BANDWIDTH AGGREGATION ACROSS MULTIPLE SMARTPHONE DEVICES

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Bradley Raymond Zeller

January 2014

COMMITTEE MEMBERSHIP

TITLE:	Bandwidth Aggregation Across Multiple Smartphone Devices
AUTHOR:	Bradley Raymond Zeller
DATE SUBMITTED:	January 2014
COMMITTEE CHAIR:	Phillip Nico, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	John Bellardo, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Foaad Khosmood, Ph.D. Professor of Computer Science

ABSTRACT

Bandwidth Aggregation Across Multiple Smartphone Devices

Bradley Raymond Zeller

Smartphones now account for the majority of all cell phones in use today [23]. Ubiquitous Internet access is a valuable feature offered by these devices and the vast majority of smartphone applications make use of the Internet in one way or another. However, the bandwidth offered by these cellular networks is often much lower than we typically experience on our standard home networks, leading to a less-than-optimal user experience. This makes it very challenging and frustrating to access certain types of web content such as video streaming, large file downloads, loading large webpages, etc.

Given that most modern smartphones are multi-homed and are capable of accessing multiple networks simultaneously, this thesis attempts to utilize all available network interfaces in order to achieve the aggregated bandwidth of each to improve the overall network performance of the phone. To do so, I implement a bandwidth aggregation system for iOS that combines the bandwidths of multiple devices located within close proximity of each other. Deployed on up to three devices, speedups of up to 1.82x were achieved for downloading a single, 10mb file. Webpage loading saw speedups of up to 1.55x.

ACKNOWLEDGMENTS

I am extremely grateful for my incredible support system. I would like to thank my beautiful wife, Taylor, who has been super patient through this entire process. She has supported me physically, mentally and emotionally. My parents, Mark and Linda, have also been extremely supportive in every way imaginable. This thesis would definitely not have been possible without them.

I would also like to thank my advisor, Dr. Nico, who encouraged me to choose a topic that I would enjoy and for guiding me throughout my thesis. Dr. Bellardo provided a wealth of knowledge to this particular domain, and I am deeply thankful for all that he contributed. And lastly, I would like to thank Dr. Khosmood for serving on my committee and for providing excellent feedback.

I would also like to thank the entire Computer Science Department at Cal Poly for the amazing environment, faculty, staff and students.

Contents

List of Tables ix

List of Figures x

1 Introduction 1

2 Background and Challenges 4

2.1 Cellular Network Overview 4

2.2 Bandwidth Aggregation 5

2.2.1 Interface Characteristics Estimation 6

2.2.2 Applications Characteristics Estimation 7

2.2.3 Scheduling 8

2.2.4 Communication Patterns 9

2.3 Challenges 10

2.3.1 Limited Resources 11

2.3.2 Closed-Source Operating System 12

2.3.3 Scheduling 13

2.3.4 High Mobility 13

2.3.5 Incentives 14

2.3.6 Privacy and Security 14

3 Related Work 16

3.1 Single Device, Multiple Interfaces 16

3.2 Multiple Devices, Single Interface 21
3.2.1 Proxy-based Aggregation 21
3.2.2 Client-based Aggregation 23
3.3 Contributions 25

4 Implementation 26
4.1 Overview 26
4.2 Overall Architecture 27
4.3 System Design 28
4.3.1 Browser 28
4.3.2 Proxy 30
4.3.3 Scheduler 31
4.3.4 URLLoader 31
4.3.5 PeerController 32
4.3.6 Interface Monitor 34

5 Algorithms and Parameter Tuning 36
5.1 Scheduling Granularity 36
5.2 Scheduling Algorithms 37
5.2.1 Round-Robin 38
5.2.2 Round-Robin with Individual Pools (RRIP) 38
5.2.3 MIME-Aware 39
5.3 Network Interface Technologies 40

6 Performance Evaluation 42

6.1 Experimental Testbed 42

6.2 Results 44

6.2.1 Baselines 44

6.2.2 Single File Downloads 45

6.2.2.1 Static vs. Dynamic Scheduling 46

6.2.2.2 Pool Size 49

6.2.2.3 Chunk-Size 49

6.2.3 Webpage Downloads 51

6.2.3.1 Scheduling Algorithms 51

6.2.3.2 Pool Size 52

6.2.3.3 Load Distributions 52

6.3 Analysis 55

6.3.1 Operating System Limitations 56

6.3.2 Cellular Connection Interference 57

6.3.3 CPU Bottleneck 59

7 Conclusion 60

7.1 Future Work 60

References 62

LIST OF TABLES

1 Packet types exchanged between members of local, ad-hoc network. . 33

2 Webpage composition measurements. 43

3 Average bandwidths for each device. 44

4 Breakdown of CPU processing tasks. 59

LIST OF FIGURES

1 3G Architecture overview as described in [15] 6

2 Single device, multiple interface architecture 17

3 Multiple devices, single interface architecture 22

4 Cell-Share system components. 28

5 Cell-Share system design. 29

6 Baseline bandwidth measured for each device. 45

7 Aggregate bandwidth using the static scheduling method for loading a single, 10mb file. 47

8 Aggregate bandwidth using the dynamic scheduling method for loading a single, 10mb file. 48

9 Time spent for connection establishment. Increasing the chunk-size reduces the amount of connection overhead. 50

10 Aggregate bandwidth for 3 devices, loading different webpages and using each of the scheduling algorithms. 53

11 The scheduled requests ratios and the actual bytes loaded are not necessarily proportional. The differences between the two ratios (for each test run using the RRQ scheduling method) are shown by each bar in the above graph. 54

12 Throughput over time while using multiple interfaces simultaneously. 58

13 CPU Usage 59

14 Best speedups observed for increasing number of devices. RRIP was the most successful algorithm used for the single file download whereas the MIME-aware algorithm was the most successful for webpage loading. 61

1 INTRODUCTION

Smartphones now account for the majority of all cellphones in use today. In fact, 55% of cell phone owners claim that their phone is a smartphone [23]. One of the most valuable features offered by these devices is their always-on and always-connected Internet connection. This connection is used by several applications such as web browsers, email clients, social networking applications and many more.

Unfortunately, the cellular networks offer much lower bandwidth compared to what we typically experience on our home, wired networks. In a study conducted across the United States in 2012[24], the average bandwidth experienced for 3G and 4G connections (for the major cellular providers) was 2.025 mbps and 6.02 mbps, respectively. The network bandwidth varies drastically depending on several factors including location, time of day, cellular provider, etc.

The lower bandwidth available on cellular interfaces often makes it undesirable to access many types of web content such as streaming videos, large webpages, downloading large email attachments, etc.

Most smartphones are multi-homed, meaning they have multiple network interfaces (ex. cellular, WIFI and Bluetooth) capable of connecting to different networks simultaneously. However, in most cases, only one interface is ever used at one time, and thus the bandwidth achievable at any moment is limited to the bandwidth of the single, active interface. Many researchers have proposed methods for combining the bandwidth of each interface to improve the overall bandwidth of the device, and therefore improving the user experience. Much research has gone into bandwidth ag-

gregation, however, very few implementations have targeted smartphones as we will see in section 3.

Cellular connections primarily used by smartphones - such as 3G, LTE and 4G - offer wide coverage with low throughput. This is opposed to connections often used by desktops and laptops, such as WIFI and Ethernet where coverage is very limited but throughputs are much higher. So it seems that improving the bandwidth in a mobile, smartphone environment is a much more pressing issue.

Consider the following use-case:

> Five friends are on a road trip, traveling in a car. Each person has a smartphone with a cellular data connection of about 1.5mbps. This bandwidth is very limiting. However, if all five friends "pool" together their individual Internet connections, then the bandwidth could increase to a theoretical 7.5mbps (1.5mbps * 5 people) for everyone. This allows each member of the aggregated connection to enjoy higher bandwidth than anyone could experience individually (however, not at the same time).

This was the goal of my thesis: to present an in-depth evaluation of what can be achieved with regards to cellular link aggregation on one of today's state-of-the-art smartphone devices, the iPhone. This thesis presents a system in which multiple iPhones, each with a cellular Internet connection, combine their bandwidth to improve networked applications' performance. The proposed system is implemented at the application-layer and its performance is analyzed and discussed.

The rest of this paper is organized as follows. The next section covers the back-

ground information and discusses several challenges of implementing this system on the Apple iOS platform. Section 3 identifies and describes several techniques that have aimed at solving this problem. Section 4 addresses the design and architecture of the bandwidth aggregation system I implemented. Section 5 discusses all of the algorithms used in this system and the parameter tuning required to achieve the best results. Section 6 presents and analyzes the system's performance results. The last section concludes this thesis.

2 BACKGROUND AND CHALLENGES

This section discusses the general concept of bandwidth aggregation and the several challenges encountered throughout the development of this thesis.

2.1 Cellular Network Overview

A brief explanation of how cellular networks work is given to provide some context into the cellular connections being aggregated. This discussion will cover both the wireless first hop and the network between the first hop and the Internet / telephone network, as explained by [15].

Figure 1 shows the 3G system architecture given by [15]. There are several components involved and we'll examine each in the order in which they are traversed from the viewpoint of a 3G user. Note that the LTE network architecture is different, but for the purposes of explaining how cellular networks work at a high-level, the 3G architecture is used.

Each user, depending on their geographic location, belongs to a cell which is served by a Base Transceiver Station (BTS), commonly referred to as a cell tower. This is the first hop that 3G users experience and the wireless link uses a combination of FDMA, TDMA and CDMA channel-sharing technologies known as Direct Sequence Wideband CDMA (DS-WCDMA). Several BTSs are managed by a Radio Network Controller (RNC), which then connects to the cellular voice network and the Internet via the core network.

The Mobile Switching Center (MSC) manages all authorization and accounting for

several RNCs. They are used by both cellular voice and cellular data core networks and perform several important functions, such as determining whether a particular user has access to the network, establishing and destroying connections and transferring mobile nodes between cells. Specialized MSCs, known as Gateway MSCs, are also employed which serve as the gateway from the MSCs to the cellular voice network.

In order to access the Internet from the RNC, two types of nodes are used: the Serving GPRS Support Nodes (SGSN) and the Gateway GPRS Support Nodes (GGSN). SGSNs interact with the MSCs from the cellular voice core network to accomplish authorization, handoff, and several other important functions. They also forward data to and from mobile nodes in the RNCs to the GGSNs. The GGSN connects several SGSNs to the Internet and appears like any other gateway router. The GGSNs abstract all of the complications of cellular technologies from the global Internet.

The LTE network differs from 3G in that the separate voice and data core networks are combined into a singe, all-IP network. This means that both voice and data are encapsulated into IP datagrams. Also, the LTE radio access network (the wireless first hop) uses orthongonal frequency division multiplexing as opposed to DS-WCDMA which allows for much higher transmission rates.

2.2 Bandwidth Aggregation

The concept behind bandwidth aggregation is to distribute the network load among all available interfaces in order to increase bandwidth and reduce user-perceived latency.

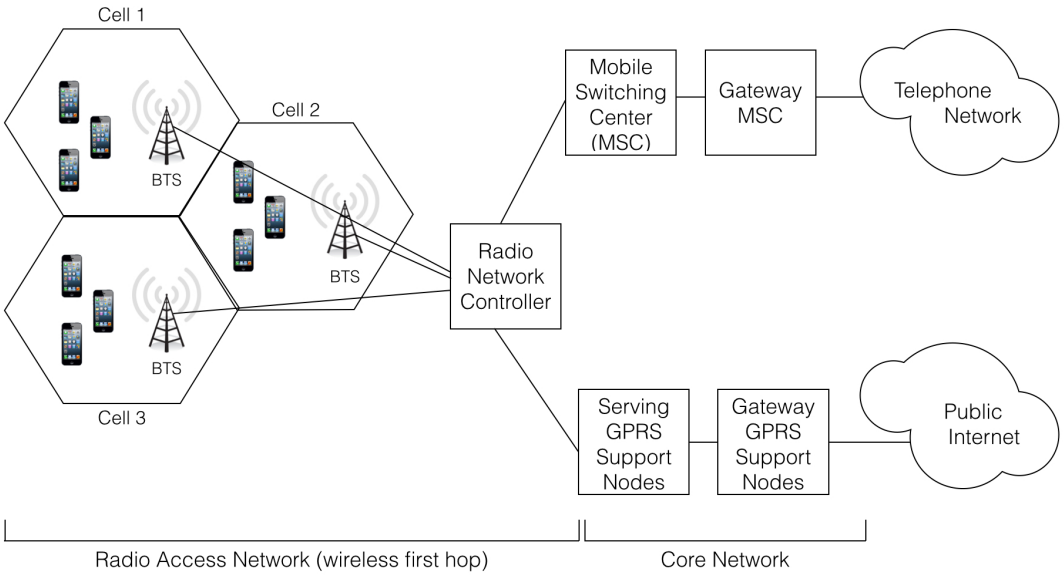


Figure 1: 3G Architecture overview as described in [15]

Bandwidth is defined as the rate at which data is received over time; latency, for the purpose of this paper, is defined as the total time taken to completely fulfill a network request (i.e. downloading a webpage or file).

How network load is distributed depends on the layer where the aggregation is implemented. [7] identifies the features offered by all existing solutions regardless of the layer in which they are implemented: interface characteristics estimation, application characteristics estimation, scheduling and communication patterns.

2.2.1 Interface Characteristics Estimation

Interface bandwidth and delay are essential characteristics used when assigning units of data to a particular interface, otherwise known as scheduling. Another important characteristic, often used by energy aware systems, is how much energy a given interface consumes.

There are several methods for estimating bandwidth on each interface such as measuring the jitter (inter-arrival time between packets) [17], using the packet-pair technique to measure the inter-arrival time between the ACKs of two packets sent back-to-back (on each interface) [4, 26], and active probing where each interface periodically connects to geographically dispersed servers and transfers a fixed size object [8, 6].

2.2.2 Applications Characteristics Estimation

Additionally, many solutions accept information from the application layer to further enhance scheduling decisions. [10] allows applications to categorize themselves into one of several predefined categories (i.e. small or large load transmissions, background or foreground execution) and [3] accepts an application's desired transport method (reliable or unreliable). Alternatively, some systems allow applications to explicitly declare how much bandwidth is needed [29]. This information enables the scheduler to forego costly mechanisms such as ensuring in-order delivery of packets (difficult to guarantee when aggregated links have very different delay and bandwidth capacities).

Note that these application specific hints or requirements are not backwards compatible with existing software at the application layer, thus increasing the barrier to deployment. For this reason, solutions like [8] make a "best effort" attempt by estimating what an application's requirements are (based on historical data) and do not require any changes to existing code.

2.2.3 Scheduling

This section discusses scheduling, a concept briefly mentioned in several of the previous sections. Scheduling refers to the mechanism that distributes network load among available interfaces. This section will cover *what* is being distributed and *how* it is distributed.

First, the *what*: in other words, the level / granularity at which scheduling will occur. [7] describes granularity as the unit of data (referred to as a scheduling-unit for the remainder of this paper) that can be assigned to a network interface. In a broad sense, we have two options: packet-level or connection-level. Packet-level granularity is much more complicated because packets belonging to a single, logical connection can be assigned to different interfaces and will appear to the endpoint as coming from multiple IPs. For this reason, packet-level granularity requires support at both the transmitting and receiving ends of the connection. Alternatively, connection-level granularity assigns packets belonging to a given connection to the same interface and avoids needing support at both endpoints.

There are many techniques describing *how* to schedule network load among interfaces:

- Round-Robin [21, 8, 6]: Often used as a baseline technique, interfaces are simply chosen in a rotating fashion disregarding any of the interface characteristics discussed above.

- Round-Robin with Queues [9]: In this technique, each interface has a logical queue with which scheduling-units are assigned. Similar to basic round-robin,

assignment occurs in a rotating fashion and work is scheduled on interfaces with the smallest queue size.

- Pull-Based [13, 1]: This technique is employed by systems where network interfaces are located on multiple nodes. The master node maintains a pool of scheduling-units needing to be processed and the worker nodes pull and process these scheduling-units (and relay the responses back to the master) whenever their interfaces are inactive or have available capacity.

- Weighted Assignment [6, 8, 5, 14, 22]: This technique takes into consideration the interface characteristics discussed in section 2.2.1, and assigns work accordingly. Therefore, work is scheduled proportional to the estimated performance of the interface.

- Energy-Aware [6]: This technique aims to minimize energy consumption and maximize throughput. The problem can be modeled and solved using linear programming.

There are several more complex scheduling algorithms described in [7], however specific discussion of these techniques is not within the scope of this thesis. The methods mentioned above are meant to highlight the importance of scheduling.

2.2.4 Communication Patterns

The communication patterns employed by each solution are largely a tradeoff between deployability and performance. For example, as mentioned in section 2.2.3, using packet-level granularity requires support at both ends of the connection. However,

this level of granularity often offers the best performance improvements. Therefore we see three types of communication patterns: end-server supported communication, proxy-based communication and legacy-server-based communication.

For end-server supported communication, the server requires modification to be aware of the multi-homed devices that communicate with it. This method sacrifices ease of deployability for performance.

With proxy-based communication, proxy servers implement the support needed for communicating with multi-homed devices and end-servers are left untouched. While this method is slightly more deployable than methods requiring the updating of end-servers, much care needs to be taken to ensure that the proxy does not become the new bottleneck.

The last communication pattern is the most deployable because support is not needed on both ends of a connection. The entire implementation is kept to the end device, therefore performance is sacrificed on behalf of ease of deployability.

2.3 Challenges

This section enumerates several of the challenges related to building a bandwidth aggregation system. Sections 2.3.1 - 2.3.3 are challenges specific to my thesis implementation. Sections 2.3.4 - 2.3.6 are general challenges that are not absolutely necessary to the implementation. These general challenges are discussed to provide a complete overview of bandwidth aggregation systems, however, they are not addressed in this thesis and are left for future work.

2.3.1 Limited Resources

As mentioned earlier, most prior work has focused primarily on desktop computing environments where resources such as CPU, power and memory are plentiful. However, mobile devices are much more constrained. The iPhone 5 and 5s are powered by a 1.3 GHz dual-core processor and contain 1 GB of memory [16]. Battery life is also an extremely important and limited resource on mobile devices.

That being said, the overhead required to combine separate connections must be kept to a minimum. There is a much smaller margin of computing capacity available on mobile devices than on traditional desktop platforms, which means that the introduced overhead could very easily overshadow any speedups achieved by the combined bandwidths.

Because a single data stream is split up into multiple streams across each interface, it will need to be stitched back together into a single data stream when aggregated. This is the additional overhead referred to above. Also, with several contributing members (possibly from different providers) there will likely be a certain degree of heterogeneity with regards to bandwidth and latency. For example, one device may belong to a provider with poor coverage in the current location, while others experience great coverage and much higher bandwidths. This disparity in link performance must be accounted for, so that informed decisions can be made when deciding which interface data should be sent out on.

Also, in order to implement this system, at least two network interfaces on each device need to be active simultaneously (one for collaborating with neighbors and one

for communicating over the Internet). For instance, the cellular and WIFI interface may both be receiving and/or sending data at the same time. These situations could potentially saturate the networking capability of the device or reduce the bandwidth realized by each interface that is active at the same time. To elaborate on the latter case, [13] found that Android OS actually turned off the cellular interface when the WIFI interface was activated at the same time, in order to conserve battery life. They were able to disable this feature by using an undocumented API.

2.3.2 Closed-Source Operating System

Probably the single most challenging obstacle for this thesis was implementing it on top of a closed-source operating system. I believe this is a ikely reason why previous work has chosen other environments to deploy these types of systems.

Typically, operating systems powering mobile devices take very conservative measures in order to provide the user with the best possible experience. Arguably the most precious resource is the battery. Therefore many mechanisms are employed to ensure that battery life is maximized which may directly conflict with the performance optimizations this system aims to accomplish [18]. The iOS developer documentation [2] mentions how accessing the network is "the most power-intensive operation you can perform," and uncoincidentally several of the power-saving mechanisms discussed by [18] target the wireless network interface cards (WNICS). Because iOS is closed-source, these features are undocumented and extremely difficult to disable without rooting or jailbreaking the devices.

Additionally, applications in iOS are sandboxed and closely monitored such that

they do not consume too many resources, especially memory. Application layer code also has a greatly reduced set of permissions and controls over the device hardware and system-wide configurations. This proved to be a very difficult challenge throughout this thesis.

2.3.3 Scheduling

I bring this topic up again because it is more complex when the aggregated interfaces are not located on a single device. The shared interfaces are essentially n-hops away (n depends on the ad-hoc topology) from any one device, meaning that the interface characterizations discussed in section 2.3.1 need to take that into consideration.

2.3.4 High Mobility

Bandwidth aggregation techniques involving multiple wireless devices, as opposed to multiple interfaces on a single device, accomplish the aggregation over an ad-hoc network (WIFI or Bluetooth). However, when a device wanders outside the range of the ad-hoc network, its cellular connection is no longer accessible by other members. Likewise, the other network connections are no longer accessible to the node that wandered out of range. Smartphones are mobile by nature and this scenario is something that must be addressed. What happens when a neighboring node leaves a shared pool after it receives data on behalf of another device? Under normal circumstances, the device would now forward this data back to the originating device. However, this would not be possible once the device leaves the ad-hoc network. Fail-safe mechanisms should be employed to recover from these situations.

2.3.5 Incentives

When aggregating bandwidth across interfaces on multiple devices, there must be incentives for a device-owner to contribute his / her resources. For what reason would an individual share their precious, usually non-free Internet connection with others? Some literature [1, 22] discuss the issues regarding incentives and present several methods for accomplishing it. This is a key step in the process of bandwidth aggregation across multiple devices, because without multiple contributors there is nothing to aggregate. [1] presents a protocol dealing with real money and provides a full-fledged marketplace for bandwidth.

Note that incentives are only necessary when aggregating interfaces on multiple devices. In the alternative case of a single device and multiple interfaces, there is only one owner and no incentive scheme is needed.

2.3.6 Privacy and Security

Because multiple users will be carrying each others traffic, mechanisms should be employed to protect each user's privacy. Users should not be able to inspect or alter traffic that is not their own. This also brings up an interesting case where users may not want to share their bandwidth with users who take part in illicit traffic.

Security is an equally important challenge in this environment. How will user authorization be supported when multiple devices from different IPs attempt to connect to protected end-servers?

As mentioned above, this is an important issue but is not necessary to implementing this thesis and it is left for future work.

3 RELATED WORK

This section discusses several approaches for accomplishing bandwidth aggregation. The discussion is divided into two sections: single device, multiple interfaces and multiple devices, single interface.

3.1 Single Device, Multiple Interfaces

These methods involve aggregating the bandwidth of multiple interfaces attached to a single device (ex. cellular, WIFI and Bluetooth), as shown in figure 2. Most literature discussing these, such as [17, 11, 12, 5], makes the claim that mobile users spend a significant amount of time within the range of multiple networks, namely WLANs (a WIFI access point) and WWANs (a cellular tower).

A recent approach aiming to implement a highly deployable bandwidth aggregation system was presented in [8]. The authors implemented and evaluated a system that aggregated two interfaces, deployed in the Windows desktop environment (the types of interfaces were not mentioned). They achieved speedups of up to 193% and discussed their techniques in detail. The authors used the scheduling techniques discussed in section 2.2.3 plus an additional technique they called Maximum Throughput. This scheduling method assigned connections to interfaces in order to maximize overall system throughput, such that the time needed to finish the current load (including the connection under consideration) was minimized. Obviously, the interface characteristics and the application's connection load were needed in order for this to be determined by the scheduler.

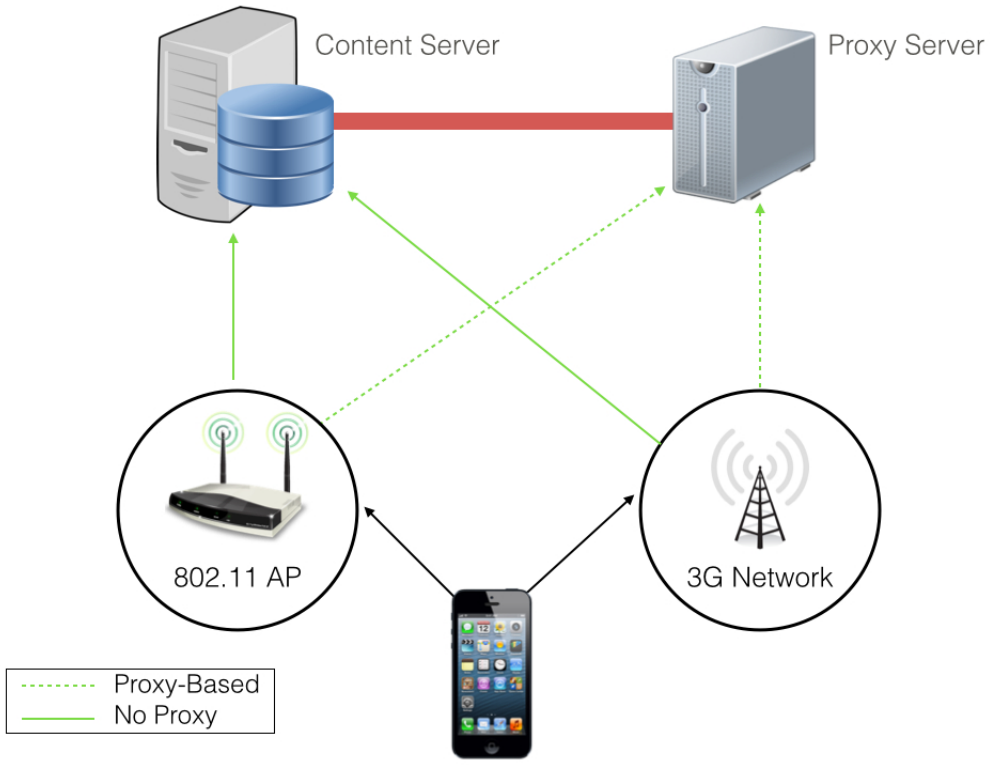


Figure 2: Single device, multiple interface architecture

[8] provided a very helpful evaluation of the different scheduling techniques and their results were very intuitive. The experiments consisted of two interfaces, one with a fixed throughput of 2mbps and the other with a varying throughput (ranging between 0.25 and 2mbps). The schedulers performance was as follows:

- Round-Robin: Performed worse than using only a single interface when interface 2 had less than 1mbps bandwidth. This was expected since round-robin did not take individual interface performance into consideration.

- Weighted Round-Robin: Was able to compensate for the high degree of interface heterogeneity by assigning a higher weight to the better interface and therefore assigning more load to it.

- Maximum Throughput: Outperformed all other scheduling techniques since connection load and interface characteristics were considered.

[27] presented another approach deployed on Android OS, however their implementation was not tested on actual Android devices but rather an Android emulator running on a desktop machine. The authors aggregated the bandwidths of an 802.11n interface and an 802.11g interface. This approach also required the support of both endpoints, raising the barrier for deployment. The authors essentially used a separate thread to manage each connection (both on the client and server). The application protocol used was not mentioned but the evaluation was based on the transfer of a 300mb file. The client continuously monitored the bandwidth of each interface and periodically computed a ratio of how much load each interface could handle; this was then sent to the server which dynamically throttled how much data it sent on each interface. Their evaluation showed that the sum of both interfaces was achieved.

Another interesting approach was presented by [28]. The authors of this paper argued that when aggregating bandwidth among interfaces that are drastically different, in terms of bandwidth capabilities, simply summing together the bandwidths would not yield performance gains perceivable to the end user. For instance, adding a 0.8mbps link with a 7mbps link would only achieve 7.8mbps, at best. This minor increase in throughput is so minimal that users would often not even notice the difference.

Instead of simply summing together the bandwidths, the authors proposed a method called super aggregation. Super aggregation used available interfaces more

intelligently in order to gain throughput gains greater than the simple summation of all throughputs. The authors described three strategies in order to utilize available interfaces more intelligently and implemented the system on an Android device.

The first technique was called Selective Offloading. The authors noted how in identical data transfers and identical environments, UDP was 30% and 70% faster than TCP on 802.11g and 802.11b interfaces, respectively. The obvious explanation for this was that UDP does not wait for any acknowledgements that data has been received, whereas TCP constantly acknowledges received packets. Due to the overheads imposed by the 802.11 protocol, even small packets like ACKs can cause significant contention with actual incoming data packets at the MAC layer. To verify this hypothesis, the authors showed how similar throughput performance resulted when sending small ACK packets back to the sender when using UDP. To avoid this contention on the 802.11 interface, the authors sent the ACKs on the cellular interface instead. IP spoofing was used such that the receiver continued to move the congestion window for the 802.11 connection. This technique improved overall throughput by 37% and 152% for 802.11b and 802.11g, respectively.

The second method was called Proxying which was used in the midst of blackouts. A blackout can occur when a link experiences severe fading or when the interface is transferring between access points. Blackouts may only last for a couple of seconds, but within that time the sender will most likely experience a retransmission timeout in which case it will enter the slow start phase. When the blackout ends, chances are that the sender will be sending at a much lower rate than it was prior to the blackout. This method of proxying sent a zero-window advertisement to the sender

on the cellular link (effectively pausing the sender, but not affecting the send rate). When the blackout ended, the receiver sent a non-zero-window advertisement which resumed the sender, and at the same send rate as before. Proxying-blackout-freezes improved TCP throughput by 87% when blackouts occurred every 20 seconds on average and by 136% when blackouts occurred every 10 seconds.

The last method was called Mirroring. Because TCP interprets packet-loss as link congestion, randomly losing a few packets can severely and unnecessarily affect the send rate of the sender. For the purpose of this explanation, let's assume that the server is sending and the phone is receiving. The mirroring method avoided this slowdown by setting up a mirrored connection to the end server on the cellular interface with the sole purpose of re-fetching the packets that were lost. When a packet was lost, this mechanism hid it from the sender such that an ACK was sent for the missing packet. The mirrored connection then requested the lost packet and inserted it back into the receiver's buffer. Again, this allowed the sender to keep increasing its send rate instead of reducing it for when a packet was lost. The greater the round-trip-time of the WIFI link, the more improvement this technique offered. For a WIFI round-trip-time of 200ms, mirroring increased TCP throughput by 175%.

[5, 11, 12, 21] are all similar approaches. [21] presented a bandwidth aggregation system through a specialized router which aggregated bandwidth from all incoming, WWAN connections. [11] discussed why application and link layer striping approaches do not produce optimal results. The authors presented an end-to-end method at the transport layer which overcame several drawbacks present in other layers. [5] applied bandwidth aggregation techniques to video-streaming where the video was divided

into several chunks and requested through a series of range requests using multiple interfaces. [12] was an extension of [5] and demonstrated the benefits of using HTTP pipelining to overcome the idle time overheads introduced by breaking up a single request into several smaller requests.

3.2 Multiple Devices, Single Interface

Like this thesis, these methods involve aggregating the bandwidth of a single interface across multiple devices, namely multiple device's cellular connection. These methods require that the devices can communicate locally over another interface (such as WIFI or Bluetooth). Each of these approaches perform inverse multiplexing. Intuitively, this is the opposite of normal multiplexing where a single, physical link is divided into several logical links. Inverse multiplexing combines several physical links into a single, logical link. The main distinction between these approaches is where the data striping occurs: at a proxy or at the client.

3.2.1 Proxy-based Aggregation

[14, 22, 27] implemented the multiplexing at a proxy server, shown in figure 3. Once a community of nearby devices was formed, each member of the community established a connection, over their cellular interface, with the proxy server. Each member indicated the community they belonged to which was maintained by the proxy. When a member made a request for data, the request was routed through the proxy. Upon receiving the response data from the origin server, the proxy split the downstream traffic among each connection belonging to the corresponding community. Each mem-

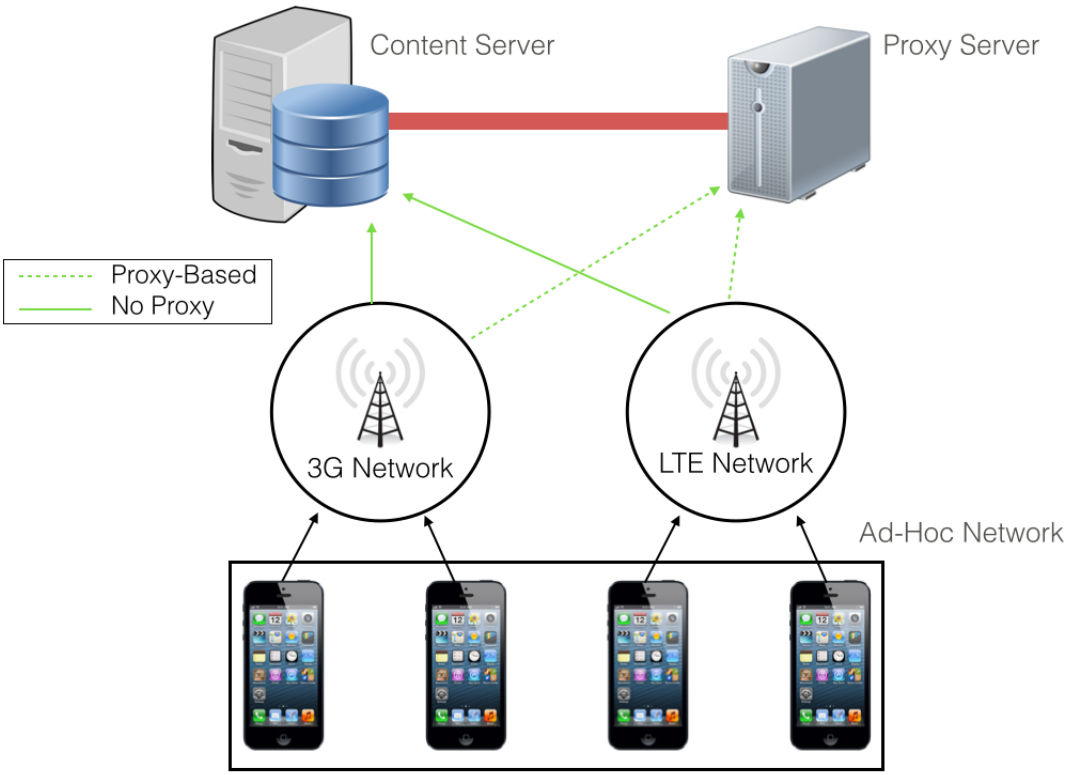


Figure 3: Multiple devices, single interface architecture

ber that received data on behalf of another member simply relayed the segments it received over the ad-hoc network. This effectively load balanced the response data across all available links, rather than being limited by the throughput of just one link. Additionally, the amount of data sent down each link was proportional to the corresponding throughput.

The advantage of such an approach was that all overhead related to scheduling and striping data flows was done at the proxy. This allowed the client members to conserve resources such as CPU. As discussed above, in a mobile environment, this is a very important factor. Additionally, since much of the workload was offloaded onto the proxy, no member needed to maintain a global system view of interface character-

istics nor did they need to perform scheduling logic. However, no matter where data striping occurs, the originating client will always be responsible for reconstructing incoming data. Something to note about the proxy server is that it can easily become a bottleneck if not properly implemented and it generally complicates the deployment process.

The authors of [14] experimented with random and round-robin distribution methods. The round-robin performed best and their speedups were 2-3 times faster (with 2 to 4 additional links) than having no additional links.

3.2.2 Client-based Aggregation

[1, 8, 13] performed the bandwidth aggregation without a proxy. [13] is a system that assumed a group of local users was interested in streaming the same video from a remote source. Each collaborating member would contribute their cellular bandwidth with which to download chunks of the video and then would subsequently distribute them to the entire group. The authors demonstrated a robust and efficient scheduling algorithm and an efficient method for communicating locally. Similar to the previous approaches, the members used the WIFI interface (ad-hoc mode) for local communication and the cellular interface for downloading the actual file chunks. They used a work-list algorithm to schedule file chunks to members, followed a push-pull distribution scheme for the dissemination of the video data to all members and utilized random network coding (a technique that combines packets together for transmission with the goal of improving network throughput) in order to reduce the amount of traffic on the local network and increase the efficiency of overhearing. They provided

several key insights for implementing this on real smartphone devices, which are discussed further in section 5, and deployed and evaluated the system on Android. The authors of this system were able to achieve nearly linear speedups as new nodes were added to the system (they tested with up to 4).

[1] is a similar approach to [13], however it did not make the assumption that the group of collaborators knew or trusted each other. Thus, much emphasis was placed on group formation, incentives and cost modeling for bandwidth sharing. These are interesting aspects but are not within the scope of this thesis. The authors also discussed in detail an implementation scheme for workload distribution. First, the file size was queried via an HTTP HEAD request. Then, the file was broken into fixed-sized chunks, which were all added to the requesting node's "work-queue." Each collaborating member queried the requester for the next chunk, downloaded the chunk by using the HTTP range header, sent the response back to the requester, and repeated the process until the work-queue was empty. This ensured that work was distributed proportional to the member's bandwidth (nodes with higher throughput would query more often for work loads than nodes with lower throughput). This was interesting because no interface characteristics were monitored. The system naturally accommodated for bandwidth differences because each device only queried for another chunk to load when its interface was idle. These authors also noted some important parameter tuning which will be discussed in section 5. Although very similar to the system implemented in this thesis, [13] was deployed and evaluated on several laptops as opposed to smartphones. The authors achieved linear speedups.

[29] is another technique for aggregating bandwidth among multiple devices. [29]

was focused on video streaming and placed much emphasis on routing within the ad-hoc network and ultimately increased the overall utilization rate by 3.5 times.

3.3 Contributions

As mentioned previously, most solutions were not actually implemented and evaluated on actual smartphone devices. Mobile environments are much different than desktop environments, which is the primary motivation for this thesis. The contributions of this thesis are:

- Using the information provided from previous work, I implemented bandwidth aggregation on a new platform: iOS.
- I conducted an exploration and comparison of what can be achieved on a smartphone as opposed to less restricting environments like desktops and laptops.
- I evaluated and identified bottlenecks in the system.

4 IMPLEMENTATION

To explore the bandwidth aggregation methods discussed in section 3, I implemented such a system, called Cell-Share, on top of the iOS platform. Cell-Share is an iOS application that allows the user to form local communities among nearby peers. This community of peers forms the pool of Internet connections that Cell-Share aggregates.

4.1 Overview

Given that the target platform was iOS, the bandwidth aggregation was implemented at the application layer, within the application itself. Due to the sandboxing constraints of iOS applications described earlier, it is not possible to insert middleware in-between the application and transport layers like many of the previous works have done.

Similar to [1, 12, 13], I focused entirely on the HTTP protocol which is the primary method of communication in the World Wide Web [20]. HTTP is a request-response protocol in which clients request resources identified by a unique URI and the corresponding server responds with a response payload.

Most approaches at the application layer [1, 12, 13, 27] evaluate the performance of the bandwidth aggregation for downloading a single payload. Cell-Share does the same, but to assess broader applicability, I also explored how bandwidth aggregation could benefit another use-case: web browsing. Webpages, in theory, lend themselves well to bandwidth aggregation in that webpages are composed of a single root object and several child objects (the average number of child objects measured in 2012 was

about 30 [20]) which can then be scheduled and distributed among all interfaces. The design and architecture of this system is described below.

The main goal of this thesis is to determine the maximum performance that iOS devices are capable of. Therefore, several other aspects of bandwidth aggregation across multiple devices mentioned in section 2.3 were not implemented. For instance, node mobility, security and user-incentives are left for future work.

4.2 Overall Architecture

The overall architecture of the system is the the same as [13] depicted by figure 3.

Each member sharing bandwidth belongs to an ad-hoc network where each device is exactly one hop away from any other member. The underlying technology used for the ad-hoc network can be Bluetooth or WIFI. This network is used for managing collaboration as well as relaying response data received from remote sources.

This system also assumes that each member belongs to a cellular network as well. The cellular interface is responsible for communicating over the Internet to the actual content servers hosting desired resources. No proxy server is needed and communication occurs directly from the iOS devices to the content server.

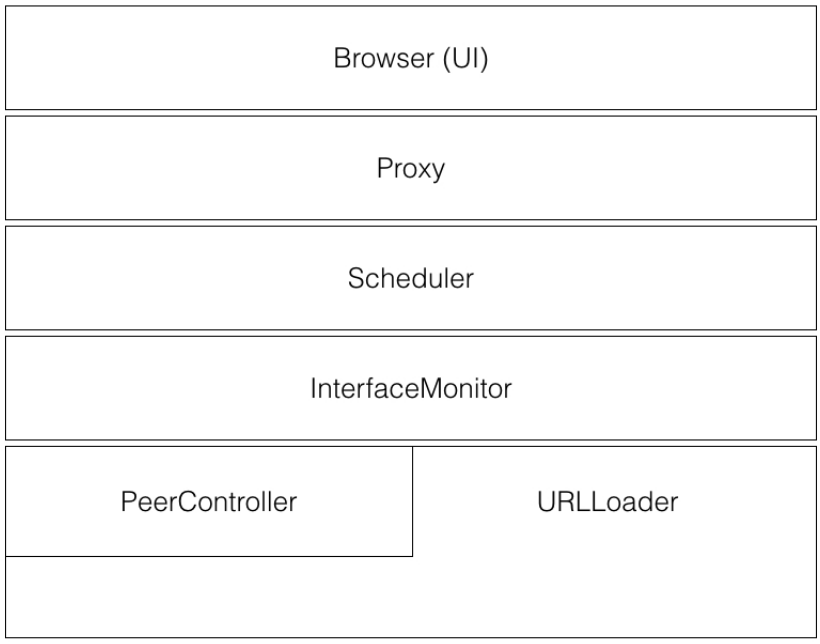


Figure 4: Cell-Share system components.

4.3 System Design

The system is designed to be modular so that each component can be swapped out with a different implementation, requiring minimal code changes to other components. The layered design is shown in figure 4 and figure 5 illustrates how all the components work together.

4.3.1 Browser

The Browser represents the user interface and is the top layer of the system, depicted by stage 1 in figure 4. The user-interface imitates a simple browser with a few additional functions. The user can select other nearby neighbors to connect with in

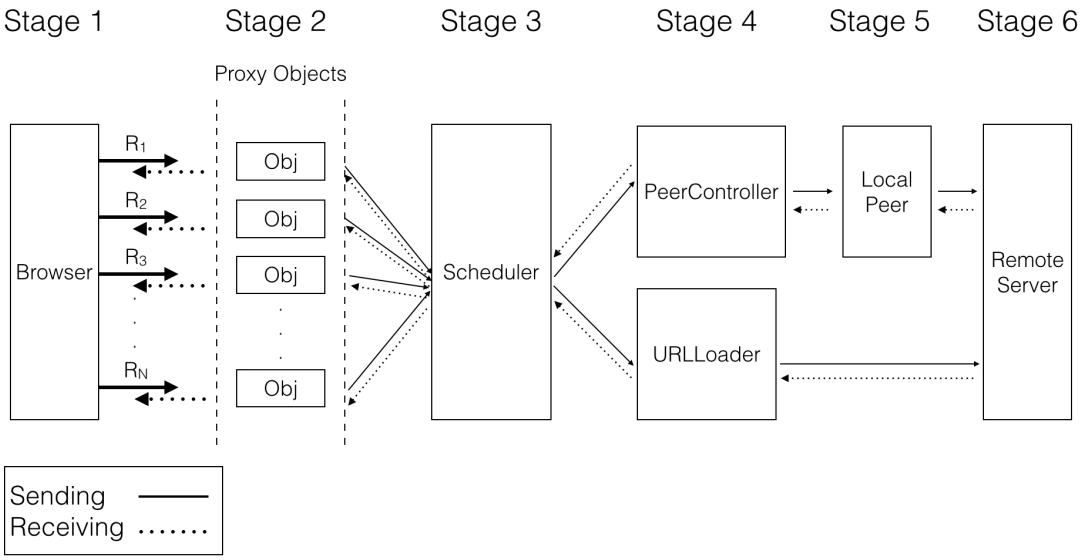


Figure 5: Cell-Share system design.

order to collaborate.

The browser also visualizes the current loading state (on each interface) for the current webpage or resource that is being loaded. This is accomplished by the InterfaceMonitor layer which periodically sends notifications to the UI informing it how many bytes have loaded and the observed bandwidth on each interface. The InterfaceMonitor is described further in section 4.3.6.

The browser uses the UIWebview object found in the UIKit framework of iOS. The UIWebview is built on top of WebKit[1] and handles all browser-related tasks such as parsing and rendering. The UI layer in Cell-Share simply allows the user to enter a URL (identifying either a webpage or a single file) into the search bar which is then passed to the UIWebview to be loaded, parsed and rendered.

In order to intercept the root and child HTTP requests generated by the UI-

[1]An opensource browser engine used by several commercial web browsers.

Webview, and thus schedule them among each available interface, a local proxy is implemented. The proxy is described in section 4.3.2.

4.3.2 Proxy

The proxy layer enables the system to intercept each request that is generated by the browser. This is key to Cell-Share when loading webpages. Because of the way that the UIWebview object encapsulates browser functionality, there is no way to control how subsequent child objects of a webpage are requested. By implementing a local proxy, I was able to leave all browser-related tasks to the UIWebview and focus entirely on the scheduling.

The proxy is a subclass of the NSURLProtocol abstract class. The class is instantiated once for every request that is made by the UIWebview, including the root object. Each instantiation receives the corresponding request and is responsible for retrieving it (in whatever way it wants; ex. from disk, from the network or neighboring peers) and delivering the response header and response body back to the UIWebview. It is here, at stage 2 of figure 4 where the system takes control over the request fulfillment and enables the scheduling decisions to be made.

To keep with the goal of modular design, the proxy hands off the request to the Scheduler. This is done asynchronously using the delegation design pattern. When the request has been fulfilled, the corresponding Proxy instantiation is notified by a callback method and can then notify the UIWebview.

4.3.3 Scheduler

The scheduler, aware of all interfaces, handles the scheduling logic. Several scheduling algorithms are supported and can be set via the application settings; each supported method is discussed in detail in section 5.

A list of interfaces is maintained by the scheduler and is periodically updated to accommodate for when members join or leave the ad-hoc network. This list always includes the "self" interface (to represent the cellular interface attached to the current user's device) and each collaborating member's device name.

As mentioned in section 2.2.1, in order to make informed scheduling decisions, all interfaces need to be monitored. The InterfaceMonitor is responsible for recording several details on each interface and can be queried by other layers to report the current status of each device and the corresponding interface. Thus, for the more intelligent scheduling algorithms, the scheduler can first consult the InterfaceMonitor.

The scheduler sits above the URLLoader and PeerController layers. Based on the scheduling decision made at stage 3 of figure 4, the request under consideration is passed to the corresponding layer. If the scheduling unit is to be loaded on the current device's own cellular connection, it is passed to the URLLoader, otherwise it is passed to the PeerController along with the chosen interface's name.

4.3.4 URLLoader

The URLLoader represents the cellular interface attached to a single device, shown at stage 4 of figure 4. The sole responsibility of this layer is to asynchronously fetch data

from the network using HTTP request messages. The delegation design pattern is used to incrementally and asynchronously inform the upper layer that data has been received and when the request has finished. As a request is processed, the URLLoader logs how much data has been received with the InterfaceMonitor.

4.3.5 PeerController

Also shown at stage 4 of figure 4, the PeerController represents the local, ad-hoc network. Again, this network is used to relay data payloads and to manage collaboration between members. This layer provides the same programming interface as the URLLoader does. Both layers abstract the request fulfillment complexities from the Scheduler. Table 1 shows the types of packets exchanged locally.

There are two types of entrances into the PeerController. The first is via the Scheduler, as discussed above. The second entrance occurs when a packet is received from a neighbor. The packet is first decoded and processed in accordance with the actions stated in table 1.

The reason for having several types of response packets is to support the pipelining of HTTP Response messages, which includes a response header and body. When a collaborating member initiates an HTTP request message to a content server on behalf of another device, there are several options for relaying the response to the requester, namely buffering the entire response or pipelining the transmission of response data to the requester, as it is received. In the former case, the ResponseBodyFull packet is used to relay both the response header and body at once. In the latter case, as soon as enough data has been received to parse the header, the ResponseHeader packet

Packet Type	Parameters	Direction	Actions Taken
Request	Request Type, Request Byte Range, URL	Initiator =>any member	Forward request to URLLoader. Upon completion, send response back to the peer who made the request.
Response Header	Request Type, Request Byte Range, URL, Content Length, Content Type, HTTP status code	Any member =>Initiator	Notify scheduler that the response header is available.
Response Body Full	Request Type, Request Byte Range, URL, Response Body	Any member =>Initiator	Notify the scheduler that the entire response body is available.
Response Body Segment	Request Type, Request Byte Range, URL, Response Body Segment	Any member =>Initiator	Notify the scheduler that the a portion of the response body is available.
Response All	Request Type, Request Byte Range, URL, Content Length, Content Type, HTTP status code, Response Body	Any member =>Initiator	Notify the scheduler that both the response header and content are available.
Response Error	Request Type, Request Byte Range, URL, error code, error domain	Any member =>Initiator	Notify the scheduler that a particular request failed with the given error.

Table 1: Packet types exchanged between members of local, ad-hoc network.

is relayed. Additionally, as the response body is received in chunks, the ResponseBodySegment packet is relayed. Similar to the URLLoader, the PeerController logs how much data has been received with the InterfaceMonitor.

4.3.6 Interface Monitor

The InterfaceMonitor is solely concerned with monitoring the status and characteristics of each interface. As mentioned in section 4.3.3, the InterfaceMonitor can be queried by any layer to provide characteristics for a given interface. Additionally, to provide feedback to the user, the InterfaceMonitor also notifies the UI throughout an entire loading event. The PeerController and URLLoader log these loading events with the InterfaceMonitor whenever data is sent or received. The following list identifies the information recorded (per interface) by the InterfaceMonitor:

- Interface/Device name (ex. "Bradley's iPhone")
- Total number of requests started
- Total number of requests completed
- A dictionary of request information which maps request URL to individual request information (start time, end time, total bytes received)
- Total number of bytes loaded
- Delay, calculated as the exponential weighted moving average (EWMA[2])

[2] $EstimatedBandwidth = (1 - \alpha) * EstimatedBandwidth + \alpha * SampleBandwidth$

- Current Bandwidth[3], sampled every 0.5 seconds for each interface. Let delta(i) represent the amount of bytes received on interface i since the last interval. The bandwidth is calculated as follows: delta(i) / 125000 / 0.5

- Average Bandwidth, same as above but calculated as the exponential moving weighed average

As I will explain in section 5.2, interface monitoring was actually not used by any scheduling algorithms in my implementation. However, this information was still needed to perform an analysis of the entire system. This information is logged and stored locally which can then be sent and analyzed by my machine.

[3] 125000 is used to change the unit from bytes to megabits

5 ALGORITHMS AND PARAMETER TUNING

This section discusses the algorithms and parameter tuning involved in Cell-Share.

5.1 Scheduling Granularity

Recall that the two forms of scheduling granularity are packet-oriented and connection-oriented. Because Cell-Share operates at the application layer and ease of deployability is a goal of this thesis, connection-oriented scheduling was adopted.

Within the context of connection-oriented scheduling and the HTTP protocol, resources identified by unique URIs are the basic unit of scheduling. When multiple resources are loaded simultaneously, as is the case for webpages, the scheduling-unit is an entire resource (web object). However, because loading a single file only consists of one HTTP request and because the HTTP protocol supports range requests (via the byte range header field), requests can further be segmented into chunks. This allows a single file download to be scheduled among multiple interfaces.

Some parameter tuning is necessary specifically when loading a single resource. Should the resource be equally divided by the number of collaborators and thus treaded the same as a webpage that contains multiple resources? Or should the resource be divided into smaller, fixed-sized chunks that can be scheduled dynamically? Both methods are implemented by Cell-Share and the performance results are given in section 6.

With regards to webpage scheduling, it is important to point out that the web objects can vary dramatically in size. For instance, a small cascading style sheet

might only be 10 kilobytes, whereas as an embedded image might be 2 megabytes. Thus, scheduling requests at the resource-level onto different interfaces may not be representative of the actual load the interface will be tasked with.

To demonstrate, consider the weighted round-robin scheduling algorithm. Say interface I_1 has an average bandwidth of 5mbps and I_2 has an average bandwidth of 1mbps. Furthermore, assume the weighted round-robin algorithm schedules 1 resource on I_2 for every 10 resources scheduled on I_1. The 10 resources assigned to I_1 may only amount to several hundred kilobytes if the resources are small web objects, whereas the single resource assigned to I_2 could amount to much more if the resource is a large image. Thus, scheduling at the resource-level is not necessarily indicative of the amount of bytes or load the interface will experience.

This is an interesting issue which could possibly render the scheduling algorithms useless. A simple and obvious solution could be to issue a HTTP HEAD request for each object to retrieve all object sizes. The object sizes could then be used to properly distribute object requests. This would pose a significant amount of overhead because it requires twice as many requests to be issued. For this reason, it was not implemented. Also, as we will see in section 6.2.3, the webpages were actually fairly uniform in object size distributions meaning that the majority of objects were comparable in size, reducing the concern of loading a given interface unproportionately.

5.2 Scheduling Algorithms

Cell-Share employs three scheduling algorithms: round-robin, a variant of round-robin that involves "pools" (referred to as RRIP) and a custom method used solely

for webpages that considers MIME types (MIME-Aware).

5.2.1 Round-Robin

As explained in section 2.2.3, this method was employed primarily as a baseline for comparison with other scheduling algorithms.

5.2.2 Round-Robin with Individual Pools (RRIP)

This technique is similar to the Round-Robin with Queues method described in section 2.2.3, but instead of queues, each interface has a "pool" of a statically chosen size with which scheduling-units are assigned to. Assignment occurs in a rotating fashion and work is scheduled on interfaces with the smallest amount of scheduling units in their pool. Interfaces can then pull these scheduling-units from their own pool sequentially or in parallel. The goal is to have a large enough pool so that interface bandwidth is not wasted and each interface can work on as many scheduling-units as they have the capacity for. If all pools are filled, scheduling-units are then assigned to a global pool where they wait for an available interface (an interface becomes available when it completes a loading task and removes an item from their corresponding pool).

By placing scheduling-units into a global pool where they wait, rather than continuing to assign them to full pools, the system has greater control over how many tasks an interface is assigned and can ensure that no interface is overrun. Note that this method implicitly handles varying interface capabilities since higher performing interfaces will be assigned more scheduling-units (because they will complete their

tasks at a faster rate than others).

Some parameter tuning is required to find the optimal thresholds for the pool sizes. If pool sizes are too small, resources may be wasted where pools spend a significant amount of time empty, waiting to be assigned a new scheduling-unit by the initiator. On the other hand, if pool sizes are too large, then the system's agility and control over scheduling is compromised. Several pool sizes were experimented with and the results are shown in section 6.2.2.

5.2.3 MIME-Aware

This method is targeted specifically at webpages. Recall how web browsers work, at a very high level: they first request the root object, often an HTML file. This HTML file is the root of the webpage and contains several child objects - identified by the parser - that must be requested in order to completely render the page. These child objects could be images, videos, javascript files, css files, etc. For each source file object, such as javascript or css, the parser may identify even more resources that are needed.

This scheduling algorithm attempts to load these requests first, and on the fastest interface available. The intent is to determine every resource that will be needed as quickly as possible so that all requests are known.

This problem is highlighted by the case where a source file is assigned to a slow interface. The subsequent resources that may be generated on behalf of that source file will not be scheduled until the source file completes loading. Any interface that is available and ready to load more requests during this time is wasted.

Thus, for each resource, the scheduler inspects the resource type (the MIME) and if it recognizes the type as a source file that could potentially generate more requests, it schedules the request on the "self" interface. The "self" interface is chosen so that the number of hops is minimized and the request and response do not have to be forwarded to and relayed by a neighboring peer. Note that if the "self" interface is significantly slower than its neighbors, the scheduling algorithm will not perform optimally. Obviously, in these cases, using a neighboring interface would be faster than using the "self" interface. This issue is not addressed in this thesis but device bandwidth was continuously monitored to ensure approximate uniformity throughout the performance evaluation.

As the secondary scheduling method, for non-source file objects, the RRIP method is used.

5.3 Network Interface Technologies

Cell-Share requires two network interfaces: one for loading data remotely and another for relaying data between peers locally. Ultimately, I chose to use the cellular interface to load data remotely and the WiFi interface for local communication. The reasoning for these decisions was based on the points outlined by [13], summarized below.

- Bluetooth and WiFi share partially overlapping sections of the 2.4 GHz ISM band which causes substantial interference. Also, WiFi and Bluetooth are often implemented on the same chip. For these reasons, the transmission rates for both WiFi and Bluetooth can be reduced when active at the same time. This was consistent with my initial experiments. Thus, using WiFi for loading data

remotely and Bluetooth for communicating locally is a poor combination. This leaves combining the cellular interface with either Bluetooth or WiFi as the only viable option.

- WiFi supports a larger number of connections and higher transmission rates when compared to Bluetooth.

6 PERFORMANCE EVALUATION

In this section, I discuss the tests performed in order to evaluate the performance of Cell-Share.

6.1 Experimental Testbed

The phones used for each experiment consisted of 1 iPhone 5 and 2 iPhone 5s devices. WiFi ad-hoc mode is only supported on iPhone 5 and above, so I was unable to test on older phones.

All phones were on Verizon's 3G or LTE network. I realize this is a significant limitation of the testbed because multiple carriers were not represented. The impacts of this limitation are discussed further in section 6.3.2.

For all tests, the phones were placed within several inches of each other and were stationary throughout. All tests were performed in an indoor-setting.

In order to measure the performance of the system, the primary metrics measured were bandwidth and latency. To measure bandwidth over time, the saved bandwidth samples calculated by the InterfaceMonitor were used. The latency was measured as the time between initiating a load for a resource (either a webpage or single file) and when that entire resource finished loading.

A link conditioner was also used in some cases. In order to estimate the impacts of aggregating cellular bandwidth across multiple carriers, Verizon's 3G and LTE networks were used in combination by the tested devices (the link conditioner was used to throttle down the LTE performance to match the 3G performance).

	apple.com	maps.google.com	surfline.com	theverge.com
Total Size (kb)	1985.84	1464.06	2043.26	5024.62
# of Objects	56	74	124	145
Max Object Size (kb)	221.42	422.32	196.01	294.03
Min Object Size (kb)	0.04	0.09	0.04	0.02
Mean Object Size (kb)	35.46	19.78	16.48	34.65
Median Object Size (kb)	8.52	7.43	5.45	6.50

Table 2: Webpage composition measurements.

Several tools were used to measure the performance of Cell-Share, namely Wireshark[4] and Instruments[5].

For single file downloads, a file of 10 megabytes stored on Dropbox[6] was used. For webpage loading, the following websites were used because of their wide variety of webpage composition: apple.com, surfline.com, maps.google.com. To demonstrate the varying page compositions, table 2 lists several key measurements for each website.

Lastly, I conducted a total of roughly 300 trials for loading both types of resources (webpages and a single file), using each of the three scheduling algorithms and with one, two and three peers. About 50 of these trials were for loading a single file and 250 trials were for loading each webpage. The averages for these trials are discussed in the following sections.

[4]Wireshark is network packet capturing program.

[5]Instruments comes bundled with the Xcode development suite. It allows for monitoring several aspects of iOS applications

[6]Dropbox.com is a widely used cloud storage platform.

	Phone 1 (3G)	Phone 2 (3G)	Phone 3 (LTE)
Individual (mbps)	1.56	1.26	1.43
Simultaneous (mbps)	0.80	0.71	1.44
Delta	-48.65%	-43.80%	0.63%

Table 3: Average bandwidths for each device.

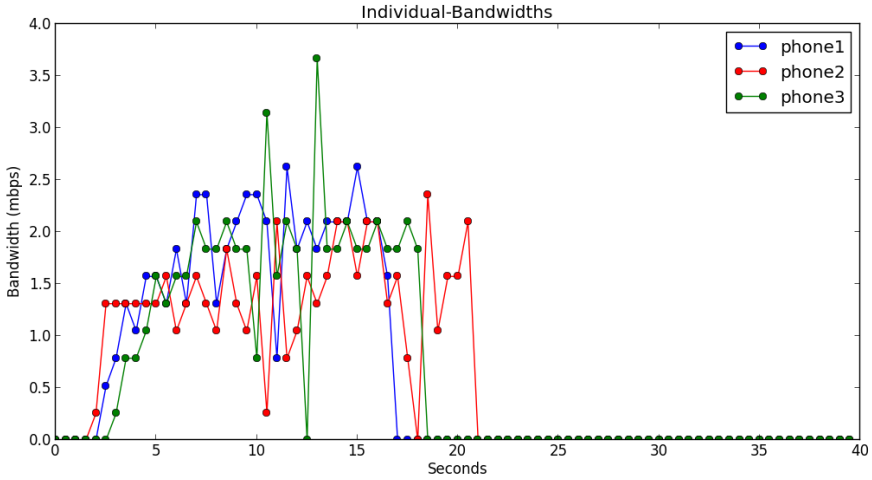
6.2 Results

6.2.1 Baselines

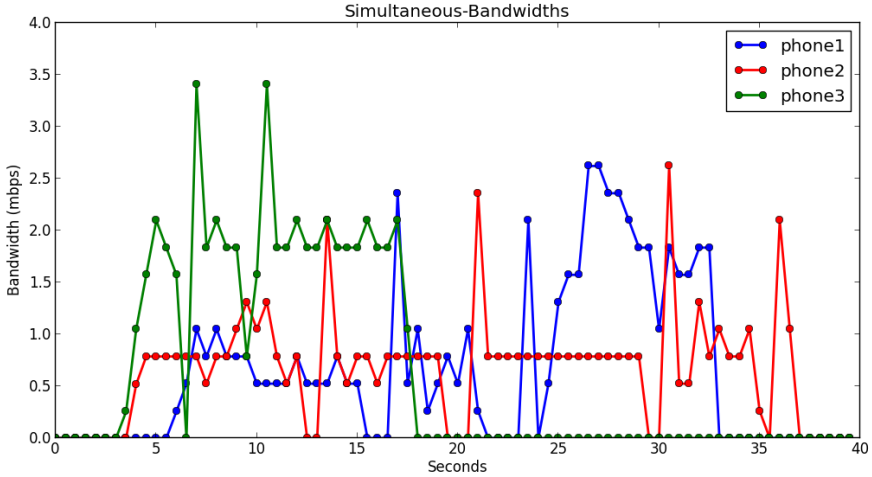
This section establishes a series of baseline measurements used for evaluating and comparing the performance results of Cell-Share.

First, I determined the rates at which Dropbox would serve the 10 megabyte file. Although not necessarily the maximum rate, using a wired connection on a standard home network, the file was downloaded at an average rate of 15.12 mbps.

Next, the bandwidth for each of the three phones was measured. This test was conducted to measure the bandwidth for each phone *individually* and *simultaneously*. For the individual case, each phone downloaded the 10 megabyte file from Dropbox at different times. For the simultaneous case, each phone downloaded the same 10 megabyte file from Dropbox at the same time. Phone1 and phone2 were using Verizon's 3G network while phone3 was using Verizon's LTE network (with the link conditioner simulating 3G performance). The bandwidth measured over time is shown in figure 6 and table 3 shows the average bandwidth per phone. Table 3 highlights an interesting finding where the throughput for the phones on the same network (3G) dropped by almost half when active at the same time. This is discussed further in section 6.3.2.



(a) Individual bandwidth.



(b) Simultaneous bandwidth.

Figure 6: Baseline bandwidth measured for each device.

6.2.2 Single File Downloads

The goal of these experiments was to measure the (hopefully) increase in bandwidth as additional devices were added, for downloading a single, 10 megabyte file.

As discussed in section 5, there are several parameters that need to be tuned:

- Static vs. Dynamic Scheduling

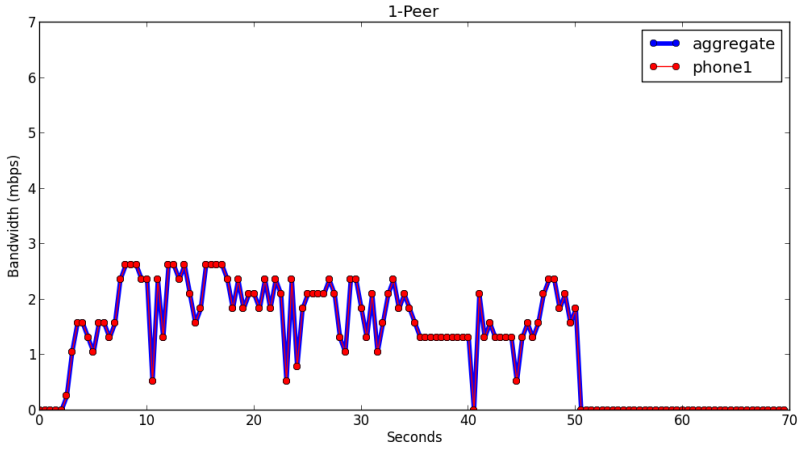
- Chunk size (in the case of dynamic scheduling)
- Pool sizes (in the case of dynamic scheduling)

6.2.2.1 Static vs. Dynamic Scheduling

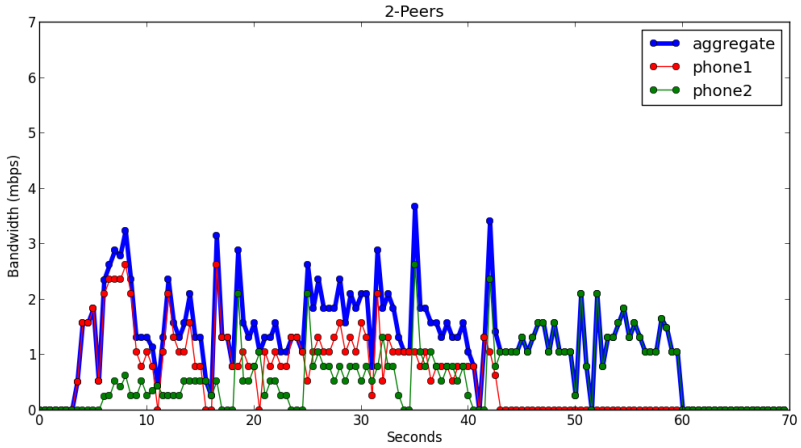
Intuitively, dynamic scheduling would seem to offer the best performance results as the workload for each interface is proportional to its capabilities, as was the case highlighted by figures 7 and 8.

Because static scheduling simply divides the workload evenly and ignores all interface performance characteristics, interfaces with poor bandwidth will slow the entire system. In figure 7, phone1 and phone3 finished loading their portion of the file around 30 seconds, while phone2 took an additional 25 seconds. Note, that when a particular phone's bandwidth flatlines, it has completed all of the work it has been assigned. For 25 seconds of the entire loading time, phone1 and phone3 were not utilized and their resources were wasted. This demonstrates the need to schedule work dynamically so that work is assigned proportional to each interface's capability and to keep all interfaces busy.

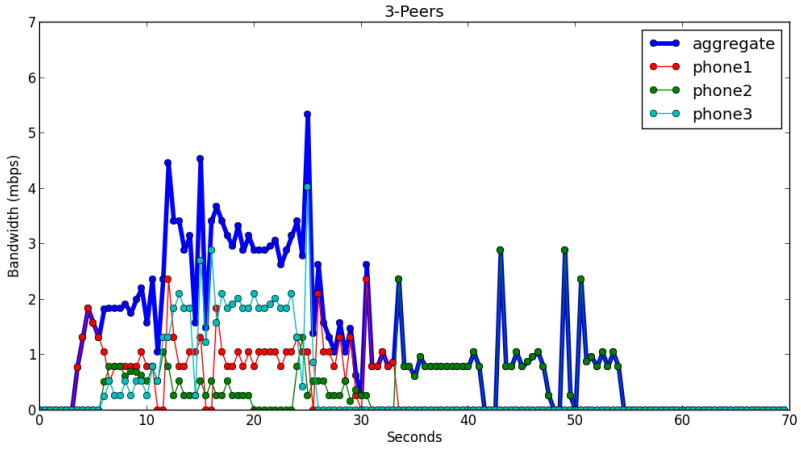
Alternatively, figure 8 shows the results of using the dynamic variant of round robin, RRIP. The un-proportional scheduling of work with RR lead to a large disparity in the times in which interfaces finished their work load. Because RRIP implicitly accounted for the interface performance, the amount of work scheduled to each interface was proportional. Thus, we see that each interface finished its work at roughly the same times, leading to significant speedups.



(a) Bandwidth for 1 device.

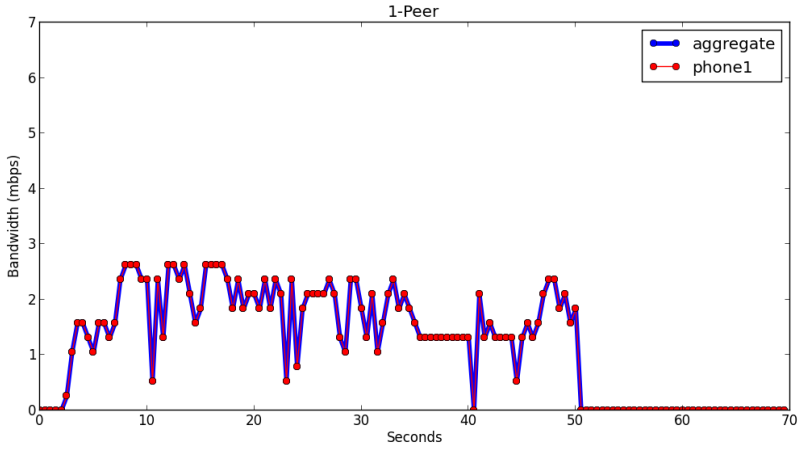


(b) Aggregate bandwidth for 2 devices.

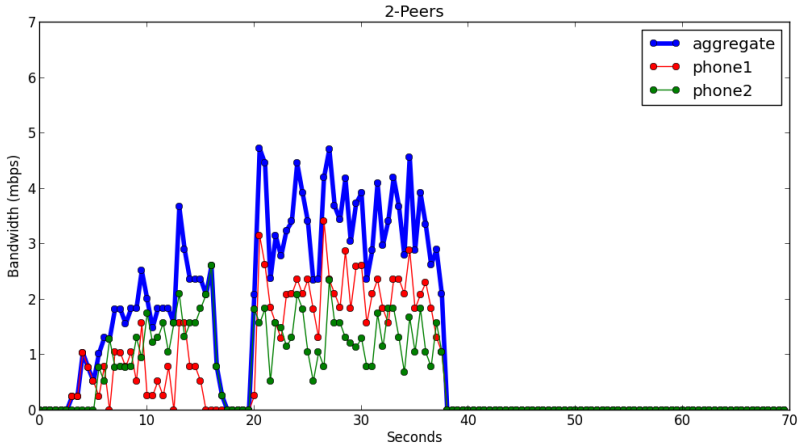


(c) Aggregate bandwidth for 3 devices.

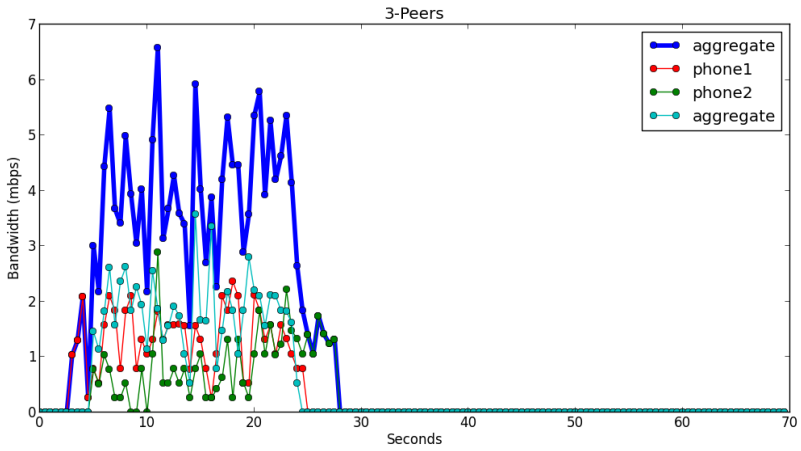
Figure 7: Aggregate bandwidth using the static scheduling method for loading a single, 10mb file.



(a) Bandwidth for 1 device.



(b) Aggregate bandwidth for 2 devices.



(c) Aggregate bandwidth for 3 devices.

Figure 8: Aggregate bandwidth using the dynamic scheduling method for loading a single, 10mb file.

6.2.2.2 Pool Size

Another parameter that needed to be tuned was the pool size for each interface. The pool sizes manage how many tasks an interface can "work on" at a given time. As mentioned in section 5.2.2, larger pools reduce the system's control over scheduling while smaller pools may not allow the system to fully utilize the network bandwidth of each interface.

[19] describes the optimal number of persistent, parallel TCP connections to a single server as 6-8. The authors mention how these values are employed by several popular web browsers such as Mozilla Firefox and Internet Explorer. For these reasons, I set the pool size of each interface to 8.

This means that the cellular interface on each device can make 8 requests in parallel. As soon as a request returns, the interface removes the request from its pool, allowing the scheduler to assign it another request.

6.2.2.3 Chunk-Size

With regards to chunk size, [1] identified that a value of 200kb yielded the best results. The authors described how a high value for chunk-size would negatively affect the agility and failure-handling of the system while a low value would impose too much overhead and reduce the overall throughput due to the high costs of opening and closing HTTP connections. These tests were conducted over 6 years ago, so the values were re-measured. A chunk-size of 200kb was used as a starting point to find the best possible chunk-size for Cell-Share in modern cellular networks.

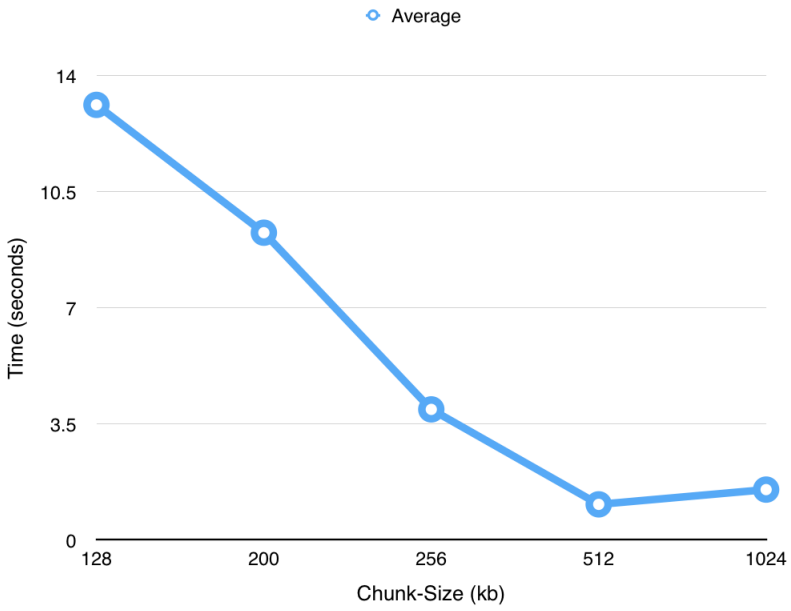


Figure 9: Time spent for connection establishment. Increasing the chunk-size reduces the amount of connection overhead.

Figure 9 shows the results of my own measurements and how different chunk-sizes affected the amount of connection overhead incurred. As the chunk-size increased, the number of requests required to download the entire file was reduced which in turn, reduced the number of connections opened. Thus, fewer connections means that less time was spent establishing them and ultimately the overall overhead was reduced.

However, remember that the trade-off for having low overhead is that the system cannot respond to varying interface performance as quickly. For this reason and because the requested file size is only 10mb, a chunk-size of 256kb was chosen. Ideally, the chunk-size would adjust dynamically based on the requested file size; this is left for future work.

6.2.3 Webpage Downloads

These experiments focused on how Cell-Share affected the loading times of the webpages mentioned above. As discussed in section 5, there were a few parameters that needed to be analyzed:

- RR vs. RRIP vs. Mime-Aware Scheduling Methods
- Pool sizes (in the case of the RRIP scheduling method)
- Load Distributions

Overall, the load times of webpages proved to be much more difficult to speed up. In many test runs, adding additional collaborators did not affect the performance of the system.

6.2.3.1 Scheduling Algorithms

The MIME-Aware scheduling method performed the best. By scheduling all source files onto the the initiator's interface (not needing to be relayed by neighbors), all the parsing could take place as soon as possible in order to generate all requests. When using other schedulers, there were several test runs where only one interface was busy, loading a source file. Once that request finished loading, the source file was parsed and further requests were identified and then scheduled on to other interfaces. This created an obvious bottleneck as other interfaces were starved for work, which the MIME-Aware method successfully minimized.

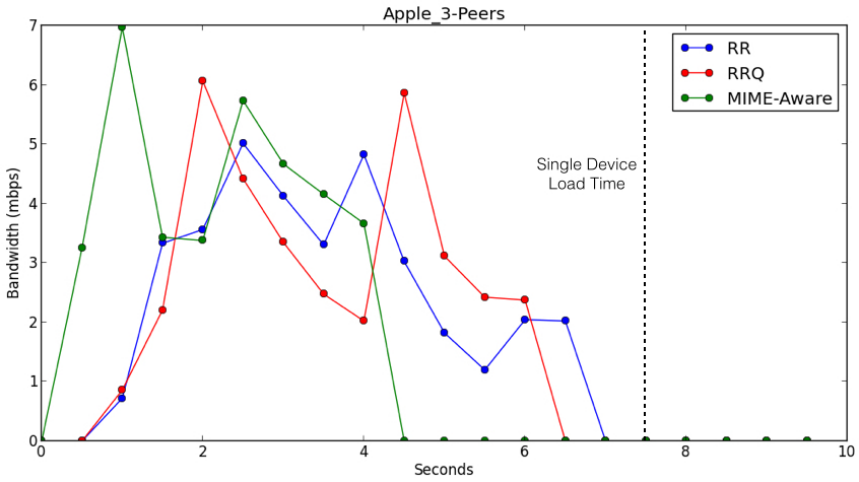
The RR method suffered from the same issues discussed in section 6.2.2 where interfaces were not fully utilized due to the work being assigned equally, regardless of interface performance. The RRIP scheduling method helped improve the loading times but by less of a margin than Mime-Aware. Overall, as mentioned above, the speed ups produced by the RRIP and Mime-Aware methods were much less significant than the improvements acheived with single file downloads; the results are shown in figure 10.

6.2.3.2 Pool Size

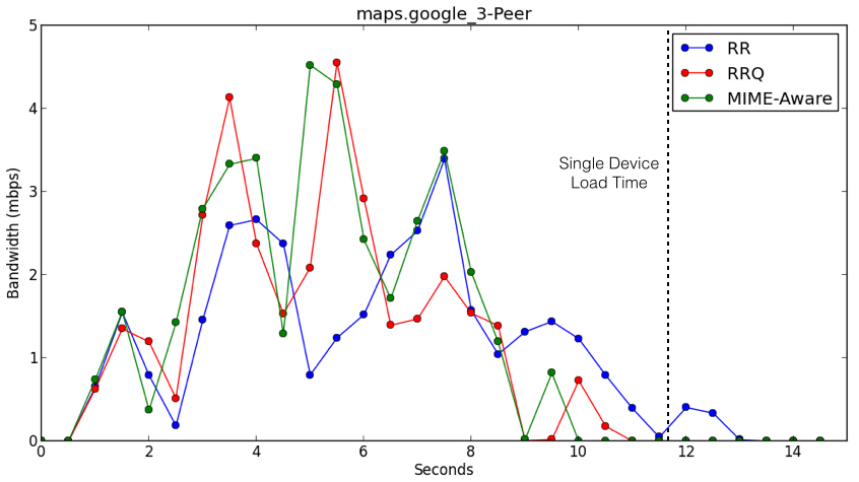
For the reasons outlined in section 6.2.2, pool sizes were set to 8.

6.2.3.3 Load Distributions

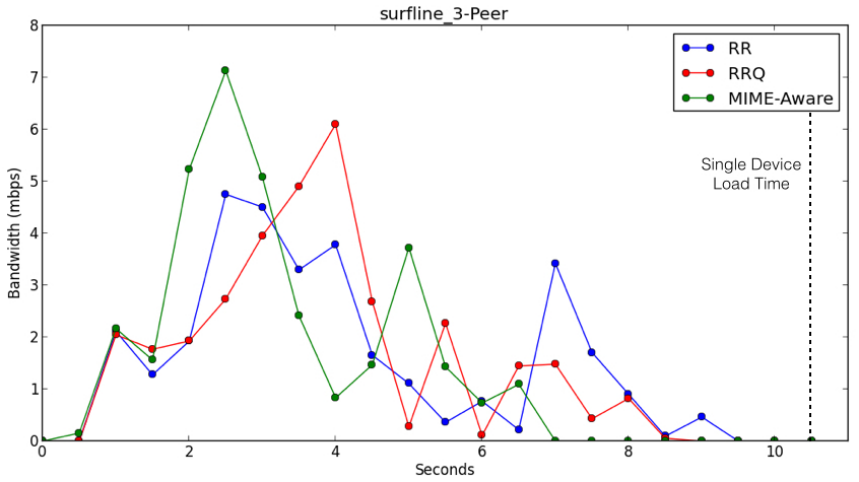
Again, the concern here was that distributing work at the request-level was not necessarily proportional to the load experienced by each interface (because of the varying response sizes). For instance, in the RRIP and Mime-Aware methods, work is assigned proportional to the interfaces' capabilities. If the basis for scheduling relies solely on the the number of requests, the scheduler may not perform as expected.



(a) Aggregate bandwidth for loading apple.com.



(b) Aggregate bandwidth for loading maps.google.com.



(c) Aggregate bandwidth for loading surfline.com.

Figure 10: Aggregate bandwidth for 3 devices, loading different webpages and using each of the scheduling algorithms.

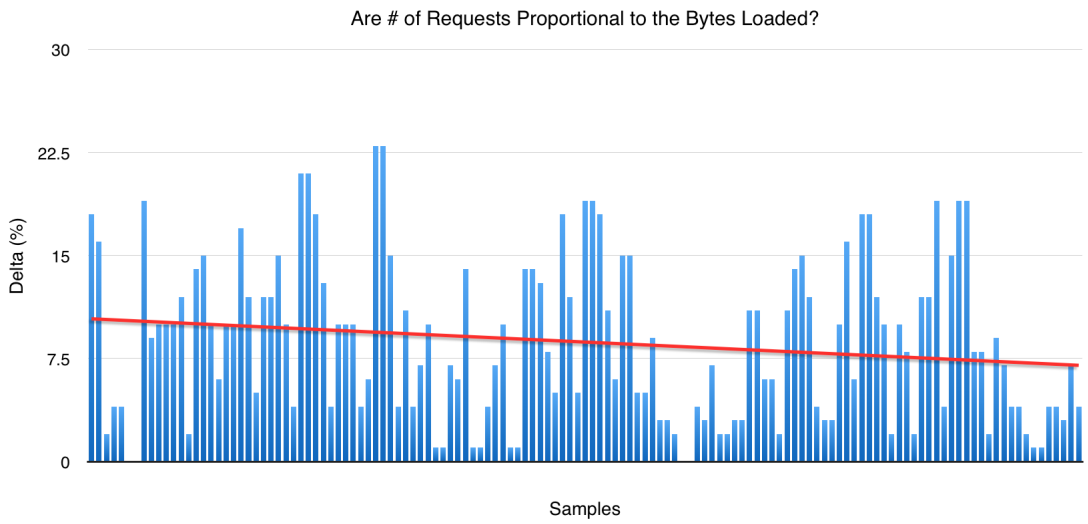


Figure 11: The scheduled requests ratios and the actual bytes loaded are not necessarily proportional. The differences between the two ratios (for each test run using the RRQ scheduling method) are shown by each bar in the above graph.

To see if this posed a real problem, I examined the ratios between the number of requests to the amount of bytes loaded for each interface. The request ratio and bytes ratio were measured for each interface, i, for several test runs (shown in equations 1 and 2).

$$RequestRatio_i = ScheduledRequests_i / TotalNumberRequests \tag{1}$$

$$BytesRatio_i = BytesLoaded_i / TotalNumberOfBytes \tag{2}$$

Ideally, the difference between the two ratios would be close to 0, indicating that scheduling at the request-level would not be a significant problem. Figure 11 shows the differences between these two ratios, for several test runs using the RRIP scheduling method; the average stayed under 10%. This suggested that the concern was fairly negligible and scheduling at the request-level was a reasonable distribution technique.

6.3 Analysis

While noticeable gains in data rates were evident, Cell-Share did not provide the dramatic performance improvement I originally anticipated. With scheduling optimized (performed in a proportional manner), and each interface fully saturated throughout the download, the aggregated data rates were appreciably higher than the individual rates. The performance, however, did not grow linearly as the number of collaborators grew, as was observed in [1, 13].

The data rate improvement was even less noticeable when loading web pages. It is worth noting that all prior work similar to this system ([1, 13, 27]), focused solely on single, large files. Transferring a single, large file is a best-case scenario for networking because the communication is half-duplex and the server can send the file as fast as the network will allow (spending the majority of the time out of the TCP slow-start phase). On the other hand, websites tend to be much smaller in size and contain several individual request that will likely be made over multiple connections (spending most of their time in the TCP slow-start phase). As the payload gets smaller and smaller, the performance of aggregating bandwidth gets worse and worse. The additional overhead of scheduling requests, tracking them, and parsing responses from the ad-hoc network seem to overshadow all performance gains.

Some possible causes for the sub-optimal performance are identified below and discussed in the following sections:

- Operating System Limitations
- Cellular Connection Interference
- CPU Bottleneck

6.3.1 Operating System Limitations

As mentioned in section 2.3.1, [13] noticed that Android was actually turning off cellular connections when a WiFi connection began concurrently, in order to conserve battery power. To see if iOS employed a similar mechanism, a packet trace was gathered during simultaneous use of the WiFi and cellular interfaces. Figure 12 shows the throughput for each interface for two situations:

1. Loading a large file over the cellular interface while a WiFi / ad-hoc connection was established between two devices (stage A, solid blue line); halfway through the cellular download, the same file was requested over the WiFi interface (stage D, dotted green line).
2. Loading a large file over the cellular interface with no other connections (stage A, dashed yellow line). Note that stages B-D do not apply to this experiment.

When interfaces were active at the same time, a dead period lasting roughly 30 seconds (shown between stages B and C) occurred for the cellular connection; no data was received during this period. The packet traces showed that at stage B,

several small packets (78 bytes) were exchanged between the two devices connected via the WiFi network. The interfaces that were communicating were labeled "AWDL0" (Apple Wireless Direct Link). The information exchanged during this period of time is unknown, as the AWDL interface has very little documentation. However, it is clear that regardless of the exchanged data, the cellular connection was effectively paused and the throughput dropped to zero. Once the activity on the AWDL0 interface ceased, the cellular connection resumed.

Interestingly, at stage D where the same file was requested from the neighboring device over the WiFi interface (AWDL0), the cellular connection remained unchanged. Obviously, the cellular connection is not turned off in *all* cases when the AWDL0 interface is active simultaneously. It could be that vital connection management information was being exchanged between stages B and D, and the OS paused all other connections to accommodate this exchange. Determination of the actual source of the cellular connection pause is left for future work but ultimately, this will have very negative affects on the performance of the system.

6.3.2 Cellular Connection Interference

In a general case, for both single file downloads and webpages, the individual bandwidths were reduced when multiple peers, using the same cellular network, collaborated at once. Table 3 shows the differences in bandwidth when requests were made simultaneously. Peer3, in all cases was on Verizon's LTE network, and its bandwidth was more-or-less consistent throughout.

Several potential causes for the significant drops in throughput shown in table 3

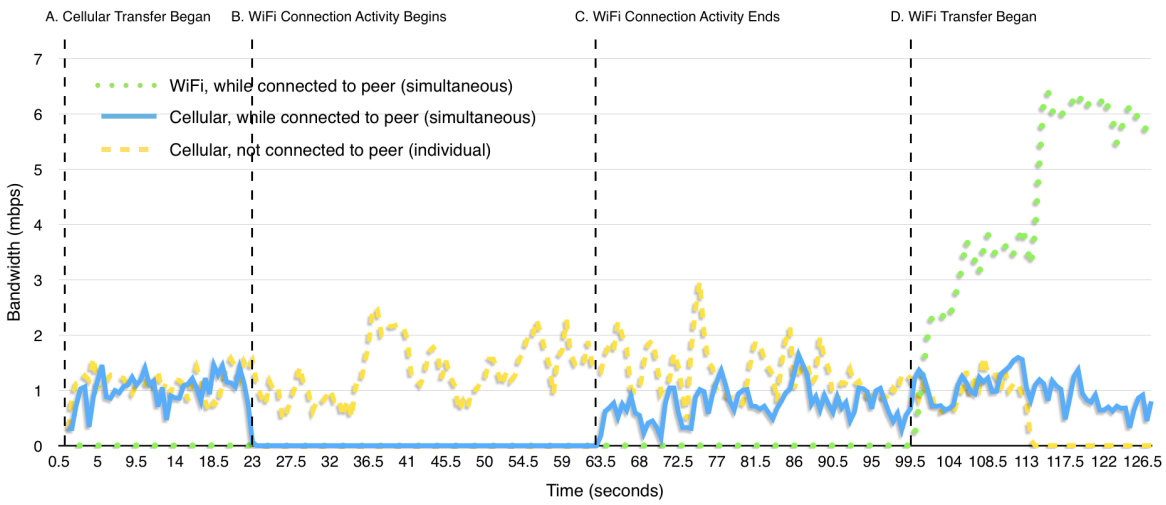


Figure 12: Throughput over time while using multiple interfaces simultaneously.

were identified, all of which seem to be associated with competing access to the BTSs. This is managed by MAC protocols like DS-WCDMA and TDMA. [25] explains how multiple versions of CDMA, namely CDMA2000 and WCDMA, can be used in adjacent frequency bands within the same area, referred to as a 3G coexistence network. The authors showed how the presence of the CDMA2000 system generated significant interference for the WCDMA system, reducing the WCDMA system's coverage and capacity. Note that this is not guaranteed to be the case all of the time, since mobile nodes are re-allocated different portions of the shared medium frequently. This was consistent with my findings in that these bandwidth reductions varied over time; some test runs showed less significant reductions, while others showed more significant reductions.

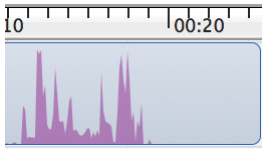
Because the multiple devices participating in Cell-Share may be allocated the interfering adjacent frequency bands described above, the throughput for each device would interfere with each other, which would explain the reductions showed in table 3.

	UI	Cellular	WiFi	Overhead
Single File, No Collaboration	53%	40%	0%	4%
Single File, w/ Collaboration	32%	16%	39%	12%
Webpage, No Collaboration	75%	20%	0%	4%
Webpage, w/ Collaboration	68%	8%	13%	10%

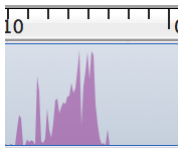
Table 4: Breakdown of CPU processing tasks.

The ideal scenario for bandwidth aggregation would therefore contain several members, each using different cellular networks, such as Verizon, AT&T, Sprint, etc. This would ensure that each device communicated with a distinct BTS, which would eliminate the interference associated with any given BTS.

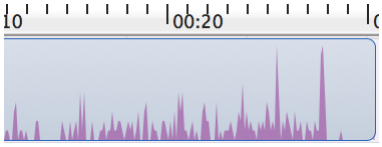
6.3.3 CPU Bottleneck



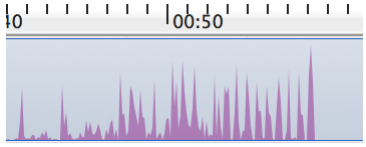
(a) Webpage, no collaboration.



(b) Webpage, with collaboration.



(c) Single file, no collaboration.



(d) Single file, with collaboration.

Figure 13: CPU Usage

Lastly, I examined the CPU usage throughout the lifetime of Cell-Share to ensure that the CPU was not a bottleneck. As shown in table 13, the CPU was never fully exhausted, confirming that the CPU was not a bottleneck.

Additionally, the aggregation overhead imposed by Cell-Share never exceeded 12% of the execution time. Table 4 shows rough estimates on how much time (relative to the total execution time) was spent for particular types of processing.

7 CONCLUSION

My primary goal of this thesis was to implement a bandwidth aggregation system on the iPhone. To validate the system, called Cell-Share, I attempted to reduce the loading times for single file downloads as well as for webpage downloads. My results show that iOS is capable of supporting such a system, but the bandwidth improvements are not linear, like they were in [1, 13]. I implemented several scheduling algorithms and evaluated each of their performances on up to three phones.

For single file downloads, the dynamic RRIP scheduling method was the best choice because work was scheduled proportional to what the interface could handle.

For webpage loading, I found that leveraging application layer information, combined with the RRIP scheduling method, proved to perform the best (Mime-Aware). Source files, such as css and javascript, were prioritized and scheduled on the "self" interface which allowed the browser parser to generate all requests at the fastest rate possible to avoid interface starvation.

The best, overall speedups are shown in figure 14.

7.1 Future Work

With regards to the Cell-Share implementation, several aspects of bandwidth aggregation were not employed. To name a few, handling node mobility where collaborating members abruptly leave an ad-hoc network, security, privacy and user incentives are functions that would benefit Cell-Share and improve its robustness.

Section 6.2.2 discussed how a chunk-size of 256kb was used. Ideally, the chunk-size

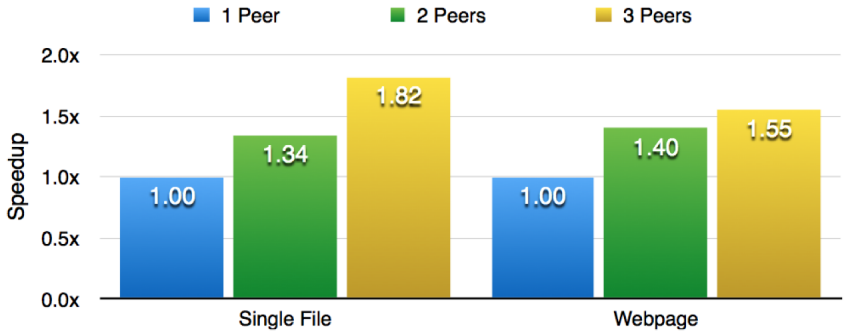


Figure 14: Best speedups observed for increasing number of devices. RRIP was the most successful algorithm used for the single file download whereas the MIME-aware algorithm was the most successful for webpage loading.

would be small enough to allow the system to adjust quickly to varying interface capabilities yet large enough to minimize TCP overhead. The "small enough" description given in the previous sentence is surely relative to the size of the requested file. For instance, a "small" chunk-size when downloading a 10kb file looks very different compared to the "small" chunk-size when downloading a 100mb file. Thus, adjusting the chunk-size dynamically based on the requested file size would also benefit Cell-Share.

The last item left for future work is further analysis into the AWDL0 interface. Remember that this interface carries the ad-hoc WiFi data between neighboring nodes and, in some situations, actually paused the cellular connections. This issue requires some further investigation in order to identify any solutions.

REFERENCES

[1] Ganesh Ananthanarayanan, Venkata N Padmanabhan, Lenin Ravindranath, and Chandramohan A Thekkath. Combine: leveraging the power of wireless peers through collaborative downloading. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 286–298. ACM, 2007.

[2] Apple, Inc. *iOS App Programming Guide: Performance Tuning*. URL: `https://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/PerformanceTuning/PerformanceTuning.html`.

[3] C Casetti and W Gaiotto. Westwood sctp: load balancing over multipaths using bandwidth-aware source scheduling. In *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, volume 4, pages 3025–3029. IEEE, 2004.

[4] Kristian Evensen, Dominik Kaspar, Paal Engelstad, Audun Fosselie Hansen, Carsten Griwodz, and Pål Halvorsen. A network-layer proxy for bandwidth aggregation and reduction of ip packet reordering. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, pages 585–592. IEEE, 2009.

[5] Kristian Evensen, Dominik Kaspar, Carsten Griwodz, Pål Halvorsen, Audun Hansen, and Paal Engelstad. Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 57–68. ACM, 2011.

[6] Karim Habak, Khaled A Harras, and Moustafa Youssef. Operetta: An optimal energy efficient bandwidth aggregation system. In *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, pages 121–129. IEEE, 2012.

[7] Karim Habak, Khaled A Harras, and Moustafa Youssef. Bandwidth aggregation techniques in heterogeneous multi-homed devices: A survey. *arXiv preprint arXiv:1309.0542*, 2013.

[8] Karim Habak, Moustafa Youssef, and Khaled A Harras. Dbas: A deployable bandwidth aggregation system. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–6. IEEE, 2012.

[9] Adiseshu Hari, George Varghese, and Guru Parulkar. An architecture for packet-striping protocols. *ACM Transactions on Computer Systems (TOCS)*, 17(4):249–287, 1999.

[10] Brett D Higgins, Azarias Reda, Timur Alperovich, Jason Flinn, Thomas J Giuli, Brian Noble, and David Watson. Intentional networking: opportunistic exploitation of mobile network diversity. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 73–84. ACM, 2010.

[11] Hung-Yun Hsieh and Raghupathy Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. *Wireless Networks*, 11(1-2):99–114, 2005. Springer.

[12] Dominik Kaspar, Kristian Evensen, Paal Engelstad, and Audun Fosselie Hansen. Using http pipelining to improve progressive download over multiple heterogeneous interfaces. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[13] Lorenzo Keller, Anh Le, Blerim Cici, Hulya Seferoglu, Christina Fragouli, and Athina Markopoulou. Microcast: Cooperative video streaming on smartphones. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 57–70. ACM, 2012.

[14] Kyu-Han Kim and Kang G Shin. Prism: improving the performance of inverse-multiplexed tcp in wireless networks. *Mobile Computing, IEEE Transactions on*, 6(12):1297–1312, 2007.

[15] Jim Kurose and Keith Ross. *Computer Networking, A Top-Down Approach 6th Edition*, chapter Wireless and Mobile Networks. Pearson Education, Inc., One Lake Street, Upper Saddle River, New Jersey 07458, 2003.

[16] Anand Lal Shimpi. The iphone 5s review, 2013. URL: http://www.anandtech.com/show/7335/the-iphone-5s-review/2, Accessed: 2013-11-16.

[17] Luiz Magalhaes and Robin Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *Network Protocols, 2001. Ninth International Conference on*, pages 165–171. IEEE, 2001.

[18] Kshirasagar Naik. A survey of software based energy saving methodologies for handheld wireless communication devices. Technical report, Department of Electrical and Computer Engineering, University of Waterloo, 2010.

[19] Preethi Natarajan, Fred Baker, and Paul Amer. Multiple tcp connections improve http throughput–myth or fact? *TR2008-333, Department of Computer & Information Sciences, University of Delaware, USA*, 2008.

[20] Rastin Pries, Zsolt Magyari, and Phuoc Tran-Gia. An http web traffic model based on the top one million visited web pages. In *Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on*, pages 133–139. IEEE, 2012.

[21] Pablo Rodriguez, Rajiv Chakravorty, Julian Chesterfield, Ian Pratt, and Suman Banerjee. Mar: A commuter router infrastructure for the mobile internet. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 217–230. ACM, 2004.

[22] Puneet Sharma, Sung-Ju Lee, Jack Brassil, and Kang G Shin. Aggregating bandwidth for multihomed mobile collaborative communities. *Mobile Computing, IEEE Transactions on*, 6(3):280–296, 2007.

[23] Aaron Smith. Smartphone ownership in 2013, 2013. URL: http://pewinternet.org/Reports/2013/Smartphone-Ownership-2013.aspx, Accessed: 2013-09-16.

[24] Mark Sullivan. 3g and 4g wireless speed showdown: Which networks are fastest?, 2012. URL:

http://www.pcworld.com/article/253808/3g_and_4g_wireless_speed_showdown_which_networks_are_fastest_.html, Accessed: 2013-03-16.

[25] Muhammad Suryanegara, Edwardo Rizky Hutabarat, and Dadang Gunawan. The interference on wcdma system in 3g coexistence network. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–5. IEEE, 2006.

[26] Atsuo Tachibana and Teruyuki Hasegawa. A deployable scheme of cmt-sctp with off-the-shelf android smartphones. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pages 861–868. IEEE, 2012.

[27] Takafumi Takiguchi, Akira Hidaka, Hiromu Masui, Yoshio Sugizaki, Osamu Mizuno, and Koichi Asatani. A new application-level link aggregation and its implementation on android terminals. In *Wireless Communications and Mobile Computing. Proceedings. 12th International Conference on*, pages 8–12. IEEE, 2012.

[28] Cheng-Lin Tsao and Raghupathy Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 337–348. ACM, 2009.

[29] Danyu Zhu, Matt W Mutka, and Zhiwei Cen. Qos aware wireless bandwidth aggregation (qawba) by integrating cellular and ad-hoc networks. In *Proceedings*

of the 1st IEEE internation conference on Quality of Service in Heterogenous Wired/Wireless Networks (QSHINE), pages 156–163. IEEE, 2004.