation Architecture Modeling:** We define the potential aggregation architectures based on the location of the functionalities and network intelligence. We build a modular network optimization model that is able to achieve the minimum deployment cost. We showcase that the widely implemented centralized Ethernet based single-edge architecture has reached its scalability limits, hence distributing the functionalities closer to the subscriber is far more efficient. A small SP will benefit more from the intelligence distribution than a larger SP and, independently of the size, the SPs should cluster the business functionalities on a different edge system. Finally, we showed that SPs would receive only a marginal benefit from higher capacity edge systems, and should rather focus on the proper allocation of the functionalities in their aggregation network.

1.2 Outline of this Dissertation

The outline of the dissertation is as follows:

- In chapter 2, we propose an Operating System fingerprinting algorithm based on association rule data mining. We evaluate the algorithm in two wireless campus networks. We use the algorithm to showcase the differences in the DHCP load generated by each device type. We propose a DHCP lease policy that can optimize the network resources. We extend our analysis to the Web traffic load generated by each device type. Based on these observations, we propose optimizations for browser and proxy caches.
- In chapter 3, we showcase that byte level caches can provide more savings compared to optimized proxy caches. We compare the two widely applicable techniques for data redundancy elimination and the effect of several parameters on the savings. We identify the system limitations of byte caches, and we propose a hybrid technique that can optimize the savings and at the same time optimize the system performance.
- In chapter 4, we propose network design enhancements that can lead to lower cost for the internet service providers.
- In chapter 5, we summarize our work and propose extensions for future work on the data deduplication systems and network design optimizations.

Chapter 2

Smartphones vs Laptops: A Traffic Study

2.1 OS Fingerprinting

It is useful for network operators to be able to dynamically identify a device in their network. In many cases Network Service Providers (NSP)s allow many kind of devices (e.g. laptops, tablets, smartphones, IP phones) to connect to their network (e.g. WiFi or 3G/4G). In order to manage and maintain the network, the NSP may need to determine what type of devices are using the network, and what kind of Operating System (OS) that device is running. Such information is useful, for example to ensure that software is up to date, to validate a users identity, to keep statistics or for security purposes.

Device identification is a superset of OS identification. OS identification simply identifies the operating system running on the device. Device identification identifies the underlying hardware. For example, variants of Mac OS runs on iPhones, iPads, and Mac OS X laptops, yet the underlying devices are different. Similarly, Linux runs on laptops and Android smartphones. The proposed tool in this chapter distinguishes the OS of the device, as well as the device itself.

2.1.1 Device Identification Background

There are two ways to infer the OS. Active fingerprinting explicitly sends messages in the device in question and determines the OS type based on how it responds (it's "fingerprint"). Probes consist of carefully constructed TCP/IP packets designed to identify the implementation differences between the OSs' network stacks [12, 13]. A list of such tools is presented in [84].

The disadvantages of active fingerprinting are that (a) it produces extra traffic into the network; (b) it may alert intruders that their presence has been detected; (c) it requires knowledge that a device is present e.g. it's IP address; (d) such probes can be blocked, e.g. by a firewall; (e) it may violate security policies.

Passive fingerprinting instead examines traffic to and from the OS via copies of captured packets taken from some vantage point in the network. By inspecting the traffic, one may infer what type of OS is running on the host. The most noteworthy tool is p0f [8]. p0f uses a fingerprint database and monitors TCP/IP traffic. For instance, in the client-originating SYN packet and the first SYN+ACK response from the server, it pays attention to factors such as the

ordering of the TCP options, the relation between maximum segment size and the window size, the progression of TCP timestamps, and the state of about a dozen possible implementation quirks. Another tool is Ettercap [5], which similarly works on the TCP/IP stack. However, these approaches have the following limitations: (a) they do not distinguish the OS from the device; (b) require constant updating as the TCP/IP stacks in the OS evolve over time; (c) packets can vary for reasons besides differences in the OSs' implementation, e.g. the value of the TCP window scale option may depend on how much memory is installed in a machine. Another approach is to use DHCP packets to fingerprint the OS [60]. Similarly to the above this requires constant update of the fingerprinting database, and it will fail to identify Virtual Machines. The latter is very important as many machines are now virtualized and hosted in a single host. Finally, the HTTP *User-Agent* request header field is nowadays widely used for passive fingerprinting. However, it has the following limitations:

- A device may not use HTTP, or use HTTP in an encrypted form (e.g. HTTPS), thus the header is not parseable.
- It is relatively easy to modify the *User-Agent*. For instance there are several add-ons in modern browsers with such a functionality.
- The header may have insufficient information to distinguish the device, e.g. Apple products may just report "Safari/OS" for iPads, iPhones and Mac OS X laptops.
- Some applications encrypt these fields to avoid detection.
- Virtualized environments (e.g. VMs, or iOS emulators) can be observed for the same MAC address.

Nonetheless, in the wireless domain we have seen little effort on OS fingerprinting. For instance in [47] the authors used a combination of HTTP *User-Agents* and MAC addresses to determine the OS. However, the MAC addresses may not be present after a switch, hence such a technique can only be implemented one switching hop from the client.

2.1.2 Device Classification

In order to address the above issues, we designed a tool that applies associative rule mining [24] in the DHCP and HTTP headers. The benefits of our approach are the following:

- It identifies patterns among different fields in the DHCP header. Thus cross-validates the outcome of a single classifier with other classifiers. Based on that, it can handle cases where the OS is configured not to report some fields.

- It can be deployed at various levels in the network hierarchy. Therefore, even if the host MAC address is not visible, device and OS fingerprinting can still be performed.
- It can identify single or dual boot end hosts, by storing the MAC layer attributes for each IP lease.
- It performs the fingerprinting per IP lease. Hence, it can identify if there are multiple OSs in one end host, i.e. VM in bridge mode. By combining higher layer fields it can classify VMs in NAT mode.

We create a set of classification rules based on the arguments that are included in the options of the DHCP Request:

- *Host-Name*: We see if it contains a device specific string. Some cellphones set their host-name to a string that can identify the type of the device. For example, many iOS cellphones have names that follow the pattern of ‘*-iPhone’, where * usually corresponds to a string related to the user.
- *Vendor-Name*: We see if it contains an operating-system specific string. Some OSs include, in their vendor-name, a string that can uniquely identify the OS. For example, most versions of Microsoft Windows include the string ‘MSFT’[7].
- *Parameter-Request-List*: We see if it contains specific DHCP options. A DHCP request contains a list of parameters indicating the set of DHCP options that a client is interested in. Some of these options (as well as their combinations and ordering) are unique for each device, as they typically indicate its auto-configuration capabilities.
- *MAC-Address*: We see if the first three bytes of the device MAC address can be associated with a specific vendor, also called the Organization Unique Identifier (OUI). Using the IANA Ethernet assignments [4], we determine the vendor of the interface and then we identify if that vendor can be directly mapped to a specific type of device.

We use association rules to determine regularities between certain of the above values. For example, an association rule can be expressed as follows: given a host-name containing the string ‘BlackBerry’, what is the probability that the vendor-name will contain the string ‘BlackBerry’ (2-itemset). We select only few of all possible association rules by using breadth-first search, and prune those that are infrequent (low support) or have low confidence (similar to *a-priori* algorithm [24]). To quantify the confidence of the rules, we used standard data mining metrics: Support $supp(X)$ is defined as the portion of all devices that satisfy the rule x . Confidence $conf(X \Rightarrow Y)$ of an association rule $X \Rightarrow Y$ is defined as $supp(X \cap Y)/supp(X)$, where $supp(X \cap Y)$ is the support of rule $X \wedge Y$, namely, the portion of all devices that satisfy both rule X and Y . The rules that have high confidence in at least one direction ($conf(X \Rightarrow Y)$ and $conf(Y \Rightarrow X)$), and are not contradictory, are broken into their corresponding itemsets X and Y . Those rules are then used for potential classification.

Table 2.1: Dataset Properties

Trace Type	Corporate	Educational
Dates (2012)	Feb 29-Mar 25	Jan 15-Feb 15
Client MAC Addresses	2980	8726
Client IP Addresses	3435	1968
Wireless Subnets	8 * /23	/21
Leases	1h or 12h	15 min
DHCP Packets	2.16M	3.48M
TCP/UDP Bytes	2.5TB	4.9TB

2.1.3 Trace Collection

In order to construct the association rules, we use DHCP header log files and summary TCP/IP log files. We capture them with a sniffer located at a vantage point of two networks. We use a university vantage point to collect data from the campus users, and an enterprise vantage point to collect data from the employees¹. Both sniffers had 3 interfaces and had packet filters set to capture only the wireless traffic. In the university network, we also added extra filters to capture a subset of the network due to high volume of traffic. The first interface receives mirrored Web traffic from the switches located at the last-hop gateway of the wireless networks. We captured TCP and UDP packets using Bro 2.0 [3] to create summary log files for TCP and UDP flows, which enables us to determine the time periods in which devices are active. The second interface received the data that is sent from the DHCP server to the clients. In the enterprise network, DHCP Relays were interleaved. We monitored all DHCP interactions to identify the leases, i.e. the time duration an IP is associated with a device. The last interface was used for remote login and management. Table 2.1 summarizes some of the captured trace properties.

2.1.4 Association Rule Development

As we have also stated, our methodology is variation of the *a-prior* algorithm [24] that is based on association rule mining. For example, [host-name contains Android] \Rightarrow [Parameter-Request-List contains '1 121 33 3 6 28 51 58 59'] happens with confidence 100%. The reverse direction [Parameter-Request-List contains '1 121 33 3 6 28 51 58 59'] \Rightarrow [host-name contains Android] happens with confidence 82.35%, and the remaining 17.63% are related to a device that neither has 'Android' in the host name (e.g., when the user has modified the default host-name) nor any other name from another device type. Now a host-name that contains 'Android' or a Parameter-Request-List that contains '1 121 33 3 6 28 51 58 59', can be used to classify Android devices. In other words, we assume no ground-truth but quantify every rule. Table 2.2 summarizes some common association rules produced with the DHCP classification process in

¹Both sniffers had multi-core Xeon processors, >16GB of RAM and libpcap 1.2.1

the corporate network.

However, there are some unique rules that contain useful information. For example, we have observed that the *Parameter-Request-List* may be slightly different across the vendors of each 'Android' device, but the same for each device manufactured by each vendor. Hence for some of these rules, we apply a Bayesian Classifier to identify the device. We classify into the following categories: Windows, Linux, Mac OS X, Other Laptop, Android, Symbian, Blackberry, iOS, Windows Mobile, Other Mobile, Cisco VoIP, and Uncategorized. Table 2.3 shows the resulting classification from our trace. We use the first identified device, as some of the laptops may run multiple OSs (e.g., multiple boot clients or those running virtual machines).

We repeat the above process, starting with the relaxed definitive-association rules that have the highest confidence, until there is no rule that classifies a device with high enough confidence. For example, [vendor-name contains 'MSFT'] \Rightarrow [parameter-list = '1 15 3 6 44 46 47 31 33 249 43'] happens with confidence 99.18%. The remaining 0.72% is due to DHCP requests that do not have any options in the parameter list, i.e., there are no other rules with the same two fields as input that can further classify them. In the reverse direction [parameter-list = '1 15 3 6 44 46 47 31 33 249 4'] \Rightarrow [vendor-name contains 'MSFT'] happens with 93%, and the remaining 7% happens due to an empty vendor-name, i.e., again there are no other rules with the same two fields as input that can further classify them. Thus any device with either the above vendor or parameter-list is classified as a Microsoft Windows laptop.

Table 2.2: Common Association Rules for Device Classification

		Association Rule (<i>Rule1</i> \Rightarrow <i>Rule2</i>)		
Device	OS	Rule 1	Rule 2	Confidence
Laptop	Windows	Vendor contains 'MSFT'	List: 1 15 3 6 44 46 47 31 33 249 43	99.18%
		List: 1 15 3 6 44 46 47 31 33 249 43	Vendor contains 'MSFT'	93.00%
	Mac OS X	Host contains 'Macbook'	List: 1 3 6 15 119 252 44 46 47	100.00%
		Host contains 'Macbook'	MAC-Vendor = Apple	100.00%
	Linux	Host contains 'Ubuntu'	List: 1 3 6 15 119 95 252 44 46 47	MAC-Vendor = Apple
Smartphone	iOS	Host contains 'iPhone iPad iPod'	List:1 3 6 15 119 252	100.00%
		Host contains 'iPhone iPad iPod'	MAC-Vendor = Apple	100.00%
		List: 1 3 6 15 119 252	MAC-Vendor = Apple	100.00%
	Android	Host contains 'Android'	Vendor contains 'dhepcd'	100.00%
		Host contains 'Android'	List: 1 121 33 3 6 28 51 58 59	100.00%
		List: 1 121 33 3 6 28 51 58 59	Vendor contains 'dhepcd'	100.00%
		List: 1 121 33 3 6 28 51 58 59	Host contains 'Android'	82.35%
		Vendor contains 'dhepcd'	Host contains 'Android'	60.87%
	BlackBerry	Host contains 'BlackBerry'	Vendor contains 'BlackBerry'	100.00%
		Host contains 'BlackBerry'	List: 1 3 6 15	100.00%
		List: 1 3 6 15	Vendor contains 'BlackBerry'	100.00%
		Host contains 'BlackBerry'	MAC-Vendor = RIM	100.00%
Other	Cisco VoIP	Vendor contains 'Cisco Wireless Phone'	List: 1 3 6 12 15 28 42 66 149 150	100.00%
		List: 1 3 6 12 15 28 42 66 149 150	Vendor contains 'Cisco Wireless Phone'	100.00%

Table 2.3: Distribution of Devices in the Trace

		Corporate		Educational	
Device	OS	#	%	#	%
Laptop	All	2176	73.02	3970	45.50
	Windows	1787	59.97	2819	32.31
	Mac OS X	385	12.92	1131	12.96
	Linux	4	0.13	20	0.23
Smartphone	All	735	23.66	4489	51.44
	iOS	577	19.36	3069	35.17
	Android	126	4.24	1334	15.29
	BlackBerry	31	1.04	84	0.96
	Win Mobile	1	0.03	2	0.02
Other	All	69	2.32	267	3.06
	Cisco VoIP	9	0.32	-	-
	Unidentified	60	2.01	267	3.06
All		2980	100	8726	100

Table 2.3 shows the distribution of the devices in the two networks. We observe that in the corporate network, we were able to identify almost 98% of the devices, and in the educational network, we are able to identify 97% of the devices. Moreover, we also observe a unique difference between those two networks. In the corporate network, the number of laptops is three times higher than smartphones. But in the educational network, the number of smartphones is almost equal to laptops. While students in the universities are more mobile, it clearly shows a trend towards smartphone devices.

Finally, the proposed OS fingerprinting algorithm could be extended to perform a finer grain classification. For instance, from the OUI we could derive the vendor of each of the Android devices (Corporate Network: 22 HTC, 19 Samsung, 13 Motorola), or by combining the UA we could identify the iPad tablets. Nonetheless, due to the low volume of these subcategories, we chose to perform the traffic analysis into the following OS categories: smartphones (iOS, Android and Blackberry) and laptops (Windows and Mac OS X). In the following sections we are going to focus on the differences and similarities in terms of the behavior of these devices.

2.2 DHCP Lease Time Allocation in the Smartphone Era

The DHCP protocol [38] allows the devices to connect to a network without a client specific configuration. However, administrators have to take into account the available address space, the device behavior, and other security policies when setting the lease policies in the DHCP servers. Moreover, the onslaught of smartphone devices has led to more transient behaviors and in some networks administrators have reduced the lease times in order to keep from having to add a large amount of additional IP addresses to the pools. For instance, in several occasions a smartphone associates with a wireless network, and therefore consumes an IP address, may the user may not use the phone to perform anything productive. Yet, it still consumes an IP address. On the other hand, short leases may result in increased broadcast traffic and unnecessary activation of wireless interfaces by power limited devices². To further understand the impact of smartphone devices on DHCP lease times, we analyze the two one-month long packet traces that have been presented in the previous section. We make the following contributions:

- We show that DHCP message exchanges vary both across device types (e.g., laptops, smartphones) and across operating systems (e.g., iOS, Android, Windows, Mac OS X, Linux), with each device type contributing a different amount of DHCP related traffic and having a varying effect on the network address utilization.
- We propose a new DHCP leasing strategy that does not require any protocol changes, and which takes advantage of the varying usage patterns per device type. Using simulation results, driven by our traces, we show that the new strategy, compared to current approaches, improves the network address utilization sixfold without considerably increasing DHCP overhead.

We summarize some of the important details of the RFC [38] in the appendix A.

2.2.1 Lease Time Analysis

Using the DHCP-based device classification of the previous chapter, we analyze the DHCP request messages. Table 2.4 shows the acknowledged DHCP request messages³ for each device type, as a percentage of the total requests, as well as the absolute mean and median values. We observe distinct behavioral differences between the corporate and educational network. In the corporate network, smartphones, especially iOS devices, generate considerably more DHCP

²A potential solution to the above problem would be to switch to Private Address spaces [16] space for IPv4 and PAT out to reach the internet. If this was done a much larger IP space and longer lease times would have been possible. Many universities avoid this practice due to the Digital Millennium Copyright Act (DMCA) notifications that the university receives and must comply with. If PAT addresses were used, the administrators would need to log all entries in the NAT translation table and store them for a long time, and then would need to compare those entries against all data provided in the DMCA notifications.

³We use the acknowledged requests so that we do not account for messages generated by DHCP relays.

Table 2.4: DHCP Requests

Type	Corporate			Educational		
	(%)	Mean	Median	(%)	Mean	Median
iOS	51.6	251	140	35.74	158	38
Android	5.88	123	58	11.44	117	37
BlackBerry	2.68	200	48	0.88	135	42
Windows	31.16	51	24	39.6	190	45
Mac OS	7.08	52	48	12.38	148	38
Other	2.2	-	-	0.4	-	-

Table 2.5: Relative (%) of DHCP Request Types

Corporate			
Type	Select	Init-Reboot	Renew
iOS	13.19	85.62	1.19
Android	72.40	17.52	10.09
BlackBerry	94.05	0.00	5.95
Windows	33.53	23.99	42.48
Mac OS X	20.33	56.18	22.49
Educational			
Type	Select	Init-Reboot	Renew
iOS	13.06	57.4	29.54
Android	28.46	10.78	60.76
BlackBerry	35.53	0	64.47
Windows	3.99	10.39	85.62
Mac OS X	4.91	8.79	86.3

requests on average as compared to laptops. In contrast, in the educational network all devices generate roughly the same number of requests. Figure 2.1(a), which shows the cumulative distribution of requests per device type, illustrates this more clearly. This difference between the corporate and educational network is due to the smaller lease time of the educational network, forcing all devices to generate frequent lease renewal requests, as shown by the larger number of requests per device in that network.

To better understand the differences, we present the distribution of DHCP request message types in Table 2.5. In the educational network, with the exception of the iOS devices, the majority of the DHCP requests are renewals. In contrast, in the corporate network a considerably smaller percentage of the requests are renewals. Given the small number of renewal requests in the corporate network, other types of requests become more prominent, revealing a number of distinctions between device types. For example, iOS devices, and to a lesser extent Mac OS X devices, generate a large proportion of init-reboot requests. In contrast, Android and BlackBerry devices generate mainly select requests, meaning once they acquire a new address, they rarely contact the DHCP server again.

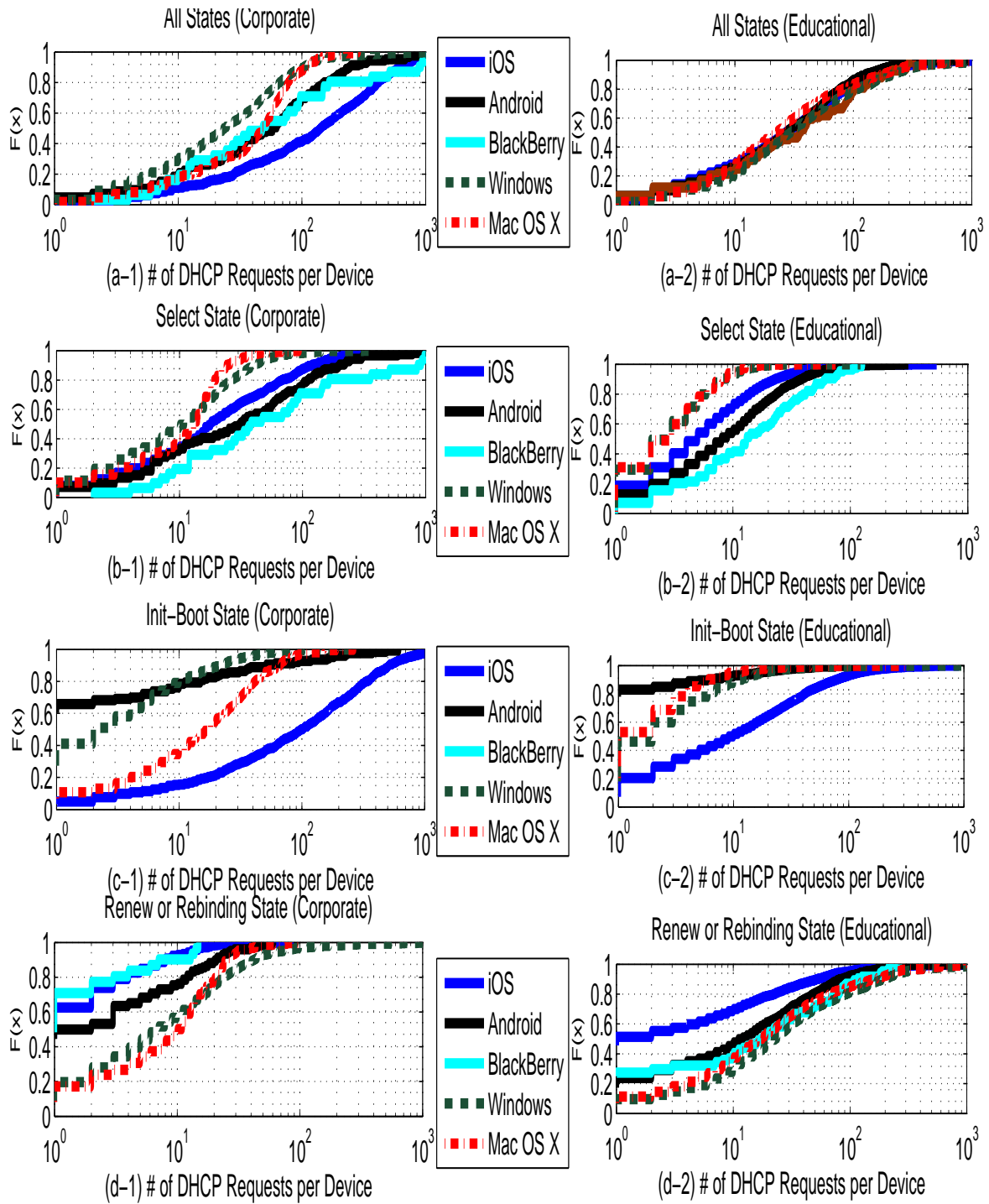


Figure 2.1: CDF of the number DHCP *Request* messages in the Corporate and Educational network

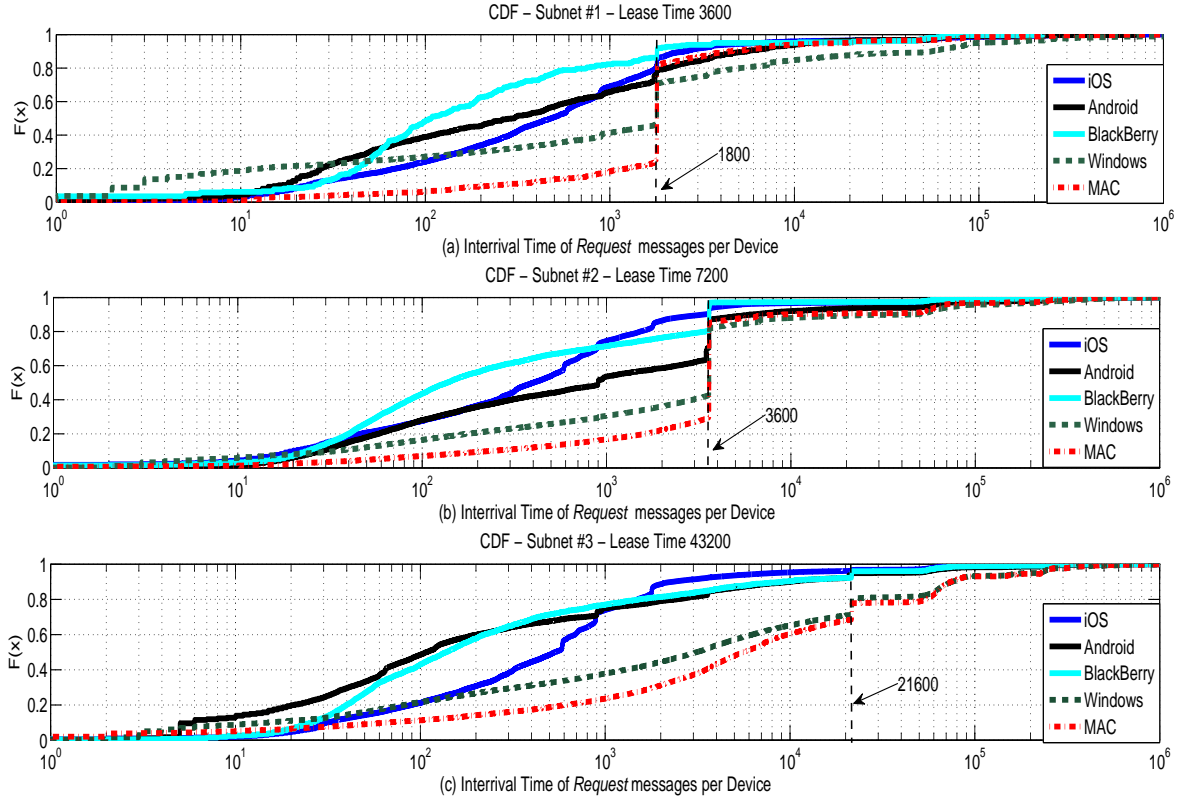


Figure 2.2: CDF of the interarrival time of *Request* messages with different *Lease Time* setting

This difference between Apple and other devices is attributed to the implementation of DNaV4 [22] in iOS and Mac OS X [28, 86]. DNaV4 optimizes the re-attachment to a previously connected network by attempting to reuse a previous but still valid configuration, by reducing the number of DHCP exchange messages and by using unicast ARP requests⁴.

Finally, in Figure 2.2, we plot the interarrival time of the DHCP request messages. The first graph corresponds to the educational network, and the second two to the corporate network (with leases of one hour and twelve hours respectively). We also indicate with a dotted vertical line the time corresponding to half of the lease time. This is the time at which a DHCP client requests a lease extension. In the educational network, we observe that the majority of the request messages are generated at half of the lease time, indicating that they are renewal requests. In the corporate network, where the lease times are larger, the devices generate far fewer requests for renewing an IP address.

⁴In our trace we observed unicast ARP requests associated with DHCP init-reboot requests coming from Apple devices.

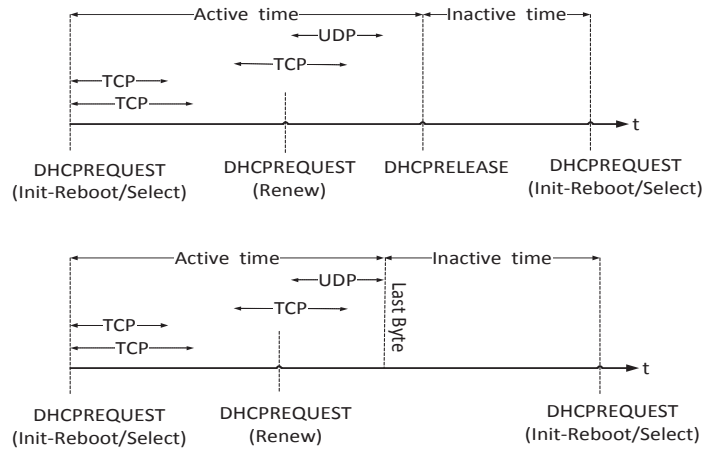


Figure 2.3: Calculating Active and Inactive times.

Network Access Patterns

Proper setting of DHCP lease times depends on the amount of time devices stay active in the network, i.e., the amount of time they send or receive data⁵. For a particular host, as identified by its MAC address, we define the following:

- *Active Time*: The time period, starting at the initial DHCP lease offer, up to the last time that any packet was generated, before the next lease offer.
- *Inactive Time*: The time period between the end of an active period and the beginning of the next active period.

Active time starts when a device receives a DHCP acknowledgement message as a response to a DHCP request message. From the request messages, we exclude those generated when the client is either in the renewing or the rebinding state, as their purpose is to update the lease duration. However, we include the ones from the init-reboot state, where the objective is to reconfigure the leases. An illustration of active and inactive times is depicted in Figure 2.3.

Figure 2.4 shows active and inactive times for the different types of devices. We observe that smartphone active times are much smaller compared to laptop active times. We also see that active times for iOS devices are smaller than the active times of other smartphones. This happens due to a combination of reasons related to: *i*) the way users use laptops and smartphones, and *ii*) the different policies related to energy management between laptops and smartphones.

⁵Note that the active time does not depend on the configuration of lease times.

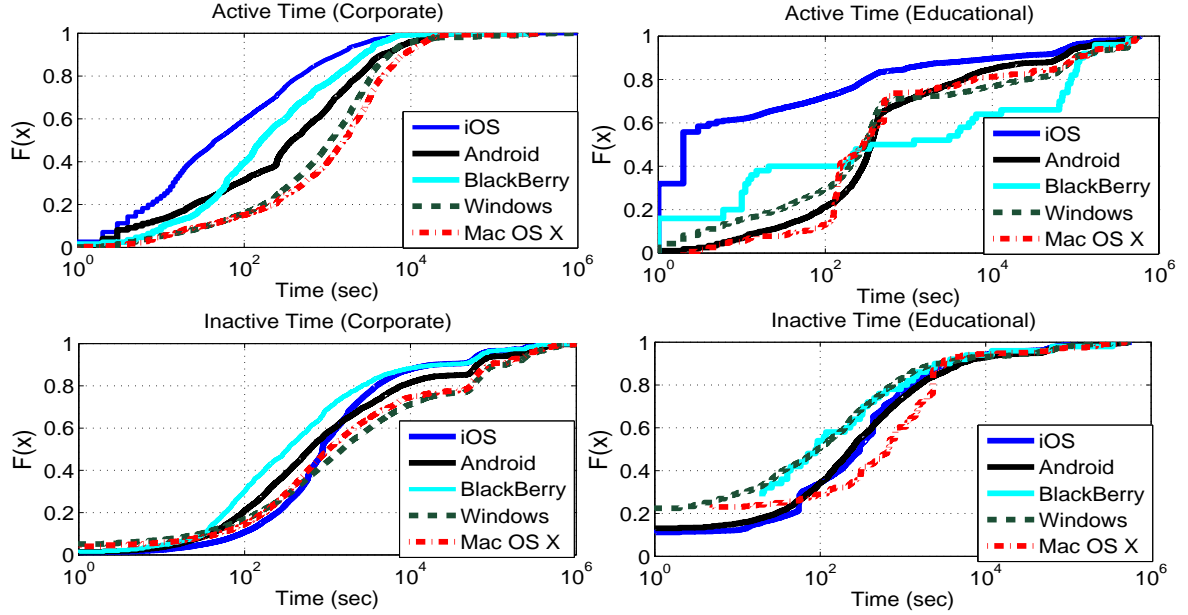


Figure 2.4: CDFs of Active and Inactive durations.

2.2.2 DHCP Lease Policies

Ideally, a DHCP lease allocation policy should account for the differences in behavior of the various mobile devices. The goals of this policy should be to minimize the following, in decreasing priority:

- *Address space utilization*: The policy should use as little of available address space as possible, in order to support the most concurrent users.
- *Broadcast traffic*: The policy should cause as few broadcasts as necessary, since broadcasts wake idle clients and consume power.
- *Server load*: The policy should minimize the load on the DHCP server, to reduce the related capex and opex expenses of running the server (including power).

We do this in two steps: first, we look at the behavior of the devices in isolation when varying the lease times over several orders of magnitude. Then, based on those behaviors, determine an approach that best meets the above goals. We wrote a simulator that, given a trace, reproduces the DHCP behavior and outputs the above metrics. We use the corporate trace in this Section ⁶.

⁶Note to reader: Our work on simulating the DHCP Leases on the educational network is undergoing.

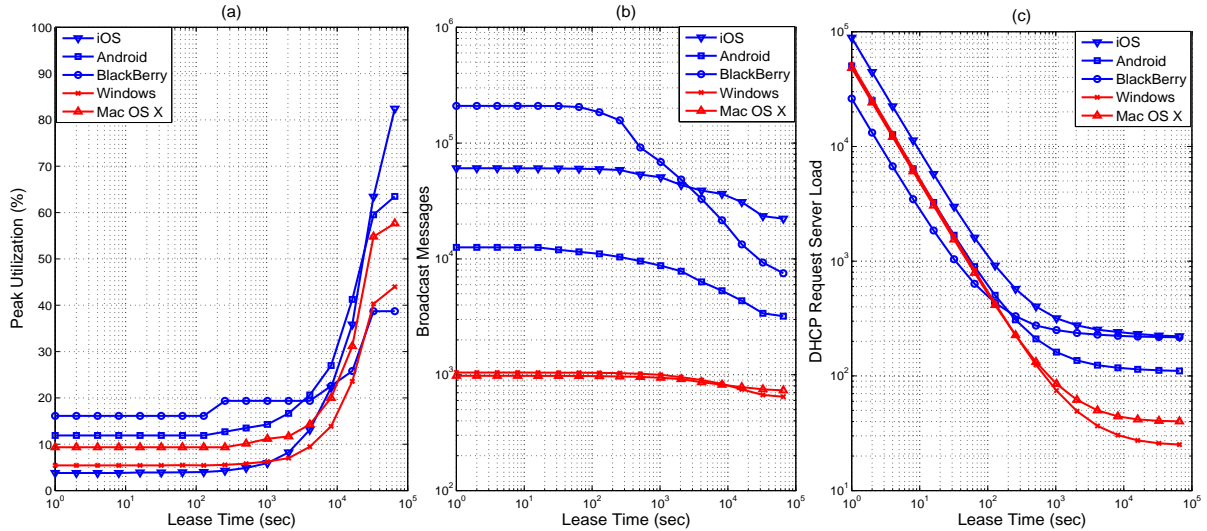


Figure 2.5: (a) Address space utilization (b) Broadcast messages (c) Server load, versus lease time, per device type

Figure 2.5 shows the results from our simulator. Broadly, one can see the tension between the goals in the three graphs. Shorter lease times utilize the address space most efficiently, but cause large amounts of broadcast traffic and high server load. Large lease times minimize broadcasts and server load, but at the expense of poor address space utilization.

Looking more closely, in Figure 2.5a we see that address space utilization stays relatively flat versus lease times for each device type up until some threshold, after which utilization starts to grow logarithmically. For iOS devices, the threshold is 10^3 seconds; for Androids and BlackBerrys, 2×10^3 seconds, and for laptops, 3×10^3 seconds. This is because of the short active periods illustrated in Figure 2.4. As lease times go up, many leases are wasted on devices that have transitioned into the inactive period.

In Figure 2.5b we observe that lease duration does not affect the number of broadcast messages generated by the laptops. Laptop users have long active times, therefore the majority of their DHCP messages are renews, which are unicast. In contrast, the number of broadcasts generated by smartphones is sensitive to the lease time. Shorter lease times incur larger numbers of broadcasts. This is because short lease time results in smartphones generating more request messages from the “selecting” state, as leases expire faster, and new leases require a full DHCP handshake, which incurs extra broadcast traffic. We also see that, under short lease times, BlackBerrys generate three times more requests than iOS devices. This is because the Apple devices use DNaV4. However, due to the volume of iOS devices in our traces, they generate the majority of broadcast messages.

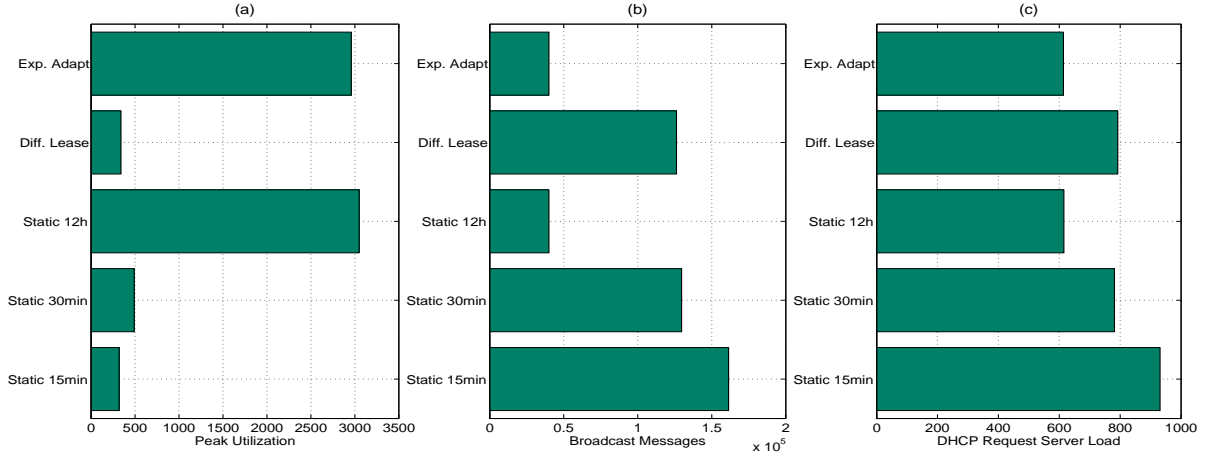


Figure 2.6: (a) Address space utilization (b) Broadcast messages (c) Server load, for various policies

In Figure 2.5c, we see that server load levels off at about 1,000 seconds for smartphones, but 10,000 seconds for laptops. This is due to the longer active times of the laptop users, as was shown in Figure 2.4.

Given these behaviors, we evaluated and compared the following DHCP lease policies:

- *Static policies*: Fixed lease times of 15 minutes, 30 minutes, and 12 hours, for all devices.
- *Exponential adaptation* [59], which allocates a short lease to a client once it arrives, and doubles the lease time every time the client renews its lease.
- *Differential lease*, which allocate different lease times based on the device type. We choose values based on our analysis of Figure 2.5: iOS devices get 1000 seconds, Android and BlackBerrys 2000 seconds, and Windows and Mac OS X 4000 seconds.

Figure 2.6 shows the results of our simulations. We see that our differential lease policy provides a good tradeoff between our goals. It is very efficient in address utilization, almost as much as the 15 minute lease policies, yet creates less broadcast traffic and server load. Exponential adaptation, on the other hand, uses a large amount of address space, but produces low amounts of broadcast traffic.

Different environments may have different priorities among the goals outlined above, depending on their address space size, distribution of clients (smartphones vs. laptops), etc. However, using static values requires a manual tuning process to determine the right tradeoff for the environment. Our differential lease policy, however, should work well across many environments, without administrative intervention.

2.3 Web study and caching implications

This section exposes some of those differences by comparing the Web browsing behavior of smartphones with a control group, namely laptops. We do this by capturing similar log files, as in the previous chapter, over a duration of 3 weeks. We use this trace to evaluate the Web browsing behavior of the devices, as HTTP makes up the bulk of the traffic. We examine standard metrics such as Web requests and responses, popularity, content type, and multimedia downloads. We focus mainly on those areas where the traffic of the two device categories differ. We study the differences of how video is delivered over HTTP in mobile devices. We also examine the effectiveness of caching in detail, both at the browser and at a proxy.

2.3.1 Trace Collection

In this chapter, we used the same captured architecture, but we have developed our own TCP reassembler and HTTP object reconstructor. The TCP reassembly is based on the TCP RFC [15]. The reason we have chosen this approach is because Libnids [94], another intrusion library, would drop all TCP connections that are not completely terminated, and Bro [3] 2.0 does not export the full HTTP header in the logs. Our overall approach is to first reconstruct the TCP streams from the component packets, and if the stream is HTTP, parse the stream for the relevant HTTP information (e.g., methods, URLs, request headers, etc.).

In any packet trace of traffic, a full connection sequence (i.e., SYN-FIN or SYN-RST) may not always be observed. We deal with this by timing out connections that do not generate any packets for over two minutes. If a connection is idle for more than that period, we consider it closed. In our trace, this happens in 11.3% of the connections.

2.3.2 Web Traffic

We first describe our methodology for extracting HTTP information from raw packet streams. We then analyze the data, digging deeper as appropriate, and then summarize our findings.

At the HTTP layer, we reconstruct the Web object by parsing the stream and calculate metrics such as the actual bytes transferred. This is not always available in the HTTP request or response headers for the following reasons. First, the *Content-Length* field is not always accurate, because many HTTP connections are aborted before the whole object gets completely downloaded [43]. Second, RFC 2616 [17] defines that responses with the *Chunked-Encoding* header must not include the *Content-Length*. We thus calculate bytes transferred based on the actual unique (non-duplicate) packets seen.

Table 2.6 presents a breakdown of the services in our trace. We observe that almost 90% of the packets are TCP, and the majority of those are HTTP. Moreover, across all devices, at

Table 2.6: TCP/IP Application Breakdown (%)

Response	iOS	Android	RIM	Windows	Mac OS
HTTP	76.17	85.16	71.38	83.99	77.13
HTTPS	6.39	9.31	28.57	10.24	17.51
Email	1.88	2.26	0	0.28	1.65
SSH	0	0	0	0.04	0.09
Other	14.85	3.19	0.05	5.48	3.62

Table 2.7: HTTP Request Method (%)

Response	iOS	Android	RIM	Windows	Mac OS
GET	92.75	96.21	95.30	95.47	95.55
POST	7.10	3.77	4.77	4.44	4.37
HEAD	0.14	0.01	0.23	0.09	0.07
PUT	0.01	0.01	0.00	0.00	0.02

least 71% of the TCP traffic is HTTP. Thus the focus of our thesis is on Web behavior.

Requests and Responses

Table 2.7 presents the breakdown of HTTP request methods in the trace. Methods that fall below the $> 0.01\%$ threshold (e.g., OPTION, DELETE, etc.) are omitted. The distribution of request methods is broadly similar across devices, with the only exception being the iOS devices. iOS devices generate a larger percentage of POST requests. POST requests are used when the user sends information to the server⁷.

Figure 2.7 provides an overview of the response codes per device type. We observe that Windows and Mac OS X laptops generate a higher fraction of 304 (Not Modified) responses compared to the smartphones. This code is typically returned in response to a client generating a conditional GET request, which validates a client’s cached copy of a document. The distribution of the conditional requests are summarized in Table 2.8. The *If-Modified-Since* is used to validate whether the copy in the browser cache is up to date. A client that has one or more entities previously obtained from the server, verifies that none of these is current by including a list of associated tags in the *If-None-Match* header field. In either conditional requests, if a copy is valid and up-to-date in the client’s cache, the server will respond with a 304 response and no actual data will be transferred. Otherwise, the full object is downloaded and a 200 response is generated. Thus, 304 responses are an indication of the browser caching activity at each of the devices. Given the lack of 304 responses in the smartphones, this suggests that browser caches on the smartphones are not as effective as those on the laptops. We examine this in more detail in Section 2.3.3.

⁷We use the methodology in [96] to identify the applications that make use of the POST requests. From the top 10 applications that generated 80% of the POST messages, 6 of them come pre-installed in iOS devices (Apple Maps, Stocks, Weather, iTunes and Browser).

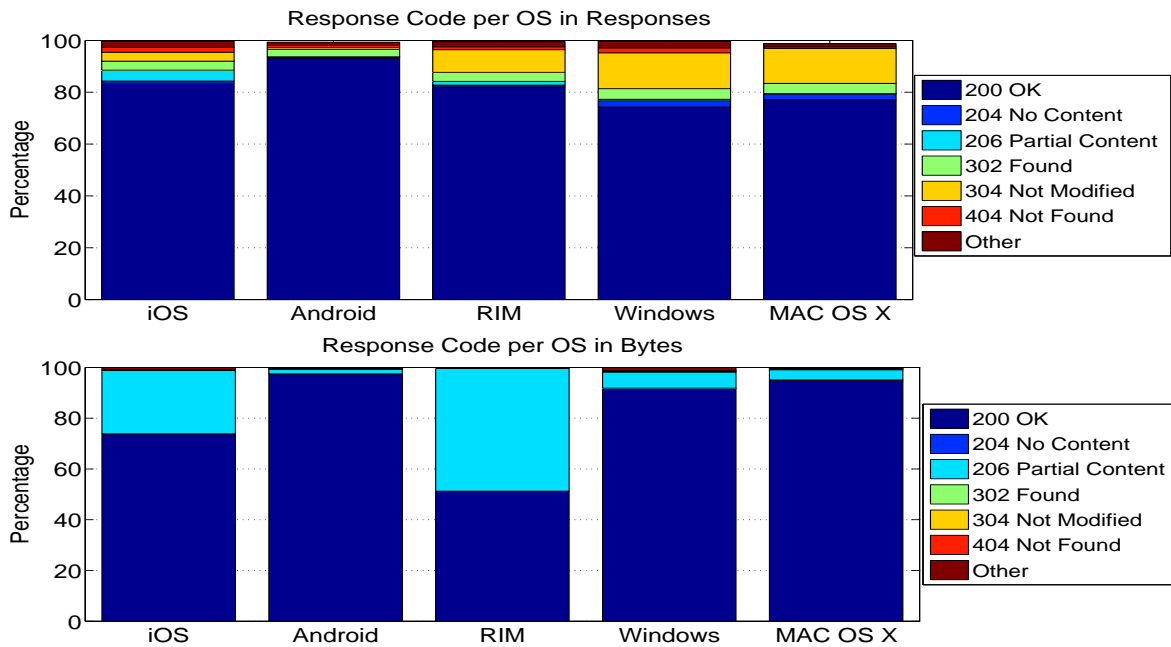


Figure 2.7: Response Code per Operating System

Table 2.8: Conditional GET Requests (%)

Response	iOS	Android	RIM	Windows	Mac OS
If-Modified	3.97	0.35	0.37	16.07	16.77
If-None-Match	1.62	0.24	0.05	8.53	8.49
If-Match	0	0	0	0.03	0
If-Range	0	0.07	0	0.08	0

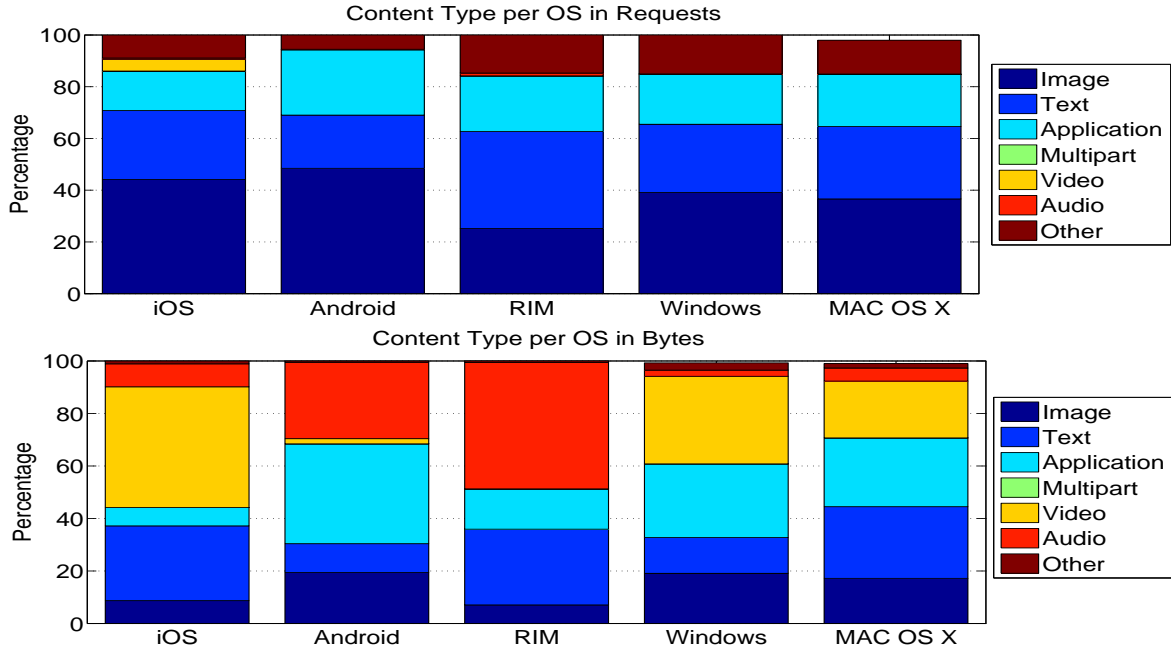


Figure 2.8: Content Types per Operating System

The second observation from figure 2.7 is that a significant fraction of bytes downloaded by iOS and BlackBerries are returned using 206 (Partial Content) responses. This status code is used when the client requests an object to be delivered in smaller chunks. For instance, such responses occur in multimedia traffic for smartphone devices, because these objects tend to be large in size. Identifying each unique chunk requires processing the *Range-Request:* field, which indicates the chunk that has been requested.

We next examine what kinds of content are downloaded by smartphones and laptops. Figure 2.8 shows the relative percentages of the various content types requested in the trace, both in terms of requests and in bytes. Across all platforms, a minority of requests for multimedia content (video and audio) generate the majority of the bytes transferred. Nonetheless, among the iOS requests, a higher number of video requests is associated with video objects than in other device types. In addition, 55% of the iOS bytes downloaded are multimedia, which is higher than seen on the other devices. There is also a significant portion of BlackBerry traffic that is associated with Audio. Finally, the distribution of content types on Windows and MAC OS X laptops are broadly similar.

We next identify, for each content type, how many devices download that content type. We do this to determine if the differences observed above are due to outliers, or if it is a behavior broadly characteristic of each device type. Figure 2.9 shows the cumulative distribution of downloaded bytes for each content type on a per-client basis. The starting point of each CDF

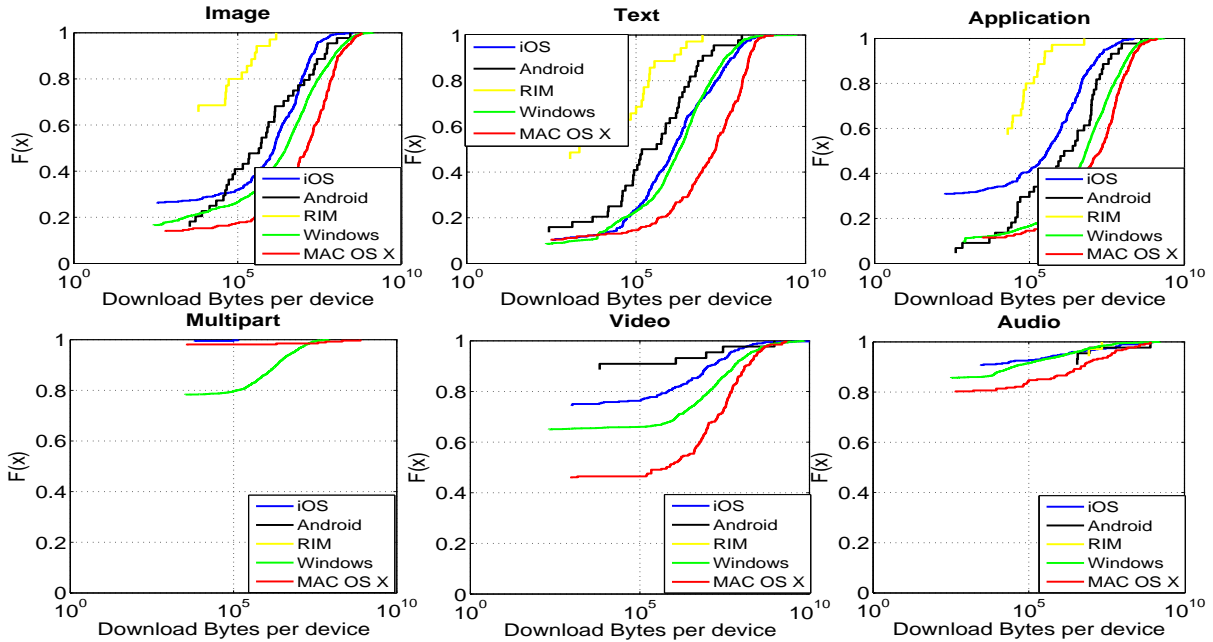


Figure 2.9: CDF of downloaded bytes per content type

indicates the relative percentage of the devices that *do not use* that content type. For example, there is a unique outlier in the BlackBerry trace for audio traffic, which requests 40% of the total HTTP traffic across *all* BlackBerries. Moreover, video traffic is more popular across laptops (Windows and Mac OS X) than it is for smartphones. For example, on average across Windows and Mac OS X laptops, 55% of them download at least one video object, whereas this happens only for fewer than 25% of the iOS devices. The median laptop device downloads almost 100 times more video bytes than an iOS device, and significantly more than the median Android device. Thus, iOS users access video content relatively more frequently than Android users.

Multimedia Analysis

Figures 2.7 and 2.8 showed, respectively, that there are a large fraction of 206 responses in our trace and that a large number of bytes are due to multimedia traffic. Here we examine the interaction of these two factors to further understand the role of multimedia content.

The stacked bars of Figure 2.10 breaks down the distribution of response codes across the devices for video and audio in terms of number of requests and bytes transferred. Only the 200, 204, 206, and 304 response codes are given since they are the only ones seen for the multimedia content. In addition, BlackBerry devices did not generate any video traffic. The 304 code was rarely returned in video requests generated by laptops. This suggests that there

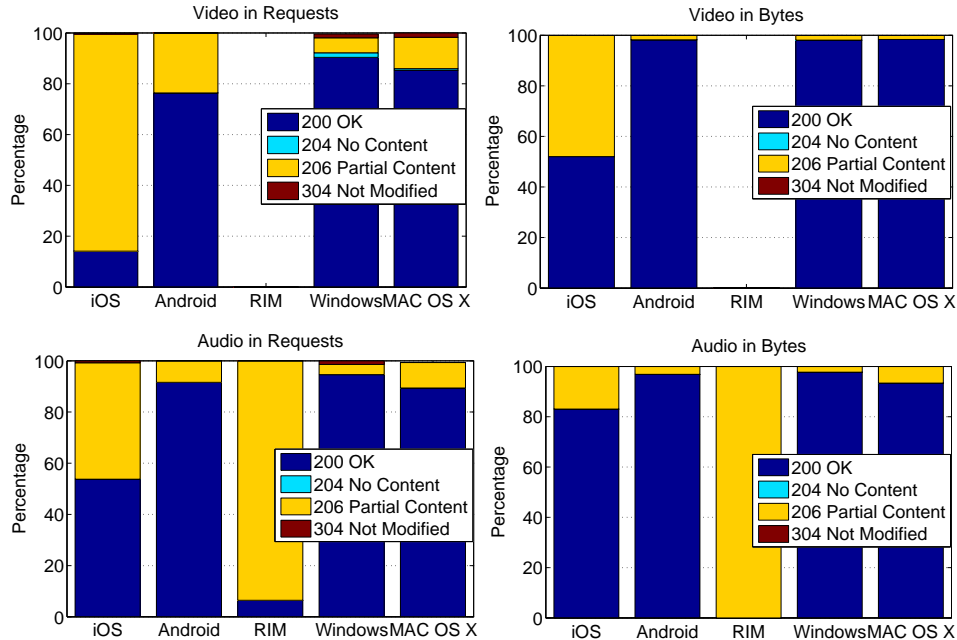


Figure 2.10: Response Codes for Multimedia Traffic across all types of devices

is little locality in video content, since the browser caches do not re-validate them using if-modified-since requests. Moreover, the majority of the responses in iOS and Androids are 206 for multimedia content. This is a unique behavior especially for iOS devices, since around 20% of the users download video traffic.

To further understand the role of 206 responses, we use and extend definitions for video from [40] to account for all the multimedia content. *Progressive Download (PD)* is multimedia content that is downloaded as a single file with the 200 OK status code. *PD with Ranges* is multimedia content that is delivered in chunks, using range requests and delivered with the 206 Partial Content status code. A third category is *HTTP Live Streaming (HLS)*, which dynamically adjusts the playback quality to match the available speed of network. Apple has proposed a standard of this in an IETF draft [80]. In HLS, objects are delivered with *PD with Ranges*, but with the difference that encoding type is MP2T (MPEG-2 Transport Stream). However, as we see in figure 2.11, only 13% of the video requests and 7% of the video bytes use HLS in iOS devices. That is because most of the iOS video content is delivered based on the MPEG-4 encoding. The remainder of the devices primarily use Flash video (which is not supported on iOS).

Consequently, we are interested in the unique properties of the multimedia traffic. Initially, we examine the User-Agents headers in iOS requests to determine the application. We observe that two User-Agents, namely *AppleCoreMedia* and *iTunes*, generate almost 98% of the video

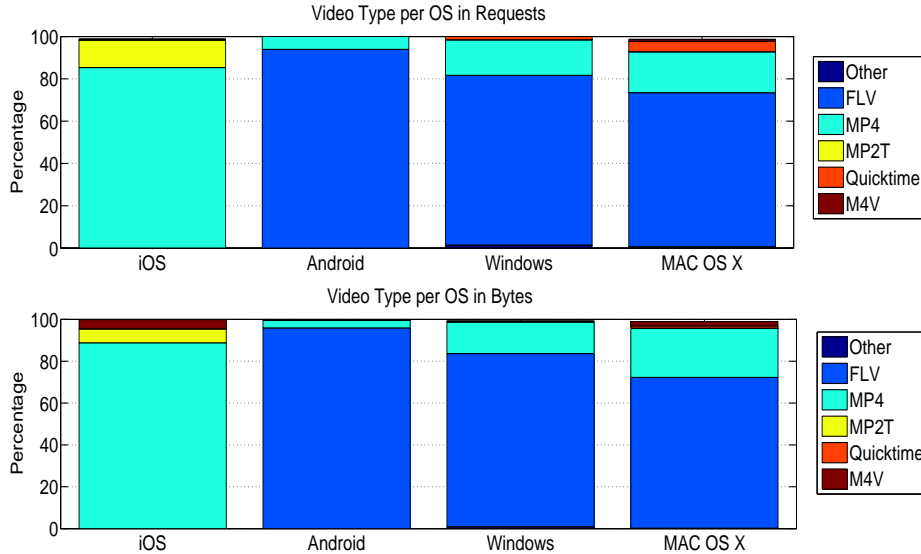


Figure 2.11: Video Content Type across all the devices

traffic. We also observe that there is a small portion of video traffic from MAC OS X that belongs to a similar User-Agent, namely *CoreMedia*. Figure 2.12 shows the breakdown of the application usage in iOS and MAC OS X for multimedia traffic. Note that the total video content downloaded is 23GB and 31 GB, for iOS and MAC OS X respectively.

The majority of the multimedia requests in iOS devices are generated by *AppleCoreMedia* and those requests contribute almost half of the bytes. While many developers may opt not to use it, the Core Media framework [2] is provided to iOS developers, as media layer for audio and video. To investigate the services that use this media component, we isolated their DNS names. We observe at least 82 unique services use it, and “googlevideo” service contributes most of the bytes (65%).

Figure 2.12 also indicates, that a small portion of *iTunes* requests contribute the other half of the bytes. Hence, the size of each *iTunes* object is larger than that of the objects related to the Core Media component. Finally, in laptops such as MAC OS X, most of the video traffic is generated by the Web browser, and only 5% of the requests are generated by *CoreMedia* and even fewer are related to *iTunes*.

We then turn our attention to the network traffic implications of these two multimedia components in iOS devices. We use the cumulative distribution function in figure 2.13 to show the object size per multimedia transfer. We observe a significant difference in terms of the object sizes. The median object handled by *AppleCoreMedia* is 100 times smaller than the median object for *iTunes*. Moreover, almost 95% of the *AppleCoreMedia* objects are < 1MB.

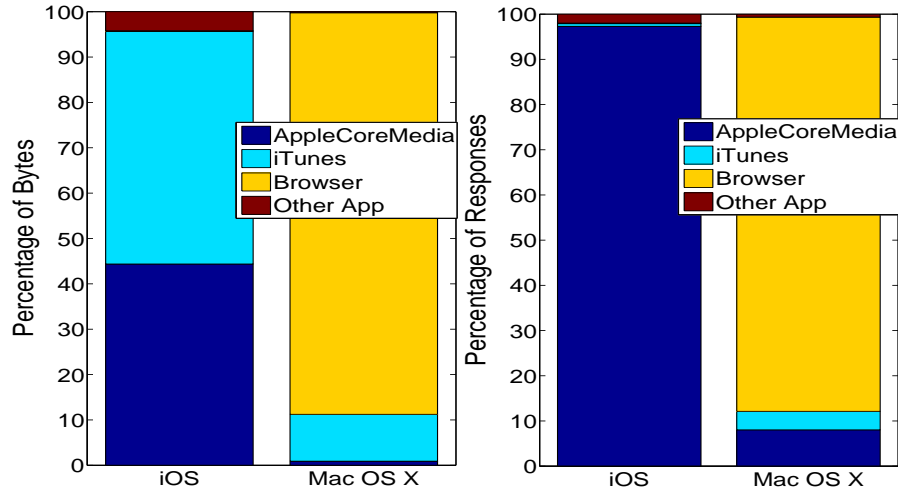


Figure 2.12: Applications that generate multimedia traffic in iOS vs MAC OS X (bytes,responses)

Finally, we see no relationship between 200 and 206 responses and object size. This suggests the iOS component will chunk the videos using the 206 response, but only if they exceed a specific size.

Aborted Transfers

In this subsection, we investigate the behavior of aborted transfers in relation to the size of the object, grouping the data in terms of laptops vs smartphones.

To examine this issue, we compare the document size of a Web object with the transfer size that is actually observed for it when it is downloaded. We normalize based on the average transfer size M for a document of size N , based on all transfers for documents of size N . We call this the *download ratio*. The results are shown in Figure 2.14. Note both axes are in log scale. If all downloads were completed, one would see a diagonal $Y = X$, which we include here for convenience. We see that, on average, the full document is not downloaded, and that for sizes greater than 4 MB, the behavior of the iOS devices diverges from the other devices. iOS devices are significantly less likely to download full objects than other devices. The second subplot (2.14 (b)) looks at the download ratio for iOS devices broken down by content type. We see here that videos are much less likely to be fully downloaded than other content types. In the third subplot, we look exclusively at video downloads on iOS and distinguish by the application generating the request. We see that streaming content, requested by the *AppleCoreMedia* component, is frequently aborted. While there are only a few *iTunes* transactions, most of them are downloaded fully.

These results indicate that aborted downloads happen mainly for streaming video requests.

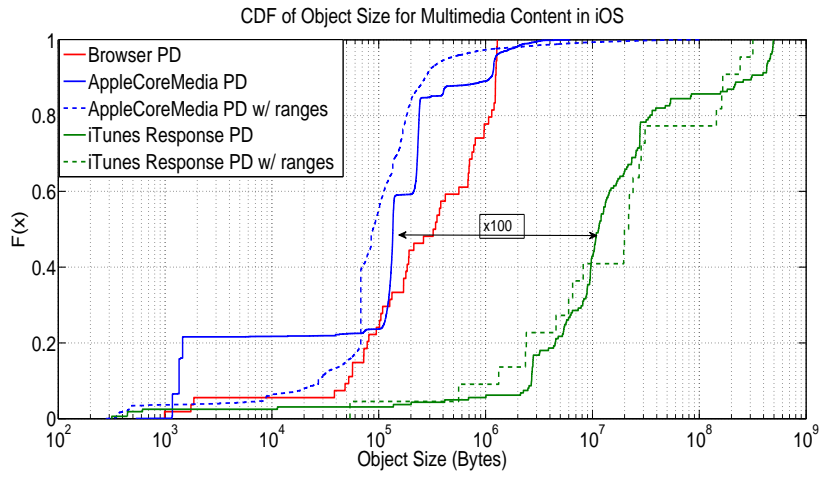


Figure 2.13: Cumulative Distribution of the iOS multimedia traffic per object size

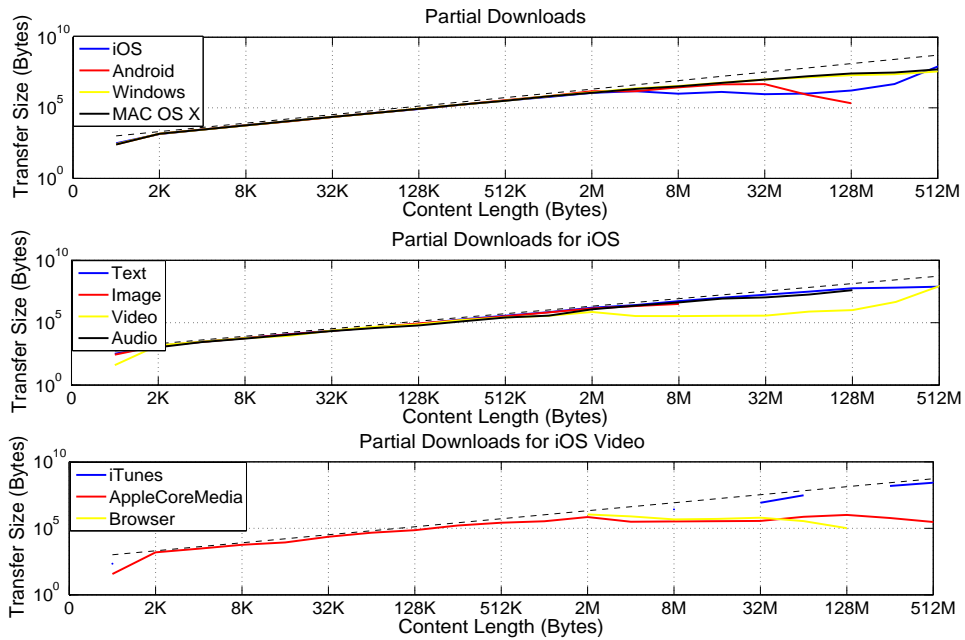


Figure 2.14: HTTP downloads per device type, iOS content type, and iOS application

Table 2.9: Zipf Law fit for Request Popularity

	iOS	Android	Windows	MAC OS
$y =$	$\frac{8136.3}{x^{0.793}}$	$\frac{48081.8}{x^{1.097}}$	$\frac{239923.1}{x^{0.93}}$	$\frac{563358.7}{x^{0.916}}$
R^2	0.9865	0.9873	0.9906	0.9910

This also suggests that proxy or browser caching policies [89] that would cache the object only if it gets fully downloaded would probably fail to provide savings for video traffic in iOS devices. We examine this issue in more detail in Sections 2.3.3 and 2.3.4.

Popularity

Popularity is frequently calculated from Web traces as a means of illustrating document access frequency and as motivation for caching Web objects.

We first investigate the object popularity, and perform a statistical fit using Matlab with several distributions. We find that a Zipf-Law fits the head of the request distribution, and show the values for fitted distributions in the table 2.9. This is an artifact of the very long tail of requests for objects that have only one reference. This indicates that caching the first few objects would provide most of the savings.

The second popularity metric that we investigate is the top Web sites visited by clients. We present the top 20 sites in tables 2.10 and 2.11, in bytes and requests respectively. There are two observations regarding the user behavior. First, we see that smartphones make relatively more references to the top 20 sites than laptops, suggesting a tighter pattern of reference. The second, is that in smartphones we see that the top 80% of the Web sites in terms of bytes are related to the top 20 applications. Those observations, as will be shown later on, are relevant to the savings from a proxy cache.

HTTP Results Summary

Here we briefly summarize our findings on Web traffic behavior for smartphones and laptops:

- Smartphone browsers are less likely than laptops to use HTTP caching features, as shown by fewer uses of the *If-Modified-Since* and *If-None-Match* headers.
- Multimedia (audio and video) is a large fraction of the bytes downloaded across all devices. Android and BlackBerry download more audio, iPhones more video.
- Most devices use Flash for video, except iPhones. Most video is downloaded using progressive downloading with range requests.
- Aborted transfers and partial downloads are common occurrences across devices, and are more likely for iPhone users. In particular, video is more likely to be aborted than other content types.
- Smartphone Web accesses are relatively more concentrated than laptops, as shown by the larger fraction of requests and bytes that are satisfied by the top 20 Web sites.

Table 2.10: Top 20 Web Sites (Requests)

Laptops	%	Smartphones	%
Search Engine	8.51	Online Journal 2	10.30
Social Networking 1	5.89	Search Engine	5.58
Advertising	3.45	Social Networking 1	4.73
Online Journal 1	2.44	Computer Vendor	3.97
CDN 1	1.82	Online Journal 1	3.52
Images Search	1.33	Tracking 2	3.32
News Site	1.12	Advertising	1.67
Analytics	1.11	Undefined IP	1.53
Antivirus Update	1.09	Feeds	1.49
Game	1.04	Video Site 3	1.29
Retail	0.98	Analytics	1.25
Video Site 1	0.89	CDN 1	1.16
Social Networking 2	0.88	Video Site 1	0.85
Tracking 1	0.85	News 3	0.83
Flash Service	0.83	News 4	0.80
News 4	0.81	Social Networking 2	0.79
Advertising 2	0.80	Retail	0.75
Computer Vendor	0.71	Advertising	0.71
Adds	0.66	Images Search	0.64
Online Journal 3	0.56	International	0.62
Total Top 20	35.73	Total Top 20	45.81

Table 2.11: Top 20 Web Sites (Bytes)

Laptops	%	Smartphones	%
Video Site 1	20.15	Computer Vendor	23.8
Computer Vendor	4.54	Video Site 4	16.34
Search Engine	3.99	Video Site 3	11.03
Antivirus Update	3.89	Video Site 2	4.34
Social Networking 1	2.07	Video Site 1	3.42
Online Radio 1	1.93	Search Engine	3.35
Video Site 2	1.86	Online Radio 1	3.09
Software Update 1	1.79	Online Journal 1	2.27
Software Update 2	1.67	Undefined IP	1.60
Images Search	0.87	TV Streaming	1.46
Advertising 2	0.86	Tracking 2	1.42
OS Download 1	0.79	News 1	1.41
University Media	0.78	File Hosting	1.23
Online Journal 2	0.61	Podcast 2	1.22
File Host	0.49	News 2	0.88
Undefined IP 2	0.47	Social Networking 1	0.82
Video Site 4	0.47	Broadcasting	0.73
News Site	0.43	CDN 1	0.6
OS Download 2	0.42	CDN 2	0.57
Retail	0.41	Online Radio	0.48
Total Top 20	47.82	Total Top 20	80.1

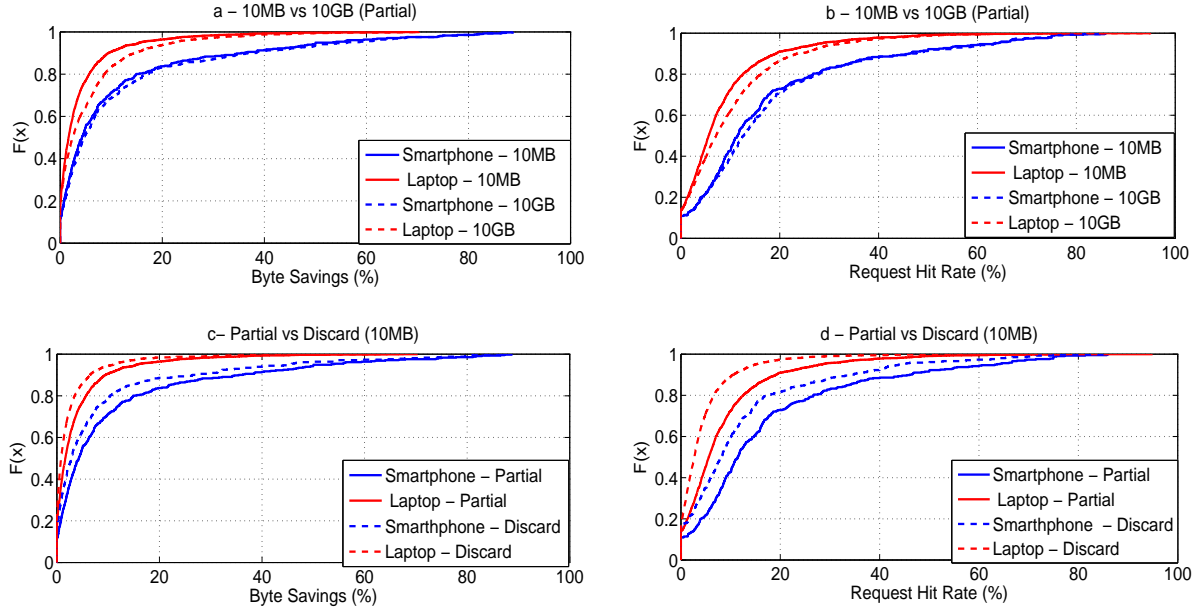


Figure 2.15: Comparing browser cache hit rates by varying (a-b) the size, (c-d) the storing policies.

Table 2.12: Median browser cache hit rates.

Size	Byte Savings		Request Hit Rate	
	Smart.	Laptop	Smart.	Laptop
10MB	5.02	2.20	12.45	6.71
10GB	6.06	3.39	13.81	8.83
Policy	Smart.	Laptop	Smart.	Laptop
Partial	5.02	2.20	12.45	6.71
Discard	3.88	1.33	9.65	3.49

2.3.3 Browser Cache Emulation

Virtually all browsers use local caches to improve the latency of requests that can hit in the cache and to reduce the network bandwidth consumed. In the mobile environment, this can also reduce energy consumption and improve battery life. In Section 2.3.2, we provided evidence that suggested that the browser caches on the smartphones were not as sophisticated as those on the laptops. In this Section, we explore this area further.

To examine the effectiveness of a local browser cache, we evaluate them by replaying the trace for each device through an additional simulated cache dedicated solely to that device. If the addition of such a cache does not provide any benefits, it suggests the the browser cache is performing well. If it does provide benefits, it suggests that the browser cache is either too small or not exploiting all the available caching features of HTTP. Our emulated browser cache follows

Table 2.13: Cacheable Bytes and Requests per OS

Location	Cacheability in	iOS	Android	RIM	Windows	Mac OS
Browser	Bytes	86.74	70.27	51.83	81.75	82.43
	Requests	70.25	68.85	55.87	63.43	65.51
Proxy	Bytes	85.20	41.10	48.42	54.74	67.21
	Requests	64.70	65.20	47.86	56.20	57.42

the cacheability requirements from RFCs 2616 [17] and 2965 [18]. We use an LRU replacement policy, both for simplicity and because it is the default in the well-known Squid [89] proxy cache.

Note that the *Cache-Control* header in an HTTP response is handled differently in a browser cache than in a proxy cache, as documents marked as *private* are considered cacheable. Table 2.13 shows the differences in cacheability across platforms, from the point of view of the browser cache and a proxy cache. We observe that the majority of the traffic in iOS, Windows and MAC OS X is considered cacheable. Interestingly, the percentage of content that is cacheable decreases significantly across most platforms when proxy caching rules are used, except in the case of iPhones.

Since smartphones experience ranged requests, aborted transfers, and partial downloads, we consider several policies in order to evaluate the performance of a a dedicated browser cache:

- *Discard*: Partially downloaded files are excluded from reuse. The cache simply discards all data that has been already downloaded. This is the default policy for Squid.
- *Partial Download*: The cache stores only the portion of the object that has already been requested by the user.
- *Conditional Download*: The entire object is only downloaded when X percentage of the object has already been requested by the user. This is available in Squid as a configurable option.
- *Full Download*: The cache continues downloading the entire object, even when the user aborts the transfer.

Since some users generate significantly more traffic than others, simply reporting an average benefit for browser caches does not sufficiently describe their behavior. Instead, we show the cumulative distribution of the benefits.

We begin with Figures 2.15 (a-b), which show the distribution of the improvement provided by an additional browser cache in terms of bytes and requests respectively. In these graphs, the partial download policy is used, and a comparison is made between a 10 MB cache and a 10 GB cache. We see several points from the graphs. First, the distribution of benefit varies widely

across the devices, based on their individual access behavior. Second, the browser cache consistently provides more benefit to the smartphones than to the laptops, adding to the evidence that the smartphone caches are not as effective as the laptop caches. Third, we see little difference between the 10 MB and 10 GB curves. This indicates the reason the smartphone caches do worse is not because they are too small, but instead because of other issues. Figures 2.15 (c-d) compare the partial and discard policies, again for bytes and requests respectively. We see the partial policy consistently outperforms the discard policy, sometimes by up to 50%. Thus a browser cache should not discard the object even if it is only partially downloaded.

Finally, we also compared a browser cache that supports range offsets with one that does not. According to the RFC [17] if the cache cannot process ranges, then it must not make any effort to cache range requests. We observe only a very slight improvement in performance for those caches that process ranges. Thus, if the implementation complexity to add the functionality that processes ranges is high, then it may not be worth caching range requests.

We summarize our conclusions as follows: *a)* laptop browser cache implementations are more sophisticated than smartphones; *b)* A small increase in the browser cache size is sufficient to capture most of the savings; *c)* A browser cache should be able to handle partially downloaded objects; *d)* A browser cache does not necessarily need to handle ranges, as the difference in savings is small; *e)* Average savings across all devices does not sufficiently describe the benefits, as traffic distributions vary widely.

2.3.4 Proxy Cache Emulation

RFC 2616 [17] defines several rules that a proxy cache must implement, and leaves several others open for different interpretations by the implementation. We thus explicitly spell out the required HTTP 1.1 compliant rules as well as the optional rules that are implemented in our emulator (as well as in popular HTTP proxy caches [89]) in the Appendix.

Given the predominance of Web traffic in our trace, it is useful to determine how effective a Web proxy cache would be in our environment, both in terms of bandwidth savings and in terms of request hits, which reduce response time. We use our cache emulation software described earlier in Section 2.3.3 with the following configuration: (a) all traffic is directed to a single proxy cache, rather than having multiple per-device caches, and (b) we follow the cacheability rules for proxy caching rather than for browser caching. We use the same caching policies described in Section 2.3.3.

One challenge in properly emulating a proxy cache's behavior is how to account for objects in the cache that are not fresh. Even if an object is not fresh, it could still be the same as the object on the origin server. Note that the time duration that an object is fresh can be determined by the *Expires* header field, or by the *max-age* and *s-max-age* sub-fields, of the

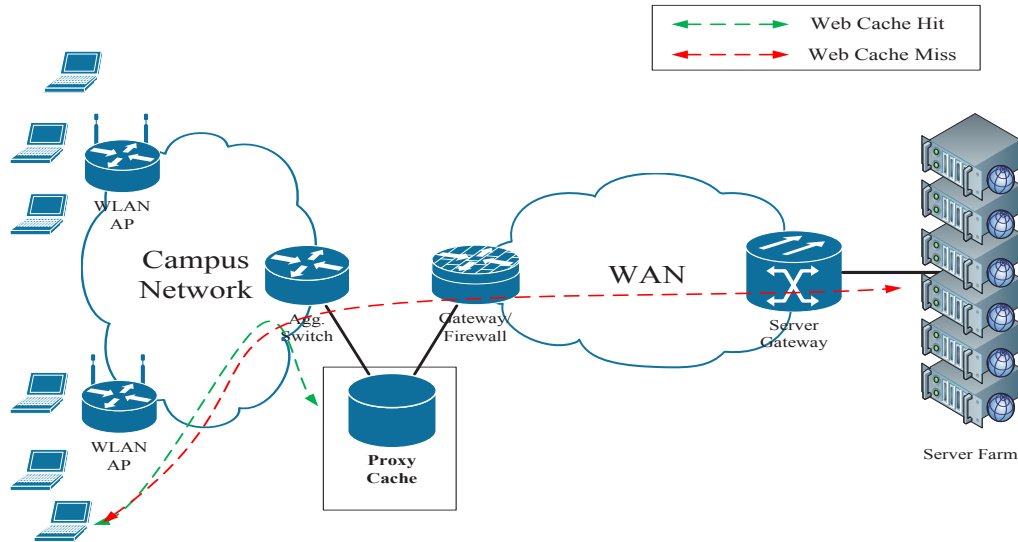


Figure 2.16: Campus Network with a Web Proxy Cache

Cache-Control header, and in accordance to the RFC2616 [17]. Thus when the proxy tries to retrieve it via a conditional GET request, the object could be returned either in full via a 200 response or simply re-validated via a 304 response. To quantify the impact of a proxy cache, both cases need to be considered. We thus take the following approach. In one configuration, *always revalidated*, a cached object is always assumed valid with the object at the server. In that case, the cache only revalidates the object and receives a 304 response. In the second configuration, *never revalidated*, the object is always assumed stale and the full object is retrieved via a 200 response. The first approach is the most optimistic case in terms of savings, while the second is the least optimistic one. In reality, of course, the actual behavior will lie between the two. To be conservative, we always evaluate using the second approach, unless otherwise stated.

Figure 2.17 shows the byte savings and request hit rates of laptops and smartphones, using the two assumptions, over a range of cache sizes. We note several observations. First, laptops tend to have better byte hit rates, but worse object hit rates than smartphones. This is a result of the wider variety of content that laptop users access, as shown in Table 2.11 (47.82% of the traffic in laptops is accumulated in the top 20 websites, and 80.1% in smartphone devices). Second, benefits achieve diminishing returns at around 100 GB, a relatively small size. Thus size, and consequently replacement policy, is not a significant issue in our environment. Finally, the validation assumption makes a large difference in the estimate of the byte savings. This is particularly true for laptops, where it can make as much of a difference as 50%. This happens

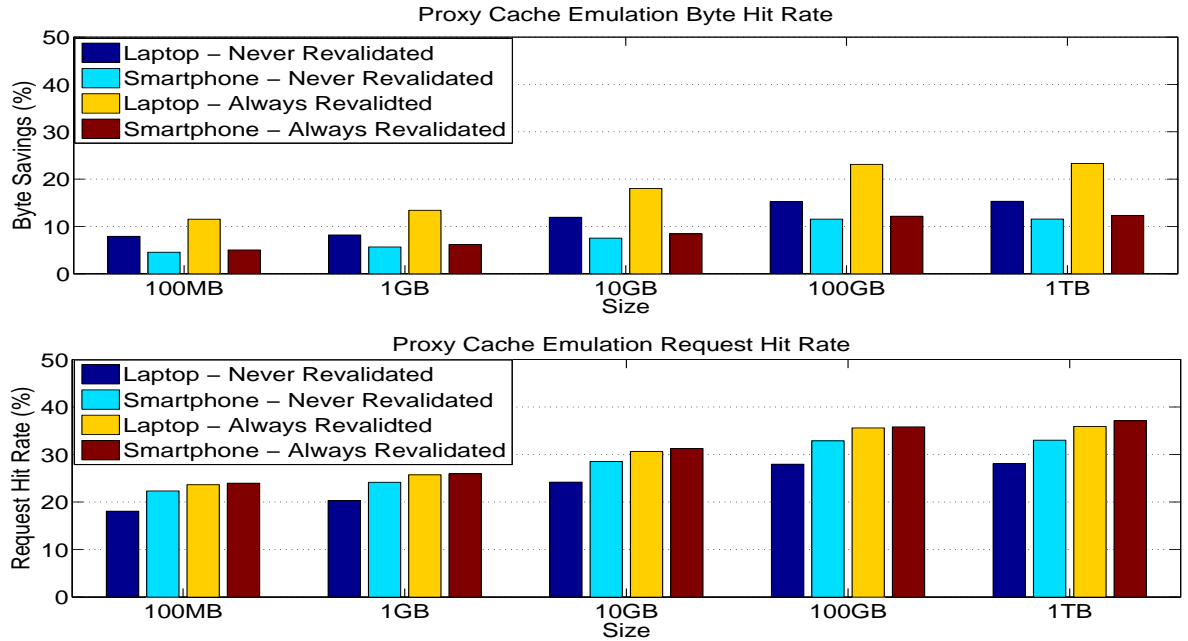


Figure 2.17: Proxy cache hit rate with several revalidation strategies and persistent storage sizes.

Table 2.14: Hit rate for infinite sized proxy space with and without handling Range Offsets

Policy	Hit Rate	Smartphones	Laptops
Partial with URL & Range	Byte	11.34%	17.28%
Partial with URL & Range	Request	38.42%	30.55%
Partial with URL & No Range	Byte	10.78%	16.32%
Partial with URL & No Range	Request	34.55%	27.87%

because the object freshness for laptops is typically smaller than for smartphones, as observed by the *max-age* header values, which tend to be larger in smartphone responses.

Table 2.14 shows the benefits of a proxy cache comparing when range offsets are handled or not. The savings for both cases are similar, indicating that caching the range requests does not add a significant amount of savings. This is an artifact of the low hit rate of the multimedia traffic. Figure 2.18 shows the effectiveness of a proxy cache for each content type across the range of devices, for both requests and byte savings. We see that video and audio hit rates are lower than other content types. On the other hand, images and text have reasonably good byte hit rates, usually $> 30\%$ across most of the devices. Hence, our results suggest that caching videos may not yield much benefit because of their low temporal locality.

Figure 2.19 (a) shows the impact of prefetching. An HTTP proxy cache, conceptually, acts as a buffer between the clients and the servers, and can potentially download more data than a user requests. This may happen either because a proxy cache is configured to prefetch data,

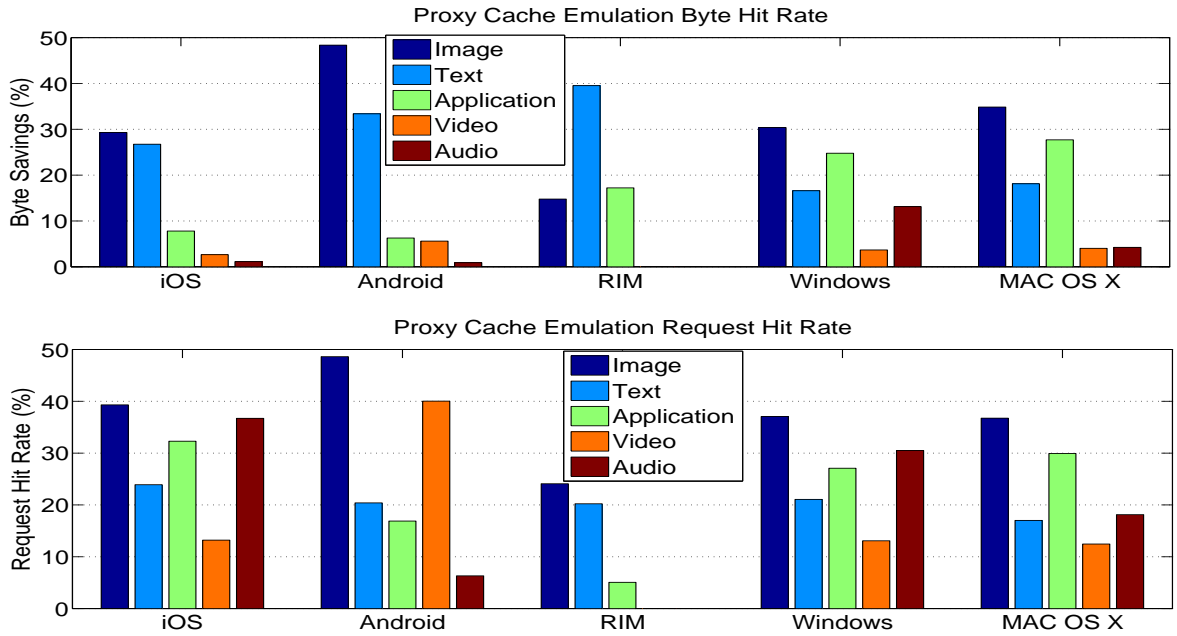


Figure 2.18: Proxy cache savings for infinite sized cache per content type for each operating system

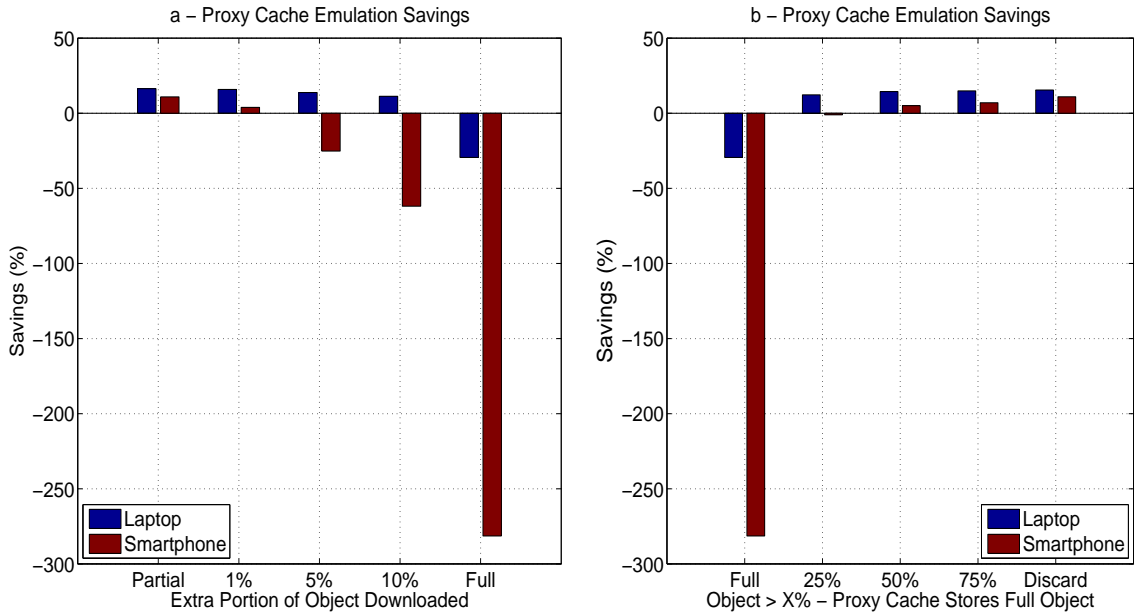


Figure 2.19: (a) Effect of Prefetching on Savings and (b) Conditional Caching for Partial Downloads

or because the connection between the server and the proxy is faster (e.g., because the packet loss rate over the wireless network is higher). While prefetching can increase the load on the network, better hit rates may be attained from having more bytes available in the cache. To evaluate this effect, we consider the case where the cache downloads $X\%$ more data than the data downloaded by the clients (up to the maximum size of the object). In the first bar, the cache stores only the amount of bytes the user is requesting. We can see that this approach provides the best savings. On the other hand, the full-download policy results in abusive bandwidth demands in smartphone devices, because of repetitive failures of large multimedia data.

Figure 2.19 (b) shows the byte savings for the various policies described in Section 2.3.3 that deal with partially downloaded objects. A partially downloaded object is cached in its entirety if more than $X\%$ of its size has been already downloaded by a user. The 0% case corresponds to the *full-download* policy, i.e., the cache downloads the full object independently of how much it has been already downloaded. The 100% case is the *discard* policy, i.e., any partially downloaded object is not cached. Any value in-between corresponds to the *conditional-download* policy. The graph shows that the full-download policy is actually detrimental, requesting content that is never viewed, particularly for smartphones.

Moreover, savings for smartphone traffic is more sensitive to the actual percentage of downloaded bytes after which an object becomes cacheable. This is because large objects are more likely to be partially downloaded by smartphones, as shown in Figure 2.19 (b), which results in more downloaded data when using the full-download and the conditional-download policies. The discard policy provides savings very close to the optimal one, indicating that partially downloaded objects are not likely to be accessed more than once.

Thus, our results indicate: *a)* replacement policies are not as important as they used to be in the wired networks; *b)* enabling prefetching in proxy caches may result in excessive bandwidth demand from smartphone devices; *c)* proxy caches should follow the *discard* storing policy above.

2.4 Related Work

Although most of the wireless networks are configured to dynamically allocate IP addresses, only a few studies focus on DHCP. Brick *et al.* [32] investigated the impact of lease times on DHCP performance. Khadikar *et al.* [59] studied the effects of longer DHCP lease times on the address space utilization. Our work is the first to differentiate the device types and study the DHCP usage patterns of smartphones. In addition, in contrast to previous studies we combine DHCP and TCP/UDP transaction log files in order to better understand network usage patterns for each device type. Finally, our work is the first to propose DHCP leasing policies that account for the various device types and their behaviors.

Moreover, in the last few years, several papers have studied the network traffic generated by smartphone devices. We summarize the relevant works in Table 2.15. Many have looked at issues related to our focus, but our work is particularly distinct in the following ways: a) We provide detailed statistics per device type (Android, iPhone, Windows laptop, etc.) in a way not done previously. Indeed, we present a novel approach for identifying different devices in Section 2.1 and quantify its advantage. b) We examine caching in depth beyond what has been done before, breaking out content types, device types, request hit rates as well as bandwidth savings (byte hit rates). c) We demonstrate the importance of partial downloads, evaluating their impact on browser and proxy caching. Given the increase in video traffic over the network, and the probability that video transfers are aborted, dealing with partial downloads is important both for the client browser cache and the proxy cache.

Several approaches use client instrumentation [41, 42, 52, 93] as a way of capturing network and application behavior. While this allows exact information, it suffers from a potential bias problem in that there may be a relatively small number of clients (and thus not statistically significant) [41, 42, 93] or that the platform is opt-in [52]. In addition, the instrumentation may be platform specific (e.g., iPhone-only), leaving out other categories of devices. Other approaches, including ours, use passive capture at a gateway [39, 40, 47, 63, 96] in order to capture all traffic of the population under observation. While the population itself may be biased (i.e., enterprise users, students), all users are measured.

Of the gateway-based research, several approaches are used to distinguish smartphones from the rest of the traffic. Some use the *User-Agent* header exclusively [40], or the *User-Agent* header combined with IP TTL information [63]. [47] validates the *User-Agent* header using the Organization Unique Identifier (OUI) from the packet's MAC address. [96] uses the available IMEI records to uniquely identify the smartphones.

Most related works do not compare smartphone behavior with laptops. [47] is a notable exception. Their work, however, focuses on network-level flow metrics in a university campus, whereas ours focuses on Web browsing and caching in an enterprise environment. Ours also goes further in distinguishing different types of smartphones and laptops, as well as providing a more accurate mechanism for identification, as shown in Section 2.1. Their work is also the only other one that studies WiFi, as most related smartphone work studies 3G. WiFi access behavior is also important given mobile user tendencies to situate in places such as home and work [96].

Table 2.15: Smartphone Related References

Ref.	Capture Approach	Net	User Population	Num Users	Trace (days)	Devices in Trace	Distinguish Handhelds?	Distinguish Smart. types	Focus
[39]	Gateway	3G	Consumer	1M	2	All	No	No	Caching w/ content-length
[40]	Gateway	3G	Consumer	3M	3	All	Yes (UA)	No	Video
[41]	Device	3G	Consumer	43	1660	Android, WinMob	N/A	No	TCP characteristics
[42]	Device	3G	Consumer	255	196	Android, WinMob	N/A	No	TCP Sessions/Applications
[47]	Gateway	WiFi	Students	32K	3	All	Yes (UA & OUI)	No	TCP and Web Traffic flows
[52]	Device	3G	Consumer	2500	95	All	No	No	Applications/Web/Carriers
[63]	Gateway	DSL	Consumer	20K	4	All	Yes	No	Web Traffic
[93]	Device	3G	Students	24	365	iPhones	No	No	Browser Cache
[96]	Gateway	3G	Consumer	600K	7	SmartPhones	Yes (IMEI)	No	Applications/Location
Ours	Gateway	WiFi	Enterprise	3000	21	All	Yes (DHCP)	Yes	Fingerprint/Web/Caching

2.5 Conclusion

The explosive growth of smartphones is expected to have significant impact on future network architectures and the technologies that will be deployed to deal with the increased traffic demands. Understanding the material differences between smartphone and laptop traffic patterns is important both to enterprise network administrators and to cellular network providers.

In this chapter, we have investigated the diverse traffic characteristics of the devices that connect to a wireless network. We first investigated issues related to the proper allocation of DHCP lease times, which have been created from the introduction of smartphone devices. We show that smartphones are primary responsible for the increase in the network address utilization, and fixed lease time policies are far from optimal, even when DHCP lease times are as low as one hour. In contrast, fix leased times of 15 minutes, while they significantly decrease address utilization, produce unnecessary DHCP related overhead. To reduce this overhead, we propose a differential lease policy that assigns different lease values to each device type. The policy makes use of a novel device fingerprinting technique done at the DHCP server, without requiring any protocol changes. The main benefit of this new DHCP lease policy is that is insensitive to the actual mixture between laptop and smartphone devices, thus removing the need to manually tune DHCP lease times as the mixture of devices continues to change.

We then studied the web traffic behavior of the mobile devices and the effectiveness of browser and proxy caching. Our results showed that smartphone browser caches are currently not as effective as laptops. In particular, properly caching partial content is important, both in the browser cache and the proxy cache. Ignoring them, as is the default in Squid, will result in lower cache hit rates, whereas using them to prefetch objects will result in wasted bandwidth. Partial content is particularly significant given that it occur more frequently with larger downloads, more frequently with video, and more frequently with iPhone users. We compared a set of policies for handling partial content, showing that a particular policy, partial download, works best, and is robust to both smartphone and laptop generated traffic. Therefore, implementing and deploying the appropriate policy should be a high priority to both vendors and users of browser and proxy caches.

Chapter 3

Data Redundancy Elimination in Wireless Networks

In the previous chapter we have used Web Proxy caching as a method of removing redundant objects. Unfortunately, Web caching does not work very well due to the growing volume of dynamic content, which is typically not cacheable, and the limited amount of sharing that has been observed in user access patterns. With the advent of streaming media websites (e.g., youtube), Web object caches have been enhanced with capabilities to store partial content of large objects such as streaming media [57].

More recently, data deduplication techniques have emerged as an effective way to accelerate networked applications by eliminating redundant data from traffic, and sending references to data instead of the actual data itself. Protocol-independent techniques that perform deduplication at the byte level have quickly emerged as commercial products in WAN and enterprise networks [35, 9, 11]. Their main advantages are that they a) perform redundancy elimination across a wider range of protocols and applications than just HTTP, b) perform deduplication at a finer granularity (bytes rather than objects) and thus can identify redundant content in partially modified files, and c) can partially or fully cache Web content that would have been classified as uncacheable by Web cacheability rules and/or HTTP header fields.

These techniques are known by different names: network data de-duplication, network redundancy elimination, etc. For simplicity and consistency, in this thesis we refer to these techniques as *byte caching*, both to stress the granularity on which redundancy is determined (sequences of bytes) and to contrast with the earlier approach of Web object caching. Indeed, Web object caching can be seen as a form of redundancy elimination, where full Web objects that have been already retrieved across the network are not retrieved again.

While byte caching has been examined in the context of enterprise networks and data centers, there has been little insight on its applicability in the wireless context. Our research both examines the characteristics of traffic being generated by mobile devices and evaluates the bandwidth savings achievable by exploiting redundancy elimination techniques. Our analysis shows that the amount of HTTP traffic being generated from mobile devices is higher than that reported from the backbone of an enterprise or university network [26]. The latter would possibly include traffic from servers or data centers, which is highly redundant. In contrast, we

perform a trace driven analysis on the effectiveness of deduplication techniques in the wireless context. We collected several traces from an educational institution in a location which at least 200 unique IPs were able to be detected, and address the following questions:

1. **How is traffic generated from mobile devices different from that of a wired network?** We observe HTTP being the major contributor to traffic. We also observe much less P2P or unidentified traffic. Since HTTP is the dominant component, we identify the content types from which redundancy is being generated.
2. **What is the optimal performance we can expect from the implementation of a redundancy elimination techniques?** The performance of a data deduplication technique is dependent on the size of the cache, data storage architecture and replacement policy. In order to identify the optimal behavior, we have assumed infinite sized caches. To this extent, we present the optimal results as well as the characteristics of traffic that is determined redundant.
3. **How effective is a byte cache and how does it compare to Web caching?** Our analysis shows that a byte-level data deduplication technique provides savings up to twice as high as a Web cache. An additional 4% of redundant content from non-Web applications is also removed by caching content using a byte cache.
4. **What is the right granularity of byte chunks for a byte cache?** We compare the benefits of different fingerprint sizes to determine the optimal one, as well as the appropriate technique to identify matching regions. We also devise an encoding technique on which a match is encoded by a pair of numbers called *location-length* pair.
5. **Where do savings come from? Are certain types of protocols, services, or content services more amenable to byte caching?** We present the distribution of bytes, objects, saved content, cacheable content for various levels: protocol, application and Web service.

Overall we find that byte caching is a much more promising technology for reducing bandwidth utilization than standard Web caching in wireless environments. Byte caching provides bandwidth reductions from 12-18 percent in our traces, versus 2-7 percent for Web caches. While these numbers are relatively small, reducing bandwidth by even a small amount as 5 percent is worth hundreds of millions of dollars to a wireless service provider [10]. In addition, byte caching is a relatively unexplored area compared to Web caching; it may be that there are approaches to improving the ability of a byte cache to identify redundant data.

3.1 Byte Caching in Wireless Networks

3.1.1 Methodology

Our goal is to understand the effectiveness of the above traffic deduplication techniques, i.e. of an object cache and a byte cache (both of the max-match and chunk-max implementation) and compare them with each other on the same bases. We use whole packet traces and we emulate actual implementations of those techniques. We measure their ability to identify redundant content as well as save bandwidth. More specifically, given a packet level trace T and a traffic deduplication technique D , we define as redundancy $R_{T,D}$ the percentage of bytes that the given technique can identify as showing up more than once in the trace. To translate the actual effectiveness of a technique to real savings, one has to consider the protocol overhead that the particular technique may incur. More specifically, given a packet level trace T and a traffic deduplication technique D , we define as savings $S_{T,D}$ the percentage of bytes saved on a hypothetical link where the given technique is applied. In order to determine the optimal savings, we assume that caches are of infinite size, such that our results are independent of the actual implementation details of the various cache replacement policies.

We present results classified across several layers of the network and application stacks. More specifically we classify traffic at three different protocol layers: a) at the network and transport layers (e.g. ARP, TCP, UDP, etc), b) at the application layer (e.g. HTTP, DNS, SMTP, etc), and c) at the HTTP content type layer (e.g. text, images, video, etc). Finally, for each of these traffic classes we measure both the relative and absolute savings. *Relative savings* of a traffic class are defined as the byte savings attributed to that class over the total bytes contribution of that class. On the other hand, *absolute savings* of a traffic class are defined as the byte savings attributed to that class over the total bytes in the trace. As such, if a traffic class has highly repetitive content it will also have high relative savings. On other hand, its absolute savings will not necessarily be high, as these savings depend on the total bytes contributed by that traffic class.

Packet Capture and Post-Processing

For capturing and processing the wireless traces, we followed a multi-stage approach which consisted of four processes each one written in C and with totally 6000 lines of code. Initially, the wireless traces were being captured with the use of airodump-ng (part of the Aircrack-ng library [1]), and by setting the wireless interface in passive mode. The traces were then being processed with a software based on libpcap library [6], such that all Rabin fingerprints were generated. To minimize the collision between two Rabin fingerprint hashes, the base for the modular arithmetic was set to be 2^{60} . For the Max-Match technique we stored all Rabin fingerprint hashes and for

the Chunk-Match approach we used the fingerprint hashes to determine the chunk boundaries. Each chunk was then referenced with a SHA-1 value. In both cases, TCP/IP headers, HTTP header information and DHCP payload information were being processed and stored. In order to keep the anonymity of the user, each IP address is encrypted with a SHA-1 hash value.

Due to the lossy nature of the wireless links, wireless traces contained a number of MAC layer transmissions. However, those packets should not have an effect on the potential savings from a cache implementation behind the BS/AP, therefore they were removed before the hashes of each approach were being generated. Moreover, we noticed that several users were far from the sniffer and would only contribute a small amount of bytes. Therefore, the users that generated less than 1000 bytes were not taken into account.

After all hashes for each method have been generated, another process would read the hashes along with the TCP/IP headers, HTTP headers and DHCP payload information would perform classification and OS fingerprinting. A protocol was classified based on the IP header. An application was selected from a finite set, and through string search based on the filters of L7 filter signatures [36]. The DHCP packets exchange was monitored to determine how much time a host using a specific IP was sending traffic and the DHCP release time was stored. TCP sessions and HTTP objects were reconstructed in order to determine partial web object matches.

In order to investigate the performance of a web cache, we emulated the partial downloading policy, mentioned in the previous chapter. We calculated the hit rate as well as the amount of cacheable content. The *object (byte) hit rate* can be defined as the number of objects (bytes) that have been referenced from the cache. *Cacheable objects (bytes)* are the number of objects (bytes) that have been stored in the cache.

Byte Cache Emulation

A byte cache implementation requires an encoding and a decoding cache. Figure 3.1 shows a typical byte cache architecture. On a cache miss, content and its fingerprints, used for indexing, are stored in the cache and the original packet is forwarded. On a cache hit, an out-of-band packet is sent from the encoding to the decoding byte cache. This out-of-band packet replaces any redundant content with a compact representation of the location of content in the cache. Then the decoding byte cache reconstructs the original packet by replacing this information with the original content found in its cache. The fingerprint window size is the number of bytes β that are required to calculate one fingerprint. Once the first index is calculated the window shifts one byte to the right and the encoding cache calculates the next fingerprint. Therefore two fingerprints that are next to each other, will have been generated from the same $\beta - 1$ bytes and 1 byte that is different. The fingerprints are computed over the packet from the beginning

of the payload of size M to the end, therefore the total number of fingerprints per packet are $M - \beta + 1$. β usually varies between 32-128 bytes [88], whereas a packet can take maximum values up to 9000 bytes for Jumbo Ethernet frames.

The fingerprints are used by the encoding cache in order to find redundant content in a packet, i.e. continues strings of bytes that have appeared before in a different packet. There are two commonly used byte caching techniques that can identify redundant content: max-match and chunk-match. Each of them comes with a different level of effectiveness and resource requirements. Max-match identifies almost all redundancies but comes with a high memory and disk access cost due to the fact that once a fingerprint match is found byte region of multiple packets have to be compared against byte to byte. Chunk-match identifies a portion of redundancies that max-match can identify and it has a higher CPU cost compared to max-match. On the other hand, it does not compare packet contents byte by byte and as such comes with a lower memory and disk access cost. Independently of the technique used in order to find duplicate content the encoding byte cache needs to replace any redundant regions with a compact representation of the duplicate content. There are various techniques that can be used to achieve that, with each of them leading to different byte cache savings.

In order to calculate the maximum attainable savings, we have implemented a scheme that is similar to compression algorithms [97]. We call this scheme *location-length encoding*. The out-of-bound communication between an encoding and a decoding byte-level cache is based on TCP, therefore packets arrive in order. For every packet that comes in, it gets stored in a particular location in the encoding cache, and at the same location when it passes the decoding cache. (i.e. both caches are fully consistent assuming no packet losses). Then for every incoming packet once a region that matches with the current data in the encoding cache is found, the region is replaced with the location of the first match (6 bytes are enough for 256TB of cache) and the length of the match (2 bytes are enough for the Jumbo Ethernet packets). For this reason we associate a 8 byte overhead for every matching region.

Next we describe how matching regions are found for each of the two byte caching techniques.

Max-Match

In the max-match approach, for every packet a fraction of the fingerprints are selected by the encoding byte cache and stored together with the full content of the packet. This subset of fingerprints is generated using a pseudo-random function that selects fingerprints based on the content of the packet from which the fingerprint was generated. In essence, this function samples the fingerprint space in a determinist manner. The size of the sample is a configurable parameter with bigger samples leading to higher potential savings but also higher memory and disk access cost. The sampled set of fingerprints is then used in order to index the packet contents. As

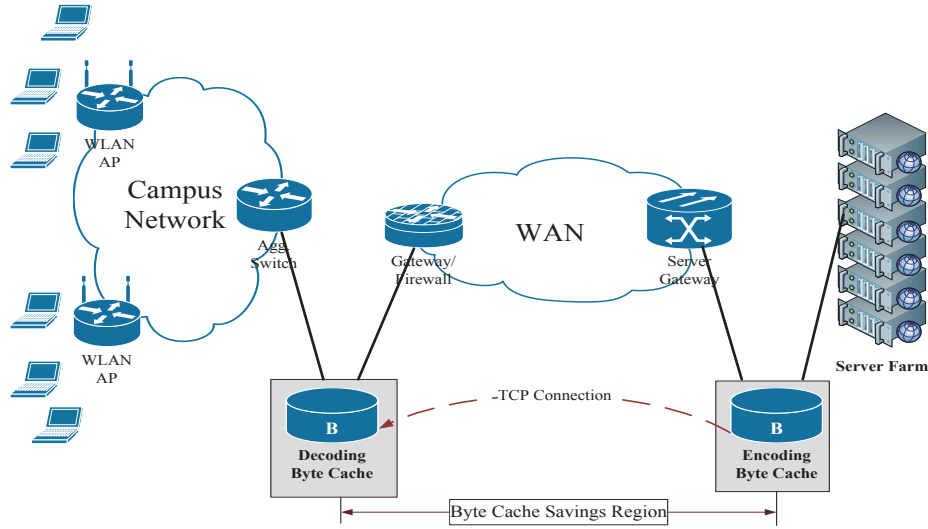


Figure 3.1: Byte Cache Architecture

such, whenever a new packet is received by the encoding cache, it generates a subset of the fingerprints and searches if any of them has already been seen before. If a match is found then the encoding byte cache tries to expand around both sides of the matching region. i.e. the region of the packet that has generated the matching fingerprint, and find a maximum match with any of the packets that has generated the same fingerprint before. More specifically, the encoding cache will retrieve all packets that have generated the same fingerprint before and then compare byte by byte the matching region of the packet that is under encoding to all other matching packets until a maximum match is found.

From the above it becomes clear that the memory and disk access cost of the encoding byte cache can be prohibitively high if many packets produce the same fingerprints. To deal with this processing overhead issue, we have selected a simple policy of retrieving only the last packet that produced a given fingerprint. In other words, each fingerprint is associated with the contents of only one packet, with that packet being always the last one that produced that fingerprint. Given that this policy can potentially lead to slightly lower savings compared to one that searches all eligible packets, we decided to make use of all fingerprints produced by each packet (i.e. we do not sample the fingerprint space¹).

¹We plan to further investigate this byte cache implementation specific issue in the future in combination with cache replacement policies, and further understand the tradeoffs between processing cost and potential savings

Chunk-Match

In chunk-match technique Rabin fingerprints are being generated over the payload of a packet. When a fingerprint matches a specified constant Y (by performing a modulo operation over Y), that fingerprint constitutes a boundary. The data between two boundaries are the chunks. Chunk identifiers are then being generated through SHA-1 hashes. While this approach is similar to [56] and [23], we have made a number modifications. The chunk-match algorithm is performed per packet and every packet generates at least one chunk. The algorithm is therefore used for any kind of traffic (TCP, UDP etc.) and the chunk sizes are small enough to identify similar redundancy as in max-match. Also, instead of sending the SHA-1 hash (20 bytes) for every chunk, we use the following approach: i) if redundancy spans over two or more chunks, a single hash value is generated for the maximum matching region (multi resolution chunking). ii) for the encoding, we perform the *location-length* encoding policy, therefore a pointer to the location is send and the size of the maximum matching region (send 8 Bytes). This resolves one of the drawbacks of this methodology, as noted in [23], that a 20 bytes SHA-1 per chunk would add a significant overhead.

Chunks that are smaller than 8 Bytes, would have a negative effect on the savings because of the encoding overhead. Therefore, a boundary is chosen only if the chunk size is greater than 8 bytes. Moreover, the start and the end of the packet are default boundaries. Thus, if the packet is bigger than β , then it will generate at least one chunk.

Content-Type Classification

One of the major functionalities of our processes were to relate an HTTP request with the appropriate HTTP response. In fact each HTTP session, begins by an HTTP request to the web server. In order to determine effectively the HTTP request, a string search over the payload was performed. Once an HTTP request was found, we stored the HTTP request in an array indexed by the TCP ACK of the last packet of the HTTP header. We also kept socket information (4-tuple) to verify that the process was flowing correctly. In fact, in many cases the HTTP request would span more than one packet, and therefore we had to keep track of the state of the HTTP request, until all the information from the HTTP header have been collected.

Once an HTTP response was seen, the TCP SEQ was used to index to the hash array. If an HTTP request was matched, then the reverse 4-tuple was being calculated and checked for a match with the HTTP request. Once, an association was found an HTTP session was open for the duration of the object length (not the Content-Length). Moreover, we kept a database of more than 120 Internet Media Types or else MIME (Multipurpose Internet Mail Extensions) classifiers. As soon as the Content-Type had a match with our database, the HTTP Content-Type was successfully found. In any other case, the content-type of the object was determined

Table 3.1: Trace File Details

	Trace A	Trace B	Trace C	Trace D
Length	9am-6pm (9h)	9am-5pm (8h)	9am-5pm (8h)	9am-4pm (7h)
Dates	21 July 2010	24 August 2010	09 September 2010	15 October 2010
Size (GB)	1.5	6.4	4.7	9.5
Packets (Millions)	2.3	10	6.9	12
Unique IPs	75	251	278	257

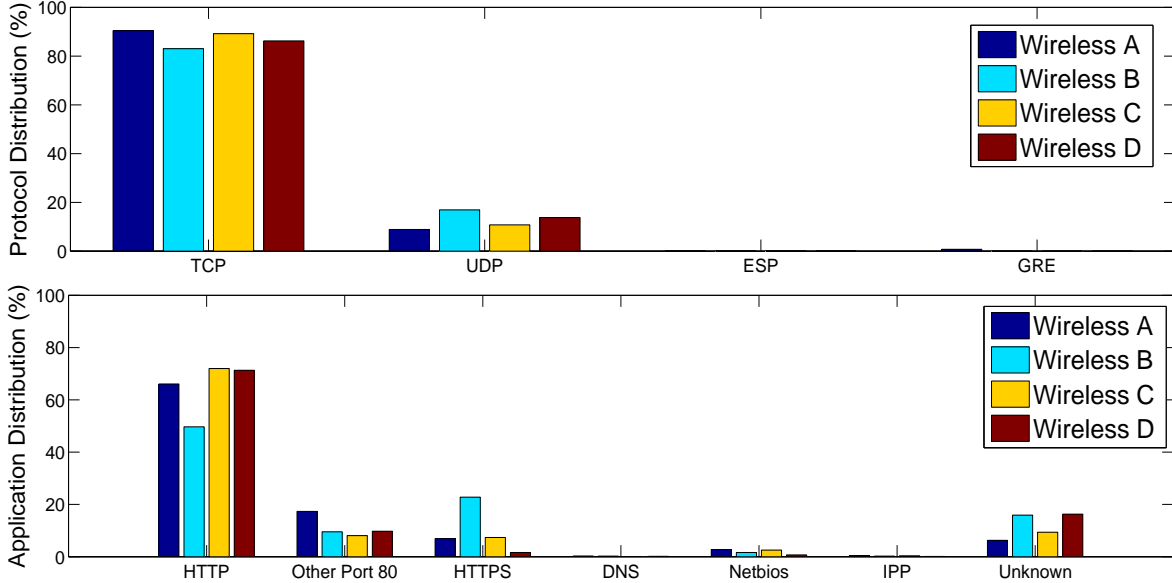


Figure 3.2: Protocol and Application Distribution of bytes in the traces

as unknown. We performed several iterations until our database had the maximum accuracy.

3.1.2 Results

In this Section we present our results. We first provide an overview of the trace characteristics.

3.2 Trace Characteristics

We captured wireless packets at four different periods for a period of 7-9 hours each. The first trace was collected during the summer semester and the other three during the fall semester (beginning, before midterm period, after midterm period). Trace details are given in table 3.1.

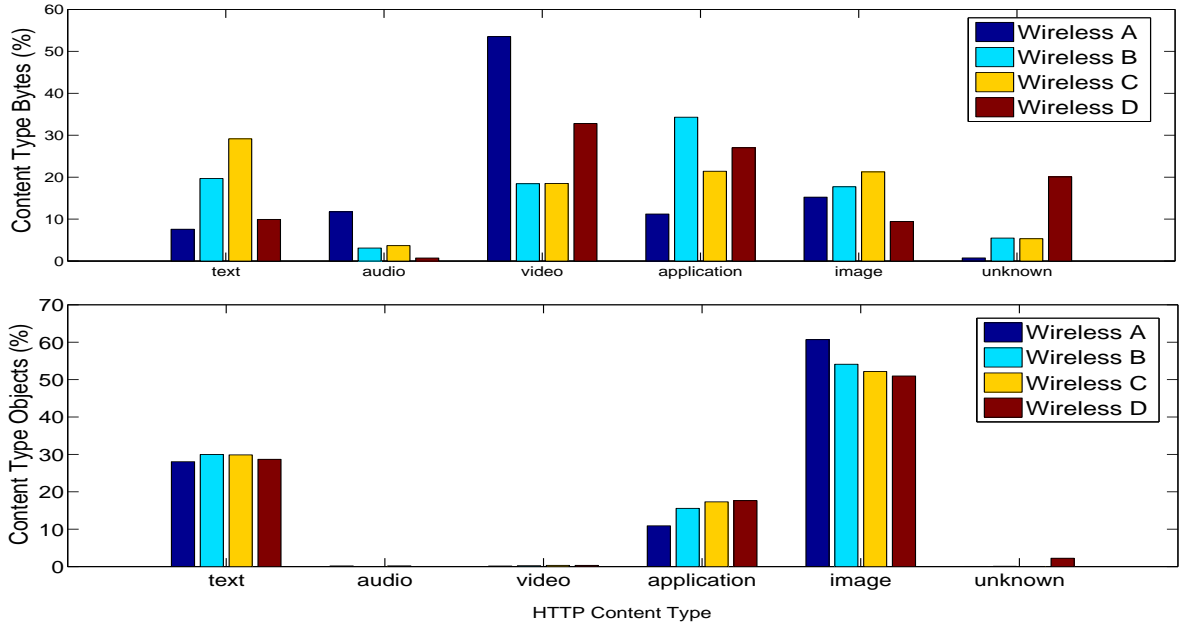


Figure 3.3: HTTP Content Type Byte and Object Distribution

The distribution of bytes transferred across protocols and applications are presented in Figure 3.2. Flows using the TCP protocol contribute the largest proportion of the traffic, with close to 90% of bytes across the traces. UDP flows are the second largest contributor, and minor portions of ESP and GRE (e.g., related to encrypted traffic). Protocols that contribute less than 0.01% are omitted from the graph. In the second subplot of figure 3.2, it is clear that most TCP traffic is observed to be HTTP. As mentioned earlier, applications that used TCP port 80 but were not identified as HTTP belong to the category “Other Port 80”. The byte contributions of HTTP, TCP Port 80 and HTTPS result in 85% of the total content, which is close to the overall 90% of TCP traffic, and shows the wide use of Web content by mobile users. The “Unknown” category is traffic that was not identified by our classification filters. Since UDP contributes around 10% of the bytes, it can be assumed that “unknown” would be mostly related to UDP.

Given that HTTP is the predominant application, in Figure 3.3 we isolate the HTTP content types. “text”, “video”, “applications” and “images” are the major contributors to bandwidth consumption. Looking deeper into the traces, we found that most of the “application” content types were related to software updates (e.g., Microsoft Windows updates). While the distribution of bytes does not vary across traces in the second subplot of figure 3.3, the distribution of object requests are consistent, in terms of HTTP content types, across all traces. Moreover,

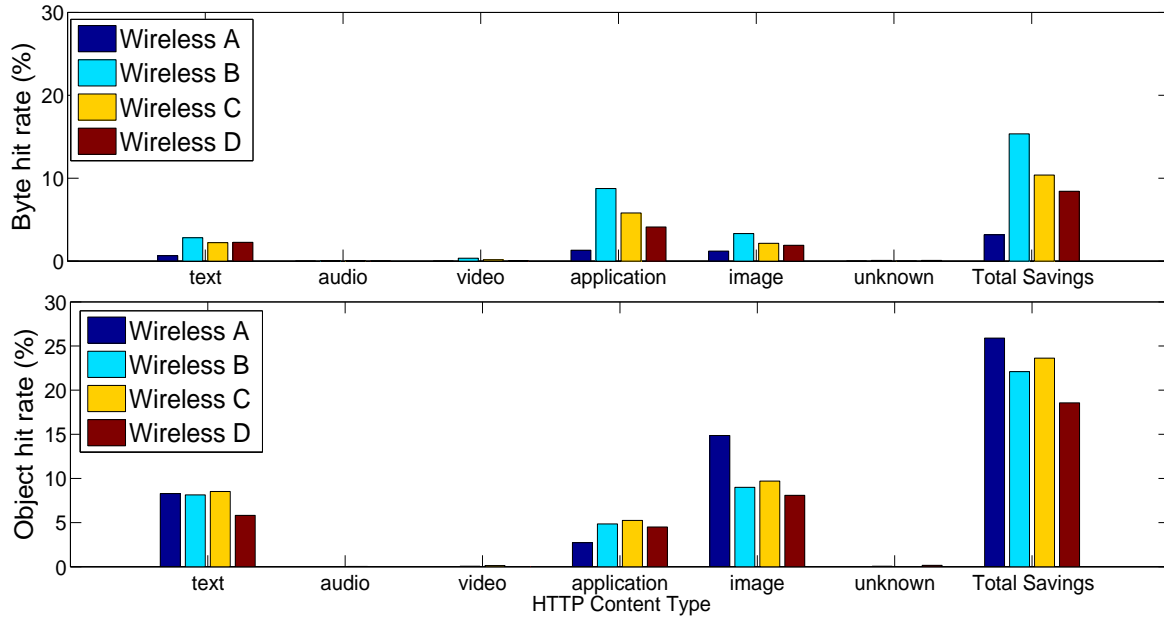


Figure 3.4: HTTP Byte and Object Hit Rate for a Web Object Cache per Content Type

while video is a major contributor in terms of bytes, the number of object requests to video content were relatively small.

3.2.1 Proxy Cache

In order to investigate the web cache savings further, we identify which objects contribute the most to the savings. As shown in Figure 3.4, most of the object redundancy comes from the "application", "text" and "images" content types. It is interesting to note that the "video" and "audio" objects contribute a small percentage to the total byte and object hit rate. A major factors is the small number of references to those content types, as was observed in the Object distribution results above.

To further understand why specific content types have low byte-hit rates, we show in figure 3.5 the relative byte savings alongside with the percentage of content that is cacheable. It is clear that the relative savings are small (most of the times $\leq 30\%$), with only "application" being an exception. There was not a consistent trend in terms of cacheability for "video" and "audio", but in some traces the cacheability was low (traces B and D). This was an interesting observation, because it suggests that VoD stream providers should make efforts to ensure their content is cacheable and thus benefit from a Web cache.

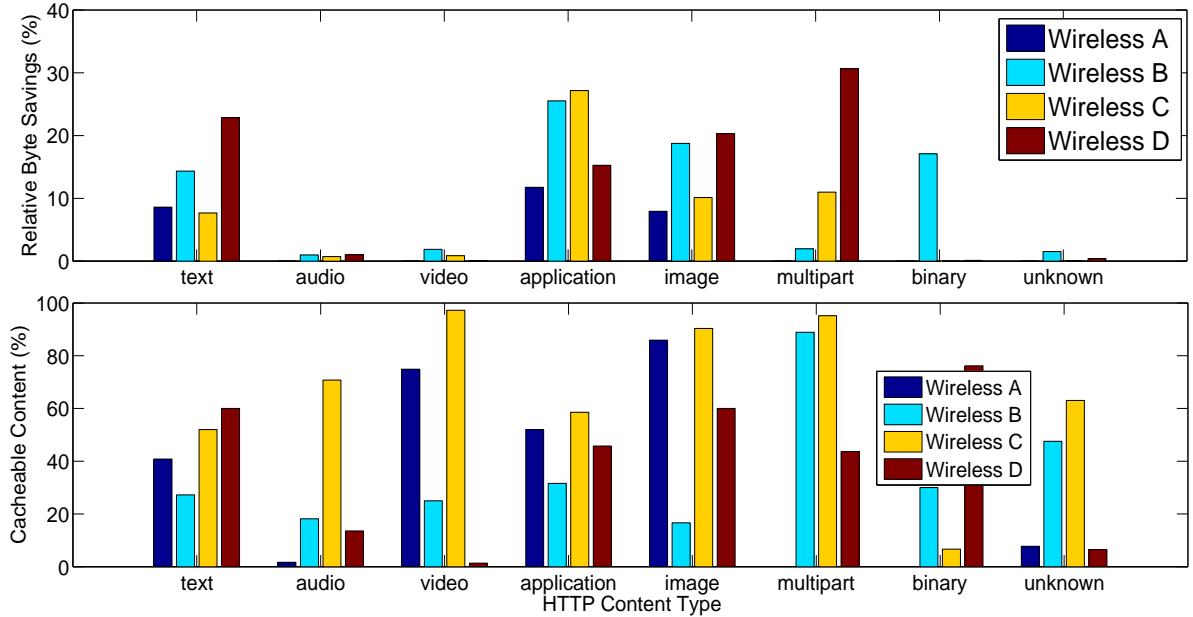


Figure 3.5: HTTP Relative Savings and Cacheable Content for a Web Object Cache per Content Type

3.2.2 Byte Cache Match-Match

In this Section we investigate the effectiveness of the Max-Match and Chunk-Match byte caching techniques. Max-Match is more effective in terms of savings, it may become computationally expensive, when trying to match redundant content that spans over the whole packet. Therefore, we believe that Chunk-Match can resolve this issue by identifying redundancy over chunks.

In order to select the optimal fingerprint size β , we have evaluated byte caching performance for various values of β s. Choosing a smaller fingerprint will provide more savings, because smaller content can be identified as redundant. However, [64] has noted that since more hashes are produced, the CPU and storage costs are higher. For space reasons, we limit the presentation to traces A and C. Similar trends are observed for the other traces. Figure 3.6 shows the relative savings for TCP and UDP traffic as a function of the fingerprint size. We see that byte savings of TCP traffic decreases very slowly as the fingerprint window size increases, and UDP traffic is more sensitive than TCP to the fingerprint window size. For values of fingerprint window higher than $\beta = 48$ bytes, UDP starts to decrease quickly, shrinking by half when $\beta = 64$. HTTP is not affected by the size of the fingerprint. For NetBIOS and unknown traffic, a fingerprint window size of more than 32 bytes would reduce the savings. Since $\beta = 32$ appears to provide good savings for both TCP and UDP, we use that value for the remainder of the experiments in the

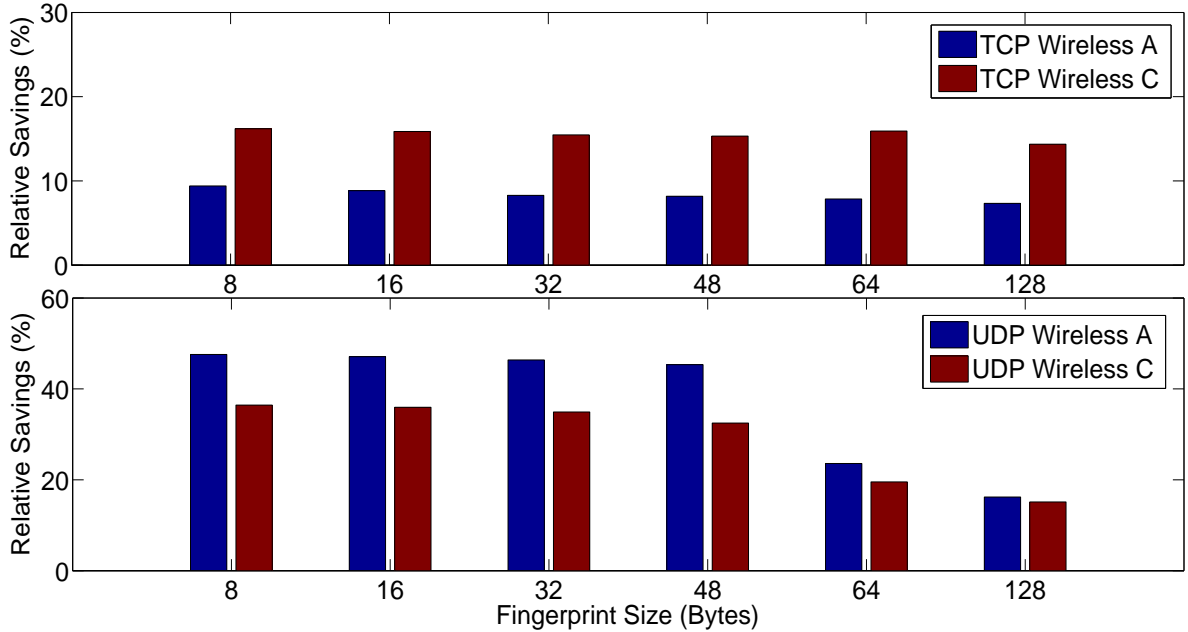


Figure 3.6: Optimal Fingerprint Size for each Protocol (Max Match byte cache)

paper.

To determine if one protocol is more amenable to byte caching than another, we analyzed the data based on the transport protocol. As shown in 3.8, TCP is the major contributor to byte savings. However, UDP appears more amenable to byte caching, as the relative savings for UDP are higher than that for TCP. Therefore, the reason that TCP contributes the most to the savings is because consists of 90% of the traffic, as observed in Section 3.2. Note that, an additional 4% of bandwidth savings can be achieved by using byte cache, since it caches UDP content, a savings not available to a Web cache.

To further identify the source of savings in TCP traffic, Figure 3.9 shows the application distribution. HTTP is again leads in terms of absolute savings, and we observe that the remainder of the applications contribute a small percentage. Therefore, targeting a byte cache to focus solely on Web traffic could potentially capture the majority of the benefit for TCP traffic. In contrast, HTTPS has a very small amount of redundant content. This is not surprising, since HTTPS uses SSL/TLS to encrypt the traffic, and encrypted traffic appears pseudo-random to foil statistical attacks. Thus, a byte cache that ignored TLS/SSL traffic would likely be effective, since no redundancy would be extracted from the stream, and no CPU, memory, or disk resources would be wasted caching HTTPS packets. Additionally, many UDP based applications are observed to be highly redundant, although their proportion of traffic was relatively

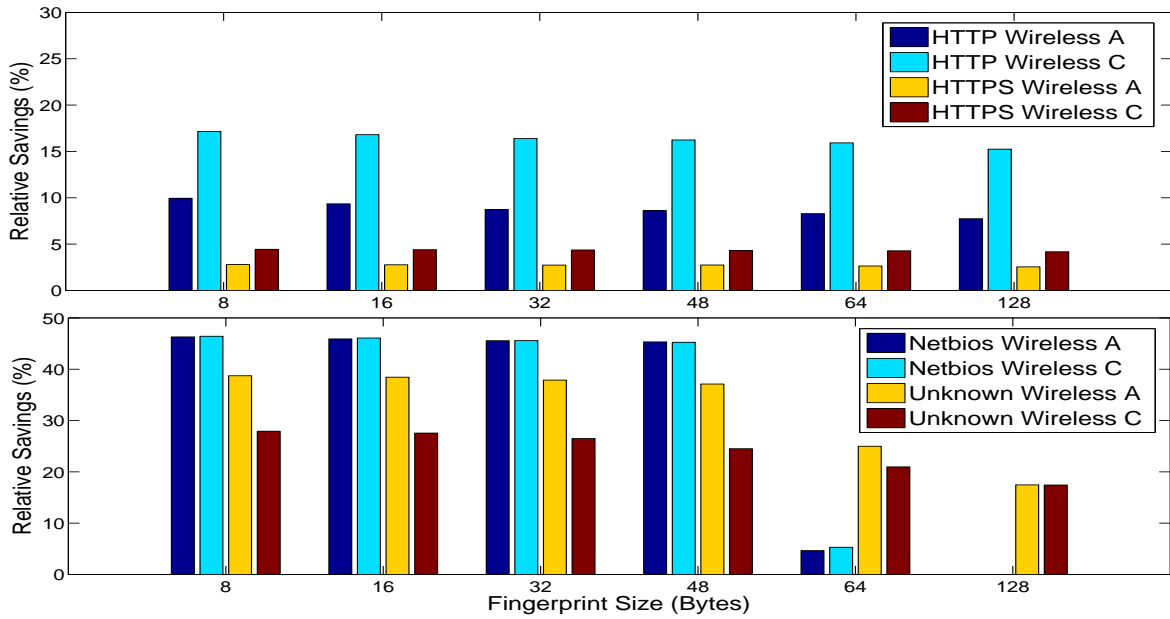


Figure 3.7: Optimal Fingerprint Size for each Application (Max Match byte cache)

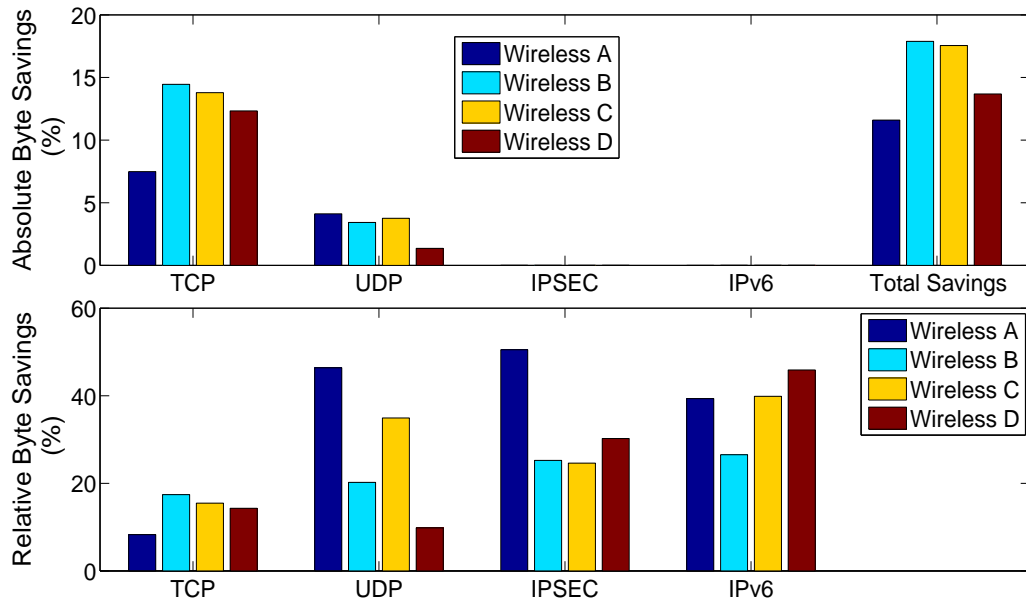


Figure 3.8: Protocol Decomposition and Savings (Max Match byte cache)

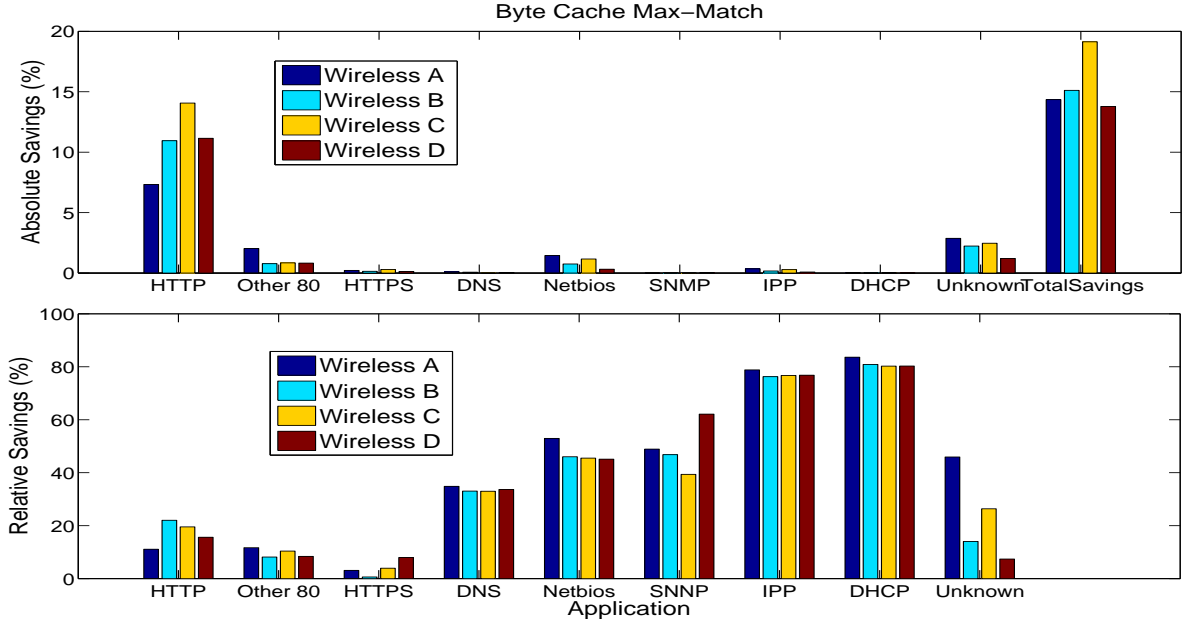


Figure 3.9: Application Decomposition and Savings (Max Match byte cache)

small. Thus, caching UDP traffic is likely worthwhile, since it is highly redundant yet would not consume much resources in terms of CPU or space in the cache.

Since HTTP content was the major contributor in both bytes and savings, we delve further to see which Web content type contribute to this effect. The “application”, “image” and “text” content types contribute the bulk of the savings. We observe a similar behavior in the Web object cache savings for those content types. However, more savings are captured in a byte cache. In addition, a byte cache is also able to capture “video” content savings. Given that video content is a major consumer of bandwidth, it is very important to extract any redundancy that exists for video content. [20] has reported a major increase on mobile video content, and this is consistent with our traces. Thus, we believe that a byte cache would do well to optimize video traffic.

3.2.3 Byte Cache Chunk-Match

In the Chunk-Match byte caching technique, the major factor that affects performance is the chunk size. The average chunk size can be controlled by the parameter Y . In [56], the authors defined chunks across TCP flows, therefore the average chunk size could vary from 32-64K bytes (the first value was selected because it was bigger than the SHA-1 cache encoding, and the second because objects might be very large). Our approach is different; we perform the selection of chunks inside the boundaries of the packet. Thus, our average chunk sizes will vary

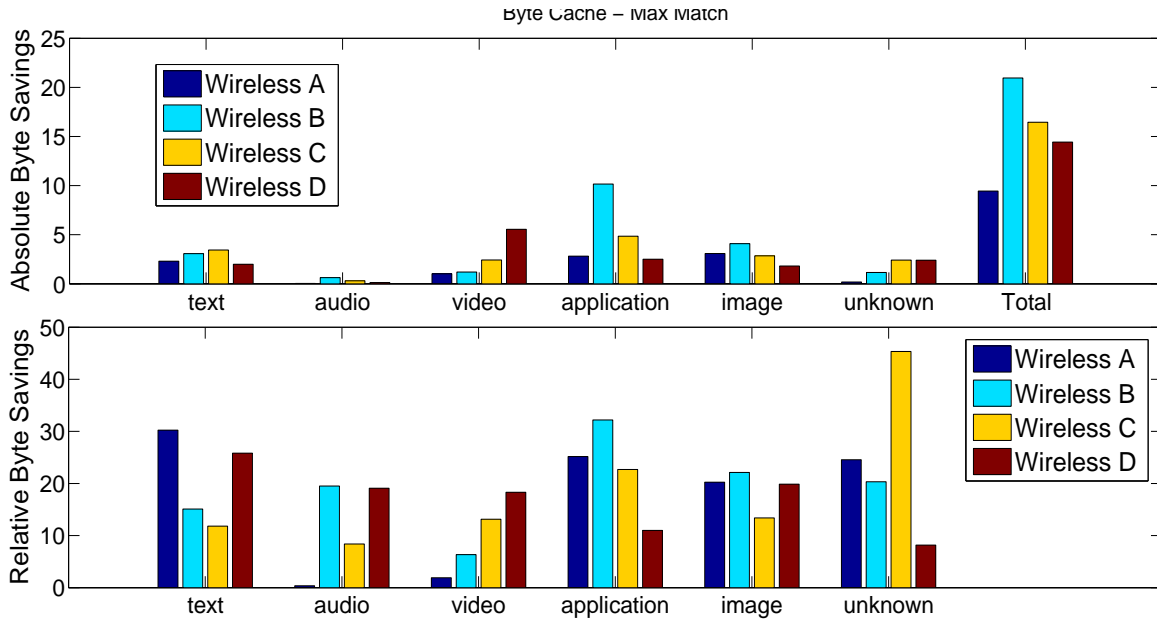


Figure 3.10: HTTP Content type decomposition and savings (Max Match byte cache)

from 8-1500 Bytes (the first value since the encoding is 8 Bytes and the second because it is the maximum size of a standard Ethernet packet).

Since the Chunk-Match technique is applied on per packet basis (as such independently of the protocol), the minimum chunk size as well as the default boundaries would have an effect on the average chunk size. We evaluate the average chunk size by varying the operator Y , which we perform the modulo to determine a boundary. The values vary slightly per trace and are presented on table 3.2.

In figure 3.11 and figure 3.12, we investigate the effect of the operator Y on the total savings as well as to the savings from each protocol. The issue of selecting the optimal Y is significant

Table 3.2: Average chunk size

Modulo Operator	Average Chunk Size
$Y = 128$	10.56 – 10.94
$Y = 256$	12.19 – 12.62
$Y = 512$	15.41 – 15.95
$Y = 1024$	21.75 – 22.46
$Y = 2048$	34.89 – 35.25
$Y = 5096$	327.81 – 363.93

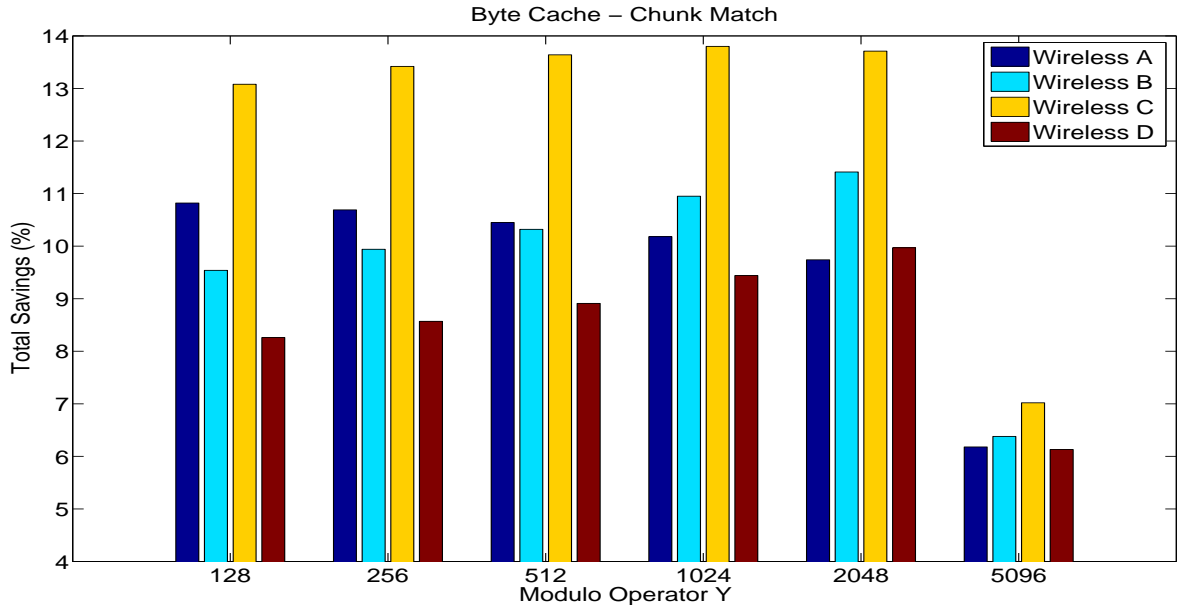


Figure 3.11: Total Savings as a function of Y

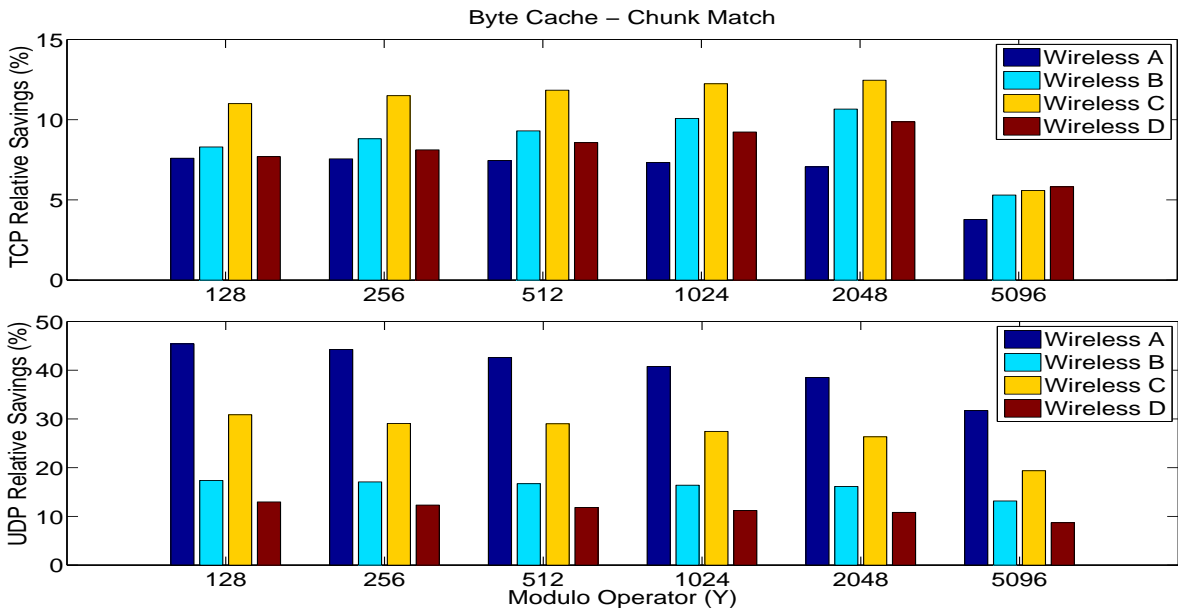


Figure 3.12: TCP and UDP Relative Savings as a function of the modulo operator Y

for the performance of the Chunk-Match technique. Since Y and chunk size are positively correlated, an increase on the Y would also affect the average chunk size. A bigger average chunk size would have a positive effect on the savings, because the overhead contribution per chunk is less. For example, for $Y = 128$ the average chunk size is close to 10 bytes, whereas the overhead is 8 bytes. Therefore, the average overhead is going to be close 80%. For a $Y = 2048$, the average chunk size is much bigger, thus the average overhead per chunk will be much less.

However, there is also trade-off for high values of Y . A high average chunk size would detect redundant content on a bigger granularity. Figure 3.11 provides a significant intuition on the appropriate value. For example, for traces B and C the savings are increasing until $Y = 2048$. For trace C, the maximum is reached at $Y = 1024$, and for A there is a negative trend starting from the smallest Y . In figure 3.12, we perform the same analysis for TCP and UDP relative savings. Increasing Y has a positive effect on the TCP savings, To this end, we believe that $Y < 2048$ is an effective solution for the Chunk-Match technique.

3.2.4 Comparison Analysis

In this Section, we examine what an appropriate solution for a wireless network might be. That is, if we get better performance from a byte cache than from a Web cache. To answer this question, we separate out the HTTP content and compare in terms of relative HTTP content. Figure 3.13 demonstrates the difference in relative savings for HTTP. A Max-Match byte cache outperforms the other two approaches. The Max-Match Byte cache outperforms a Web cache by 7-10 percentage points. The variation is based on the characteristics of the trace, e.g., when large objects are referenced from the Web cache, then the contribution of smaller content that is redundant for byte cache would be smaller. Chunk-Match is also better, in most cases, than a Web cache.

Figure 3.14 shows that savings among all the application types in stacked bar graph. The savings from a Max-Match byte caching are at least 2x-7x larger compared to Web caching. HTTP Savings from a byte cache is 4 – 8% higher than from a Web object cache. Since byte caching is application independent, an additional 4% of bandwidth savings are contributed by non-HTTP traffic.

In Figure 3.15, we plot the savings of a byte cache and a Web cache as a function of time over the trace. We see that a Max-Match byte cache implementation always outperforms a Web object cache. The Max-Match and Chunk-Match byte caches have similar performance, with Chunk-Match having slightly less savings. The byte cache savings show much higher variability than that of the Web cache, which tends to stabilize to $\pm 1\%$ of their last value, after 50% of the trace has been processed.

Finally, in Table 3.3 we present the memory required to store all the hashes for each of

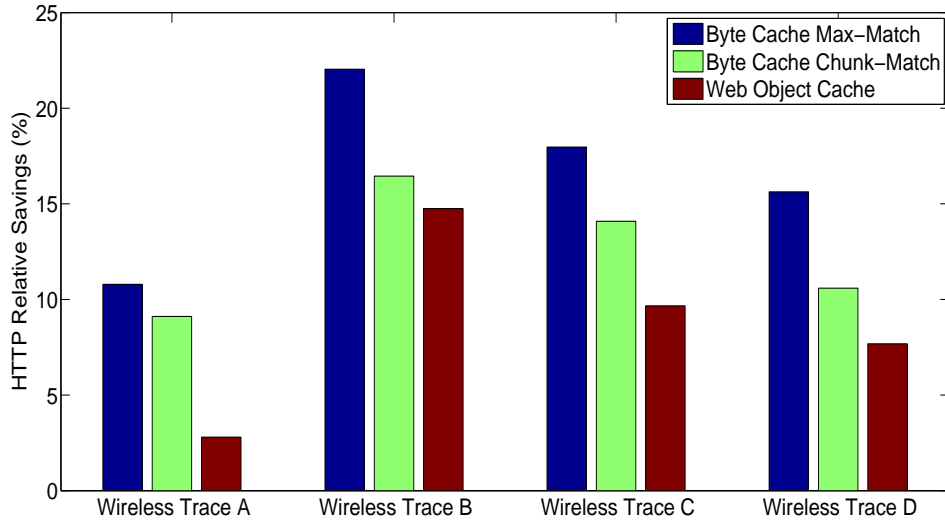


Figure 3.13: Byte Cache Max-Match and Chunk-Match vs Web Cache (identifiable HTTP content)

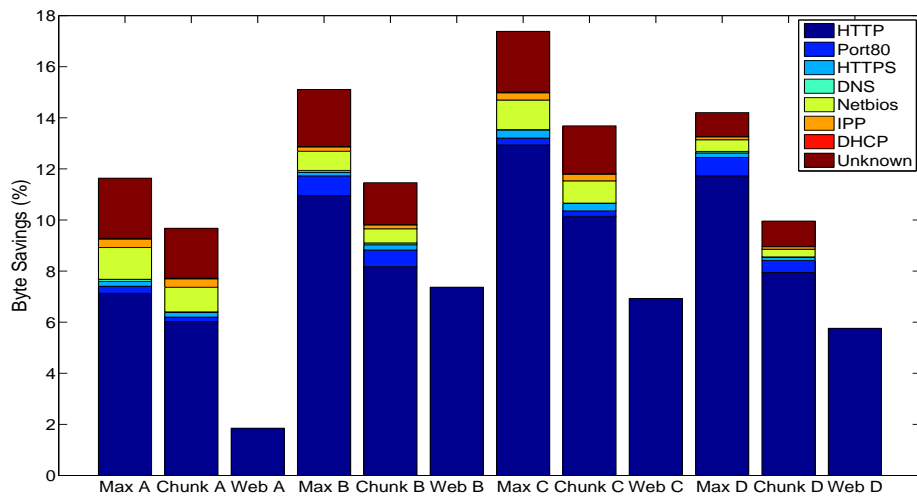


Figure 3.14: Byte Cache Max-Match and Chunk-Match vs Object-level Cache (all traffic)

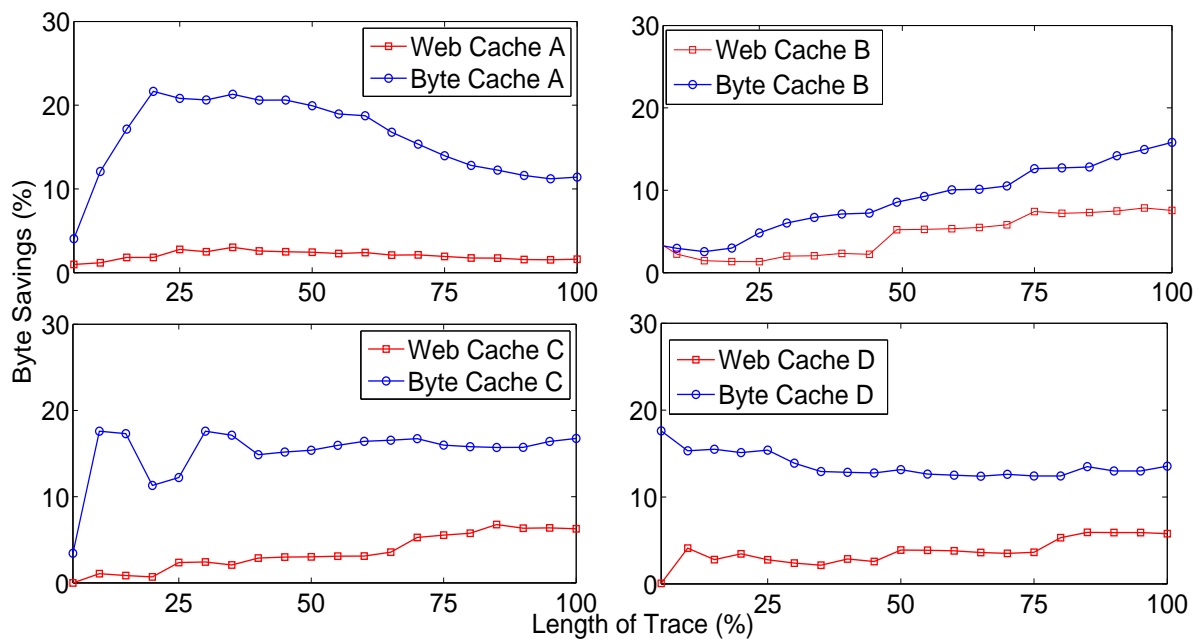


Figure 3.15: Max-Match Byte Cache vs Web Object Cache (all traffic)

Table 3.3: Hash Memory Requirements (GB)

Trace	Max-Match	Chunk-Match	
		$Y = 512$	$Y = 2048$
A	19.08	2.88	1.40
B	56.99	7.80	4.67
C	52.15	6.20	3.42
D	128.00	9.49	6.40

the two byte cache approaches. For even vary small chunk sizes, the storage requirements for Chunk-Match are much less than those from Max-Match. One of the main reasons is that two Rabin fingerprints that are adjacent to each other would have a difference of only a single byte. On the other hand, a Chunk-Match approach identifies independent chunks. Therefore, in terms of storage Chunk-Match is much more efficient.

3.3 Object and Byte Caching: A Hybrid Approach

Initially, chunks were identified inside each packet. However, in [56] the authors proposed a WAN optimization system that identifies duplicate chunks on top of the TCP layer. TCP based chunks are bigger than packet based chunks, but smaller than objects. The advantage of this approach is that it could remove the redundant bytes even if they span over many packets.

As mentioned in the previous section, a byte caching system requires the installation of two middleware boxes; one closer to the server (encoder) and one closer to the client (decoder). As the data flow from the server to the client, they pass through the boxes and they are broken into chunks. The chunks are stored on persistent storage and the corresponding hashes are stored in the memory of each box (e.g. a 1KB stream can be represented by a collision-free 20B hash). Since both boxes contain the same data, they are *synchronized*. A second reference to a chunk would mean that the encoding box would generate an out-of-band TCP connection with the decoding box and send the hash value instead of the chunk [88].

In byte caches², fingerprinting is performed based on the Rabin algorithm [81]. A sliding window moves byte and byte, generating the fingerprints. Each fingerprint is compared with a global constant to derive the boundaries of each chunk. However, performing these steps over all of the data may create a bottleneck in higher bandwidth links [65]. Moreover, the hash overhead per object in proxy caches is much smaller than the hash overhead per chunk in RE systems. Nonetheless, there is a vast amount of web data that are not cacheable per RFC 2616 [17] that a proxy cache would never cache.

In this chapter, we propose a hybrid redundancy elimination technique. The proposed system consists of a scheduler, an RE module (or byte cache) and a proxy cache module. The decision on which entity the byte stream is going to flow is taken by a scheduler that precedes those modules. The benefits of such an approach can be summarized as:

- Fewer hash computations are performed, therefore allowing our hybrid system to be deployed within higher bandwidth links.
- A TCP level compression scheme, instead of the standard IP packet based RE, leading to better savings and storage overhead.
- A reduction in the memory overhead compared to a standalone RE system. Cacheable content does not need to be broken in chunks and hash generation and storage can be omitted.
- The proposed approach can be implemented on the encoding box of a WAN optimization system without significant architectural modifications.

²In this chapter "Redundancy Elimination (RE)" and "Byte Cache" are used interchangeably

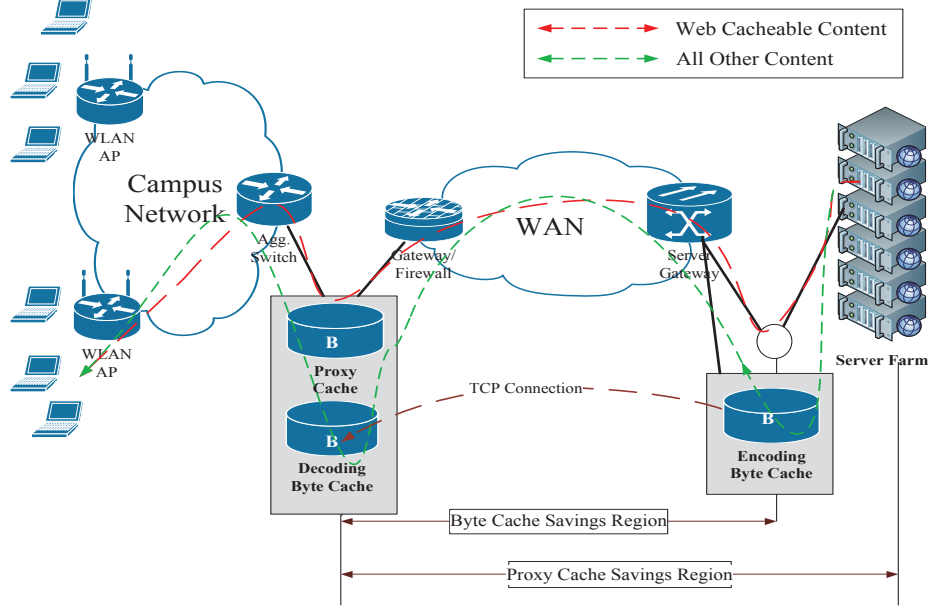


Figure 3.16: Proposed Architectural Approach of a Hybrid Cache

3.3.1 System Design

We initially assume a standard architectural approach of a byte cache [88]. The decoding box is located closer to the access network and the encoding box closer to the server. The encoding box performs fingerprinting of the byte stream, indexing and lookup, and data storage. The task of data reconstruction is performed at the decoding box. The proposed implementation is shown in Figure 3.16. In the encoding middleware we assume three separate entities (a) the scheduler, (b) the RE cache (or chunk based cache), and (c) the proxy cache. The memory and the persistent storage are shared among the caching modules.

Scheduler

The scheduler precedes the two caching modules, and is responsible to decide whether to forward the flow to the encoding RE or to the proxy cache. The scheduler operates at multiple layers. It first checks the IP header to identify the protocol. All TCP traffic is passed to the higher layers for further examination.

For every TCP connection that uses destination port 80, we look into the bytes of the TCP buffer to determine whether they contain web data. Once we have performed the application classification, we process the HTTP headers. Note that HTTP headers may span multiple packets or may be delivered in the application layer in different TCP buffer reads. By processing

the HTTP request headers the scheduler determines whether the object is cacheable or not, based on the RFC 2616 [17]. All cacheable content is directed towards the proxy cache module. All non-cacheable content is directed towards the RE module, since there would be no benefit from flowing through the proxy cache module.

For the remaining TCP and UDP traffic, the scheduler decides if they will flow through the RE module or they will not be processed by the hybrid system. However, unless mentioned specifically, we will assume that the scheduler decides not to allow the hybrid system to process those data. We follow this approach because, for non-web traffic, the RE module could show additional savings, whereas the proxy cache would not do that. Therefore for fairness in comparison we will focus on web traffic optimization. In the last section we will show further results on other types of traffic.

3.3.2 Module Design Principles

Proxy Cache Module

In the HTTP proxy cache module a hash value of the object's URL (host name and URI), along with some metadata, needs to be stored in non-volatile memory. The object's content is stored locally on the persistent storage. When an object is still valid in the proxy cache, it is retrieved from the cache without further communication with the end server. Otherwise, it is fetched from the end server. The proposed web proxy module is fully compliant with RFC 2616 [17]. This RFC defines several rules and leaves several others open for implementation. For the latter, we follow the most recent version of Squid [89] with LRU replacement policy. However, there are some cases for which we use an optimized approach compared to Squid.

First, since recent works [46] have indicated that several users tend to partially download an object (e.g. watch a small number of seconds from a video and then jump to another one), we assume a *partial downloading* policy. This is different from the default policies in Squid, in which an object is cached only if an $x\%$ of the object has been downloaded (this configurable argument in squid is called *range offset limit*). In other words, the proxy cache only stores the portion of the object that has been already requested by the users (which is the optimum policy in terms of cache overhead and savings).

Second, when the content provider sends the objects in fragments³, different fragments of the object will have the same URL. Every fragment may be sent on a separate TCP connection and the client replies with an HTTP *206 Partial Content* response. The content provider uses the *Range Offset* field in the HTTP header to uniquely identify a) the size of the object, b) the position of the fragment inside the object stream. Our proxy cache is able to parse the *Range*

³Youtube and Netflix videos displayed on smartphones are sent to the phone in fragments. This technique is called *adaptive streaming*

Offset and uniquely identify the object.

Lastly, the proxy cache module incorporates video optimizations (such as uniquely identifying videos even if their URLs contain user-related information), similar to the video cache Squid plugin [14]. For example, accessing Youtube service is done in two phases: 1) content look-up, 2) content download and playback. The first phase is performed on the web browser (or a custom application that embeds Youtube video) by selecting a video uniquely identified by an 11 byte *videoID*. In the second phase, the Youtube video contacts the Youtube CDN to collect a copy of the video object solely identified by a 16 byte *cacheID*. While the PC-player and Mobile-player may have some differences [46], our proxy cache module is able to handle those cases from Youtube and from several other video service providers. The main caching directives of our implementation are described in the Appendix-B.

Byte Cache Module

Our byte cache module follows the *Chunk-Match* principles from [23] with several major differences. In [23] the redundancy elimination was performed on a per packet basis. A Rabin fingerprint [81] is generated for every substring of length β . The base of the modular arithmetic, for generating the fingerprints, was set to 2^{60} , such that the collisions among hashes are minimum. When a fingerprint matches a specified constant γ ($hash \bmod \gamma = 0$), the fingerprint constitutes a boundary. In our case, the fingerprint generation is performed per web object. One of the main advantages of this approach are the higher upper bounds of compression. Assuming a perfect match, the upper bound of the packet level compression is $1 - \beta/MTU$, whereas in our approach it may reach $1 - \beta/object_size$.

Packets that contain the same data are usually of the same size, therefore determining the boundaries is relatively easy. Since the proposed RE module sits on top of the TCP layer, the TCP buffer size may change based on the network conditions. In a naive approach, the whole object would be allocated in RAM, and then processed byte-by-byte. However, this has problems related to unnecessary memory allocation (especially for some large objects) and the process of loading the whole object in memory and then fingerprinting the data is serial.

To tackle this issue we use a small circular buffer per TCP connection. We copy the byte stream of the TCP buffer to this circular buffer. The fingerprints are computed over the bytes of the corresponding circular buffer. Assuming a temporal size M of the circular buffer, the total number of fingerprints per buffer read is $M - \beta + 1$. Since the Rabin algorithm determines the fingerprints in a sliding window manner, we are aware of the fingerprints from the start of the TCP circular buffer and up to β bytes before the end. We remove all processed bytes from the circular buffer, and keep the last $\beta - 1$ bytes in the circular buffer. Once a new TCP buffer is available for this connection, we allocate the new byte steam after the $\beta - 1$ bytes and

Table 3.4: RE cache module constants and variables definition

Constants	
Fingerprint window size	β
Module operator for chunk boundaries	γ
Minimum chunk size	δ
Temporal Parameters	
Size of circular TCP buffer	M
Chunk size	L
Hash size	N

perform the fingerprint generation. Hence, the size of the circular buffer per TCP connection must be at least the size of the fingerprint window size. For fairness in comparison, we have implemented an LRU policy for the RE module. All hashes are stored in RAM, and the content in the persistent storage.

Finally for the byte cache (RE), we assume a minimum chunk size, namely δ . This is because: (a) We want to avoid very small chunk sizes because the overhead will be high. Assuming N as the hash size, if $\delta < N$, it will result in positive redundancy but negative savings. (b) Smaller chunks tend to contribute much less than bigger chunks in savings. (c) A smaller δ may decrease the CPU utilization as the modulo operations, for chunk boundary determination, are computed only when $L > \delta$. (d) For security reasons, since an attacker can flood the RE cache with very small or very big chunks⁴. The proper selection for the value of δ is shown in section 3.3.4.

Memory Overhead

The memory overhead is the amount of bytes that need to be stored in memory. Those bytes are related to the hashes that point to the disk location, where the actual content is stored.

A hash of the URL along with some metadata are stored in memory, and the object itself on persistent storage. The index entry in Squid is 80 bytes. An optimized indexing methodology [30] would not change the overhead considerably.

For the byte caches, the chunks are stored in persistent storage. For every chunk the representing hash value is stored in memory. Let us assume that all chunks are very small, e.g., $L = 8B$. A hash value of size $N = 4B$ would produce 4.2 billion unique entries for 8B chunks, viz. 24TB of data on the HDD. The 4B hashes are stored in a doubly linked LRU list, for which a 2x3B virtual memory pointers (forward and backward connection pointers) would suffice for an 134GB hash memory space. The chunk size, which is also stored in memory, is 2B; the log generation output 1B and the disk number 1B (in case a disk array is implemented). The total

⁴Since the Rabin fingerprint determines the boundaries based on the content, any long sequence of 00 would potentially generate a chunk boundary or an a priori knowledge of the γ , which would enable an attacker to choose specific content that does not generate boundaries.

overhead per chunk can therefore be 14B.

Transmission Overhead

The transmission overhead is the amount of bytes that will be sent instead of the actual data. It is expressed in terms of percentages of the actual data. For example, if each 60B chunk/object is encoded with a unique 6B hash, the transmission overhead is 10%. Therefore, we define:

- **Savings** *as the amount of bytes the provider would not transmit due to the implementation of a redundancy elimination technique.*
- **Redundancy** *is the amount bytes that a technology would determine redundant.*

Note that the difference between savings and redundancy is the transmission overhead.

For the proxy cache, once an object is found in the cache and it is fresh, the object does not have to be downloaded from the server. As described above, the implemented partial downloading policy does not add extra overhead.

In the RE module each chunk is replaced by a hash, therefore the transmission overhead is equal to N/L . However, there are many cases in which the redundancy spans over more than one chunks. In the proposed implementation, the encoding cache sends the location of the chunk in memory (4 bytes are enough to map a memory of 4TB) and also 2B that indicate the maximum matching region. Thus, the transmission overhead has a lower bound of 6B per chunk.

3.3.3 Implementation and Data Set

To understand the tradeoffs of each type of cache (standalone web cache, standalone chunk based cache, hybrid cache) we have developed an emulator for each one. The emulators read packet-level traces from Tcpcdump [6] and emulate various scenarios. We use Libnids library [94] for TCP stream reassembly, which emulates a Linux Kernel 2.0.x IP stack. Object reconstruction and each of the caching techniques are developed as separate libraries in C.

We captured two wireless packet traces in an aggregation router of an educational institution. One in a low bandwidth link (Trace A) and one in a high bandwidth link (Trace B). The details of the traces are shown in Table 3.5. We did not process the TCP connections, if only one half-stream appears in the traces. This is important because one stream may contain the HTTP request and the other the HTTP response. In figure 3.17, we showcase the protocol and application distribution in the captured traces. We may observe that the majority of the generated traffic is TCP and a smaller portion of UDP traffic. From the application analysis, we may see that HTTP is the dominant application, with a small portion of HTTPS, and 15% of other types of traffic. As we will show in the following section other types of traffic may add a portion in the savings.

Table 3.5: Trace File Details

	Trace A	Trace B
Length	9am-4pm (7h)	11am-2pm (3h)
Dates	15 Oct 2010	11 Jan 2011
Size (GB)	9.5GB	19GB
Packets (Mil)	12	36
Unique IPs	257	651

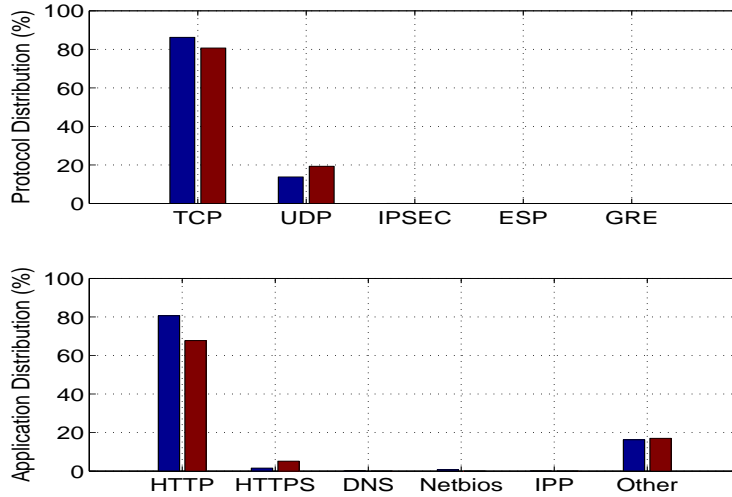


Figure 3.17: Protocol and application distribution of bytes in the traces

3.3.4 Results

Optimal Parameter Selection

The three main parameters of the RE module are the fingerprint size β , the average chunk size (which is a derivative of the modulo parameter γ) and the minimum chunk size δ . We have performed multiple runs of the emulator to determine the optimal combination of those parameters. We modified the scheduler to send all data to the RE module. Finally, we used the biggest trace (Trace B), such that the highest possible amount of data flow through the RE module.

We estimated the savings and memory overhead using a Response Surface Methodology (RSM) [31]. We varied the three parameters from 8 – 64B with a double step each time. $f_1(\beta, \gamma, \delta)$ represents the 3-dimensional response for the savings and $f_2(\beta, \gamma, \delta)$ the corresponding one for the memory overhead. Their responses are shown on figure 3.18. Note that in the graphs, the red values (higher) are better for the savings and the blue values (lower) are better

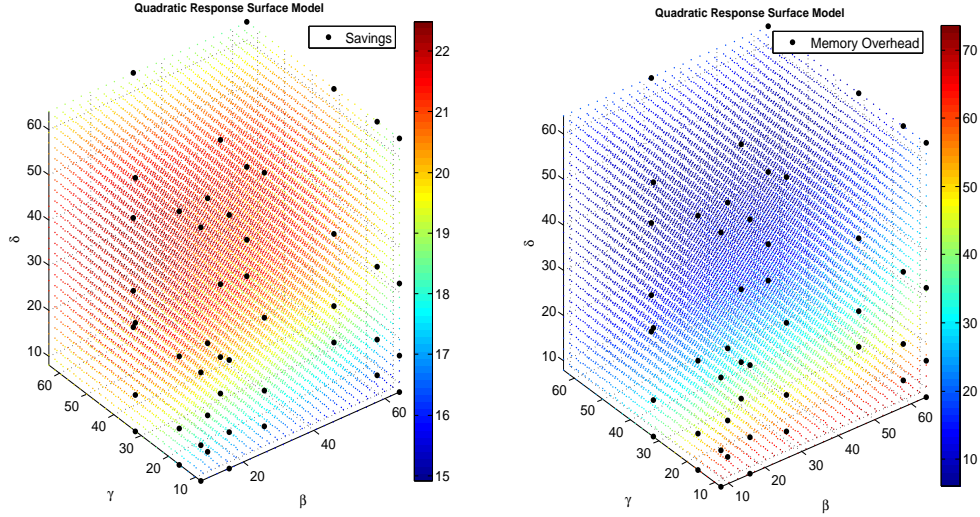


Figure 3.18: Savings and memory overhead as a function of the input parameters.

for the memory overhead. We can observe that the vector $\{\beta, \gamma, \delta\} = \{8, 32, 32\}$ provides the best savings, and the smallest memory overhead. A $\gamma = 32B$, would produce chunk size of average length around $64B$. For the rest of the results we are going to use these values. The best savings for web data using an RE module is 21% and this is attained with a memory overhead of 22%. In other words, for every 1TB of data in the disk, 220MB needs to be stored in memory.

The parameters of the statistical regression are shown below and have been estimated with a squared coefficient of multiple correlations $R_1^2 = 0.83$ and $R_2^2 = 0.92$.

$$f_1 = 12.7 - .01\beta + .217\gamma + .273\delta - .001\gamma\delta - .002\gamma^2 - .003\delta^2$$

$$f_2 = 103 - 0.03\beta - 2.04\gamma - 2.05\delta + .016\gamma\delta + 0.01\gamma^2 + 0.01\delta^2$$

For representation purposes, we have omitted the combinations that had minimal contribution (second most significant digit). The parameters of the polynomial provide an indication of the effects of each of the parameters to the savings and memory overhead. Obviously, the fingerprint size β has a smaller contribution compared to the other parameters, and the minimum chunk size δ has the highest contribution.

The Hybrid Approach

In this subsection the scheduler forwards the non-cacheable content to the RE module and the cacheable to the proxy caching module. This was derived from the observation that this

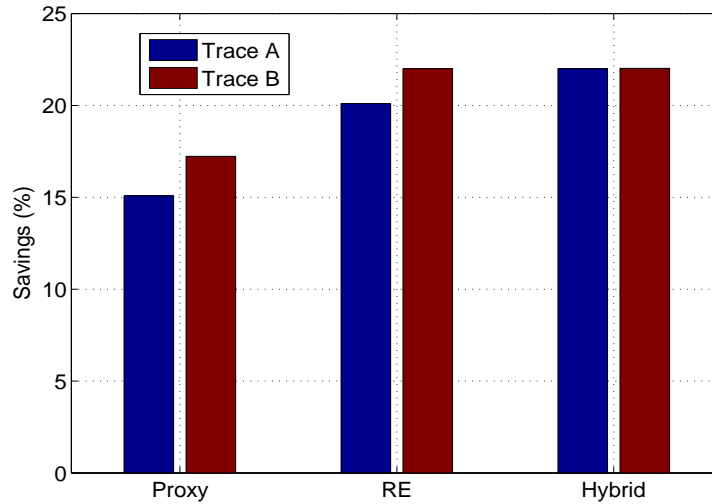


Figure 3.19: Savings for proxy cache standalone, RE standalone and hybrid system.

non-cacheable content can be more than 35% of the web content. For example, we found that the non-cacheable content is 49% for trace A and 37% for trace B.

In figure 3.19, we showcase the performance of a proxy standalone system, a Redundancy Elimination standalone system and the hybrid system that we propose. Obviously the savings from an RE standalone system would be higher than a proxy cache system. In our traces we find that RE provides 33% higher savings compared to a proxy cache. However, the most interesting is that a hybrid system provides higher savings than a standalone RE. Given that the RE has higher redundancy compared to the hybrid system in all traces (figure 3.20), we may conclude that the higher savings for the hybrid system are an artifact of the less transmission overhead to encode and send the chunks to the decoding side.

An additional advantage of the hybrid system is shown in the memory overhead graph of figure 3.21. We can see that the memory overhead for the RE system ranges from 15 – 22%. The hybrid system though requires half or one third of the memory compared to a standalone RE implementation.

The final micro-benchmark that we performed is based on the bandwidth that the system can support, or else the amount of time that the RE module requires to process the web data. We used a 16-core Intel Xeon X5560 with CPU 2.8GHz and 16GB of RAM on a 64bit Linux OS. By using the GNU profiler *gprof* [44], we isolated the cumulative time per trace to perform the chunk based RE functions. These functions are related to hash generation and fingerprinting. We observed that those functions take almost 95% of the CPU time, and the majority of the time was spent for generating the Rabin fingerprints (~ 2.31 msec/call).

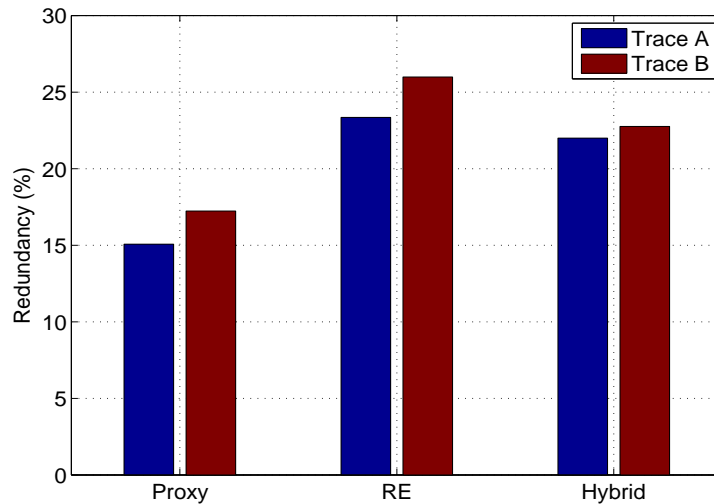


Figure 3.20: Redundancy for proxy cache standalone, RE standalone and hybrid system.

By dividing the bytes that would flow from each module with the cumulative time spend to process the data, we could derive the bandwidth of the system. Figure 3.22 indicates that hybrid approach can support up to 1.5 Gbps, viz. a 3x speedup compared to an RE only module. This speedup comes from the fact that the RE module in the hybrid case needs to process much less traffic (only the non-cacheable portion), whereas the RE standalone cache needs to process all web data. Since a hybrid cache can have higher or close to the RE standalone savings such a speedup is highly beneficial. Note that the hash computation for the proxy cache is minimal, thus it is not shown on figure 3.22.

Finally, we need to point that this speedup does not include the disk I/O operations. Yet newer disks can support close to Gbps of throughput, thus an RE only module would potentially create a CPU bottleneck.

Finite Sized Cache System

In this subsection we are going to use our biggest trace (Trace B). We perform two emulations of the hybrid system, one with a disk capacity of 1GB and another one with 10GB. The two emulations are plotted in figures 3.23a and 3.23b. Note that the x-axis represents the amount of bytes allocated to each module, in the format {RE module size - Proxy module size}. Those two figures depict two things: (A) They showcase the percentage of disk space allocation for each functionality in order to get the optimal savings. This is important as the memory and disk space are shared entities. (B) They present the difference in the savings between a proxy cache standalone system and a hybrid system (red and blue line). This is because the savings

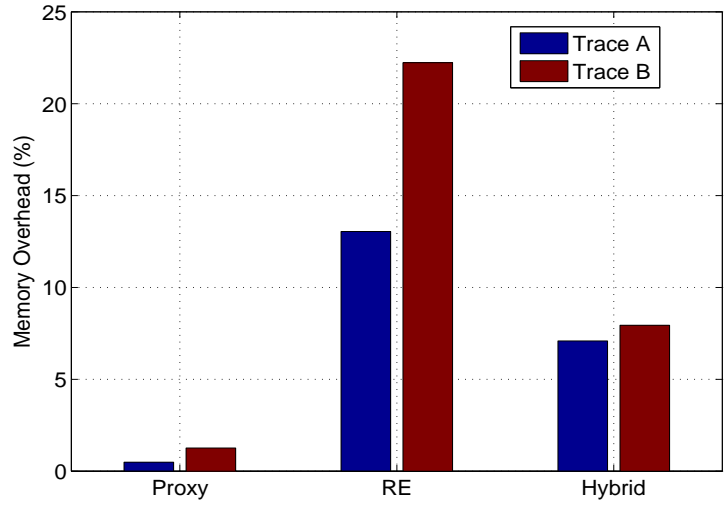


Figure 3.21: Memory requirement to represent data for proxy cache standalone, RE standalone and hybrid system.

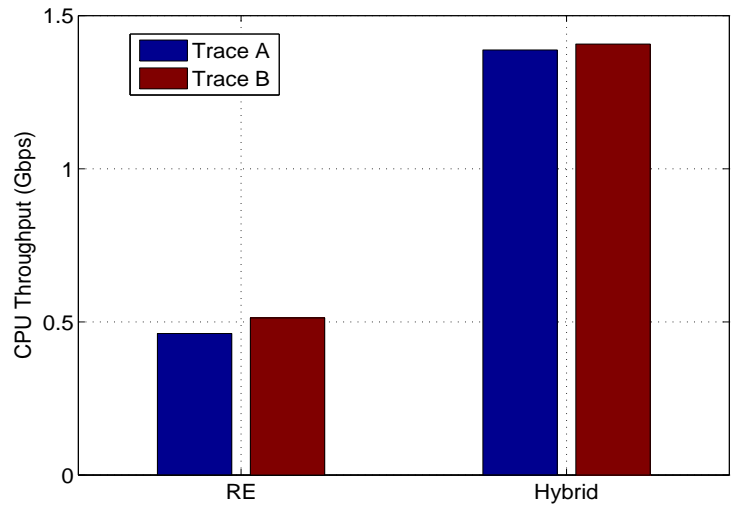


Figure 3.22: CPU throughput for an RE standalone and the hybrid system.

of the proxy cache module do not change when implemented as part of the hybrid system.

Therefore, from figure 3.23a, we may observe that the maximum attainable savings, for the 1GB hybrid system, are 19%. Whereas for the 10GB hybrid system the maximum attainable savings are 22%. Since an infinite sized cache⁵ provides savings close to 22%, we may conclude the total storage space of a hybrid system needs to be 10GB. Figures 3.23a and 3.23b present similar savings for the RE module. This indicates that the RE module has diminishing returns only after 400MB have been used. Given that the non-cacheable content in the trace is much bigger, we may conclude that a 400MB allocated to the RE module would be enough to provide the optimal savings. Apparently the decrease in the savings for the hybrid system with size 1GB, compared to the one with 10GB space, is an artifact of the proxy cache. Thus the proxy cache module needs to be at least 10GB to get the maximum savings. This shows that the RE module requires only 4% of additional storage space to provide an increase of 28% in the savings.

We also modified the scheduler to send all web data to the RE module (emulating again an RE standalone system). We observed a difference of 3% between a 1GB and 10GB disk storage. This indicates that the reason that only 400MB of RE module are required in the hybrid system is because the non-cacheable content is not as repetitive as the cacheable content. Non-cacheable content tends to include user specific information, which may not be the same across different users.

Additional Savings

In section 3.3.2 we proposed an RE module, that operates on three "sublayers"⁶. The first sublayer is responsible to receive the byte stream from the TCP buffers and perform the corresponding functionalities. The second sublayer is the RE module itself, viz. the fingerprint, indexing and lookup. The last sublayer is the storage in the shared memory and disk. Modifying any of these sublayers does not affect the other ones. For this reason, we have added an additional sublayer parallel to the first one. In this sublayer, we have implemented a packet based processing. Effectively the modified system is able to perform chunk based caching per object and per packet, at the same time.

We performed this step to showcase that additional savings can be attained from processing UDP and other than web TCP traffic. Since UDP traffic is connectionless, only a packet based RE could be used. In section 3.3.3, we showed that there is around 15% of the traffic that is unknown, and most of it is UDP. A proxy cache would not process those data. Hence we modified the scheduler to send all not-HTTP traffic to the modified first sublayer. Our results

⁵Infinite size cache is a cache whose storage space is higher than the size of the trace, therefore no content replacing is taking place.

⁶We define them as "sublayers" as layers usually refer to the OSI layering system.

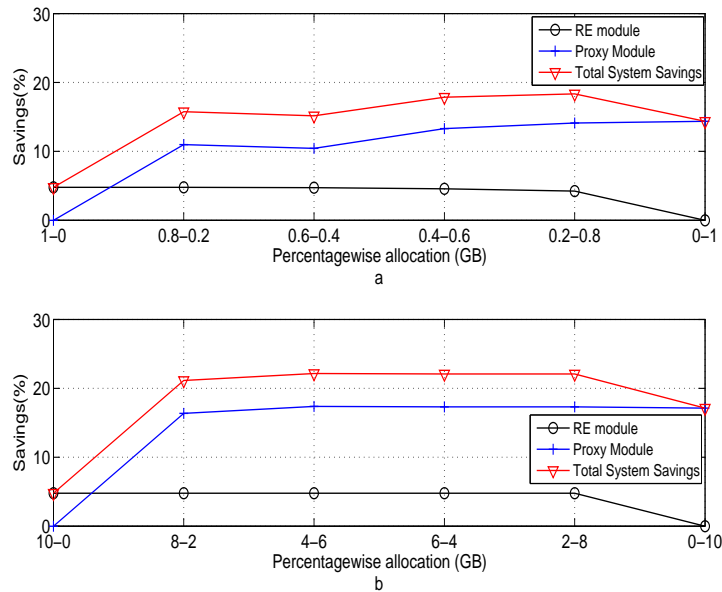


Figure 3.23: Disk Capacity Allocation for each module for an 1GB (a) and 10GB (b) hybrid system (RE module size - Proxy module size)

indicate that such a sublayer will add 2% in savings for trace A and 2.6% in trace B. Since these savings are only going to be apparent in an RE implementation, the difference in the savings between the hybrid cache and the proxy cache would increase to 10%.

Finally, we need to note that such an additional sublayer may potentially create other issues, e.g. system complexity, performance of RE module in encrypted (e.g. SSL traffic etc.). Moreover, one can find several system optimizations that would provide a more efficient memory or disk space utilization. For this reason, we plan to extend our emulator to a prototyped system.

3.4 Related Work

In the last decade, as the dominant traffic shifted from the Web to P2P protocols, new protocol independent redundancy elimination techniques emerged [88, 23]. Due to the increasing demand for WAN bandwidth optimization and given their deployment limitation compared to Web caching, these new traffic deduplication techniques became popular mainly in enterprise environments [35, 11, 9]. A quantitative study of the benefits of redundancy elimination techniques on wireline enterprise and university traces can be found in [26].

Unfortunately, apart from the analysis in [63], little is known about the characteristics of traffic generated by mobile handheld devices, and the impact that traffic deduplication tech-

niques have on mobile workloads. This is specifically important since: a) it is well known that wireless cellular networks have recently become extremely bandwidth constrained both at the wireless and the backhaul links of their radio access networks (RANs), b) the traffic mix both in the Internet and especially in wireless cellular networks is shifting back to HTTP, mainly due to the large growth of video traffic transported over HTTP seen in recent years, and which is expected to continue in years ahead [20].

While both object caching and byte caching have been studied before, there has been no direct in depth comparison between them, and especially for traffic mixes that are becoming prominent in wireless cellular networks.

A byte cache is a redundancy elimination scheme that identifies regions of redundant content, indexed by Rabin fingerprint [81]. The first paper that mentioned the idea of protocol-independent deduplication in computer networks was [88] and showed that potential savings can be close to 20%. A byte cache can potentially increase the savings compared to Web object cache, because it identifies redundant content at the byte level. However, it is less flexible in terms of deployment because it requires pairs of devices (encoding and decoding) to function, and it is more computationally expensive, as deduplication happens at the granularity of packets rather than flows.

Anand et al. [26] have improved the efficiency of byte caching by incorporating a more effective selection of fingerprints and performed an analysis based on enterprise and university traces. In earlier work the authors have implemented a similar redundancy elimination at the router level [25] and network wide [27]. Martynov [64] has shown that a byte cache might provide savings in a corporate network, but a practical approach may create a bottleneck itself. Other studies have shown that protocol independent deduplication can be effective in the developing world, where bandwidth is major issue [56].

In [23] two approaches are investigated for end devices, "Chunk-Match" which identifies chunks that repeat in full across data blocks, and "Max-Match" which identifies maximal matches around fingerprints that are repeated across blocks. However, the authors do not investigate the cases, in which a unique hash is generated for multiple and continuous matching chunks (multi-resolution chunking as named in [56]). In [56], the authors incorporate the above, but perform the "Chunk-Match" at the flow level (implemented as part of a Web-proxy), where chunks are bigger compared to chunks produced at the packet level.

In our work, we incorporate the above functionalities, and devise an efficient methodology for encoding based on *location-length* pair. In addition, our goal in this work is to perform a direct comparison of an optimal object cache against both types of byte cache. Finally, we perform the first in-depth investigation of the effectiveness of traffic deduplication techniques at the HTTP content-type level, enabling a further understanding of what types of Web-applications (e.g. video, images, dynamic content, etc) are more amenable to the different traffic deduplication

techniques.

3.5 Conclusion

A distinct feature of the wireless traces compared to wireline is that their major component is HTTP traffic. Thus, it was not clear whether a protocol independent data deduplication technique would outperform a Web cache. In this chapter, we have thoroughly investigated the benefits of the implementation of a protocol independent data deduplication middleware in a wireless network. Based on the perceived savings and comparison analysis with a Web object cache, we claim that a byte-level cache would be beneficial if implemented adjacent to the Access Point or Base Station. In fact the savings from a data deduplication technique at the byte level, would be more from the HTTP content itself, as well as from other applications. More specifically, our major findings can be summarized as follows:

- **Byte Cache:** The savings from a protocol independent byte cache range between 10% – 20%. We showed that such savings are much larger than is the case for a Web object cache. A byte cache not only outperforms a Web cache for HTTP content, but also provide an extra 4% savings from other protocols that a Web cache would not be able to capture.
- **Max-Match vs Chunk-Match:** From the two byte cache approaches the first one has proven to be more effective in finding redundant content, but in expense of resources. In fact the Chunk-Match approach would require almost 10 times less memory space to store all the hashes.
- **Hybrid Cache:** A cache that uses both object and chunk based deduplication techniques can provide two times more savings than a standalone proxy cache, and better savings than standalone byte cache. It requires three times less memory storage, and provides a speedup equal to 3 compared to standalone byte cache.

Chapter 4

Network Design of Aggregation Architectures

The well established 80/20 rule for client-server versus local traffic has driven network designers to address problems with pure L2 switches acting as hubs, and L3 routers performing the routing towards different paths. However, the rise of on demand video streaming (e.g., Netflix, Hulu), Peer-to-Peer (P2P) television over the internet (e.g., TVUPlayer, PPLive, QQLive, PP-Stream), and the advent of mobile traffic has triggered a number of actions from broadband providers who are trying to achieve flexibility, scalability and efficiency [68]. According to several studies, the annual global traffic will be doubling in volume every two years, while a quarter of this is projected to be video traffic [21, 20]. Current 4G mobile service provider architectures are bandwidth constrained, and the cost to build out new networks to increase the available bandwidth is prohibitively expensive.

Therefore, it is of vital importance for the Internet Service Providers (ISPs) to engineer their infrastructure to meet these challenges. In this paper, we focus on answering the question of *where to place certain functionalities* in the aggregation network. We model this through a Network Design Problem (NDP) that the SP may use to achieve the maximum efficiency with the minimum deployment capital cost.

In the past, NDPs have been used to address the problem of cost optimization when developing *new* infrastructures. Nonetheless, most ISPs already possess an access infrastructure. In order to address this issue, we propose an optimization formulation that uses the current infrastructure (in terms of design), but defines the optimal distribution of *functionalities* instead. Another important aspect is that devices used in the Ethernet aggregation, and edge part of the network, are not anymore purely layer 2 switches (in the NDP literature they are sometimes called “aggregators” [79]) and layer 3 routers. Most of the vendors have engineered multipurpose edge “systems” that incorporate different functionalities and network intelligence in modular “sub-systems” e.g., high-end backplanes in which multiple interface and functionality line cards are added. Therefore, design problems for such architectures need to be revisited.

Considering the above challenges, in this paper we classify the aggregation architectures based on the location of the functionalities and network intelligence, and then proceed to build a modular optimization model that is able to achieve the minimum deployment cost. Our contribution is summarized as follows:

- We define the possible aggregation architectures and group them into three main categories: 1. *Centralized Single-Edge*; 2. *Centralized Multi-Edge*; and 3. *Distributed Single-Edge*. On top of these categories, another subcategory is added, whether to cluster the edge systems or keep the functionalities in a single box.
- We develop a network design model that goes beyond the well investigated *location* and *dimensioning* problems. Our modeling approach is applied to both design an aggregation architecture, as well as upgrading the current infrastructure.
- We propose models based on edge “systems”, rather than network elements. Edge systems may support different types of functionalities either on their own, or as attached “sub-systems” (line cards).
- We formulate constraints that account for physical characteristics (line card capacity, port capacity), bandwidth (device and port bandwidth), and layer 2 versus layer 3 functionalities (such as switching and routing, VLAN/IP termination capacity, or business functions).
- We propose two novel heuristics for scaling down the problem and for decreasing the execution time of the problem.
- We evaluate our model with two close-to-real-case scenarios and with multiple traffic profiles. We show that, with the current trends, the SPs will need to re-engineer their aggregation infrastructure.

The paper is structured as follows: In the next section, we provide an overview of the current state of the art in NDPs and aggregation design methodologies. In section III, we define and explain the architectures according to the distribution of the edge systems. In section IV, we describe the proposed modeling methodology and assumptions. Given the plurality of symbols and notations, all of them are summarized in the tables of section IV. In section V, we explain the modeling approach for the traffic flows and in Section VI we deploy the cost optimization model. In section VII, we explain the heuristics that have been used to scale-down the problem, and in section VIII we showcase an evaluation of two multi-service scenarios over metro area networks based on architectural values from EU ISPs and actual system values from vendors. Section IX, includes the discussion and further remarks, and we conclude with the final section.

4.1 Architectural Comparison

Since each edge system (BNG, MSE, Video BNG) may support differential and modular functionalities, various topologies may be implemented even when the edge router’s backplane re-

mains the same. For example a BNG (into a single chassis) that incorporates both IP termination and video functionalities. Another approach that is followed by some vendors is to add several L3 functionalities on L2 devices (e.g., IP termination) [29]. We propose the following architectural combinations for an ISP aggregation network: Centralized or Distributed based on the intelligence; Single or Multi edge based on the services per location; Clustered and Unclustered based on the functionalities.

4.1.1 Network Elements in an Aggregation Network

L2 Aggregation devices: are high capacity L2 aggregation switches used as a second level of aggregation, and perform low cost VLAN policy enforcement. They usually support Gigabit Ethernet ports and multiple filtering polices per VLAN. They are the simplest aggregation devices.

BNG: Broadband Network Gateways are IP devices terminating the layer 2 access and route over IP/MPLS with support of a full set of MPLS and IP routing protocols, including multicast routing (PIM-SM/IGMP [45]). They enforce sophisticated IP QoS per service and per-content/source differentiation. They usually terminate PPP sessions or IP tunnels and can support up to hundred of thousands of subscribers and Gbps capacities. Edge routers, according to [19], can support additional functionalities related to Authentication, Authorization and Accounting (AAA) and are dynamic devices that mainly focus on residential customers. Moreover several next generation BNGs have the capability to support mobile blades to provide mobile termination functionalities.

MSE: Multi-Service Edge Routers are responsible for routing business traffic, which is usually dedicated bandwidth. There is no need to authenticate the business users as these are leased lines, hence MSEs support less intelligence.

Video BNG: Video Broadband Network Gateways are dedicated routers that have been introduced to handle the increasing demand for video traffic (e.g. IPTV, Netflix etc.). A video BNG does not implement subscriber management functions (e.g., PPP termination, per user QoS), since these functions are likely to be performed by the other network elements. In fact the video BNG may be the point of insertion of an IPTV flow and/or the replication point (L3 Multicast).

4.1.2 Centralized Single-Edge Architecture

This type of design was very important in the early evolution of aggregation networks. In this type of architecture, the L2 Metro Ethernet aggregates the traffic from multiple access points and delivers the Virtual Local Area Network (VLAN) to the IP Edge network, as shown on Fig. 4.1. Some of the characteristics of this architecture are: 1) all types of traffic are backhauled to the BNGs and then to a single P-Router location, which is connected to the

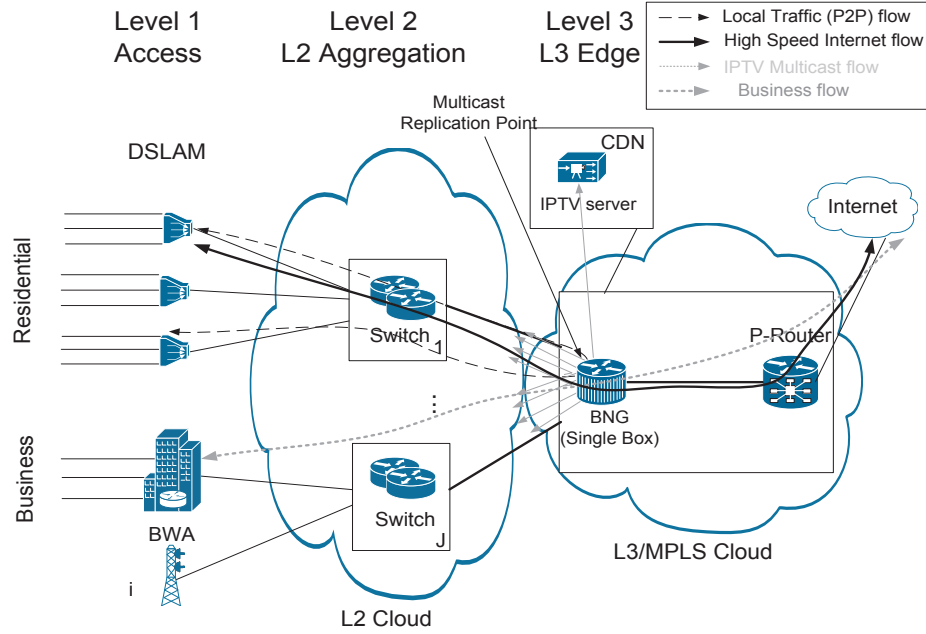


Figure 4.1: Centralized Single Edge Overlay Architecture

ISP backbone (P-Router is part of the backbone and sometimes called PoP); 2) Subscriber termination functionality, multicast replication and IP QoS policies are executed in the BNG deeper in the network; 3) Multicast traffic for broadcast video is transmitted from the edge router over L2 multicast VLANs to all customer premises (Wireless BS, DSLAM).

4.1.3 Centralized Multi-Edge Architecture

In the multi-edge architecture there are different types of edge routers (BNG, Video BNG and MSE) that handle separate classes of subscriber and different types of traffic, as shown in Fig. 4.2. Residential subscribers are usually terminated in the BNG, whereas the Business VLANs or leased lines are routed through the MSE. Some providers, in order to enforce specific QoS policies on their video channels (IPTV), implement a separate 'Video BNG' [19]. Therefore, the Centralized Multi-Edge architecture benefits from incumbency, since it is easier to evolve from the existing architecture.

4.1.4 Distributed Single-Edge Design

A distributed IP Edge approach is being considered by many SPs as an alternative architecture to satisfy the bandwidth requirements for future applications. As shown in Fig. 4.3, the edge network is comprised by both L2/L3 routers. Video and HSI are backhauled over separate

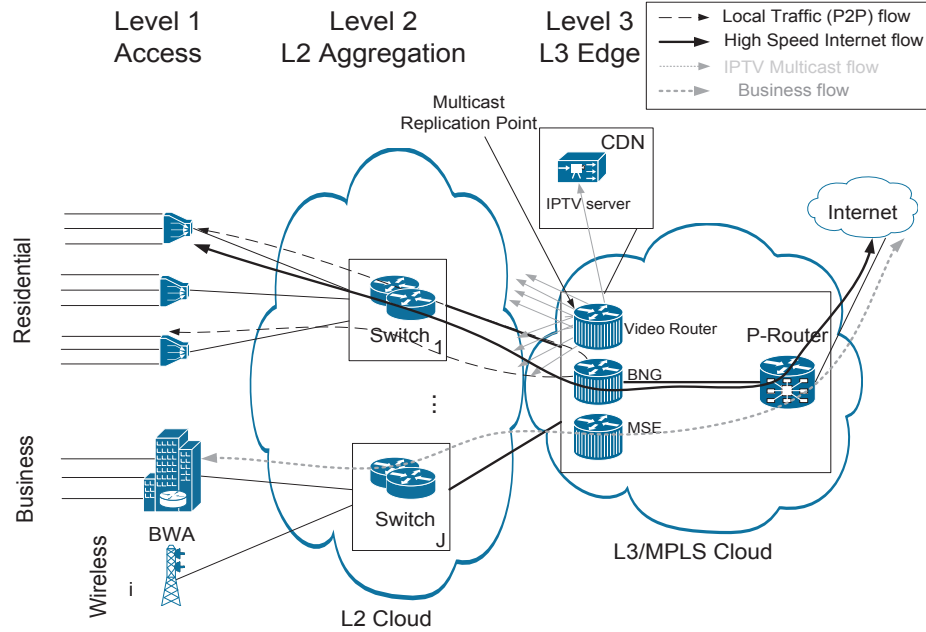


Figure 4.2: Centralized Multi Edge Overlay Architecture

VLANs to the Edge Routers, where services and access to the IP network is controlled. The scalability is increased, since the amount of state information in the BNG is decreased (less subscribers are terminated per BNG) and IP QoS policies are enforced closer to the last mile. IP multicast routing is used across the L2/L3 Carrier Ethernet network for delivery of broadcast video services. In the single edge case, all services flow through a single device distributed closer to the subscriber.

Furthermore, based on the allocation of subscribers the aforementioned architectures can be further divided into:

- *Clustered*: Allocating the subscribers to a particular service over many systems located in the same PoP.
- *Unclustered*: Allocating all subscribers for a particular service to one system.

With the current centralized design, the increasing demand of video channels over IP networks leads to unavoidable bandwidth problem in the aggregation networks. Distributing the replication functionalities in multiple aggregation levels may result in less congestion of traffic bandwidth and VLANs, since a single unicast flow is required from the CDN to the distributed Video BNG. Moreover, the evolution of P2P IPTV sets new standards on how users interact.

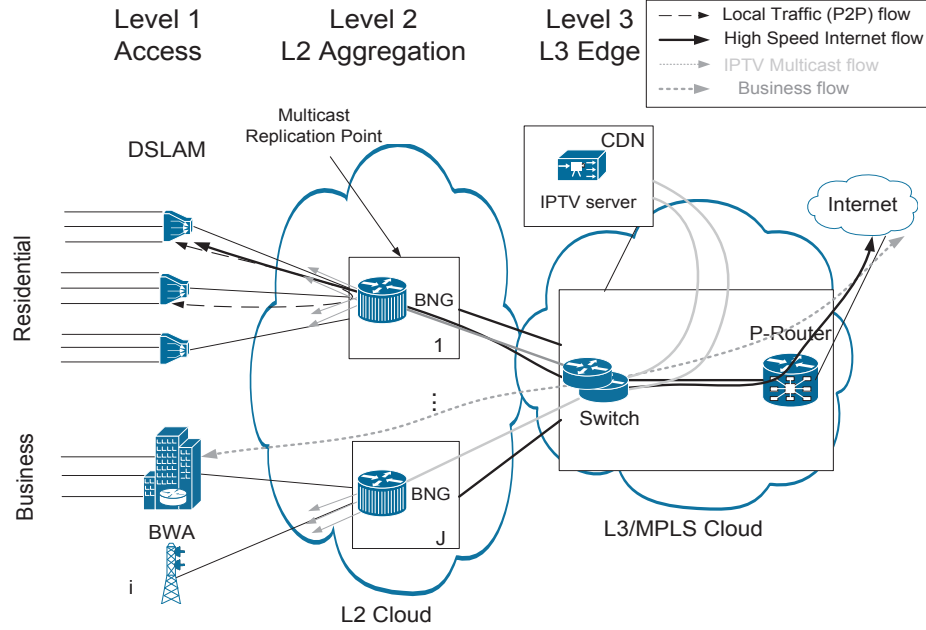


Figure 4.3: Distributed Single Edge Overlay Architecture

Distributing the IP intelligence can deal more effectively with traffic flows that tend to be “local”, since traffic is terminated and routed closer to the subscribers. In addition, several wireless providers are already facing issues on how to backhaul the increasing demand of mobile traffic. For this reason our work focuses on determining which architecture is the optimal one.

Finally, the distributed multi-edge architecture has been abandoned due to the increasing complexity of distributing multiple type of boxes over the aggregation network. Hence, we exclude it from the potential solutions.

4.2 Methodology and Assumptions

4.2.1 Overview

The main objective of our work is to create a single modular and portable model that is able to provide the cost optimal solution among different architectural candidates. We model this through a mixed integer programming model. We divide the variables in qualitative (0-1) and quantitative. The qualitative variables are used to select the appropriate architectural approach (Centralized vs Distributed), whereas the quantitative to determine the number of systems per location. The model is a non linear mixed integer; the non-linearity appears in the constraints. Such problems can be a challenging and daring venture, because they combine all difficulties

of their subclasses: the combinatorial nature of integer programs (MIP) and the difficulty in solving non-convex nonlinear programs (NLP). Hence, we have transformed the problem to a linear one in the expense of extra constraints. In addition, we decreased the search space by defining upper bounds for the number of elements and interfaces. By using these heuristics the optimization problem was solvable in a finite amount of time.

In this model, we assume a tree topology (bipartite) with multiple aggregation levels. In fact, most of the ISPs usually implement small and medium aggregation sites and several core sites. The number of locations varies based on the size of the ISP. Small locations may vary from 10-10,000, the medium locations from 10-1,000 and the core location from 1-100. Every core site may support different kinds of areas (rural or non-rural) with various customer distributions. Thus, we are modeling a single three-level aggregation network. The results can then be extrapolated to include the whole ISP aggregation infrastructure (multiple aggregation networks). In other words, solving the problem for the whole ISP network would require running the model multiple times for each aggregation network with different input parameters (e.g., subscribers, levels of aggregation).

4.2.2 Definitions

On the first level there may exist A access locations. In each access location $a = [1, \dots, A]$, the L2 access devices are able to handle a finite number of subscribers. On the remaining levels the model is going to define how to distribute the higher layer functionalities and how to handle different kinds of service flows.

Architecture

The first aggregation level is comprised of I locations ($i = [1, \dots, I]$), the second level of J locations ($j = [1, \dots, J]$) and the last level of $K = 1$ single location (single core site since we are modeling one aggregation area). The core site incorporates the P-Router which is responsible to route the non local traffic to the backbone or traffic from Tier-2/3 SPs (LAC/LNS functionality). However, it was not included as a variable in the model as it does not play any role on the functionality distribution. The insertion point of the video traffic from the CDN is assumed to be the core location. From the results, we noticed that if the insertion point was the Video BNG, the solution would have a very small variation.

Links

In order to model the links between each aggregation level, we have used boolean constants $u'_{n-1,n}$ for every aggregation level $n \in \{i, j, k\}$. If $u'_{n-1,n} = 1$, then a lower level location $n - 1$ is connected with a higher level location n . However, it may be the case that the problem

identifies that less aggregation levels are needed. Thus, in a similar manner $u'_{n-1,n+1}$ represents the connection with links between two different layers.

Parameters

Table 4.1 summarizes the parameters of the model. Our model is based on a network that has at most three aggregation levels. Our approach is able to determine, whether all three are needed or the ISP may decide to implement fewer levels. In the same table, we also present the properties of each of the devices. Note that edge systems have the subscript t , which corresponds to different types. In regards to the port number, we make the following simplification: Edge system backplanes contain a finite amount of slots. In each slot, line cards are added that have a finite number of ports. Hence, instead of having so many variables, we have decided to associate the number of ports per device as the one to vary P_t^{L3} . Each port is also associated with a cost $co^{L3,c}$.

Edge System Functionality

We also assume six different types of edge routers $t \in T = \{A, B, C, D, E, F\}$, based on the functionality, the characteristics and the services that they support. We define the following edge system functionalities:

- High Speed Internet (HSI): $T_1 = \{A, B, C, D\} \subset T$ because they require IP termination;
- Business: $T_2 = \{A, C, E\} \subset T$ for business subscribers; since they lease VLANs they do not require any termination.
- Video replication (Multicast): $T_3 = \{A, B, F\} \subset T$.

Table 4.2 summarizes the types, the functionalities, the edge design that they form, and the potential combinations with other edge systems at the same location. More specifically, in order to avoid redundancy of functionalities and therefore unnecessary cost, not all types of edge routers should be used in (a) a single location and (b) in any path from access node to P-router. For instance, if a BNG (type A) is used in a location i , then it could perform everything in a single box, thus no other BNG should be installed in the same path (or location) towards the P-Router.

4.2.3 Variables

The objective function of the optimization problems is, given the appropriate locations, corresponding links and traffic demands, to optimally allocate the network elements and interfaces,

Table 4.1: Modeling Input Parameters

	Description	Symbol
General	Access Location	$a = [1, \dots, A]$
	1st Level Aggregation Locations	$i = [1, \dots, I]$
	2nd Level Aggregation Locations	$j = [1, \dots, J]$
	Core Locations	$k = [1, \dots, K]$
	Number of subscribers	SUB
	Subscribers per Residential Location a	res_a
	Subscribers per Business Location a	bus_a
	DSLAM subscriber capacity a	S_a
	Number of IPTV Channels from CDN	Ch
	% of concurrent subscribers watching IPTV	w_i
Capacity of a Port	$c = \{1G, 10G\}$	
Agg. Switch	Cost (\$)	co^{L2}
	Bandwidth (Gbps)	C^{L2}
	VLAN capacity	$vlan$
	Number of slots for Ports	$P^{L2,c}$
	Cost (\$) of switching ports	$co^{L2,c}$
Edge Systems	Cost (\$) for Type t	co^{L3t}
	Bandwidth (Gbps) for Type t	C_t
	IP Termination Capacity for Type t	sub_t^{L3}
	Number of slots for Ports for Type t	$P_t^{L3,c}$
	Cost (\$) of routing ports	$co^{L3,c}$

Table 4.2: Types of Edge Functionalities t

Type	Functionality	Edge Design	Combinations
A	All	Single Edge	None
B	HSI and Video	Single/Multi Edge	only type E
C	HSI and Business	Single/Multi Edge	only type F
D	HSI	Single/Multi Edge	type E,F
E	Business (MSE)	Multi Edge	type D,F or type B
F	Video	Multi Edge	type D,E or type C

and identify where to place the routing functionalities. For this, there are two different types of variables, qualitative and quantitative¹:

Qualitative Variables (0-1)

- $u_{n,t}^{L3}$ is a boolean variable that specifies which type of system should be used at location n . If the edge router of type t is chosen at location n , then it is equal to 1; otherwise it is 0.
- u_n^{L2} is a boolean variable that specifies if a switch is going to be placed at location n or not. If it is, then it takes the value 1; otherwise it is 0.
- u_n^0 is a boolean that takes the value 1, if no device is installed at all. That is $u_{n,t}^{L3} = 0, \forall t \in T$ and $u_n^{L2} = 0$. Evidently $u_{n,t}^{L3} + u_n^{L2} + u_n^0 = 1$.
- u_n^{1G} is a boolean variable that takes the value 1, if 1Ge interfaces (line cards) are chosen to be installed at location n ; otherwise it is 0.
- u_n^{10G} is a boolean variable that takes the value 1, if 10Ge interfaces (line cards) are chosen to be installed at location n ; otherwise it is 0.

Thus if $u_{n,t}^{L3} = 0$, then either a switch or another type of edge system or nothing is installed. If $u_{k,t}^{L3} = 1$ for any $\forall t \in T$, then the optimal architecture is the centralized one (according to the definition of the previous section). Whereas, if $u_{n,t}^{L3} = 1$, for $n \neq k$, the optimal architecture is the distributed. For the centralized case, if $t = A$, then the programming model will have selected the single-edge architecture.

Quantitative Variables

As quantitative variables we define those variables that are natural numbers, \mathbb{N}_0 . In this set there can be: "real" variables, those associated with a specific characteristic and affect the objective function; "dummy" variables, those that are used for other purposes and are not part of the objective function. The latter ones are used in the locations that the provider does not need to open or use. More specifically:

- $Y_{n,n+1}^{x,c,up}$ and $Y_{n-1,n}^{x,c,dn}$ are the integer variables specifying the number of interfaces of capacity c attached to the network element of layer x and located in location n . Those interfaces are attached to links that connect, either the uplink with one level higher ($n + 1$), or the downlink with one level lower ($n - 1$).

¹The variables are without an accent. Lower-case u is used for qualitative variables that take values 0 or 1, and upper-case Y is used for quantitative variables that are natural numbers

Table 4.3: Volumes of traffic

Traffic Volume per subscriber	
HSI traffic (Mbps) for residential customers	HSI_a^{res}
HSI traffic (Mbps) for business customers	HSI_a^{bus}
Bidirectional P2P (Local traffic) in Mbps	$P2P_a$
IPTV Bandwidth (Mbps) of each channel ch	$iptv_{ch}$
Internet (Non local Traffic) flowing	
through the location n	x_n^{HSI}
IPTV Traffic flowing	
through the location n	x_n^{IPTV}
from the CDN to the first IP replication point	x_n^{CDN}
P2P (Local Traffic) flowing	
through the location n	x_n^{P2P}
from one level below	$x_{n,dn,in}^{P2P}$
from one level above	$x_{n,up,in}^{P2P}$
to one level below	$x_{n,dn,out}^{P2P}$
to one level above	$x_{n,up,out}^{P2P}$

- $Y_{n,t}^x$ is the integer variable specifying the number of network elements x at location n and of type t . If $Y_{n,t}^{L3} > 1$ then it means that the location n needs its routing functionalities to be clustered.
- Y_n^0 is the integer dummy variable that is used whenever neither routing nor switching functionality is installed in location n . If all the n locations are in the same level have $Y_n^0 > 0$, then this means that it is cost optimal to have less aggregation levels.
- $Y_{n,n+1,dn}^0$ is the integer dummy variable that is used to depict a fictional number of interfaces, whenever no device is determined to have been installed at the $n + 1$ location, $Y_{n+1}^0 = 0$.

4.3 Modeling the Traffic Flows

In our modeling approach, we assume that the beginning of the traffic is the first aggregation point, or else any access location a . This was done for several reasons: a) the number of subscribers may range to millions, making the problem hard to be solved with that amount of variables and constants; b) network architects usually use mean (or some percentage) traffic demands over subscribers of a bigger geographical area [20, 62]; c) according to the VLAN policy implemented, traffic subscriber VLANs are bundled into a bigger VLAN per service from the access location (e.g. DLSAM in wired and nodeB in 4G wireless) until the edge location

(BNG in ethernet aggregation or RNC in 4G wireless); d) the service providers do not tend to incorporate L3 type capabilities and management functionalities that close to the subscribers. For example HSI_a^{res} (on table 4.3 is calculated by taking into account the harmonic mean of the bandwidth consumed per subscriber that is served by the access location a . However, an ISP may properly configure the per subscriber usage in order to assume a worst case (or some percentile) and possibly overprovision the network [66]. In the evaluation, we are investigating a variety of network throughput and showcase the effect of the traffic demand into the optimal solution.

The traffic classes and service types are generally available to the ISPs, since BNGs incorporate DPI or flow classification functionalities [34]. For modeling reasons, we have grouped the traffic flows into three main classes, i.e. IPTV, Internet (Non Local) and P2P (Local). The traffic demands are shown on Table 4.3.

We categorize the traffic flows according to the effect (utilization, direction etc.) that they have on each aggregation level $n \in \{i, j, k\}$. Hence we define three main categories of traffic class:

4.3.1 Internet (non local) Traffic Class

Internet traffic class is any type of residential or business traffic that will flow through all the levels of the aggregation network. It is usually a client-server type of traffic and its major portion is usually web traffic [62]. In terms of network optimization, this type of traffic satisfies the conservation theorem, unless redundancy elimination devices are included in any level of the network [70]. The Internet traffic per aggregation level is represented by the following equations.

$$\begin{aligned}
 x_i^{HSI} &= \sum_a u'_{a,i} (x_a^{res} + x_a^{bus}) \\
 x_j^{HSI} &= \sum_i u'_{i,j} x_i^{HSI} \\
 x_k^{HSI} &= \sum_a (x_a^{res} + x_a^{bus})
 \end{aligned} \tag{4.1}$$

The non local traffic that flows through an access location a is based on the number of subscribers and the corresponding network usage per subscriber. Thus $x_a^{res} = res_a \cdot HSI_a^{res}$ and $x_a^{bus} = bus_a \cdot HSI_a^{bus}$, indicate the bundled traffic that arrives in an access location, for residential and business subscribers respectively. The reason that we use separate volumes for each type of subscriber is because the network usage is different and their connections towards the “edge systems” are different.

4.3.2 P2P (Local) Traffic Class

P2P traffic class includes those applications that exhibit locality in their behavior. Those traffic flows are distributed across the aggregation network over several directions and may either remain local in the same geographical area (e.g., P2P or a business unit that has a local server), or in the same routing domain, and depend on how peers are interconnected. In P2P systems the files are split in smaller chunks and are downloaded from various sources. The system is therefore initiating various flows to various users. The P2P flow will have different performance based on the placement of the routing functionalities. Let us assume that users in the same access location are downloading P2P traffic with a certain throughput $P2P_a$ Mbps from sources that are either in the same edge network or outside the aggregation area. Therefore, the total P2P traffic that flows through an access location is $x_a^{P2P} = sub_a P2P_a$.

The distribution of functionalities is also affected by the overlay P2P network. If the user is downloading a file from peers in the same first aggregation level then we denote this probability with p_1 . Similarly if the two peers coexist at the same second aggregation level location, their probability is p_2 , and p_3 for the third aggregation level location. Since the P2P files are usually divided in chunks, the above probabilities can be regarded as equal to the portion of traffic that is coming from users at the same aggregation level. Practically those probabilities are related to the direction of the traffic, and can be calculated by the ISP through performing flow identification (such functionality can be found in both L2/L3 devices). Equivalently for the flow that goes outside of the network $1 - \sum_{l=1}^3 p_l$. The question now is how this probability may affect the cost of the architecture.

The following equations show how the P2P traffic flows over the networks. Effectively the first equations bundles the P2P traffic into a VLAN that will be offered to the first aggregation layer i . The existence of a router of type t at any level k , as defined by the variable $u_{k,t}^{L3}$ will affect the direction of the flow. For example, if a router that performs IP termination ($t \in T_1$) is placed at the first aggregation layer ($u_{i,t}^{L3} = 1$), all traffic that would remain local at the first aggregation layer would not have to flow through the rest of the layers. Similarly for the rest of layers

$$\begin{aligned}
 x_i^{P2P} &= \sum_a u'_{a,i} x_a^{P2P} \\
 x_j^{P2P} &= \sum_{t \in T_1} [u_{k,t}^{L3} x_k^{P2P} + \sum_i u'_{i,j} x_i^{P2P} (u_{j,t}^{L3} + u_{i,t}^{L3} \sum_{k=2}^3 p_k)] \\
 x_k^{P2P} &= \{(1 - \sum_{t \in T_1} u_{k,t}^{L3})(1 - \sum_{k=1}^2 p_k) + \sum_{t \in T_1} u_{k,t}^{L3}\} \sum_j x_j^{P2P}
 \end{aligned} \tag{4.2}$$

4.3.3 Multicast Traffic Class

Multicast is required in order to make efficient use of network resources when delivering broadcast content. Mainly, the desired goal is to support multicast optimization by controlling the flooding of Ethernet multicast frames. The IGMP/PIM-SM agents can locally adjust replication filters on the device, such that packets are replicated only on those ports that have specifically requested to be part of the multicast group [45]. However, the location of the multicast functionality (either as IGMP snooping on L2 devices, or as IP multicast on L3 devices) is based on the VLAN architecture. In the following, we assume that the VLAN policy implemented by the ISP requires the replication to happen in the edge system.

A unicast IPTV traffic flow is assumed to be initiated from the CDN(s) and is replicated at the Edge Router, where multicast is implemented. The streaming bit rate is usually between $iptv_{ch}^{SD} = 1\text{Mbps}$ for Standard Definition (SD) and $iptv_{ch}^{HD} = 10\text{Mbps}$ for HD channels and the user is assumed to choose either the channel in one definition or in the other [87]. Several studies have shown that the selection of IPTV channels $ch \in \{1, \dots, Ch\}$, follows a Zipf Law distribution $p_{ch} = \alpha/ch$, given that the channels are arranged by channel popularity [91] (α is a constant). It is also assumed that the percentage of residential users per a access locations (e.g., DSLAM, Wireless Base station) that have selected the IPTV service is w_a .

In order to calculate the required bandwidth for the unicast transmission of IPTV channels, from the CDN to the first replication point, the number of requested channels, from the residential subscriber population, needs to be determined. The probability that at least one user is watching a channel c , and belongs at the subtree formed by the location n , can be determined as $1 - P\{\text{none watching channel } ch\}(t) = 1 - (1 - p_{ch})^{\sum_a w_a \cdot res_a \cdot u'_{a,n}}$. Therefore, regarding the unicast flow from the CDN to the first IP replication point, the bandwidth can be defined by the following equation

$$x_n^{CDN} = \frac{iptv_{ch}^{HD} + iptv_{ch}^{SD}}{2} \cdot \sum_{c=1}^{Ch} 1 - (1 - p_c)^{\sum_a w_a \cdot res_a \cdot u'_{a,n}} \quad (4.3)$$

After the flow reaches the Video router all flows are distributed to the subscribers through multicast. However, the distribution of video routing functionality plays a role on the distribution of IPTV flows over the network. Note that if a multicast routing functionality has been determined to exist in that location, then the flow conservation theorem does not hold (since the video router will replicate the IPTV flow to the corresponding IPTV VLANs). For example, if a single BNG is placed in the first aggregation level, then the multicast is going to happen closer to the subscribers. Therefore the IPTV flow that every access location is demanding is $x_a^{IPTV} = w_a \cdot res_a \cdot IPTV_{ch}$ (percentage of users w_a from the population of sub_a connected to

access location a), then the flows are going to be distributed in the network as follows

$$\begin{aligned}
x_i^{IPTV} &= \sum_a u'_{a,i} x_a^{IPTV} \\
x_j^{IPTV} &= \sum_i u'_{i,j} \left[\sum_{t \in T_3} u_{i,t}^{L3} x_j^{CDN} + \left(1 - \sum_{t \in T_3} u_{i,t}^{L3}\right) x_i^{IPTV} \right] \\
x_k^{IPTV} &= \sum_a x_a^{IPTV} \sum_{t \in T_3} u_{k,t}^{L3} + x_k^{CDN} \left(1 - \sum_{t \in T_3} u_{k,t}^{L3}\right)
\end{aligned} \tag{4.4}$$

For the first definition of the above equation, x_i^{IPTV} is equal to the bandwidth that all IPTV VLANs generate. For the x_j^{IPTV} , there are two possibilities. If a type T_3 router is placed at any of the dependent first aggregation levels ($u_{i,t}^{L3} = 1$), then the bandwidth of the IPTV flows is equal to $\sum_i x_i^{IPTV}$. If this is not the case, then the bandwidth is equal to the eq. (1). The third equation takes into account the property $\sum_{t \in T_3} (u_{i,t}^{L3} + u_{j,t}^{L3} + u_{k,t}^{L3})$. Thus, if a router that includes IPTV functionality is included at the central location k , then the IPTV bandwidth is $\sum_a x_a^{IPTV}$. Otherwise it means that an IPTV capable router has been placed in either the 1st or the 2nd aggregation level and the bandwidth for the flow is $x_k^{CDN} \simeq \frac{iptv_{ch}^{HD} + iptv_{ch}^{SD}}{2}$. From the above, it is readily seen that if users tend to watch similar programs, less bandwidth is used, decreasing the infrastructure cost. In addition, integrating the multicast functionality (i.e. $t \in T_3$), closer to the access network, decreases the bandwidth too.

Therefore the total amount of traffic that arrives in every aggregation location is $x_n = x_n^{IPTV} + x_n^{P2P} + x_n^{HSI}$.

4.4 Edge Design Model

The objective function of the optimization problem is to determine the optimum deployment cost by optimally distributing the functionalities over the aggregation network

$$\min \sum_{n \in \{i,j,k\}} \left[\sum_c \sum_x (co^{x,c} Y_n^{x,c}) + \sum_t co_t^{L3} Y_{n,t}^x + co^{L2} Y_n^{L2} \right] \tag{4.5}$$

The above objective function includes the cost of the interfaces based on the capacity $c = [1G, 10G]$, that will be implemented in a node of layer $x = [L2, L3]$ at location n , as well as the cost of the network elements that will be placed at the same location.

4.4.1 Constraints

Non Linear to Linear Transformation

The problem has two sets of variables per location, $Y_{n,t}^{L3}$ and $u_{n,t}^{L3}$. The multiplication of those variables results in a non linear problem. Hence, we try to transform the model from a non-linear to a linear, by adding some extra constraints along with two big constants $BIGD_n$ (for the devices) and $BIGI_n$ (for the interfaces). For example, devices per location may range between 10-100, therefore $BIGD_n$ can be as high as 1000. Any value higher than that may result in an unnecessary increase of the search space. In the next section, we provide a heuristic to determine an approximate upper bound. The extra constraints are expressed as follows:

$$\begin{aligned}
 & Y_{n,t}^{L3} \leq BIGD_n \\
 Y_n^{L2} \leq & BIGD_n \left(1 - \sum_{t \in T_1} u_{n,t}^{L3}\right), n \in \{i, j, k\} \\
 & \{t \in T_1 | n = i, j\} \text{ or } \{t \in T | n = k\}
 \end{aligned} \tag{4.6}$$

The above constraints specify that (a) the $Y_{n,t}^{L3}$ must be bigger than a big constant, and (b) in every location n there can be only an aggregation switch or an edge system. In terms of functionalities, the two sets define that for the first and second aggregation levels $n = \{i, j\}$, only IP termination functionalities can be distributed ($t \in T_1$). The last set effectively excludes the multi-edge architecture from the possible solutions. We follow a similar approach for the interfaces.

$$\begin{aligned}
 & Y_n^{L3,c} \leq BIGI_n \sum_{t \in T_1} u_{n,t}^{L3}, \\
 & \{t \in T_1 | n = i, j\}, \{t \in T | n = k\} \\
 Y_n^{L2,c} \leq & BIGI_n \left(1 - \sum_{t \in T_1} u_{n,t}^{L3}\right), n \in \{i, j, k\} \\
 & Y_n^{L3,c} + Y_n^{L2,c} \leq BIGI_n u_n^c, n \in \{i, j, k\}
 \end{aligned} \tag{4.7}$$

The first constraint, of the above set, defines that an edge system interface will be chosen if at least one edge system is installed in that location. The second constraint shows that an aggregation switch at location n must not be installed if an edge router is installed at this location. The last constraint indicates that either a 1Ge or 10Ge interface can be installed at a network device.

Avoiding redundancy of functionality

In order to avoid the redundancy of functionality in a path, only a certain combination of routers can be used per path. The path starts from the access node and ends at the P-Router, and can be represented by the multiplication of the following binary constants $u'_{i,j}u'_{j,k}$. In that path, each functionality must not be repeated (which is represented by the sum of the binary variables in the path must not be greater than one) and certain functionality must not be used.

$$\begin{aligned}
u'_{i,j}u'_{j,k} \sum_{t \in T_1} u_{i,t}^{L3} + u_{j,t}^{L3} + u_{k,t}^{L3} &\leq 1 \\
u'_{i,j}u'_{j,k} \left(\sum_{t \in \{A,C\}} (u_{i,t}^{L3} + u_{j,t}^{L3}) + \sum_{t \in T_2} u_{k,t}^{L3} \right) &\leq 1 \\
u'_{i,j}u'_{j,k} \left(\sum_{t \in \{A,B\}} (u_{i,t}^{L3} + u_{j,t}^{L3}) + \sum_{t \in T_3} u_{k,t}^{L3} \right) &\leq 1 \\
u'_{i,j}u'_{j,k} &\leq \sum_{t \in T_1} u_{i,t}^{L3} + u_{j,t}^{L3} + u_{k,t}^{L3} \\
u'_{i,j}u'_{j,k} &\leq \sum_{t \in \{A,C\}} (u_{i,t}^{L3} + u_{j,t}^{L3}) + \sum_{t \in T_2} u_{k,t}^{L3} \\
u'_{i,j}u'_{j,k} &\leq \sum_{t \in \{A,B\}} (u_{i,t}^{L3} + u_{j,t}^{L3}) + \sum_{t \in T_3} u_{k,t}^{L3}
\end{aligned} \tag{4.8}$$

Capacity Constraints

The capacity constraints define the amount of traffic that either an aggregation switch or an edge system can support. This is effectively the backplane capacity. For the case of a distributed router $u_{n,t}^{L3} = 1$ selected at location $n = \{i, j\}$, the following two constraints are defined

$$u_{n,t}^{L3}(x_n^{CDN} + x_n^{res} + x_n^{bus} + x_n^{P2P}) \leq Y_{n,t}^{L3} C_t^{L3}, \tag{4.9}$$

$$\forall n = \{i, j\}, t = \{A, B\}$$

$$u_{n,t}^{L3}(x_n^{IPTV} + x_n^{res} + x_n^{bus} + x_n^{P2P}) \leq Y_{n,t}^{L3} C_t^{L3}, \tag{4.10}$$

$$\forall n = \{i, j\}, t = \{C, D\}$$

Type A and B do have multicast functionality, therefore they will replicate the traffic. Hence, only the flows that correspond to the requested channels need to be routed x_n^{CDN} . Nonetheless, if type C or D are selected to be distributed, then the replication of the channels is done in the core location, and they will need to route the bundled flows for the multicast traffic. Starting from the top, type A router handles all traffic, therefore the throughput at a central k must be

less than the total throughput of the routers of type A. Note that type A handles all types of traffic, thus it serves all the traffic $x_k^{CDN} + x_k^{bus} + x_k^{res} + x_k^{P2P}$. Similarly for the rest, with the main difference that only a subset of functionality are support per edge system.

$$\begin{aligned}
u_{k,A}^{L3}(x_k^{CDN} + x_k^{bus} + x_k^{res} + x_k^{P2P}) &\leq Y_{k,A}^{L3}C_A \\
u_{k,B}^{L3}(x_k^{bus} + x_k^{res} + x_k^{P2P}) &\leq Y_{k,B}^{L3}C_B \\
u_{k,C}^{L3}(x_k^{res} + x_k^{bus} + x_k^{P2P}) &\leq Y_{k,C}^{L3}C_C \\
u_{k,D}^{L3}(x_k^{res} + x_k^{P2P}) &\leq Y_{k,D}^{L3}C_D \\
u_{k,E}^{L3}x_k^{bus} &\leq Y_{k,E}^{L3}C_E \\
u_{k,F}^{L3}x_k^{CDN} &\leq Y_{k,F}^{L3}C_F
\end{aligned} \tag{4.11}$$

For the switching capacity things are more complicated as the capacity is affected by the distribution of the edge system functionality. For example, the closer the multicast functionality is to the subscribers, the closest the replication takes place. Equation (4.12) indicates the switching capacity constraints. The first is related to the locations in the first aggregation level i . This constraint indicates that if an edge system is installed $u_{i,t}^{L3} = 1$ then there is no need to care about the switching capacity. The second constraint is related to the locations in the second aggregation level j . If multicast replication takes place at lower level $u_{i,A}^{L3} + u_{i,B}^{L3} = 1$, then only a single flow per channel needs to be switched x_{CDN} . If it is taking place at the core location, $u_{i,C}^{L3} + u_{i,D}^{L3} = 1$ or $\sum_{t \in T_3} u_{k,t}^{L3} = 1$ (T_3 is the set of the routers that support multicast), then a separate flow per subscriber for the IPTV needs to be switched. Finally, if a router is placed at the second aggregation level $u_{j,t}^{L3} = 1$, then there is no need to switch any traffic. If a switch is placed in a core location and also an MSE or Video BNG is placed at the same location, then they will route business and IPTV traffic and therefore those two portions need to be excluded from the switching capacity $x_k^{bus}(1 - \sum_{t \in T_2} u_{k,t}^{L3}) + x_k^{CDN}(1 - \sum_{t \in T_3} u_{k,t}^{L3})$. Those constraints can be found in the inequality set (4.12) at the top of the next page.

$$\begin{aligned}
& (1 - \sum_{t \in T_1} u_{i,t}^{L3})(x_i^{IPTV} + x_i^{res} + x_i^{bus} + x_i^{P2P}) \leq Y_i^{L2} C_{ags} \\
\sum_i (u_{i,j}(u_{i,A}^{L3} + u_{i,B}^{L3})x_k^{CDN} + (u_{i,C}^{L3} + u_{i,D}^{L3})x_i^{IPTV}) + \sum_{t \in T_3} u_{k,t}^{L3}x_j^{IPTV} + (x_j^{res} + x_j^{bus} + x_j^{P2P})(1 - \sum_{t \in T_1} u_{j,t}^{L3}) & \leq Y_j^{L2} C_{ags} \quad (4.12) \\
(1 - \sum_{t \in T_1} u_{k,t}^{L3})(x_k^{CDN} + x_k^{res} + x_k^{bus} + x_k^{P2P}) + x_k^{bus}(1 - \sum_{t \in T_2} u_{k,t}^{L3}) + x_k^{CDN}(1 - \sum_{t \in T_3} u_{k,t}^{L3}) & \leq Y_k^{L2} C_{ags}
\end{aligned}$$

$$\begin{aligned}
& (1 - \sum_{t \in T_1} u_{i,t}^{L3}) S_i \leq Y_i^{L2} sub_{ags} \\
\sum_i u'_{i,j} ((u_{i,A}^{L3} + u_{i,C}^{L3}) J + (u_{i,B}^{L3} + u_{i,D}^{L3}) (S_i^{bus} + J)) + \sum_{t \in T_1} u_{k,t}^{L3} S_k & \leq Y_i^{L2} sub_{ags} \quad (4.14)
\end{aligned}$$

Subscriber Termination/VLAN Capacity

The following equalities determine the number of VLANs per location. A distinction is made between the residential and the business VLANs.

$$\begin{aligned}
S_i^{res} &= \sum_a u'_{a,i} S_a^{res}, S_i^{bus} = \sum_a u'_{a,i} S_a^{bus} \\
S_j^{res} &= \sum_i u'_{i,j} (1 - \sum_{t \in T_1} u_{i,t}^{L3}) S_i^{res} \\
S_j^{bus} &= \sum_i u'_{i,j} (1 - \sum_{t \in \{A,C\}} u_{i,t}^{L3}) S_i^{bus} \\
S_k^{res} &= \sum_j (1 - \sum_{t \in T_1} u_{j,t}^{L3}) S_j^{res} \\
S_k^{bus} &= \sum_j (1 - \sum_{t \in \{A,C\}} u_{j,t}^{L3}) S_j^{bus}
\end{aligned} \quad (4.13)$$

For routers of type A, B, C and D which may exist at any of the levels $n = \{i, j, k\}$ the subscriber termination constants are similar to the capacity ones. They are expressed as follow $u_{n,t}^{L3} S_n \leq Y_{n,t}^{L3}$, where $S_n = S_n^{res} + S_n^{bus}$ for Type A/C, $S_n = S_n^{res}$ for B/D. $S_n = \sum_a w_a BUS_a$ for Type E, and $S_n = \sum_a w_a RES_a$ for Type F. The latter is expressed lie that as all these must be terminated at the Video BNG, because only a portion w_a of subscribers are watching a channel and the total number of IPTV VLANs are determined by $\sum_a w_a res_a$

For VLANs that need to be switched, the case is a bit more complicated. For the second level of aggregation and if a customer has been already been terminated in the previous level, then $J - 1$ VLANs will be required to switch traffic among the same level of switches and one more that will send the traffic to the P-router. If however only the residential subscribers have been terminated in the previous level ($u_{i,B}^{L3} + u_{i,D}^{L3} = 1$) then the business VLANs will still remain ($S_i^{bus} + J$). The constraints for the VLAN capacity are mentioned in the next page, inequality (4.14)

Link and Interface Constraints

Each access location is assumed to have N_a devices and each location has both uplink and downlink interfaces. The minimum number of downlink interfaces from the first aggregation level is equal to the number of access devices that are connected to this location, i.e. $u'_{a,i}N_a$. In every of the first and second level aggregation locations either a router or a switch or nothing is placed. Therefore, two of the $Y_{n-1,n}^{L3,c,dn}$, $Y_{n-1,n}^{L2,c,dn}$ and $Y_{i,j}^{0,dn}$ are going to be zero.

$$\begin{aligned}
u'_{a,i}N_a &\leq \sum_c (Y_{a,i}^{L3,c,dn} + Y_{a,i}^{L2,c,dn}) \\
u'_{i,j}(\sum_{t \in T_1} Y_{i,t}^{L3} + Y_i^{L2}) &\leq Y_{i,j}^{0,dn} + \sum_c Y_{i,j}^{L3,c,dn} + Y_{i,j}^{L2,c,dn} \\
\sum_{t \in T_1} Y_{j,t}^{L3} + Y_j^{L2} + \sum_i Y_{i,j}^{0,dn} &\leq \sum_c Y_{j,k}^{L3,c,dn} + Y_{j,k}^{L2,c,dn}
\end{aligned} \tag{4.15}$$

The network elements also have line card capacity. Each line card is a separate module that has a specific capacity of ports and functionalities. We assume two types of line cards, one for switches and one for routers. Each element may take ports of the same type, 1Ge and 10Ge, and the port capacity is determined by multiplying the number of line card slots to the number of ports per line card. Since the number of ports per line card is affected by the bandwidth of the interfaces, the same will hold for the total number of ports per network element. The constraints are expressed as follows.

$$\begin{aligned}
\sum_a u'_{a,i} Y_{a,i}^{L2,c,dn} + \sum_j u'_{i,j} Y_{i,j}^{L2,c,up} &\leq Y_i^{L2} P_i^{L2,c} \\
\sum_a u'_{a,i} Y_{a,i}^{L3,c,dn} + \sum_j u'_{i,j} Y_{i,j}^{L3,c,up} &\leq \sum_{t \in T_1} Y_{i,t}^{L3} P_{i,t}^{L3,c} \\
\sum_i u'_{i,j} Y_{i,j}^{L2,c,dn} + Y_k^{L2,c,up} &\leq Y_j^{L2} P_j^{L2,c} \\
\sum_i u'_{i,j} Y_{i,j}^{L2,c,dn} + Y_k^{L2,c,up} &\leq \sum_{t \in T} Y_{j,t}^{L3} P_{i,t}^{L3,c}
\end{aligned} \tag{4.16}$$

As we can see above, the first constraint makes sure that there is at least one link between the access node and the first aggregation level. However if the solution determines that no device is going to be installed, then $d \rightarrow \infty$ and $Y_i^0 = 1$.

Interface Capacity Constraints

The interface capacity constraints are the most complicated, since they must be determined without any knowledge of the placement of the devices. In every device there are going to be

interfaces that connect both the uplink and the downlink.

$$Y_n^{x,c} = Y_{n-1,n}^{x,c,dn} + Y_{n,n+1}^{x,c,up} \quad (4.17)$$

gives the total number of interfaces of capacity c per location n . Moreover, in all cases the uplink capacity is equal to the downlink capacity of the higher layer. For instance for the link between the access location a and first, the constraint is as follows $u'_{a,i}(x_a^{IPTV} + x_a^{res} + x_a^{bus} + x_a^{P2P}) \leq \sum_c \sum_x Y_{a,i}^{x,c,dn} C^c$. For the links between i and j , The bandwidth of the uplink of the first aggregation level is calculated to be: a) the internet and business traffic (non local flows); b) if a multicast functionality is placed at the first level, $u_{i,A}^{L3} + u_{i,B}^{L3} = 1$, the flow of the channels from the second aggregation level x_j^{CDN} ; c) if there is not a multicast functionality $1 - \sum_{t \in T_1} u_{i,t}^{L3} = 1$ on the first level, the IPTV flow of each subscriber; d) if there is routing functionality $\sum_{t \in T_1} u_{i,t}^{L3} = 1$ the P2P portion that is not first level local $(1 - p_1)x_i^{P2P}$. For simplicity we mention only the downlink constraints (the uplink i to j are the same). Similarly the remaining interface capacity constraints are calculated for the other two aggregation levels.

$$u'_{i,j}((u_{i,A}^{L3} + u_{i,B}^{L3})x_j^{CDN} + \sum_{t \in T_1} u_{i,t}^{L3}(1 - p_1)x_i^{P2P} + (1 - \sum_{t \in T_1} u_{i,t}^{L3})x_i^{IPTV} + x_i^{P2P}) + x_i^{HSI} + x_i^{BUS} \leq \sum_c \sum_x Y_{i,j}^{x,c,dn} C^c \quad (4.18)$$

$$x_j^{HSI} + x_j^{BUS} + (1 - \sum_{t \in T_1} u_{j,t}^{L3})(1 - p_1 - p_2)x_j^{P2P} + (1 - \sum_{t \in T_3} u_{j,t}^{L3})x_k^{CDN} + \sum_{t \in T_3} u_{j,t}^{L3}x_j^{IPTV} \leq \sum_c \sum_x Y_{i,k}^{x,c,dn} C^c \quad (4.19)$$

$$x_k^{HSI} + (1 - \sum_{k=1}^3 p_k)x_k^{P2P} + x_k^{CDN} + x_k^{BUS} \leq \sum_c \sum_x Y_k^{x,c,up} C^c \quad (4.20)$$

Table 4.4: Input Parameters for Aggregation Network Characteristics and Interface Properties

Symbol	Descr/Units	Small SP	Big SP
A	Locations	1000	2000
I	Locations	6	50
J	Locations	3	5
K	Locations		1
SUB	Subscribers	200K	400K
res_a	Subscribers	160K (80%)	320K (80%)
bus_a	Subscribers	40K (20%)	80K (20%)
S_a	Subscribers		200
Ch	Channels		100
w_i	% of viewers		50%
C_{1G}^{L2}/C_{10G}^{L2}	Bytes		1G/10G
C_{1G}^{L3}/C_{10G}^{L3}	Bytes		1G/10G
$co^{L2,1G}$	\$		1K
$co^{L3,1G}$	\$		2K
$co^{L2,10G}$	\$		2K
$co^{L3,10G}$	\$		4K

4.5 Model Implementation

As a mixed integer programming model, the problem is considered to be NP-hard (polynomial time hard). We applied two heuristics to decrease the number of variables. After this phase, we used the branch-cut-algorithm from the CPLEX 11 engine. Finally, we used the CPLEX pre-solver to eliminate redundant rows and columns. In the following we describe the two heuristics:

Heuristic 1: Minimizing number of variables. Generally the number of access locations A varies from hundreds to thousands making the problem hard to be solved. For this, we implemented a clustering algorithm to determine the number of access locations. The algorithm starts with a single cluster $cl = 1$, increases the number of clusters in every iteration, and terminates when both the time to convergence is smaller than a predefined value t' and the optimality tolerance is also smaller than a predefined value ac' . This means that the number of access locations will be $A = cl * I$, where cl is the number of clusters. The divisive clustering procedure is as follows

- 1: **procedure** CLUSTER(I, t', ac')
- 2: $cl \leftarrow 1$
- 3: $A \leftarrow I$
- 4: Run model to determine t and ac
- 5: **while** $t \leq t'$ and $ac \leq ac'$ **do**

```

6:       $cl \leftarrow cl + 1$ 
7:      Perform K-means clustering
8:       $A \leftarrow cl * I$ 
9:      Run model to determine  $t$  and  $ac$ 
10:   end while
11:   return  $cl$ 
12: end procedure

```

In every iteration, the K -means instance [58] is called with attributes related to the technical characteristics of the access devices. In our case we used as attributes the L2 subscriber capacity of the access technologies. After running the optimization model several times, we determined that if the model does not converge to a solution in less than $t' = 120sec$, it will hardly manage to converge (soft threshold). Finally the optimality tolerance was set to 10^{-6} . In any case, a potential error in terms of Kbps will not affect the solution of the model (since all system capacities are at the order of thousands of Gbps).

Each access location is assumed to incorporate the same access device. However, the definition of the “location” in our model is not strictly a geographical location. Therefore, if in the same geographic location different access technologies are implemented, then these are treated as two separate locations.

Heuristic 2: Minimizing execution time. The optimization has integer variables that are either binary (qualitative) or non-binary (quantitative). In order to transform the problem to integer programming model, the values of $BIGI_n^c$ and $BIGD_n$ were introduced. If these values are selected to be very big, then the solver would require a significant amount of time to converge to an optimal solution. If the values are selected to be small, the solution would not be the optimal (or the problem could be infeasible). Thus the values were selected by taking into account the amount of traffic that flows and the number of VLANs in the corresponding location. A logical upper bound for the number of devices per aggregation location can be derived from

$$BIGD_n = \max\left\{\frac{X_n}{\min\{C_t^{L3}, t \in T\}}, \frac{sub_n}{\min\{sub_t^{L3}, t \in T\}}\right\} \quad (4.21)$$

For an upper bound on the number of interfaces, we used the above upper bound multiplied by the most numbers of ports

$$BIGI_n^c = BIGD_n * \max\{P_t^{c,L3}, t \in T\}, c \in \{1G, 10G\} \quad (4.22)$$

Using the above values the solver was determining the solution in less than 1min on an Intel Core2 Duo T7700 with 4GB RAM.

Table 4.5: Network Element Features

Edge System	Cost $co_{L3,t}$ (K\$)	Capacity C_t (Gbps)	IP Term. $subL3, t$	1Ge $P_{L3,t}^{1G}$	10Ge $P_{L3,t}^{10G}$
All	300	160	64000	192	24
HSI/VideoBNG	600	640	64000	480	64
HSI/MSE	220	40	32000	96	12
HSI	340	280	32000	140	28
MSE	180	20	4000	96	12
Video BNG	200	280	10000	140	28
	Cost K\$	Switching C_t (Gbps)	VLAN $vlan$	1Ge P_{L2}^{1G}	10Ge P_{L2}^{10G}
Agg. switch	270	280	64000	140	28

4.6 Evaluation and Results

In this section, we use two exemplar architectures to evaluate the aforementioned edge design model. The first architecture, namely "small SP", is based on an EU ISP with 200K subscribers. The second architecture, namely "big SP" is based on another EU ISP with twice the number of subscribers, and ten times more first level aggregation locations. We use several traffic scenarios per subscriber (very low up to very high). We performed this investigation such that a future traffic growth is incorporated in our modeling. In table IV, the details of the architectures are given. In table V the values for each edge system are showcased. These values were provided by vendor and are representative of high-end systems at the time of submission². The number of interfaces in the table is derived by multiplying the number of ports per line card with the capacity of line cards per edge system. In addition the local (P2P) traffic does not exceed 20% of the total internet traffic [62] and P2P locality probabilities p_k get values from a uniform distribution. The questions that we try to address are as follows

- Which of the functionalities should the provider distribute closer to the subscriber?
- Does the size of the ISP affect the choice of the optimal functionality placement?
- Should the providers choose faster (but more costly) line cards or a faster backplane in an Edge Router?
- What is the usage (or how much redundancy exists) per edge system for throughput, termination capabilities and port availability?

²The prices of those systems may vary over time and offer. However those values are representative at the time of submission through private communication with at least one vendor. An incremental change of those values does not affect the formulation of the problem.

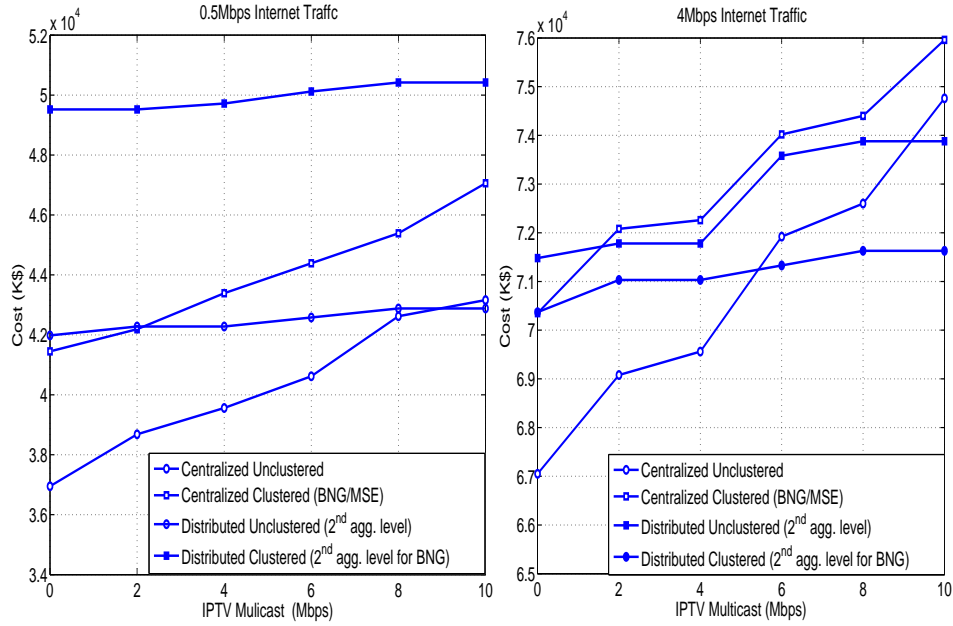


Figure 4.4: Cost comparison for the big service provider per aggregation network for 0.5Mbps and 4Mbps of Local & Non Local traffic (Internet)

- How does a change in the number of subscribers affect the intelligence distribution?

In Fig. 4.4 the cost of multiple architectures is shown for the big service provider. The centralized and 2nd level distributed architectures are investigated. The 1st level aggregation is the first level above the access network and similarly for the rest. Due to the number of 1st level aggregation locations, we found that the cost of distributing the functionalities at the 1st level was higher than in the other two. For this reason, the centralized and the 2nd level distributed architectures were chosen to be plotted. In the case of a small amount of multicast flows, the centralized unclustered seems to be the cost optimal architecture. This is because most of the traffic goes through the central locations to the backbone. However, an increase of multicast traffic leads to significant increase in the cost of the centralized architecture.

On the other hand, the rate of cost increase for the distributed is smaller, and after some point it becomes cheaper. The point at which the graphs meet is important. It effectively shows at which point the SP will need to distribute the functionalities. For small amounts of internet traffic (0.5Mbps per user) the meeting point is around 8 Mbps of multicast traffic, and for a much larger internet traffic (4Mbps per user) it is around 4Mbps. In such large amounts of traffic, either the capacity constraints of the elements or the port constraints are met, leading to more devices/interfaces. Therefore, *the distribution of IP intelligence is preferable, because*

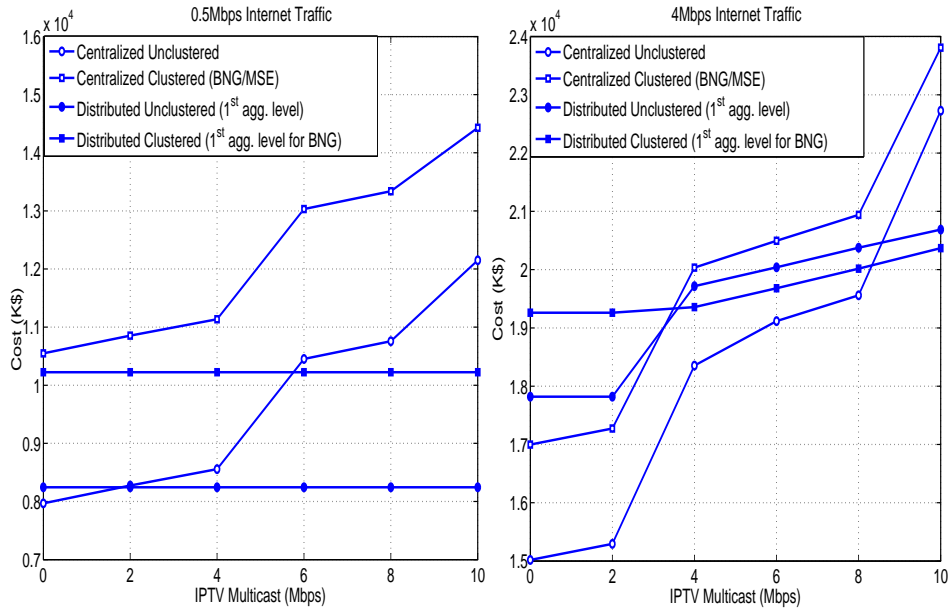


Figure 4.5: Cost comparison for the small service provider per aggregation network for 0.5Mbps and 4Mbps of Local & Non Local traffic (Internet)

(a) it alleviates the bottleneck due to P2P traffic that in any other case would need to go the central location and (b) the multicast traffic is replicated closer to the access network. Finally, clustering the devices does not have a positive impact (compared to the unclustered case) to the cost. In the Big SP there is enough "capacity" (by "capacity" here we mean the shadow price of the subscriber, traffic and port constraints) to support all different services.

In Fig. 4.5 we performed the same comparison for a smaller provider. In this architecture the differences among the implementations become more apparent. For low values of multicast traffic, centralized is optimal. But the increase of multicast traffic affects significantly the cost of a centralized architecture. Moreover assuming a constant internet bandwidth, an increase of multicast traffic would not affect the cost of a distributed architecture. Getting deeper into this issue, the IPTV flows are unicast, until they reach a router, in which case they become multicast. The number of unicast flows are at most equal to the number of channels the SP provides. For the worst case scenario of 100 different channels to be watched at HD (10Mbps), the total traffic will not exceed the capacity of an interface (1Gbps or 10Gbps). A cost breakdown showed that it is optimal to use 10Gbps interfaces, which leaves plenty of bandwidth for even more channels to be offered without having to increase the number of interfaces.

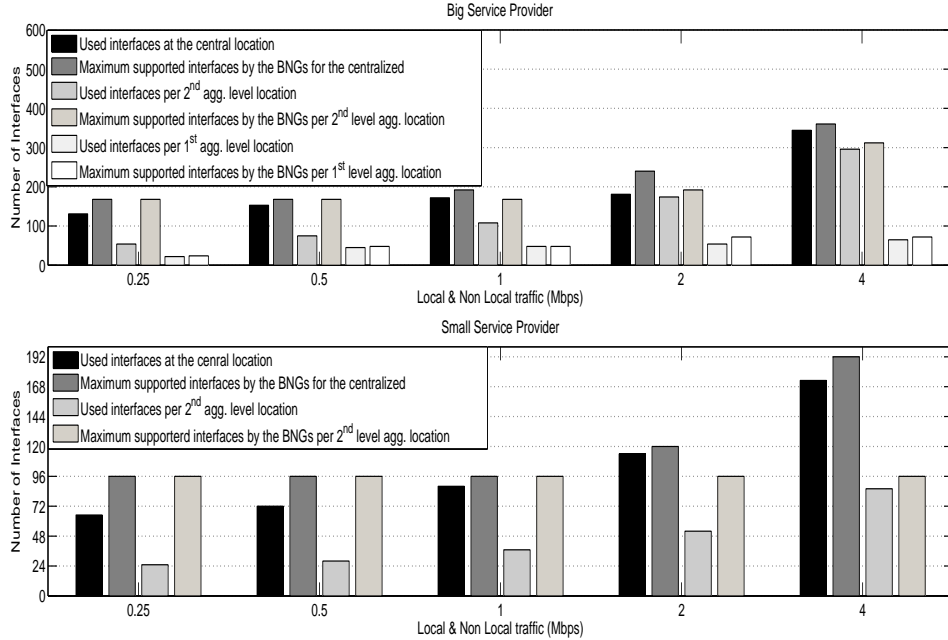


Figure 4.6: Used ports vs available ports in an aggregation location with variable internet and constant IPTV=6Mbps traffic

Since the replication functionality is closer to the access networks, the increase of multicast traffic will only affect the downlink ports of the routers. A single 1Gbps port, that is connected to an access location, will be able to handle 100 HD IPTV flows. Since every DSLAM is having a subscriber capacity of 200 customers and 50% of customers are assumed to be registered to watch TV, then there is no effect on the number of ports. Combining those results with the sensitivity analysis from Fig. 4.6, it looks that there are enough slots to handle an even bigger amount of multicast flows (therefore interfaces). The optimal cost will only be affected by the cost of the interfaces (shadow price). Thus, *distributing the replication functionality seems to be cost efficient*. In similar traffic scenarios for the small SP, the distributed architecture costs two times less than the centralized.

In Fig. 4.6, sensitivity analysis was performed to determine if and how much the number of ports is affecting the solution of the problem. In this figure the unclustered case is plotted for variable internet traffic. The bars are presented in sets; the first bar is the number of used ports at an aggregation location and the second bar is the sum of the number of available interfaces at the same location. Thus, the second bar must be always larger than the first bar, in order for a constraint to be satisfied. The steps of the y -axis are equal to the 10G interface capacity of a BNG (24 ports).

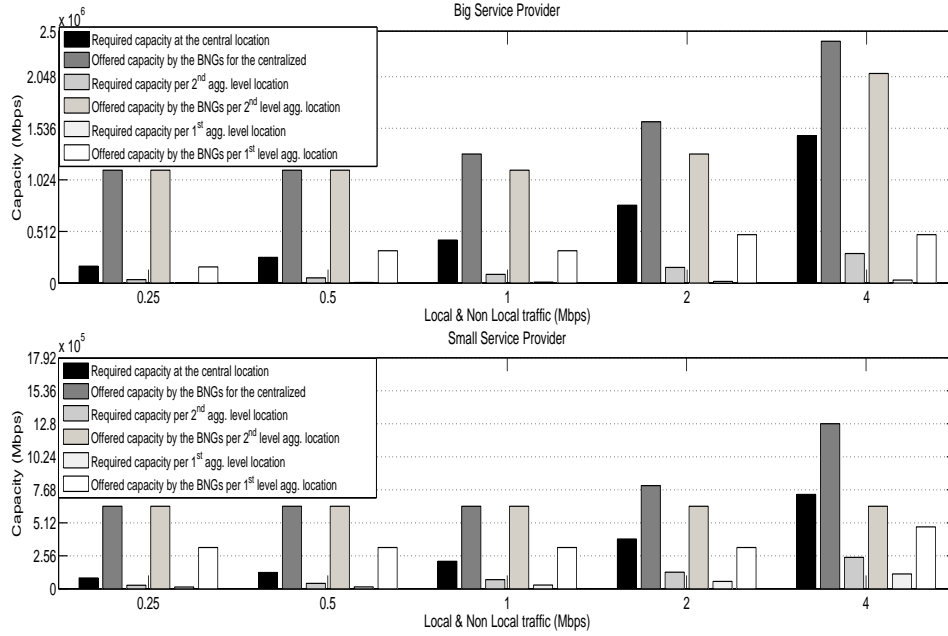


Figure 4.7: Traffic capacity and bandwidth of edge locations plotted with variable internet traffic and constant IPTV=6Mbps traffic

It is clearly shown that in the centralized unclustered topology, and for both a small SP and a big SP, the number of required interfaces determines the number of edge systems to be used (for example for 0.5, 1 and 4Mbps if one less BNG was used the port constraint would have been violated). A similar conclusion can be derived for the distributed unclustered case at the first aggregation level. Therefore, *increasing the interfaces bandwidth such that they handle more traffic or increasing the capacity of the chassis for line cards would decrease the number of required edge systems (and therefore the cost).*

In Fig. 4.7 the amount of traffic that flows through a specific location and the amount of traffic that the edge systems can handle at the same location is presented. Similarly to the above case, the steps of the y -axis are equal to the backplane capacity of a BNG (160Gbps). In most cases, it is clearly shown that the bandwidth of a router does not have an effect on the number of edge systems. For example, even if half of the edge systems are being used, the set of the bandwidth constraints would still not be violated. Therefore, *an increase in the capacity of the edge systems does not have a significant impact on the aggregation networks.*

In Fig. 4.8, it is shown that for the Big SP with low volumes of internet traffic per subscriber and centralized topology, the number of edge systems seems to be affected by the VLAN capacity. However, as the traffic increases, the interface capacity constraint becomes the one that

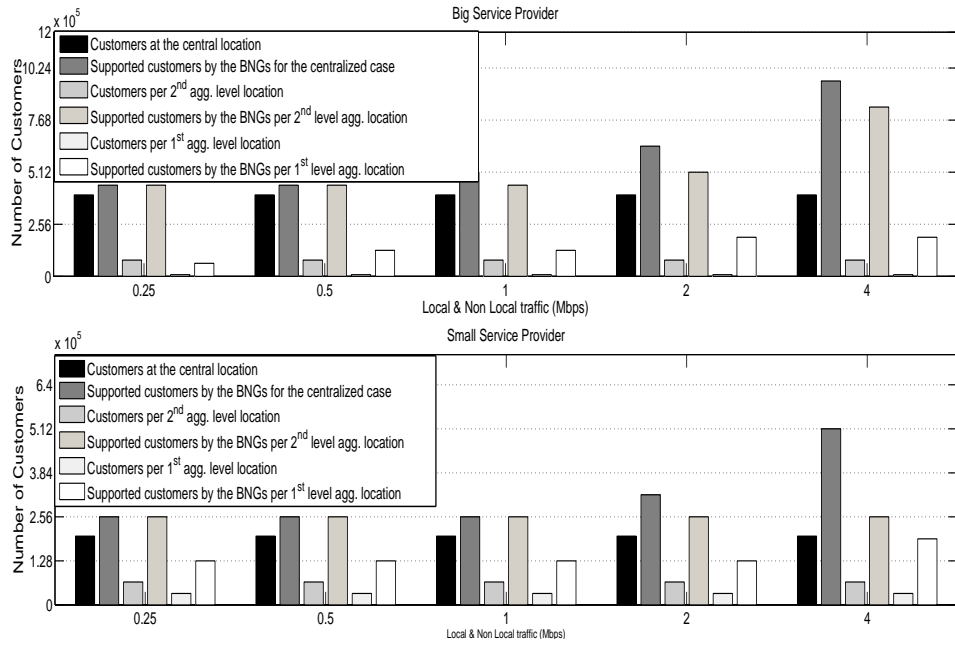


Figure 4.8: Number of subscribers that are terminated per location with variable internet traffic and constant IPTV=6Mbps traffic

Table 4.6: Cost Breakdown in thousands of dollars for IPTV=6Mbps and HSI=1Mbps

SP Architecture	Aggregation Switch (L2)	L2 Ports	L2 Ports	Edge Router (L3)	L3 Ports	L3 Ports
		1G	10G		1G	10G
Centr. Small	6480	1884	276	1200	0	276
Distr. Small (2nd)	5400	1884	42	3600	0	444
Distr. Small (1st)	1350	0	168	3600	2784	0
Centr. Big	31050	0	5230	2100	0	460
Distr. Big (2nd)	27540	0	4570	10500	0	1080
Distr. Big (1st)	3240	0	480	15000	7100	0

requires the addition of more edge systems. A similar pattern is followed by the small SP. An interesting point is that *the subscriber termination capabilities have no effect on the distributed architectures*.

Finally, in table IV, a breakdown of costs is shown for 1Mbps HSI and 6Mbps multicast traffic. For both the small and the big SP and when a distributed topology is implemented, the edge router cost becomes dominant. Due to the fact that the traffic is distributed over the access locations, when distributing the functionalities closer to the subscribers, it is cost optimal to use 1G line cards. In case of a centralized topology, the optimal solution consists of 10G line cards for the edge systems.

4.7 Discussion

Our results showed that re-engineering the edge infrastructure is of importance in order to accommodate intelligent aggregation. More specifically:

1. For a small SP the distributed architecture is cheaper, since only 2-3 BNGs are required per 1st aggregation level. For the big SP the distributed architecture is also less expensive, but the difference with the centralized is smaller.
2. For a small SP the number of BNGs is mainly affected by the number of ports. Yet, if the SP wants to add another edge system in order to have spare ports for resiliency or/and service flexibility, the total cost would not surpass the cost of a centralized architecture.
3. For the big SP, the number of free interfaces are less and the subscriber termination constraints have a small shadow price, leading to smaller flexibility.
4. Distributing the functionalities simplifies the operational flexibility since the network administrator only has to deal with the interface constraints. Moreover, for both the small and the big SP, the cost of the distributed architecture does not increase when the number of IPTV channels increase.
5. Centralized single-edge architectures have been proven to reach scalability limits, whereas clustered, multi-edge or distributed architectures offer better architectural scalability, since smaller number of subscribers are terminated per system.

Distributed architectures have some further advantages. Scalability from the system resources point of view is better, since the amount of state information (memory overhead) in the edge system can be substantially less than in a centralized architecture. In fact distributed architectures benefit from policy enforcement close to the subscriber (no need to backhaul traffic that can be dropped).

One other important aspect is related to the operation of the network. More specifically, single-edge architectures allow provisioning of all subscribers and services on a single system, thus facilitating QoS provisioning using hierarchical schedulers, as well as minimal operational expenditures (OPEX). On the contrary, multi-edge architectures lead to distributed provisioning for services destined to the same subscriber, thus requiring intelligent protocols for proper QoS management and generation of significant amount of state information.

One can name more aspects that our study has not included, but due to space limitations our main contribution was to construct a modularized model and provide several insights for the service providers. We plan to extend this modeling approach into a 4G mobile architecture [48, 54], as well as investigate how caching architectures [73] affect the flow distribution.

4.8 Related Work

During the Internet era the extensive need for bandwidth became a crucial issue for Internet SPs. This led the research community to focus on Network Design Problems that require advanced optimization techniques to be solved. These problems can be divided into two main categories: *Locationing problems*, which are pure topological design problems and where the demand volumes are not taken into consideration; *Dimensioning problems*, which incorporate the demand volumes. Both classes of problems have been extensively analyzed in [67, 79, 82, 83].

A sub-category of the dimensioning problems, are the two-level hierarchical network problems [49, 55]. In [37], the authors investigate an hierarchical network problem for fast moving users. These problems assume a set of candidate locations, capabilities of concentrators and routers, and determine the optimal location placement. However, more and more vendors are developing network systems that support on the same backplane (or chassis) multiple functionalities (line cards or blades). Moreover, the high demand for video traffic affects the way IPTV functionalities are distributed in Next Generation Networks [61].

Hence, on the basis of a *dimensioning problem*, our proposed approach is different from previous work in the following points: (a) We propose a modeling approach where micro-optimization is required in each location to determine the appropriate intelligence. (b) The proposed modeling approach may be applied to network designs as well as network upgrades. For example, a network architect can use the model to determine the cost of distributing the functionality closer to the subscriber. (c) The proposed design is not limited to a single hierarchical design (e.g. an IP termination functionality can be placed before or after the switch). (d) Each service may have different characteristics. For example video traffic for IPTV is usually multicast [53]; P2P applications tend to generate multiple flows, some of which are geographically concentrated or distributed over long distances [33]; Internet traffic is usually based on a client-server behavior [69].

In [85], the authors have solved a Network Utility Maximization (NUM) problem for triple play services. The authors propose three utility functions for each offered service. However, NUM problems address the issue of fairness and require the a priori knowledge of the utility function, which in many cases is hard to compute. In our work, we propose three different flows, which have directional characteristics and are quantifiable by performing Deep Packet Inspection (DPI) or flow inspection and classification [34].

In [95], the authors considered an ADSL access network consisting of subscribers, Digital Subscriber Line Access Multiplexers (DSLAMs), metro Ethernet switches and a Broadband Remote Access Aggregation Server (BRAS), and have developed analytical expressions for dimensioning the access network in the upstream direction. Moreover, in [50], the authors performed a cost investigation of transport architectures based only on demand and physical layer capabilities. In our work, we use an Ethernet based next generation aggregation network in which the BRAS has been replaced by multi-purpose Broadband Network Gateways (BNGs), Multi-Service Edge Routers (MSEs) and Video BNGs [29, 51]. We also propose multiple architectures based on the distribution and clustering of the edge functionalities. Our approach is based on the TR-101 broadband forum’s report [19], which standardizes the Ethernet based aggregation design.

In [71] we presented a comparison of centralized vs distributed unclustered topologies, through two optimization problems. In this paper, we combine the problems into a single cost optimization model and extend the above work to include most of the possible ISP architectures. Finally, we evaluate our methodology with architecture designs provided by three major European service providers.

4.9 Conclusion

This chapter focused on a quantitative, cost-optimal hierarchical network model for differentiated services. The model was based on edge “systems”, which can receive extra functionalities as attached sub-systems. The modeling approach incorporates physical characteristics, bandwidth and VLAN/IP termination capacity, L2 versus L3, multicast and business functionalities. In order to scale down the problem, and decrease the execution time, we applied a set of heuristics that are based on clustering algorithms.

Our results indicate that the widely implemented centralized Ethernet based single-edge architecture has reached its scalability limits, hence distributing the functionalities closer to the subscriber is far more efficient. A small SP will benefit more from the intelligence distribution than a larger SP and, independently of the size, the SPs should cluster the business functionalities on a different edge system. Finally, we showed that SPs would receive only a marginal benefit from higher capacity edge systems, and should rather focus on the proper allocation of

the functionalities in their aggregation network.

Chapter 5

Conclusions

5.1 Summary of this Dissertation

This dissertation focused on comparing different types of devices in a corporate and education wireless network, and on identifying their unique traffic characteristics. Based on these traffic profiles, we proposed enhancements in several layers of the network and in deduplication techniques. Our main conclusions can be summarized as follows:

- We performed **OS fingerprinting** by applying a data mining technique in multiple DHCP header fields. Hence, we could follow the traffic generated by each device through an IP lease. We were able to identify more than 97% of the devices in two wireless campus networks. One can further enhance the accuracy by using higher layer header fields, e.g. TCP or HTTP header information.
- We proposed a **Differentiated Lease Policy**, in which the DHCP server is set to allocate different leases on a per device type basis. The benefits of this approach are that:
 1. It decreases the address space utilization by taking advantage of the different sleeping policies and the shorter transient times in smartphones, as well as the intra-campus mobility of the users;
 2. it decreases the DHCP broadcast messages and the server load, by not renewing unnecessary renews;
 3. it is insensitive to the actual mixture between laptop and smartphone devices.
- Our main findings on **Web traffic behavior** for smartphones and laptops are as follows:
 1. Smartphone browsers are less likely than laptops to use HTTP caching features, as shown by fewer uses of the *If-Modified-Since* and *If-None-Match* headers;
 2. multimedia (audio and video) is a large fraction of the bytes downloaded across all devices. Android and BlackBerry download more audio, iPhones more video;
 3. most devices use Flash for video, except iPhones. Most video is downloaded using progressive downloading with range requests;

4. aborted transfers and partial downloads are common occurrences across devices, and are more likely for iPhone users. In particular, video is more likely to be aborted than other content types;
 5. smartphone Web accesses are relatively more concentrated than laptops, as shown by the larger fraction of requests and bytes that are satisfied by the top 20 Web sites.
- In **browser caches** some of the key outcomes are:
 1. Laptop browser cache implementations are more sophisticated than smartphones;
 2. a small increase in the browser cache size is sufficient to capture most of the savings;
 3. a browser cache should be able to handle partially downloaded objects;
 4. a browser cache does not necessarily need to handle ranges, as the difference in savings is small;
 5. average savings across all devices do not sufficiently describe the benefits, as traffic distributions may vary widely.
 - From the **proxy cache** perspective our results indicate:
 1. Replacement policies are not as important as they used to be in the wired networks;
 2. enabling prefetching in proxy caches may result in excessive bandwidth demand from smartphone devices;
 3. proxy caches should follow the discard storing policy above.
 - We propose a **hybrid byte cache**, with both a proxy and a byte cache included as modules in a system, and a scheduler that decides which deduplication technique to apply. This system architecture has the following advantages:
 1. Increases the savings compared to a standalone byte cache, and almost doubles the savings compared to a standalone proxy cache;
 2. provides a speedup equal to 3, which means that it can be deployed in higher bandwidth links;
 3. decreases the memory utilization by 50%.
 - We finally study the **Ethernet Aggregation Architectures** and show that ISPs must re-engineer the edge infrastructure in order to accommodate intelligent aggregation. Our main results are as follows:
 1. For a small SP the distributed architecture is cheaper, since only 2-3 BNGs are required per 1st aggregation level. For the big SP the distributed architecture is also less expensive, but the difference with the centralized is smaller.

2. For a small SP the number of BNGs is mainly affected by the number of ports. Yet, if the SP wants to add another edge system in order to have spare ports for resiliency or/and service flexibility, the total cost would not surpass the cost of a centralized architecture.
3. For the big SP, the number of free interfaces are less and the subscriber termination constraints have a low shadow price, leading to smaller flexibility.
4. Distributing the functionalities simplifies the operational flexibility since the network administrator only has to deal with the interface constraints. Moreover, for both the small and the big SP, the cost of the distributed architecture does not increase when the number of IPTV channels increase.
5. Centralized single-edge architectures have been proven to reach scalability limits, whereas clustered, multi-edge or distributed architectures offer better architectural scalability, since a smaller number of subscribers is terminated per system.

5.2 Future Work

Data deduplication techniques try to remove repetitive patterns in the traffic stream. Compression has been widely used in Web traffic (GZIP, DEFLATE, etc.). However, most compression techniques are application specific, require both client and the server to support compression, and perform deduplication inside a single object. HTTP delta encoding identifies the delta differences between Web objects and may provide extra savings compared to the proxy caches. Nonetheless, the savings are limited to web traffic. In the previous chapters, we have described how a DRE system (byte cache) can identify savings independently of the traffic protocol. Nonetheless, byte caches have some inherited inefficiencies. In some occasions, they may become a bottleneck, as the production of hashes and chunks is CPU intensive. To address this issue we have proposed a hybrid byte cache with a static scheduler in which cacheable content flows through a proxy cache and uncacheable content through a byte cache. As a future work we plan to extend our work in the following topics.

5.2.1 Dynamic Scheduler for the selection of caching modules

In extension to the hybrid cache, we plan to use a dynamic scheduler that would dynamically select the appropriate caching modules. The scheduler would decide the module based on a machine learning process. It would take into account system characteristics and several other quantities including the savings. The benefits of this approach are that parallel caching modules act as independent modules and can process independent traffic flows, hence the system can benefit from pipelined/multi-threaded CPU architectures.

5.2.2 Quality of Service in Byte Caching

In a second aspect, we plan to develop a system that can provide a Quality of Service (QoS) solution in deduplication systems. We propose a system that consists of multiple caching modules in a box, and a feedback controller. The controller monitors the system performance and ensures that the Service Level Agreement (SLA) characteristics of each flow are met, by dynamically allocating the system resources to each caching module. Hence the ISP can provide differentiated speedup for certain applications or customers. Moreover, since each class is an independent entity, a potential fail of caching module will partially affect the performance of the system. Hence further redundancy/failover is assured with our proposal.

5.2.3 Distributed Byte Caching for Mobile Networks

Byte caches have been introduced as a point to point optimization. This is because they require two middleware devices, the encoder and the decoder. We plan to extend the byte deduplication concept in a mobile aggregation setting, in which the byte cache architecture is extended to a distributed point-to-multipoint deduplication architecture. Our plan is to develop an algorithm that would synchronize the distributed devices, hence even if a mobile device moves from one Base State to another (or Radio Network Controller), the deduplication can continue seamlessly. The benefits of such an approach are significant as most 4G architectures have not been designed to accommodate the smartphone data explosion.

REFERENCES

- [1] Aircrack-ng, tools for auditing wireless networks. www.aircrack-ng.org.
- [2] Applecoremedia. http://developer.apple.com/library/IOs/#documentation/CoreMedia/Reference/CoreMediaFramework/_index.html.
- [3] The Bro network security monitor. <http://bro-ids.org/>.
- [4] Ethernet number registration. <http://www.iana.org/assignments/ethernet-numbers>.
- [5] Ettercap. <http://ettercap.sourceforge.net/>.
- [6] Libpcap, packet capture library. www.tcpdump.org.
- [7] Microsoft DHCP Vendor and User Classes. <http://support.microsoft.com/kb/266675>.
- [8] P0f versatile passive os fingerprinting tool. <http://lcamtuf.coredump.cx/p0f.shtml>.
- [9] Peribit WAN optimization. www.juniper.com.
- [10] Personal communication. Telco sales representative (identity omitted for anonymous submission).
- [11] Riverbed networks. www.riverbed.com.
- [12] The TCP behavior inference tool. www.icir.org/tbit.
- [13] A TCP/IP network testing tool and active OS fingerprinting. syscan.sourceforge.net.
- [14] Video cache squid plugin. cachevideos.com/.
- [15] RFC 793 TCP transmission control protocol. <http://www.ietf.org/rfc/rfc793.txt>, Sep 1991.
- [16] RFC 1918 - Address Allocation for Private Internets. IETF RFC, February 1996.
- [17] RFC 2616 hypertext transfer protocol – HTTP/1.1. <http://www.ietf.org/rfc/rfc2616.txt>, June 1999.
- [18] RFC 2965 HTTP state management mechanism. <http://www.ietf.org/rfc/rfc2965.txt>, Oct 2000.
- [19] Migration to ethernet-based DSL aggregation, TR-101, DSL forum. Technical report, April 2006.
- [20] *Sandvine Global Internet Phenomena Report*. Spring, 2011.
- [21] Visual networking index: Usage. Technical report, Cisco, 2011.
- [22] B Aboba, J Carlson, and S Cheshire. RFC 4436 - Detecting Network Attachment in IPv4 (DNAv4). IETF - <http://www.ietf.org/rfc/rfc4436.txt>, March 2006.

- [23] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese. EndRE: An end-system redundancy elimination service for enterprises. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010.
- [24] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
- [25] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: The implications of universal redundant traffic elimination. In *SIGCOMM*. ACM, 2008.
- [26] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: Findings and implications. In *ACM SIGMETRICS/IFIP PERFORMANCE*. ACM, 2009.
- [27] A. Anand, V. Sekar, and A. Akella. SmartRE: An architecture for coordinated network-wide redundancy elimination. In *Proceedings of ACM SIGCOMM*. ACM, 2009.
- [28] Apple. DHCP client software. <http://www.opensource.apple.com/source/bootp/bootp-198.2/IPConfiguration.bproj/dhcp.c>.
- [29] P. Arberg, T. Cagenius, O.V. Tidblad, M. Ullerstig, and P. Winterbottom. Network infrastructure for IPTV. *Ericsson Review*, 84:3, 2007.
- [30] A. Badam, K.S. Park, V.S. Pai, and L.L. Peterson. Hashcache: Cache storage for the next billion. In *6th Network Systems Design and Implementation (NSDI)*, pages 123–136. USENIX Association, 2009.
- [31] G.E.P. Box and N.R. Draper. *Response surfaces, mixtures, and ridge analyses*, volume 527. Wiley-Interscience, 2007.
- [32] V. Brik, J. Stroik, and S. Banerjee. Debugging DHCP performance. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference*, pages 257–262. ACM, 2004.
- [33] M. Cha, P. Rodriguez, S. Moon, and J. Crowcroft. On next-generation telco-managed P2P TV architectures. In *Usenix 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.
- [34] Umang K. Chaudhary, Ioannis Papapanagiotou, and Michael Devetsikiotis. Flow classification using clustering and association rule mining. In *Computer Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD), 2010 15th IEEE International Workshop on*, pages 76 –80, dec. 2010.
- [35] Cisco. Wide-area application services (WAAS). www.cisco.com/web/go/waas.
- [36] Clear Foundation. Application layer packet classifier for linux. <http://17-filter.clearfoundation.com/>.

- [37] F. De Greve, F. Van Quickenborne, F. De Turck, I. Moerman, and P. Demeester. Aggregation network design for offering multimedia services to fast moving users. *Springer Quality of Service in Multiservice IP Networks*, pages 235–248, 2005.
- [38] R. Droms. Dynamic host configuration protocol. IETF RFC, March 2007.
- [39] J. Erman, A. Gerber, M. Hajiaghayi, Dan Pei, S. Sen, and O. Spatscheck. To cache or not to cache: The 3g case. *Internet Computing, IEEE*, 15(2):27–34, march-april 2011.
- [40] J. Erman, A. Gerber, KK Ramakrishnan, S. Sen, and O. Spatscheck. Over the top video: The gorilla in cellular networks. In *IMC*. ACM, 2011.
- [41] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *IMC*. ACM, 2010.
- [42] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *MobiSys*. ACM, 2010.
- [43] A. Feldmann, R. Caceres, F. Douglass, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *INFOCOM*. IEEE, 1999.
- [44] J. Fenlason and R. Stallman. The GNU profiler.
- [45] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. RFC4601 protocol independent multicast - sparse mode (PIM-SM). protocol specification. *IETF*, 2006.
- [46] A. Finamore, M. Mellia, M. Munafo, R. Torres, and SR Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *IMC*, 2011.
- [47] A. Gember, A. Anand, and A. Akella. A comparative study of handheld and non-handheld traffic in campus wi-fi networks. In *Passive and Active Measurement*, pages 173–183. Springer, 2011.
- [48] Z. Ghebretensaé, J. Harmatos, and K. Gustafsson. Mobile broadband backhaul network migration from tdm to carrier ethernet. *Communications Magazine, IEEE*, 48(10):102–109, 2010.
- [49] István Gódor and Gábor Magyar. Cost-optimal topology planning of hierarchical access networks. *Comput. Oper. Res.*, 32(1):59–86, 2005.
- [50] S. Han, S. Lisle, and G. Nehib. IPTV transport architecture alternatives and economic considerations. *Communications Magazine, IEEE*, 46(2):70–77, 2008.
- [51] C. Hermsmeyer, E. Hernandez-Valencia, D. Stoll, and O. Tamm. Ethernet aggregation and core network models for efficient and reliable IPTV services. *Bell Labs Technical Journal*, 12(1):57–76, 2007.
- [52] Junxian Huang, Qiang Xu, Birjodh Tiwana, Zhuoqing Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *MobiSys*, pages 165–178, 2010.

- [53] Y. Huang, Y.F. Chen, R. Jana, H. Jiang, M. Rabinovich, A. Reibman, B. Wei, and Z. Xiao. Capacity analysis of mediagrid: A P2P IPTV platform for fiber to the node (FTTN) networks. *IEEE Journal on Selected Areas in Communications*, 25(1), January 2007.
- [54] D. Hunter, A. McGuire, and G. Parsons. Carrier ethernet for mobile backhaul [guest editorial]. *Communications Magazine, IEEE*, 48(10):92–93, 2010.
- [55] Aníbal Alberto Vilcapoma Ignacio, Virgílio José Martins Ferreira Filho, and Roberto Diéguez Galvão. Lower and upper bounds for a two-level hierarchical location problem in computer networks. *Comput. Oper. Res.*, 35(6):1982–1998, 2008.
- [56] S. Ihm, K.S. Park, and V.S. Pai. Wide-area network acceleration for the developing world. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. USENIX Association, June 2010.
- [57] S. Jin, A. Bestavros, and A. Iyengar. Network-aware partial caching for Internet streaming media. *Multimedia Systems*, 9(4):386–396, 2003.
- [58] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 881–892, 2002.
- [59] M. Khadilkar, N. Feamster, M. Sanders, and R. Clark. Usage-based DHCP lease time optimization. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference*, pages 71–76. ACM, 2007.
- [60] Eric Kollmann. Chatter on the wire: A look at DHCP traffic. <http://myweb.cableone.net/xnih/download/chatter-dhcp.pdf>, 2007.
- [61] G.M. Lee, C.S. Lee, W.S. Rhee, and J.K. Choi. Functional Architecture for NGN-based Personalized IPTV services. *Broadcasting, IEEE Transactions on*, 55(2):329–342, 2009.
- [62] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102. ACM, 2009.
- [63] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Passive and Active Measurement*. Springer, 2010.
- [64] M. Martynov. Challenges for high-speed protocol-independent redundancy eliminating systems. In *Proceedings of 18th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2009.
- [65] M. Martynov. Experimental study of protocol-independent redundancy elimination algorithms. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*. ACM, 2010.
- [66] M. Menth, R. Martin, and J. Charzinski. Capacity overprovisioning for networks with resilience requirements. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 87–98. ACM, 2006.

- [67] M. Minoux. Networks synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19(3):313–360, 1989.
- [68] R. Nagarajan and S. Ooghe. Next-generation access network architectures for video, voice, interactive gaming, and other emerging applications: Challenges and directions. *Bell Labs Technical Journal*, 13:69–86, Spring 2008.
- [69] L. Newell and M. Sif. Optimizing the Broadband Aggregation Network for Triple-Play Services. *Annual Review of Broadband Communications*, page 15, 2006.
- [70] Ioannis Papapanagiotou, Robert D. Callaway, and Michael Devetsikiotis. Chunk and object level deduplication for web. optimization: A hybrid approach. In *International Communications Conference (ICC), 2012 IEEE*, jun. 2012.
- [71] Ioannis Papapanagiotou and Michael Devetsikiotis. Aggregation network design methodologies for triple play services. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1 –5, jan. 2010.
- [72] Ioannis Papapanagiotou, Fabrizio Granelli, Dzmitry Kliazovich, and Michael Devetsikiotis. A metamodeling approach for cross-layer optimization: A framework and application to voice over wifi. *Simulation Modelling Practice and Theory*, 19(9):2117–2129, 2011.
- [73] Ioannis Papapanagiotou, Erich M. Nahum, and Vasileios Pappas. Smartphones vs. laptops: comparing web browsing behavior and the implications for caching. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '12, pages 423–424, New York, NY, USA, 2012. ACM.
- [74] Ioannis Papapanagiotou, Georgios S. Paschos, and Michael Devetsikiotis. A comparison performance analysis of qos w lans: approaches with enhanced features. *Adv. MultiMedia*, 2007(1):1:1–1:13, April 2007.
- [75] Ioannis Papapanagiotou, Georgios S. Paschos, Stavros A. Kotsopoulos, and Michael Devetsikiotis. Extension and comparison of qos-enabled wi-fi models in the presence of errors. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 2530 –2535, nov. 2007.
- [76] Ioannis Papapanagiotou, Dimitris Toumpakaris, Jungwon Lee, and Michael Devetsikiotis. A survey on next generation mobile wimax networks: objectives, features and technical challenges. *Communications Surveys Tutorials, IEEE*, 11(4):3 –18, quarter 2009.
- [77] Georgios S. Paschos, Ioannis Papapanagiotou, Stavros A. Kotsopoulos, and George K. Karagiannidis. A new mac protocol with pseudo-tdma behavior for supporting quality of service in 802.11 wireless lans. *EURASIP J. Wirel. Commun. Netw.*, 2006(3):8:1–8:9, June 2006.
- [78] Georgios S. Paschos, Ioannis Papapanagiotou, Efstathios D. Vagenas, and Stavros A. Kotsopoulos. Performance analysis of admission control for 802.11 wlan networks in ad hoc mode. 2006.

- [79] M. Pioro and D. Medhi. *Routing, Flow and Capacity Design in Communication and Computer Networks*. Elsevier Inc and Morgan Kaufmann Publisher, New York, 2004.
- [80] Pantos R. and May W. Ed. HTTP live streaming. <http://tools.ietf.org/html/draft-pantos-http-live-streaming-07>.
- [81] M.O. Rabin. *Fingerprinting by Random Polynomials*. Center for Research in Computing Tech., Aiken Computation Laboratory, Harvard Univ., 1981.
- [82] M.G.C. Resende and P.M. Pardalos. *Handbook of optimization in telecommunications*. Springer Verlag, 2006.
- [83] B. Sanso and P. Soriano. *Telecommunications network planning*. Kluwer Academic Publishers, Ottawa, 1999.
- [84] J. Schwartzberg. Using machine learning techniques for advanced passive operating system fingerprinting. Technical report, University of Twente, 2010.
- [85] L. Shi, C. Liu, and B. Liu. Network utility maximization for triple-play services. *Computer communications*, 31(10):2257–2269, 2008.
- [86] David Simmons. Rapid DHCP redux. http://cafbit.com/entry/rapid_dhcp_redux.
- [87] D.E. Smith. IP TV bandwidth demand: Multicast and channel surfing. In *IEEE INFOCOM*, pages 2546–2550. IEEE, 2007.
- [88] N.T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *SIGCOMM*. ACM, 2000.
- [89] The Squid Project. Web proxy caching. www.squid-cache.org.
- [90] John S. Vardakas, Ioannis Papapanagiotou, Michael D. Logothetis, and Stavros A. Kotsopoulos. On the end-to-end delay analysis of the ieee 802.11 distributed coordination function. In *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on*, page 16, july 2007.
- [91] E. Veloso. A hierchical characterization of a live streaming media worklad. In *ACM SIGCOMM IMW*, November 2002.
- [92] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. Why are web browsers slow on smartphones? In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 91–96, New York, NY, USA, 2011. ACM.
- [93] Zhen Wang, Felix Xiaozhu Lin, Lin Zhong, and Mansoor Chishtie. How far can client-only solutions go for mobile browser speed? In *WWW*, 2012.
- [94] Rafal Wojtczuk. Libnids: E-component of network intrusion detection system. libnids.sourceforge.net.
- [95] K. Xiong, H. Perros, and S. Blake. Bandwidth provisioning in ADSL access networks. *International Journal of Network Management*, 19(5):427–444, 2009.

- [96] Q. Xu, J. Erman, A. Gerber, Z.M. Mao, J. Pang, and S. Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *IMC*. ACM, 2011.
- [97] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

APPENDICES

Appendix A

Dynamic Host Control Protocol

In the first chapter of the Appendix, we describe a summary of the DHCP protocol. We summarize the details that are important in this thesis; the prospective reader can refer to the [38] for further details.

A.1 DHCP Functionality

DHCP [38] enables automatic network configuration of hosts in TCP/IP networks, with a message exchange between hosts and DHCP servers. To perform these actions there are eight message types.¹ A *discover* message is broadcasted to locate available servers. The listening server replies with an *offer*, which contains the client MAC address. The client generates a *request* (“selecting” state) asking for offered parameters from one server and implicitly declining offers from all others. However, there are other occasions in which a client issues a *request* message, such as confirming correctness of previously allocated address after, e.g., system reboot (“init-reboot” state), or extending the lease on a particular network address (“renewing” or “rebinding” state²).

The server responds to a request with either an *acknowledgment*, if the request is granted, or a *negative acknowledgment*, in the case where the parameters are incorrect or the lease has expired. The *acknowledgment* contains the lease time for which the network address will be valid, either as a new lease, or as an update. The client may extend its lease with subsequent *request* messages sent periodically after half the lease period. For example, if the lease time is 14400 seconds, and the client is still active after 7200 seconds, it can generate a *request* message at that time to notify the server. If the lease time expires, the server assumes that the device has been disconnected from the network. The client can issue an explicit *release* message, but this is not mandated by the RFC.

Finally, an *inform* message is sent from the client to the server to ask for local configuration

¹The RFC refers to these messages as DHCPDISCOVER, DHCPREQUEST, DHCPPOFFER, DHCPRELEASE etc. For simplicity, in this chapter we are going to use the *italics* notation for the DHCP messages.

²The RFC defines the “renewing” and “rebinding” as different states. Their only difference is the way they request a lease extension, i.e., broadcast or multicast.

```

Frame 75: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits)
Ethernet II, Src: 42:09:02:fa:03:00 (42:09:02:fa:03:00), Dst:
Internet Protocol Version 4, Src: , Dst:
User Datagram Protocol, Src Port: bootps (67), Dst Port: bootps (67)
Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 1
  Transaction ID: 0xa57f8cd9
  Seconds elapsed: 0
  Bootp flags: 0x0000 (unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 9.2.240.4 (9.2.240.4)
  Client MAC address: Apple_92:f4:73 (00:26:b0:92:f4:73) First 3 Bytes of MAC Address
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
  Option: (τ=53,1=1) DHCP Message Type = DHCP Request
  Option: (τ=55,1=6) Parameter Request List
    Option: (55) Parameter Request List
      Length: 6
      Value: 0103060f77fc
      1 - Subnet Mask
      3 - Router
      6 - Domain Name Server
      15 - Domain Name
      119 - Domain Search [TODO:RFC3397]
      252 - Private/Proxy autodiscovery
    Option: (τ=57,1=2) Maximum DHCP Message Size = 1500
    Option: (τ=61,1=7) Client identifier
    Option: (τ=50,1=4) Requested IP Address =
    Option: (τ=51,1=4) IP Address Lease Time = 90 days
    Option: (τ=12,1=6) Host Name = "iPhone" Host Name
  End option
  Padding
  
```

Figure A.1: Exemplar DHCP Request message and the important information for the OS Fingerprinting

parameters. This only happens when the client already has an externally configured network address.

A.2 DHCP Request Header

As part of the DHCP OS Fingerprint that was introduced in Chapter 2, we used several fields from the DHCP Request. In figure A.1 we present an exemplar of a DHCP Request message. Nonetheless, not all messages are the same.

A DHCP Request can be issued by the client in four different occasions:

- A client responding to a *Offer* message from a server (“selecting” state). In this case the client inserts the address of the selected server in the ‘server identifier’ option of the *Request* message and the ‘requested IP address’ is filled in with the value from the *Offer* message.
- A client seeking to verify a previously allocated, cached configuration (“init-reboot” state). In this case the *Request* message does not have the ‘server identifier’ filled in, and the ‘requested IP address’ option is filled in with the client’s notion of its previously assigned address.

- A client requesting to extend its address (“renewing” or “rebinding” state³). In this case the *Request* message has neither the ‘server identifier’, nor the ‘requested IP address’ fields filled in. Only the ‘ciaddr’ is filled in with the client’s IP address. In this situation the client is completely configured, and is trying to extend its lease.

Moreover, other options fields include parameters that are related to the host, such as *Host Name*, *Vendor Name*, *MAC address*, etc.

A.3 DHCP Behavior and Sleeping Policies

In the second chapter, we are investigating the DHCP behavior of the smartphone and laptop devices. However, there is little knowledge on how the sleeping policies affect the DHCP network traffic. Here we will share some of our thoughts on how devices behave, when attaching to the network with an unexpired lease⁴.

In the case that a device thinks it has an unexpired leases, and the device’s network interface has just brought the physical LINK up, then the client starts in the DHCP “init-reboot” state (or “init” state). There are two events that can take place.

The IP address of the unexpired lease is not valid on the new network to which the device is now attached. In other words, the client connected to a network that is different from the one it was previously attached to. In this case, the device would issue a DHCP request message in the “init-reboot” state, and the server would response with a DHCP NAK because the requested IP address is topologically inappropriate on the new network (or even if the old IP address is appropriate, it may be leased to some other client on this new network, or there may be some other reason the DHCP server does not want the client to use that IP address on the new network.). The DHCPNAK should drive the client back to the DHCP INIT state so the client may obtain a lease for this new network.

Alternatively, if the device thinks it has an unexpired lease, and has kept LINK up continuously since going to sleep, then the client could start in the BOUND state when awaking. (If link has never toggled since the last time the obtained/renewed its DHCP lease, it’s still on the same network.).

So one of the reasons that mobile devices exhibit a different DHCP behavior is related to whether the device was able to keep the LINK up while asleep. Some devices attempt to this by setting the CPU into a lower power state, but keep the wireless interface up, whereas other switch off their wireless interface after several seconds (if they are not connected to a power source). Similarly, some vendors (e.g. Apple) have implemented Dनाव4 [22]. Dनाव4 has been

³The RFC defines the “renewing” and “rebinding” as different states, Their only difference is the way they request for a lease extension, i.e. broadcast or multicast.

⁴We would like to thank Irwin Tillman from Princeton University for his advices.

used to decrease the time required when moving between networks and when trying to obtain (or continue using) an unexpired IP address. This is important as the re-attachment time may be a significant fraction of the total handover latency.

Appendix B

Proxy Cache Details

B.1 Caching Metrics

We use the following three metrics to assess cache effectiveness. The first metric is the *request hit rate*, i.e., the percentage of requests that are served by the cache. It captures the expected reductions in request response times. The second metric is the *byte hit rate*, defined as the percentage of the bytes that are served by the cache. It shows the amount of traffic that can be served locally without going to the origin server. Finally, the third metric is the actual *savings*, i.e., the traffic reduction achieved through caching. Note that savings can be negative, when caching policies generate more traffic than requested by the users.

B.2 Proxy Cache Rules

At a high level, an HTTP proxy caches responses, usually by storing them in a non-volatile memory, and uses them potentially in the future to generate other responses. Cached responses are generated from the locally stored content either without interacting with the origin server or after consulting with it.

RFC 2616 makes explicit: *a)* what HTTP responses cannot be cached, *b)* how to validate the cached HTTP responses, and *c)* how to revalidate cached HTTP responses that are no longer valid. Note that the cacheability rules in the RFC are not inclusive, i.e., HTTP content can be cached even if it is not explicitly defined in RFC 2616, as far as it is not in conflict with any of the rules that define non-cacheable content. For example, a later RFC (RFC 2965 [18]) clarifies how requests and responses that contain cookies can be cached. RFC 2616 defines that only responses to requests with *GET*, *POST* or *HEAD* methods are potentially cacheable. All other HTTP methods are not cacheable.

In the absence of *cache-control* headers and the *expires* header, an HTTP 1.1 compliant cache can implement its own caching algorithm for *GET*, *POST* and *HEAD* requests. For example, most popular caches [89] follow an HTTP 1.0 compliant caching algorithm. In other words, in the absence of *cache-control* and *expires*, they will consider requests with ‘?’, ‘cgi-

bin', in the URL as uncacheable. Similarly for the POST messages. Similarly in the presence of *cache-control* and *expires*, cacheability is dictated by those arguments only. However, there are some additional rules that are left open for implementation. In order to derive the maximum potential savings: *a*) responses with no validator or expiration time are cacheable (RFC 2616 Sec. 13.4 defines that they may not be cached), and *b*) responses with codes 200, 203, 206 that include *range* or *content-range* headers, 300, 301 or 410 are cacheable (RFC 2616 Sec. 13.4 defines that they may be cached).

Cookies and Caching: Our emulator does not consider the presence of cookies when deciding whether a response can be cached and whether a request can be served with cached content. This is based on the RFC 2965 which states that *set-cookie* headers can be used in public as well as private objects. It is the server's responsibility to include additional caching directives in the HTTP headers in order to enable or disable caching of HTTP objects with cookies.

Range Requests and Caching: Although HTTP 1.1. defines that is optimal to cache request *content-range* headers, our emulator determines differences in requests based on the reported ranges. Thus a cache hit means that both the URL and the byte ranges need to match.

Exceptions for Video Caching: Currently, many popular video streaming Web sites make their video non-cacheable across multiple users by attaching user-specific information in their video file URLs. For example, YouTube video file URLs include various metadata related to user preferences as well as keys unique to each user. Using the full URL as an object name prevents effective caching given that the same video file results in different URLs when accessed by different users or even when it is downloaded from different CDN servers. For this reason, we have developed a number of exceptions when dealing with major video contributors (YouTube, Google Video, Daily Motion) that allow caching of video content. This feature is available in other proxies such as Squid [14, 89]. More specifically, rather than referring to each video using its full URL, we use only the part of the URL that uniquely identifies a video file. In addition, for video file responses that have *cache-control:private* headers, we ignore those headers and we assume that the video file can be cached publicly.

Appendix C

Other Research

C.1 Research on Performance Analysis of Wireless Networks

During my thesis, I have also worked in some other interesting areas mostly related to the analytical decomposition of wireless protocols. Some of these areas were extensions of my undergraduate research. More specifically, I studied the Quality of Service (QoS) of WiFi networks (IEEE 802.11e). The motivation for this study was twofold; first, some IEEE 802.11e properties were not properly captured by prior analyses, and the current standards do not define principles for strict QoS.

I used three common analytical approaches for wireless networks, namely Discrete Time Markov Chains, Closed Queueing networks, and analysis using elementary conditional probabilities. I proposed several enhancements that included (a) QoS class differentiation, (b) Block-Acknowledgments, as defined in the IEEE 802.11e standard, and (c) erroneous channel conditions. I demonstrated that Block-ACKs may improve the QoS under erroneous channel conditions. I also proposed a methodology to incorporate non-markovian traffic to the analytical approaches [75, 74].

In addition, my colleagues and I devised a scheme that combines TDMA for the VoIP class, and CSMA/CA for the rest of the QoS classes [77]. Our analytical results indicated that the saturation throughput was higher and the end-to-end delay was lower [90]. Moreover, we showed that with the proper Admission Control, the aforementioned scheme may provide QoS guarantees [78]. I verified all these results through Discrete Event Simulations. I extended the simulation analysis, on metamodeling concept, in which the simulation outcomes are fed into an optimization cross-layer problem. The programming result is to optimally tweak some of the parameters from different layers, such as the maximum number of users that can connect to the network [72]. Some of these concepts could potentially be extended in 4G networks [76].