Performance and Power Characterization of Cellular Networks and Mobile Application Optimizations

by

Junxian Huang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2013

Doctoral Committee:

Associate Professor Z. Morley Mao, Chair
Associate Professor Jason N. Flinn
Associate Professor Robert Dick
Technical Staff Subhabrata Sen, AT&T Labs Research

To my family and my love.

ACKNOWLEDGEMENTS

I became a graduate student in the University of Michigan in the fall of 2008. Five years have passed and I would like to thank many people during this long and unique Ph.D. journey in my life. The completion of this dissertation would not be possible without anyone of them.

First of all, I would like to sincerely thank my advisor, Professor Z. Morley Mao. I first met her when I was an intern in Microsoft Research Asia in 2007, wondering about my future, and she offered me much helpful advice for pursuing a Ph.D. degree in the U.S. Morley has provided me excellent and professional guidance on various research projects throughout my Ph.D. study, given her expertise in the computer networking and mobile computing, as well as delicious birthday cakes, when even I forgot my birthday myself. Moreover, she helped me gain the confidence, ability and most importantly, the enthusiasm to tackle difficult real-world problems. Her diligence and enthusiasm has set a great model for me and the experiences working with her would be my life-long treasure.

I am greatly thankful for my mentors Kun Tan and Yongguang Zhang in Microsoft Research Asia (Beijing 2007), Yinglian Xie and Fang Yu in Microsoft Research Silicon Valley (California 2010), Subhabrata Sen and Oliver Spatscheck in AT&T Labs - Research (New Jersey 2012). I am fortunate to work with these knowledgeable and excellent researchers. I enjoyed my internships, which would be my unforgotten memories. I also enjoyed the conversations with them about exciting research ideas, and about career and life as well. Many of their words have kept ringing in my mind, helping me to improve.

I would like to express my deep gratitude to Professor Jason Flinn, Dr. Subhabrata Sen

and Professor Robert Dick for serving as my thesis committees. Their valuable suggestions and comments have greatly helped me improve this dissertation. My grateful thanks are also due to Professor Scott Mahlke, Professor J. Alex Halderman for their excellent teaching on my core computer science courses, and Dr. Ming Zhang and Dr. Paramvir Bahl for their priceless help for me in writing and presenting the my first conference paper [1].

It is hard for me to forget the great professors during my undergraduate study in Tsinghua University who taught me the basics in computer science and helped me find the joy in this area. Their names include Xiping Mao, Wei Chen, Hong Wang, Jianhua Feng, Zhongxin Xu (who passed away in 2010 and is long remembered), Shimin Hu, Ning Su, *etc.* I owe my special thanks to Professor Andrew Chi-Chih Yao, who included me in his first special pivot class, where I have received great education in computer science, and he himself also demonstrated what a great computer scientist and a Turing Award winner is like.

I would like to thank all my friends and colleagues at Michigan. I have worked closely with Dr Feng Qian and Qiang Xu since the first day of my Ph.D. study. Dr. Ying Zhang, Dr. Xu Chen and Dr. Zhiyun Qian are former members of our research group and they are also my good friends. I am grateful for my roommate Lujun Fang. My thanks also go to my other friends and colleagues in the Computer Science department at Michigan, including, but not limited to Bin Liu, Yunjing Xu, Yudong Gao, Birjodh Tiwana, Zhaoguang Wang, Lide Zhang, Mark Gordon, Caoxie Zhang, Fangjian Jin, Zhe Chen, Kee Shen Quah, Li Qian, Sanae Rosen, Bangxin Hu, Jie Yu, Xinyu Zhang, Timur Alperovich, Yihua Guo, Yuanyuan Zhou, Haokun Luo, Qi Chen. I am thankful for University of Michigan Table Tennis Club (UMTTC) and University of Michigan Chinese Drama Club for bringing me many good days and good friends: Weilun Xu, Hang Zhao, Allen Hung, Ason Chiang, Bo Bi, Yang Zhou, Tomas Fuentes-Afflick, Lang Ming, Yuhua Wang, Beimin Zhu, *etc.* I want to thank my piano teachers Alexandra Lynelle James, Yuanchang Zhou, Shuai Wang, and my table tennis coaches David Kleshchik, Naihui Liu, Qing Miao for helping me fulfill

my childhood dreams. I would also like to thank my other friends outside Michigan for appearing in and shaping my life: Jianfeng Gao, Qi Xie, Wentao Han, Jingyue Wu, Hui Min, Ruizhe Li, Chengwei Guo, Yifang Yuan, Jiajie Zhu, Haohui Mai, Yini Xu, Ning Ding, Shan Zhou, Zhaojie Cheng, Dongdong Peng, Simin Zhang, Tian Guo, Yi Chen, Di Zhang, Zhenqiang Gong, *etc*.

I could not love University of Michigan more. "For today, Goodbye; For tomorrow, Good Luck; Forever, GO BLUE!" I will remember Ann Arbor, the most beautiful town in the world. I will miss Mr. CSE raccoon, Mr. North-Campus turkey, unnamed geese and deer, and I will be a fighting wolverine wherever I go!

Finally, I would like to express my deepest thanks to my beloved parents, Xuemeng Huang and Zhaoxia Wang, and my dearest grandparents Shu Huang, Lingzhi Liu, Shengqin Wang (who passed away in 2008 to my greatest sadness) and Xiuhua Ma for their enduring love, support and encouragement throughout my life. This dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

ABSTRACT

Performance and Power Characterization of Cellular Networks and Mobile Application Optimizations

by

Junxian Huang

Chair: Z. Morley Mao

Smartphones with cellular data access have become increasingly popular with the wide variety of mobile applications. However, the performance and power footprint of these mobile applications are not well-understood, and due to the unawareness of the cellular specific characteristics, many of these applications are causing inefficient radio resource and device energy usage. In this dissertation, we aim at providing a suite of systematic methodology and tools to better understand the performance and power characteristics of cellular networks (3G and the new LTE 4G networks) and the mobile applications relying upon, and to optimize the mobile application design based on this understanding.

We have built the MobiPerf tool to understand the characteristics of cellular networks. With this knowledge, we make detailed analysis on smartphone application performance via controlled experiments and via a large-scale data set from one major U.S. cellular carrier. To understand the power footprint of mobile applications, we have derived comprehensive power models for different network types and characterize radio energy usage of various smartphone applications via both controlled experiments and 7-month-long traces col-

lected from 20 real users. Specifically, we characterize the radio and energy impact of the network traffic generated when the phone screen is off and propose the screen-aware traffic optimization. In addition to shedding light to the mobile application design throughout our characterization analysis, we further design and implement a real optimization system RadioProphet, which uses historical traffic features to make predictions and intelligently deallocate radio resource for improved radio and energy efficiency.

CHAPTER I

Introduction

Smartphones with cellular data access have become increasingly popular across the globe, with the wide deployment of 3G and emerging LTE [2] networks, and a plethora of applications of all kinds. In the third quarter of 2009, the global smartphone shipments reached 41.4 million units [3]. As of the third quarter of 2012, the global smartphone shipments reached 173.7 million [4] with 61.3% year-on-year increase in average. It is expected that in the next few years smartphone sales will continue to grow. Vendors, such as Samsung, Apple, and HTC offer a variety of smartphones equipped with increasingly faster CPUs and larger memory, though still lagging behind desktop or laptop systems. With access to various high-speed 3G networks, such as EVDO and UMTS, and the LTE 4G networks, they are powerful enough to run modern operating systems and sophisticated network applications such as web browsing, email, and streaming media. However, the performance and power characteristics of these smartphone applications are not well-understood, and many of these applications are inefficient in radio resource and energy usage, which mainly attribute to the unawareness of the cellular specific characteristics. In order to solve this challenge, in this dissertation, ***we devise a suite of systematic methodology and tools to accurately measure the performance and power characteristics of cellular networks and mobile applications, and to optimize the mobile application design.***

Unlike traditional Internet-based applications, whose performance is mostly constrained

by the wired network, network application performance on smartphones with limited physical resources also heavily depends on factors including hardware and software on the phone as well as the quality and load of wireless link. Understanding the application performance on smartphones is important for the purpose of assisting consumers in choosing carriers and phones and guiding application developers in designing intelligent software. Moreover, cellular network operators and smartphone hardware and software vendors can use this knowledge to optimize networks and phones for better end-user experiences. Similarly, content providers can leverage this knowledge to better customize content for mobile users. However, this task is quite challenging since the performance of network applications on smartphones is poorly understood thus far, due to a lack of a systematic approach for controlled experiments and comparative analysis, and especially because the network performance of the underlying cellular networks is not well understood. In this thesis, we take one of the first steps to thoroughly study the performance of cellular networks and smartphone applications.

In addition to network performance aspect, we also study the energy footprint for smartphone applications. Todays cellular systems operate under diverse resource constraints: limited frequency spectrum, network processing capability, and handset battery life. Optimizing energy footprint for smartphone applications is important for end-users. In the meanwhile, optimizing the radio resource usage is of great interest for mobile operators to minimize cost and guarantee quality of service.

In this dissertation, we dedicate five chapters to present our study in this space and these chapters are broadly classified into three categories as summarized in Table 1.1. Chapter III and Chapter VI discuss the network and power characterization of cellular networks and smartphones. Chapter IV and Chapter V focus more on mobile application behaviors and the interactions between mobile applications and cellular networks. Chapter VII presents our work in optimizing resource utilization efficiency for mobile applications in cellular networks.

Chapter(s)	Content category
Chapter III, Chapter VI	Network and power characterization
Chapter IV, Chapter V	Application performance and interplay with network protocol
Chapter VII	Optimization to improve resource utilization efficiency

Table 1.1: Roadmap.

1.1 Characterizing Cellular Network Performance

Since 2008, We have been working on devising systematical methodologies and developing tools for characterizing cellular network performance directly from end users. The tools developed includes 3GTest [5], 4GTest [6] and *MobiPerf* [7] [1], which have cumulatively over 150,000 users from over 190 countries or regions. In these measurement tools, we have devised methods to accurately measure round-trip time (RTT), DNS lookup time, uplink/downlink bandwidth, loss rate, and other network performance metrics for 3G, WiMAX and LTE 4G networks and compare those with WiFi networks.

Our study is among one of the first studies of the network characteristics of commercial LTE networks. Initiated in 2004 by *3rd Generation Partnership Project* (3GPP), the *Long Term Evolution* (LTE), commonly referred to as a type of 4G wireless service, aims at enhancing the *Universal Terrestrial Radio Access Network* (UTRAN) and optimizing radio access architecture [2]. Since 2009, LTE starts entering the commercial markets and is available now in more than 10 countries, with an expectedly fast-growing user base. The targeted user throughput is 100Mbps for downlink and 50Mbps for uplink, significantly higher than the existing 3G networks, with less than 5ms user-plane latency [11]. Understanding the actual user-perceived network performance for LTE network and how it compares with its predecessor 3G and its competitors, *e.g.,* WiFi and WiMAX, is important, yet not straightforward. Our forementioned tool 4GTest, with enhanced measurement design and global server support, allows us to characterize network performance of LTE

[1] Notably, *MobiPerf* has received both the *Open Internet App Award* and the *People's Choice App Award* in the *FCC Open Internet Apps Challenge* [8]. It is now an open-source project [9] that are being actively worked on, with joint collaboration among University of Michigan, M-Lab [10] and University of Washington.

and other mobile networks [12].

1.2 Anatomizing Smartphone Application Performance

In order to understand the key factors that affect smartphone application performance, we develop a systematic methodology for comparing this performance along several key dimensions such as carrier networks, device capabilities, and server configurations [1]. We perform detailed analysis to help carriers, phone vendors, content providers, and application developers gain insight. For example, for carriers, we infer various network level problems, *e.g.,* high latency or high loss rate, which they can directly take action on. For phone vendors, we identify performance bottlenecks on the devices or issues associated with the content. These issues can be resolved either independently or by cooperating with content providers. And for application developers, we evaluate factors such as the overhead of HTML rendering and Javascript execution given a particular software configuration.

Compared with 3G, LTE significantly improves the network performance. Meanwhile, device processing capability and software design have also improved remarkably over the last two years [1]. To understand the potential performance bottleneck shift for smartphone applications, we perform longitudinal case studies of several popular applications on Android, especially for web browsing applications. With the help of CPU, network and power traces, we compare the determinant factors on smartphone applications in 2009 [1] and in 2011 [12]. We identify that the performance bottleneck for web-based applications lies more in the devices processing power than in the network for the LTE networks.

1.3 Studying Effect of Network Protocol and Application Behavior on Performance for LTE Networks

Despite it fast increasing user base, the interplay between mobile applications, protocol and the network for the commercial LTE networks still remain unexplored. We thoroughly

study these topics of the LTE network with a data set covering around 300,000 real LTE users in a large metropolitan area for 10 days. We revisit basic network metrics in the LTE network and compare with previously studied network conditions. We also observe that a high downstream queueing delay, likely due to bufferbloat, has caused TCP congestion window collapse upon one packet loss. With the help of TCP Timestamps option, we have devised a lightweight passive bandwidth estimation algorithm, allowing us to observe that for 71.26% of the large flows, the bandwidth utilization ratio is below 50%. We find that TCP may not fully utilize the fast-varying available bandwidth when RTT is large in the LTE network. Upon further analysis, we identify 52.61% of all downlink TCP flows have been throttled by TCP receive window and data transfer patterns for some popular applications are both energy and network unfriendly.

1.4 Understanding Power Characteristics of 4G LTE Networks

Besides higher bit rate, lower latency and many other service offerings for LTE, *user equipment* (UE) power saving is an important issue and there has been increasing interest in understanding the power characteristics of LTE networks, compared with 3G/WiFi networks. LTE employs *Orthogonal Frequency Division Multiplex* (OFDM [13]) technology, which suffers from poor power efficiency. To save power, LTE uplink uses an special implementation of OFDM called SC-FDMA for uplink, with improved power efficiency. *Discontinuous reception* (DRX) has been employed by existing wireless mobile networks to reduce UE energy consumption. In UMTS [14], during the idle state, UE periodically wakes up to check paging messages and sleeps for the remaining time. LTE supports DRX for both **RRC_CONNECTED** and **RRC_IDLE** modes [15], seeking more opportunities to conserve UE battery. DRX is configured at a per-UE basis and its configuration incurs tradeoff among UE power saving, channel scheduling delay, and signaling overhead.

To understand this tradeoff, existing studies use either total DRX-on time to estimate UE power usage [16, 17], or a simplified LTE power model [18, 19], which ignores the im-

pact of downlink/uplink data rates. In this paper, we develop the first empirically derived comprehensive power model of a commercial LTE network, which accurately models UE energy usage with less than 6% error rate. Also, existing studies [16, 17, 18, 19] heavily rely on synthetic packet models instead of real user traces. Our study is the first that leverages a comprehensive real user data set, we call UMICH, consisting of 5-month traces of 20 smartphone users, to analyze the impact of LTE parameter configuration on realistic application usage patterns. We carefully investigate the energy usage in 3G, LTE, and WiFi networks and evaluate the impact of configuring LTE-related parameters. Despite several new power saving improvements, we find that LTE is as much as 23 times less power efficient compared with WiFi, and even less power efficient than 3G, based on the user traces and the long high power tail is found to be a key contributor.

1.5 Optimizing Energy Usage in Cellular Networks

With the knowledge of performance and power characteristics of 3G/4G cellular networks, we study how we can optimize the resource (radio and energy) utilization of smartphone applications. Cellular networks are typically characterized by limited radio resources and significant device power consumption for network communications. The battery capacity of smartphones cannot be easily improved due to physical constraints in size and weight. Hence, battery life remains a key determinant of end-user experience. Given the limited radio resources in these networks and device battery capacity constraints, optimizing the usage of these resources is critical for cellular carriers and application developers. Achieving such energy efficiency for mobile devices when connected to cellular networks without incurring excessive network signaling overhead, even despite diverse application and user behavior, still remains a rather difficult and yet important challenge to tackle. Energy use due to network access, particularly cellular networks, is becoming increasingly dominant due to numerous network-based smartphone applications. In many cases, achieving network energy savings must reside on the mobile device's OS to effectively and cen-

trally manage the data scheduling decisions transparent to applications and with minimal changes to the network.

The key mechanism that determines the energy consumed by cellular network interface is the radio resource control (RRC) state machine [20] pre-defined by carriers (covered in more details in Section 2.1) that governs when radio resources are acquired and released. Previous studies [21, 20, 22, 12] have shown that the origins of low resource efficiency comes from the way radio resources are *released*. Radio resources are only released after an idle time (*a.k.a.* Radio Resource Control (RRC) tail [21]) controlled by a statically configured inactivity timer. The tail is necessary and important for cellular networks to prevent frequent state promotions (resource allocation), which can cause unacceptably long delays for the UE, as well as additional processing overheads for the radio access network [23, 24]. During the tail time, radio energy is essentially wasted. Values as large as 11.6 seconds are configured [12] in current networks, contributing to about half of the total radio energy on user handsets (UEs) spent in idle times for common usage scenarios.

Without knowing when network traffic will occur, large tail timer settings are essentially a conservative way to ensure low signaling overhead due to state transitions, as signaling is known to be a bottleneck for cellular networks. Furthermore, they also help minimize performance impact experienced by users caused by state promotion delays incurred whenever radio resource is acquired. Given that application and user behavior are not random, using a statically configured inactivity timer is clearly suboptimal. Smaller static timer values would help reduce radio energy, but is not an option due to the risk of overloading cellular networks caused by signaling load increase.

An attractive alternative is to configure the timer dynamically — adaptively performing radio resource release either signaled by the UE or triggered by the network itself by monitoring the UE traffic, accommodating different traffic patterns, improving the overall resource efficiency. But the key challenge is determining *when* to release resources, which essentially comes down to accurate and efficient prediction of the idle time period.

Clearly, the best time to do so is when the UE is about to experience a long idle time period, otherwise the incurred resource allocation overhead (*i.e.,* signaling load) is wasteful due to unnecessary radio state transitions, and the achieved resource savings are very small. Therefore, accurate and efficient prediction of the idle time period is a critical prerequisite for dynamic timer schemes.

We propose RadioProphet (RP), a practical system that makes dynamic decisions to deallocate radio resources based on accurate and efficient prediction of network idle times. Using 7-month-long real-world cellular traces, we comprehensively evaluate it using various traffic features and machine learning algorithms. Properly configured, it correctly predicts 85.88% of idle time instances, achieving radio energy savings of 59.07%, at the cost of 91.01% of additional signaling overhead, significantly outperforming existing proposals. It incurs negligible energy overhead and has fast response times, demonstrating the practicality of deploying the system on contemporary smartphones.

Besides, we also consider a novel angle to the above problem orthogonal to RP and explore the impact of **screen status**, *i.e.,* whether the screen is on or off, on the device's network traffic patterns. We find that off-screen traffic accounts for 58.5% of the total radio energy consumption although their traffic volume contribution is much smaller. Such unexpected results are attributed to the unique cellular resource management policy that is not well understood by developers, leading to cellular unfriendly mobile apps. We then make a further step by proposing screen-aware optimization, given that the screen status is easy to monitor for most mobile OSes. We propose that the screen-off traffic should not be treated the same as the screen-on traffic for traffic optimization purposes, and the former can be optimized more aggressively. The main intuition is that the user (and possibly application) behavior have significant differences when the screen is on *v.s.* off, resulting in different traffic patterns and different performance requirements. When the screen is off, there is a much higher chance that the user is not actively interacting with the device and the network traffic is most likely to be more delay tolerant. Hence we can be more aggressive in

optimizing this traffic using techniques such as batching and fast dormancy. In contrast, when the screen is on, it is harder to predict the delay sensitivity of the network traffic and aggressive optimizations may harm the user experience. To validate this intuition, we characterize the screen-off traffic for a real-world user data set and evaluate the benefits of using "screen-aware" optimization for balancing UE energy savings and the resulting overheads in radio resource usage and response delay. The proposed screen-aware optimization focuses on the overall device traffic, and is complementary to other efficiency improvement efforts, *e.g.,* better application design. Our proposal can better balance the key tradeoffs in cellular networks.

1.6 Thesis Organization

This dissertation is structured as follows. Chapter II provides necessary backgrounds for resource management in cellular networks. We present the design of MobiPerf tool and network performance characterization in Chapter III followed by the smartphone application performance study in Chapter IV. We study the effect of network protocol and application behavior on performance for one large commercial LTE Network in Chapter V. We analyze the power characterization of smartphone applications in Chapter VI and present the design, implementation and evaluation for the smartphone energy optimization system RP in Chapter VII. We summarize the related works in Chapter VIII, before concluding in Chapter IX.

CHAPTER II

Background

This section provides sufficient background on resource management in cellular networks, especially for the LTE networks.

2.1 Radio Resource Control (RRC) State Machine

To efficiently utilize the limited resources, cellular networks employ a resource management policy distinguishing them from wired and Wi-Fi networks. In particular, there is a radio resource control (RRC) state machine [20] that determines radio resource usage based on application traffic patterns, affecting device energy consumption and user experience. Similar RRC state machines exist in different types of cellular networks such as UMTS [20], EvDO [25] and 4G LTE networks [12], although the detailed state transition models may differ.

RRC States. In 3G UMTS networks, there are usually three RRC states [20, 26]. **`RRC_IDLE`** is the default state when the UE is turned on, with no radio resource allocated. **`CELL_DCH`** is the high-power state enabling high-speed data transmission. **`CELL_FACH`** is the low-power state in between allowing only low-speed data transmission. In 4G LTE networks, the low-power state is eliminated due to its extremely low bandwidth (less than 20 kbps) so there are only two RRC states named **`RRC_CONNECTED`** and **`RRC_IDLE`** [27, 28].

State Transitions. As shown in Figure 2.1, regardless of the specific state transition

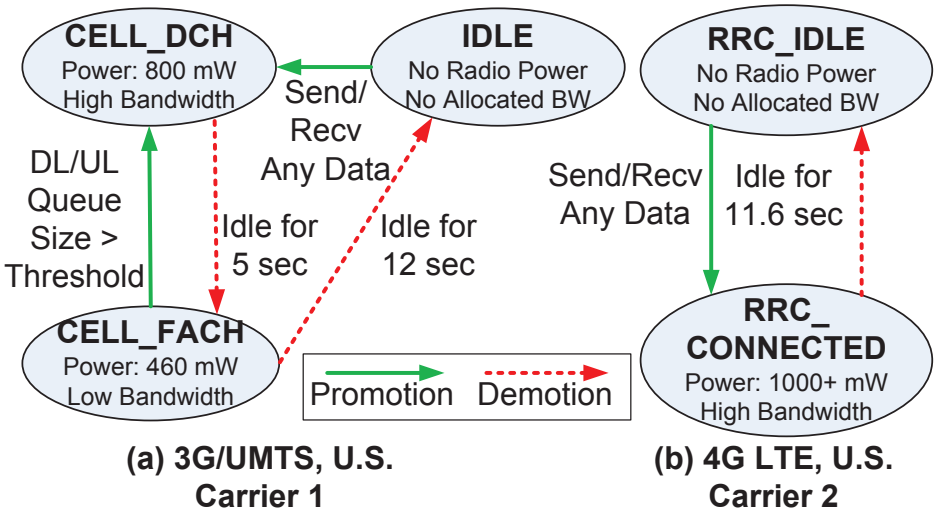


Figure 2.1: RRC State Machine of (a) a large 3G UMTS carrier in the U.S., and (b) a large 4G LTE carrier in the U.S. The radio power consumption was measured by a power monitor on (a) an HTC TyTn II smartphone, (b) an HTC ThunderBolt smartphone.

model, there are two types of state transitions. State promotions switch from a low-power state to a high-power state. They are triggered by user data transmission in either direction. State demotions go in the reverse direction, triggered by inactivity timers configured by the radio access network (RAN). For example, as shown in Figure 2.1b, at the **RRC_CONNECTED** state, the RAN resets the **RRC_CONNECTED** $\rightarrow$ **RRC_IDLE** timer to a constant threshold T_{tail}=11.6 seconds whenever it observes any data frame. If there is no user data transmission for T_{tail} seconds, the **RRC_CONNECTED** $\rightarrow$ **RRC_IDLE** timer expires and the state is demoted to **RRC_IDLE**. The two timers in 3G UMTS networks use similar schemes (Figure 2.1a).

State promotions and demotions incur *promotion delays* and *tail times*, respectively, which distinguish cellular networks from other types of access networks.

State promotions incur a long "ramp-up" delay of up to several seconds during which tens of control messages are exchanged between the UE and the radio access network (RAN) for resource allocation. Excessive state promotions increase the signaling overhead at the RAN and degrade user experience, especially for short data transfers [29, 22].

State demotions incur *tail times* that cause waste of radio resources and the UE en-

ergy [21, 20]. A tail is the idle time period matching the inactivity timer value before a state demotion, *e.g.,* the tail time is 11.6 seconds in Figure 2.1b. During the tail time, the UE still occupies transmission channels, and its radio power consumption is kept at the corresponding level of the RRC state. As an example of the negative impact of the tail effect, periodically transferring small data bursts (*e.g.,* every one minute) can be extremely resource-inefficient in cellular networks due to the long tail appended to each periodic transfer instance which is small in size and short in duration [30].

Adaptive Release of Radio Resources. Why are tail times necessary? First, the overhead of resource allocation (*i.e.,* state promotions) is high and tail times prevent frequent allocation and deallocation of radio resources. Second, the current radio resource network has no easy way of predicting the network idle time of the UE, so it conservatively appends a tail to every network usage period. This naturally gives rise to the idea of letting the UE actively request for resource release: once an imminent long idle time period is predicted, the UE can actively notify the RAN to immediately perform a state demotion. Based on this intuition, a feature called fast dormancy has been proposed to be included in 3GPP Release 7 [31] and Release 8 [32]. It allows the UE to send a control message to the RAN to immediately demote the RRC state to `RRC_IDLE` (or a hibernating state called `CELL_PCH`) without experiencing the tail time. Fast dormancy is currently supported by several handsets [32], which can dramatically reduce the radio resource and the UE energy usage while the potential penalty is the increased signaling load when used aggressively [29, 22]. In this work we propose robust methodology for predicting the idle time period, enabling more effective usage of fast dormancy.

2.2 RRC and Discontinuous Reception (DRX) in LTE

In this section, we provide more details for RRC in LTE in addition to Section 2.1, as well as Discontinuous Reception (DRX) mechanisms in LTE.

As shown in Figure 2.2, at `RRC_CONNECTED` state, UE can be in one of the three modes:

Symbol	Full name	Measured value	Description
T_i	DRX inactivity timer	100ms	UE stays in Continuous Reception for T_i before DRX starts when idling
T_{is}	Short DRX cycle timer	20ms	UE remains in Short DRX for T_{is} before entering Long DRX when idling
T_{tail}	RRC inactivity timer	11.576s	UE stays in RRC_CONNECTED for T_{tail} before demoting to RRC_IDLE
T_{on}	RRC_CONNECTED On Duration timer	1ms	The on duration of UE during each DRX cycle in RRC_CONNECTED
T_{oni}	RRC_IDLE On Duration timer	43ms	The on duration of UE during each DRX cycle in RRC_IDLE
T_{ps}	Short DRX cycle	20ms	The cycle period of Short DRX in RRC_CONNECTED
T_{pl}	Long DRX cycle	40ms	The cycle period of Long DRX in RRC_CONNECTED
T_{pi}	RRC_IDLE DRX cycle	1.28s	The cycle period of DRX in RRC_IDLE

Table 2.1: Important LTE RRC and DRX parameters.

Continuous Reception, Short DRX, and Long DRX. While at **RRC_IDLE** state, UE is only in DRX mode. Table 2.1 enumerates a list of important LTE parameters, which have significant impact on UE's radio energy consumption, user experience, and signaling overhead for cell towers. The terms in Table 2.1 are used consistently throughout this paper.

If UE is initially in **RRC_IDLE** state and receives/sends one packet, regardless of the packet size, a state promotion from **RRC_IDLE** to **RRC_CONNECTED** occurs with a relatively stable delay, similar to the promotion from IDLE to DCH/FACH in UTMS network [20]. We define the LTE promotion delay to be T_{pro}[1] consistently throughout this paper. During this period, radio resources are allocated to the UE.

After being promoted to **RRC_CONNECTED**, UE enters Continuous Reception by default and keeps monitoring the *Physical Downlink Control Channel* (PDCCH), which delivers control messages to UE. UE also starts the DRX inactivity timer T_i, which is reset every time UE receives/sends a packet. Upon T_i's expiration without seeing any data activity, UE enters the Short DRX mode.

Discontinuous Reception (DRX) [15, 33], illustrated in Figure 2.3, is adopted by LTE

[1] T_{pro} is a measured system property, different from the configurable LTE parameters in Table 2.1.

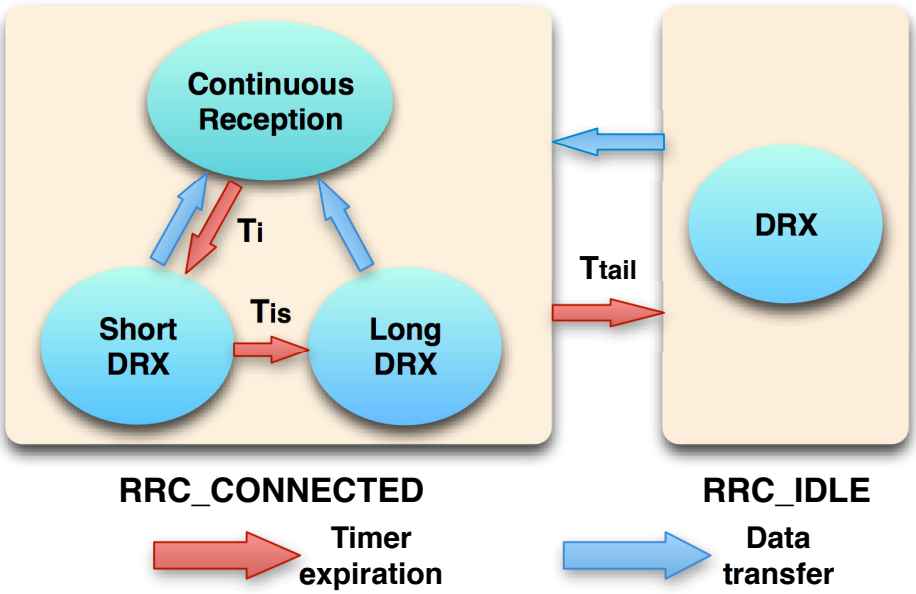


Figure 2.2: RRC state transitions in LTE network.

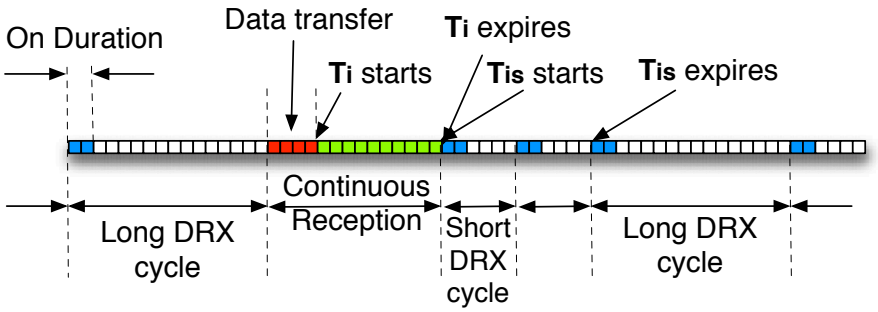


Figure 2.3: Illustration of the LTE DRX in **RRC_CONNECTED**.

for UE to "micro-sleep" to reduce power consumption while providing high QoS and connectivity. DRX in **RRC_CONNECTED** and **RRC_IDLE** have similar mechanisms, but different parameter settings. A DRX cycle includes an On Duration during which the UE monitors PDCCH. UE rests for the rest of the cycle to save energy. The tradeoff between battery saving and latency is the guideline for determining the parameterization of DRX cycle. With a fixed On Duration, a longer DRX cycle reduces energy consumption of UE while increasing user-perceived delay, and a shorter DRX cycle reduces the data response delay at the cost of more energy consumption. Short DRX and Long DRX modes, having the

same On Duration and differing in cycle length, are to meet these conflicting requirements.

When UE enters Short DRX, Short Cycle Timer T_{is} is started. Upon T_{is}'s expiration, if there is no data activity, UE switches to Long DRX; otherwise, UE goes back into Continuous Reception. For our measurement, T_{is} coincidentally equals T_{ps}, so only one cycle of Short DRX is expected to take place before T_{is} expires. Every time UE enters Continuous Reception when there is any data transfer, UE starts the tail timer T_{tail}, which is reset every time a packet is sent/received. When T_{tail} expires, UE demotes from **RRC_CONNECTED** to **RRC_IDLE** and the allocated radio resource is released. Notice that T_{tail} coexists with T_i and T_{is}.

CHAPTER III

MobiPerf: Characterizing 3G/4G Network Performance

Given the wide adoption of smartphone platforms, such as iOS and Android, there is a growing number of popular mobile applications designed for these platforms. For many of these applications, including web browser, email, VoIP, social networks, network access is required. Even for games that are often run locally, ranking systems and online peer matching systems are widely adopted which also requires network access, *e.g.,* Game Center for iOS. As a result, mobile data traffic volume is sky-rocketing. For example, AT&T's mobile data volumes surged by a staggering 8,000% from 2007 to 2010 [34]. Hence, it is critical to understand the *network performance* in cellular networks, and such understanding is a prerequisite to study smartphone application performance and optimizations. Smartphone customers want to know the cellular network performance in order to choose carriers and devices to use; mobile network operators care about cellular network performance to ensure the quality of service.

Systematically quantifying the cellular network performance is not straightforward. The challenges are multifold:

- It is not easy to reuse existing open-source network performance measurement tools due to smartphone operating system constraints. For example, iOS does not allow us to run a command-line program unless we jailbreak the device.

- Conducting measurements from a single or a few vantage points is not sufficient for

large cellular carriers. This is because the network condition and user load may vary across locations and without a reasonable number of sample users, the measurement results may not be representative. This forces us to abandon the idea to carry out all measurements ourselves, but instead to provide a tool for real smartphone users to use for network performance measurement.

- As the measurement tool is intended to be run by real smartphone users, who do not necessarily have any computer science background, we have to design the tool in a way that is easy for these users to make network measurements.

- The selection of the set of metrics for quantifying the cellular network performance is critical, as we want to collect sufficient information about the network performance within a period of user-tolerable time. Existing similar tools, such as Speedtest.net [35] and FCC's broadband test [36], only measure bandwidth and latency in cellular networks and ignore other important metrics such as DNS lookup time, *etc.* The measurement methodology also requires careful design, *e.g.,* bandwidth measurements in cellular networks need support from geo-distributed server nodes.

In the following of this chapter, we first discuss the design and implementation of MobiPerf in Section 3.1 and following that, we discuss the measurement results collected via MobiPerf and their implications.

3.1 MobiPerf Design and Methodology

Inspired by previous work in the Internet, *e.g.,* Netalyzr [37], which collects measurement data from volunteers, we develop a measurement platform, MobiPerf, used by real users on their smartphones to build a comprehensive data set for cellular networks. The public deployment of MobiPerf overcomes the limitation of a single vantage point and short time duration for locally conducted measurements and provides a representative data set on cellular network performance in the real world.

MobiPerf covers a more comprehensive set of metrics than existing public network performance measurement tools available in iOS or Android, such as DNS lookup, Ping to the first hop, *etc.*. We next describe the metrics we use for evaluating network performance and how we compute them. To minimize the impact of the performance limiting factors in the Internet path, we leverage the M-Lab [10] support and make MobiPerf always choose the closest server node(s) for measurement. MobiPerf server suite is deployed to 46 M-Lab nodes across the world, covering North America, Europe, Asia, and Australia. Each node has 4-core 2.00 GHz Intel Xeon CPU and our virtual slice has 4GB memory and 100Mbps Ethernet network access, which ensures that the network bottleneck is unlikely on the wired network path. Specifically, 23 nodes are within the U.S., spreading across major cities in different parts of the country.

To characterize cellular network performance, we use TCP throughput, downlink RTT, retransmission rate, local DNS lookup time, TCP handshake time, and Ping latency to the first responsive IP hop as our metrics. TCP is of particular interest, since most network applications use TCP. An application session usually requires DNS lookup, and every TCP connection begins with a TCP handshake. Hence these two factors contribute heavily to user-perceived performance of many network applications. Ping latency to the first responsive hop provides an estimate of the latency of the wireless hop.

DNS lookup For the DNS experiment, MobiPerf sends DNS requests to resolve a list of selected popular domain names. We use Alexa [38] top sites to select the top URLs and the list we use was downloaded in 2009. We list the top 20 domains as follows:

```
// List of top 20 domain names for DNS lookup test in C++
char* DOMAIN_NAMES[] = {"yahoo.com", "google.com", "youtube.com",
    "live.com", "facebook.com", "msn.com", "myspace.com",
    "wikipedia.org", "blogger.com", "yahoo.co.jp", "baidu.com",
    "rapidshare.com", "microsoft.com", "google.co.in",
    "google.de", "hi5.com", "qq.com", "ebay.com", "google.fr",
```

```
    "sina.com.cn"};
```

By tuning the size of the list and going through the list sequentially twice, we ensure that during the second lookup the names are highly likely cached at the local DNS (LDNS) server in carrier networks but not on the phone based on observed latencies. This is achievable since compared to the phone the LDNS server typically has a larger DNS cache.

RTT and variation test LTE has significantly smaller latency compared with 3G [11], hence the network distance between users and the measurement servers for the wired Internet path becomes less negligible. Given that the GPS coordinates for M-Lab nodes are known, in MobiPerf, a nearest M-Lab node is selected for a user based on the current GPS location if available, or the IP address otherwise, with the help of a IP address to GPS coordinates mapping [39]. Such a mapping is sufficient for our purpose of finding coarse-grained location estimates for server selection.

To measure RTT and variation, MobiPerf repeatedly establishes a new TCP connection with the server and measures the delay between SYN and SYN-ACK packet. Both the median of these RTT measurements and the variation are reported to our central server. For some cellular ISPs, traffic may be redirected to a middlebox, which replies a SYN-ACK packet to the client on behalf of the server. In this case, the measured RTT is between the client and the middlebox. However, this RTT is the user-perceived delay for TCP connection establishment and we think it is fine to use it for comparison purpose.

To measure TCP handshake to server nodes in diverse physical locations, MobiPerf sends TCP *connect* requests to different M-Lab nodes distributed across the U.S. To characterize Ping latency, our tool Pings `www.google.com` with increasing TTL values starting from 1 and records the IP address and the corresponding RTT. MobiPerf also Pings different M-Lab nodes to obtain the delay distribution to diverse Internet locations.

TCP throughput Since single-threaded TCP measurement is more sensitive to packet loss and hence less accurate [40], we use multi-threaded TCP measurement in MobiPerf to estimate the *peak channel capacity*, *i.e.,* three nearest server nodes in M-Lab are selected for each user at runtime to start concurrent threads for throughput test. Despite the shared nature of M-Lab nodes with other test tools, it is unlikely that all three selected nodes are overloaded in terms of CPU and network.

A throughput test lasts for 20 seconds, to balance across bandwidth usage, user waiting time and measurement accuracy. The initial 5 seconds are ignored empirically due to TCP slow start. The remaining 15 seconds are separated into 15 1-second bins. The average throughput for each bin is calculated and the median of all bins is the measured throughput. Compared with using average throughput, median more accurately estimates the steady peak throughput by reducing the impact of abnormal bins, *e.g.,* a very high-value bin due to initial buffering or a low-value bin due to temporary signal problem. Uplink and downlink tests share the same methodology. Packet traces are collected at the server side to calculate TCP retransmission rate.

3.2 MobiPerf Deployment and User Statistics

MobiPerf [7] project was initiated in 2008 and the name of the measurement tool was 3GTest [7, 1] at the time. We made it publicly available on different smartphone platforms, which allows us to characterize cellular network performance in multiple cellular carriers at diverse locations over an extended duration. With significant UI and methodology improvements, we changed the name to be 4GTest [6] in 2011. In 2012, we further decided to change the tool name to be MobiPerf and made it open source [9]. Initially, MobiPerf was supported for iOS, Android and Windows Phone platforms and later, due to time limit and OS constraints, we decided to focus our efforts on the Android platform. At the time this dissertation is being written, MobiPerf is under active development with joint efforts from University of Michigan, University of Washington, and M-Lab [10], and it is being used as

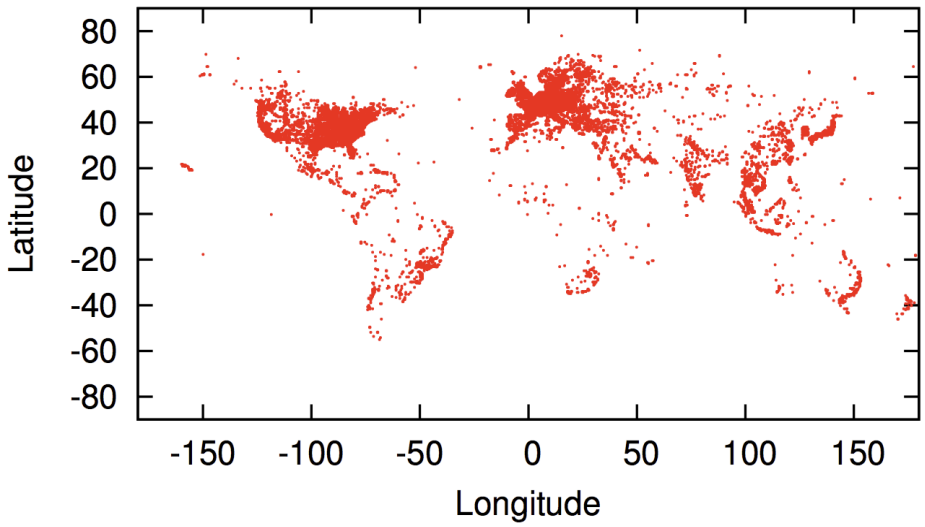


Figure 3.1: MobiPerf user coverage between August 2009 and April 2011.

	Android	iOS	Win Mobile	All
User	39.3K 39.7%	47.0K 47.4%	12.8K 12.9%	99.1K
Run	273.8K 62.3%	127.0K 28.9%	38.7K 8.8%	439.5K

Table 3.1: MobiPerf User and run breakdown of different platforms.

basis for research projects and commercial products of multiple organizations.

We publicly deployed the MobiPerf application in August, 2009, distributed via Apple's App Store, Google's Android Market and Microsoft's Windows Marketplace for Mobile. Ever since the initial deployment, we have been continuously improving and releasing updates for iOS and Android version of our app. Till April, 2011, 99.1K users from across the world have run our app for 439.5K times. The number of users and runs for three different platforms, including iOS, Android, and Windows Mobile, is listed in Table 3.1. The average number of runs for each Android user is larger than the other two platforms, because for the Android version of our app, we give an option to the users to periodically run the tests. We observe users from 179 countries or regions according to the collected GPS information. Since GPS information for Windows Mobile users is seldom available, for most top countries of the other two platforms, we do not observe any Windows Mobile users. GPS information may sometimes be unavailable as well on Android and iOS, due to

Referred to as	iPhone	Palm	Samsung	G2	HTC
Carrier	AT&T	Sprint	Verizon	T-Mobile	AT&T
Network	UMTS	EVDO	EVDO	UMTS	UMTS
Advertised Downlink(Mbps)*	0.7-1.7	0.6-1.4	0.6-1.4	0.6-1.0	0.7-1.7
Advertised Uplink(Mbps)*	0.5-1.2	0.35-0.5	0.5-0.8	0.3-0.7	0.5-1.2
Vendor	Apple	Palm	Samsung	HTC	HTC
Device	iPhone	Treo800w	SCHi760	Android G2	TyTnII
Memory (MB)	128	128	64	192	128
Processor (ARM)	1176	1136	920T	1136EJS	1136EJS
CPU frequency (MHz)	620	333	400	528	400
OS	iPhone OS 2.1	WM 6.1†	WM 6.1	Android 1.6	WM 6.1
Browser	Safari	IE	IE	Browser App	IE

Table 3.2: Device specifications and 3G network carriers$^\natural$.

*All advertised data rate was for the time in 2009.
†WM stands for Windows Mobile
$^\natural$At the time of this piece of study (2009), these devices represent state of the art of smartphones.

signal problem or users not wishing to share their location information. Among all 93.3K users, 63.7K (68.27%) have GPS readings and 52.24% of them are from the U.S., and among these 63.7K users, about 1.0K (1.57%) users have run our app in more than one countries or regions. We also observe more than 800 carrier names. However, carriers may adopt different names in different countries, making it difficult to accurately estimate the actual number of carriers. Figure 3.1 shows the user coverage of MobiPerf, with one dot representing one run of MobiPerf. Given the wide coverage of regions, we believe our data set is fairly representative of the entire smartphone population, especially for North America with denser user distribution. In this study, our analysis mostly focuses on U.S. users.

3.3 3G Network Characterization

We first focus on characterizing the performance of commercial 3G networks with the MobiPerf data set, complemented by local controlled experiments.

Table 3.2 lists the devices used and carriers studied in the local controlled experiments. We studied four major carriers in the U.S., AT&T, Sprint, Verizon, and T-Mobile. They split

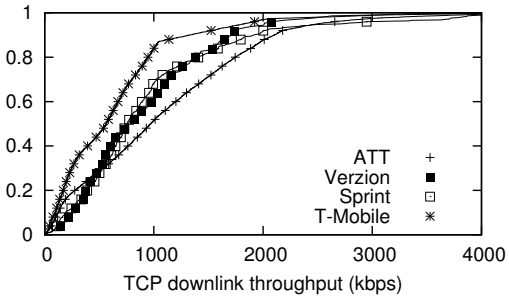
between UMTS/HSPA (AT&T and T-Mobile) and EVDO (Sprint and Verizon). AT&T has the highest advertised downlink and uplink data rates. The actual data rates that a user can attain depend on many factors, such as signal strength, location, and background traffic. One of our goals is to understand how the actual data rates match the advertised ones and which factors have the biggest impact on network performance.

3.3.1 Comparison across Carriers

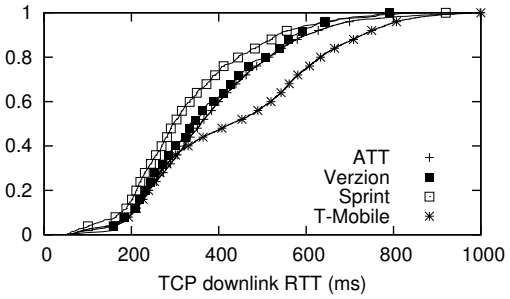
Figures 3.2(a) illustrates measured TCP downlink throughput. Given stable TCP throughput is roughly inversely proportional to RTT and to the square root of packet loss rate [41], we also analyze RTT and retransmission rate. In Figure 3.2(b), all carriers show comparable RTT distributions, with T-Mobile showing slightly larger RTT values and correspondingly lower downlink TCP throughput. Various reasons contribute to large RTT in 3G networks, *e.g.,* queueing delays at the base station or other internal nodes, such as RNC, SGSN, and GGSN in UMTS networks. Large RTTs may also be due to packet loss recovered through link layer retransmission, which we do not have direct information about.

Figure 3.2(c) plots measured TCP uplink throughput. Unlike downlink throughput, AT&T and T-Mobile have lower uplink throughput compared with Sprint and Verizon. One of the reasons could be the lack of support for UMTS/HSUPA on the phones used for AT&T and T-Mobile. Even the latest version of iPhone 3GS does not claim to support HSUPA. The median uplink throughput for AT&T and T-Mobile ranges from 200 kbps to 300 kbps, while that for Sprint and Verizon is around 400 kbps.

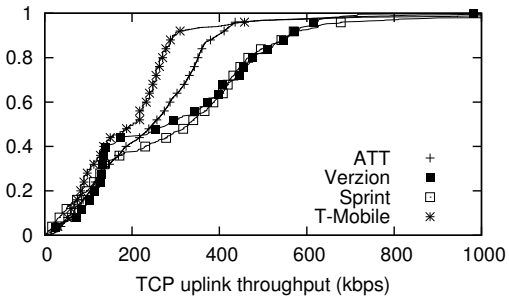
Figure 3.2(d) shows that Verizon and Sprint exhibit slightly higher TCP retransmission rate, matching observations from our local experiments. On average, AT&T's downlink throughput outperforms that of the other carriers due to its relatively lower RTT and loss rate. The median of TCP downlink throughput for all carriers ranges from 500 kbps to 1 Mbps. Median RTT varies from 300 ms to 500 ms, suggesting 400 ms is a representative delay value to emulate 3G networks. AT&T and T-Mobile have a median retransmission



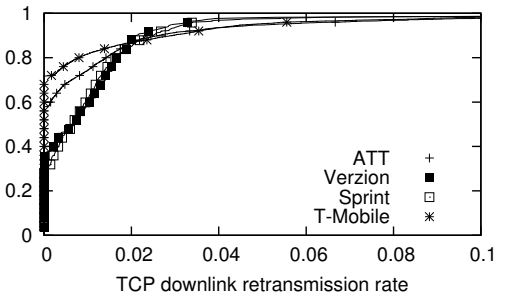
(a) CDF of downlink TCP throughput



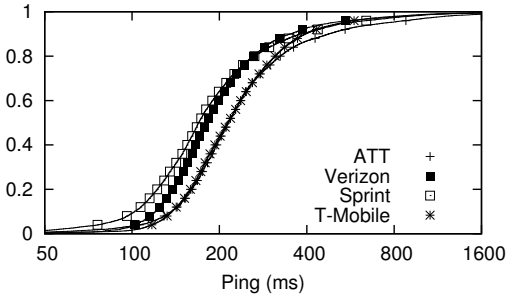
(b) CDF of downlink TCP round trip time



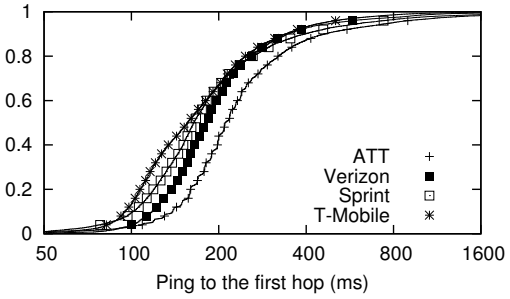
(c) CDF of uplink TCP throughput



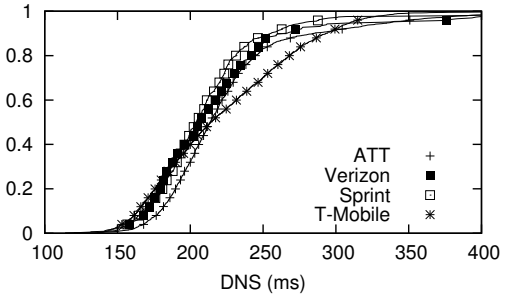
(d) CDF of downlink TCP retransmission rate



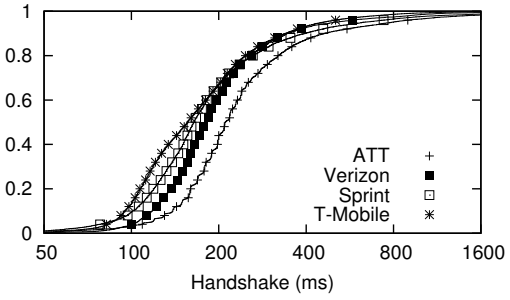
(e) CDF of Ping latency to landmark servers



(f) CDF of Ping latency to the first hop



(g) CDF of DNS lookup time



(h) TCP handshake time to landmark servers

Figure 3.2: TCP performance comparison among carriers (data from deployed application MobiPerf, only considering U.S.).

rate of 0%, while that for Sprint and Verizon is 0.7%.

Figures 3.2(e)(f) show that Ping latency to the first responsive hop is close to that to

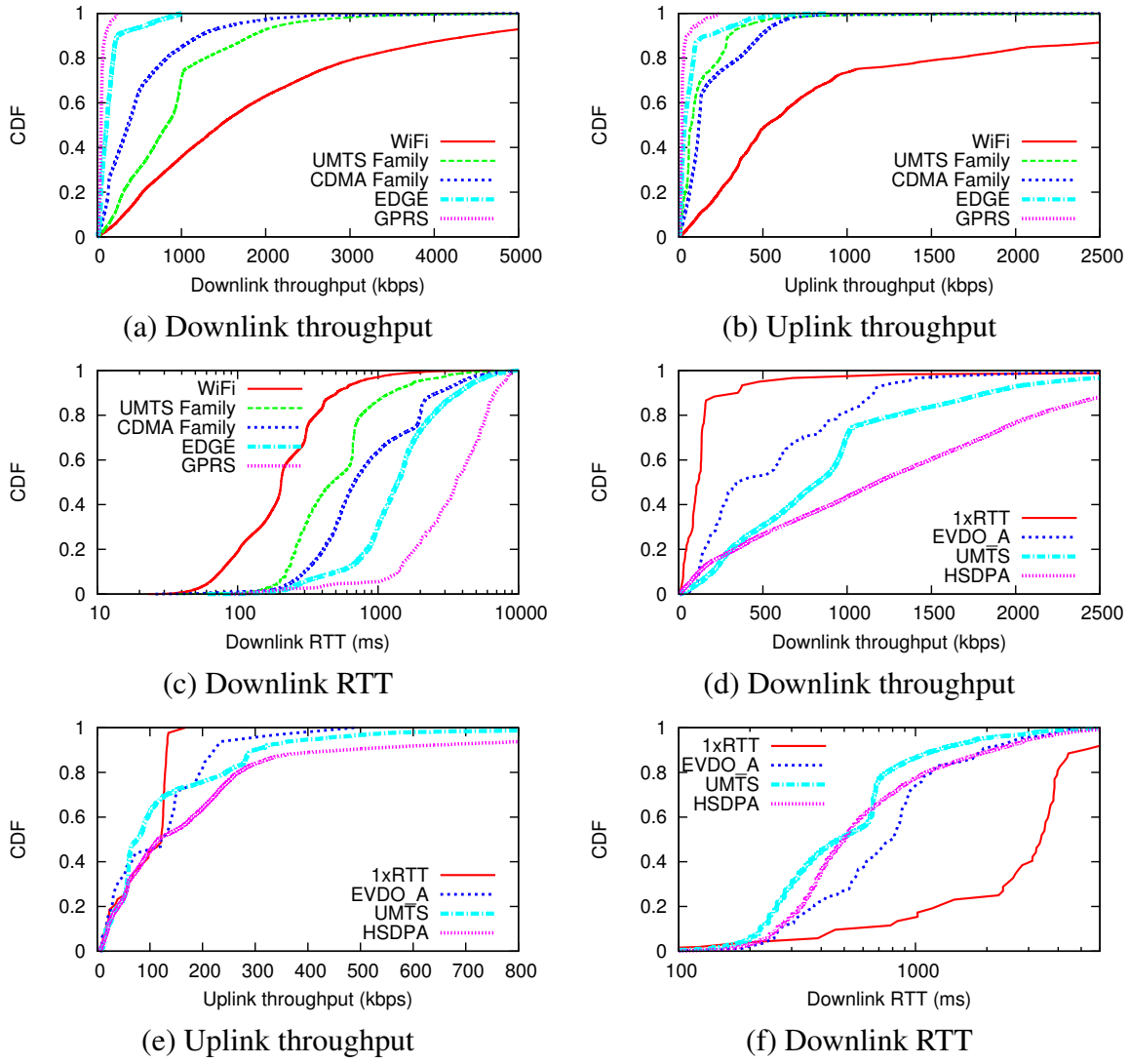
landmark servers, suggesting that the first responsive hop consisting of 3G wireless link contributes to most of the delay along the end-to-end network path. Note that Ping latency to the first responsive hop actually refers to the first IP hop responding to ICMP probing. For AT&T and T-Mobile, the first IP hop, when TTL is set to 1, does not respond in most cases. Only the second IP hop replies with a private IP address. For Sprint and Verizon, the first IP hop does reply with a public IP address. The median latency to the first responsive hop ranges from 150 ms to 200 ms, while that to landmark servers is between 180 ms and 250 ms. We observe that both the Ping latency and TCP handshake time are smaller than RTT values measured in TCP downlink experiments.

Figure 3.2(g) shows DNS lookup performance. We design the experiment in a way that all DNS lookups are cached at the LDNS server but not locally on the phone (Section 3.1). This allows us to more accurately estimate the delay to the LDNS servers. The LDNS servers studied tend not to respond to ICMP packets, making it challenging to directly measure the network delay between the phone and LDNS server. From the results, we found that all carriers exhibit similar trend with median values close to 200 ms. Given that the DNS lookup delay is already close to Ping latency to the first responsive hop, there is limited room for improving DNS lookup performance.

As shown in Figure 3.2(h), the median of TCP handshake delay ranges from 160 ms to 200 ms, close to the Ping latency to the first responsive hop in Figure 3.2(f). We also observe that the relative ranking among all carriers is consistent with that in Figure 3.2(f). Compared with Figure 3.2(b), large packets with size close to MTU (*e.g.,* 1348 bytes in AT&T) are found to have 2 – 4 times of the RTT for small packets.

3.3.2 Performance Comparison among Mobile Network Technologies

We compare the cellular network performance among technology types. We first break down technology types into WiFi, UMTS family, CDMA family, EDGE and GPRS. UMTS and CDMA are considered as 3G, while EDGE and GPRS are two types of older tech-



(a) Downlink throughput (b) Uplink throughput

(c) Downlink RTT (d) Downlink throughput

(e) Uplink throughput (f) Downlink RTT

Figure 3.3: Downlink/Uplink performance comparison among different types of networks.

nologies. Within 3G family, we select 4 major network types, including HSDPA, UMTS (without HSDPA), 1xRTT and EVDO_A, since these network types cover most 3G users. Notice that at the time of this part of study [42], LTE has not come into market yet and we leave the study of LTE in Section 3.4.

Downlink throughput is compared in Figure 3.3 (a). WiFi has the best performance with median throughput of 1.46 Mbps. For 3G network, UMTS family appears to outperform CDMA family, with median downlink throughput of 964 kbps compared to 368 kbps. EDGE lags with median downlink throughput 112 kbps and GPRS is the slowest at

45 kbps. The ranking of downlink retransmission rate is consistent with that of downlink throughput, except that UMTS and CDMA have similar retransmission rate distribution. In Figure 3.3 (c), UMTS's median RTT is 495 ms, smaller than CDMA's 680 ms. This helps explain the throughput gap between UMTS and CDMA, since TCP throughput is lower with higher RTT and loss rate. Among 3G networks shown in Figure 3.3 (d), HSDPA has the highest median downlink throughput of 1.18 Mbps and 1xRTT has the smallest throughput of 115 kbps, since it is one of the earliest CDMA 3G technologies. Similarly, we observe high RTT in Figure 3.3 (f) and high retransmission rate for 1xRTT correlated with its low throughput.

We closely study the variation of TCP downlink RTT, often known as jitter. It is an important metric to evaluate network performance for streaming video, VoIP, and online gaming applications, *e.g.,* high variation causes intermittent video playing, voice calls and lags for online games. In Figure 3.3(f), each data point in this figure corresponds to the standard deviation of all RTT samples in one downlink TCP flow, *i.e.,* one user's downlink experiment. Compared with WiFi, whose median of RTT standard deviation is 41 ms, UMTS has a higher value of 93 ms and 233 ms for CDMA. If applications running on smartphones fail to tolerate this high variation in RTT, user experience would be degraded.

In Figure 3.3 (b) , the uplink throughput difference between UMTS and CDMA is less obvious compared with downlink. For example, at 50^{th} percentile, UMTS's uplink throughput is 110 kbps and CDMA's is 120 kbps. Within the 3G family, as shown in Figure 3.3 (e), all network types experience less than 150 kbps median uplink throughput.

These results can be taken into consideration by network application developers to have better support of various network conditions.

3.3.3 Time of Day Correlation

Understanding whether traffic patterns exhibit any time of day behavior is useful for improving the design of applications and mobile network infrastructure. We expect smart-

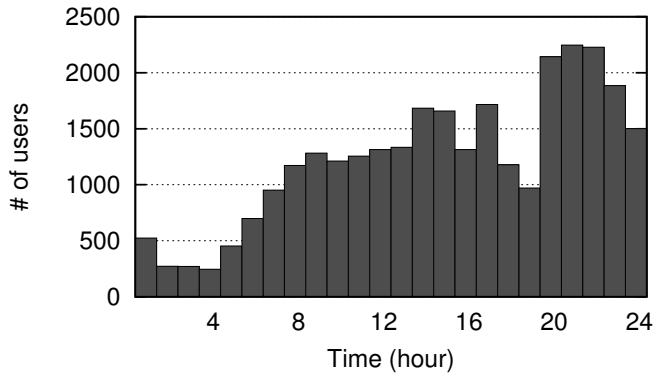
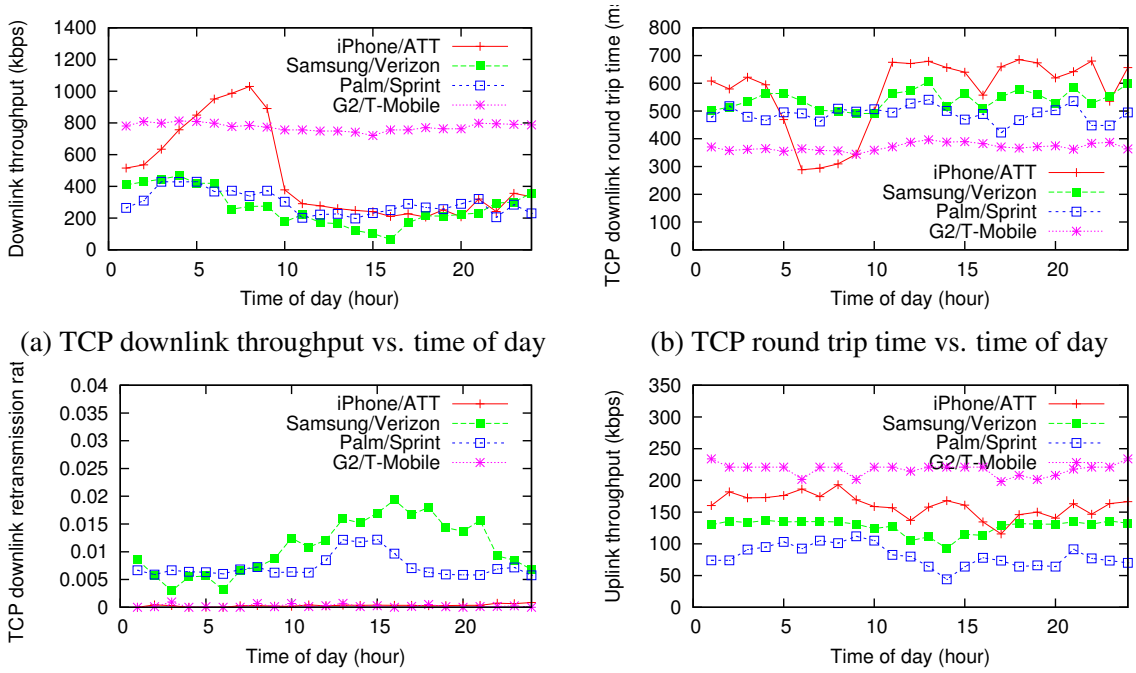


Figure 3.4: Number of MobiPerf users vs. time of day.

phone users to have diurnal patterns in their behavior. For example, we can observe such a pattern in Figure 3.4. To further understand its impact on performance, we resort to conducting local controlled experiments.

To measure the network performance over a long time period, we created an internal version of MobiPerf and installed it on the smartphones listed in Table 3.2. MobiPerf is modified to record the signal strength on the Samsung and Palm phones, and continuously conducts measurements every 10 minutes to collect one week's data (excluding weekends) in Ann Arbor, MI. We make sure that the phones are placed at the same location with excellent signal strength during the entire measurement study. Since the data is collected continuously for a long period of time, it can be used for characterizing the time-of-day effect. The results for the 5 contiguous weekdays are shown in Figure 3.5.

First, time of day effect is less pronounced for uplink throughput compared to downlink throughput, comparing Figure 3.5(a) and (d). This is likely due to higher demand for downlink capacity by popular applications such as web browsing and video streaming. Second, we observe an obvious time pattern for AT&T's downlink throughput. At night and early morning hours, between 2AM and 8AM, the downlink throughput can reach 1 Mbps. However, the downlink throughput of lower than 500 kbps is observed at other times. This phenomenon is possibly due to the large number of iPhone users and the large traffic volume brought by various network applications. For Sprint and Verizon, we observe



(a) TCP downlink throughput vs. time of day

(b) TCP round trip time vs. time of day

(c) TCP retransmission rate vs. time of day

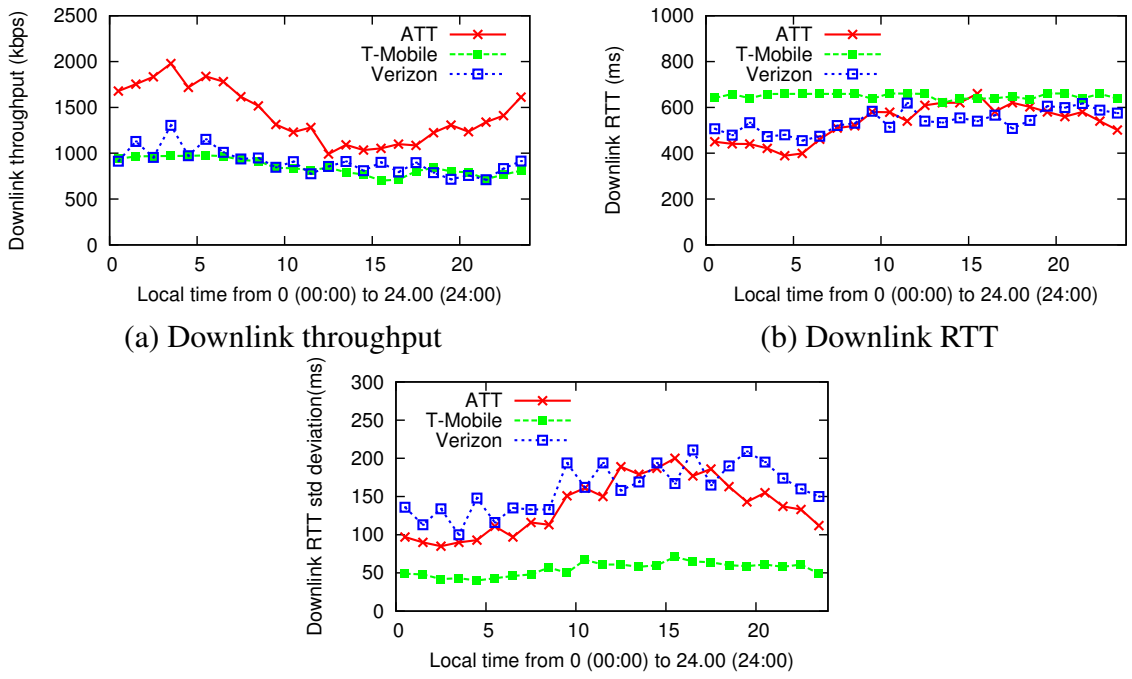
(d) TCP uplink throughput vs. time of day

Figure 3.5: Correlation between TCP performance and time of day (local controlled experiments).

similar though less prominent trend compared to that for AT&T. For T-Mobile, the TCP downlink throughput is more stable, which we conjecture is due to the fact that its 3G service has only recently become available at our location.

Figures 3.5(b)(c) indicate that RTT and retransmission rate exhibit time of day pattern for some carriers. For AT&T, the downlink throughput is found to be mostly affected by RTT values, likely to be caused by queueing delays in AT&T's 3G networks. RTT varies from 300 ms during late nights to as high as 700 ms at peak times. For Verizon and Sprint, the RTT values are more stable, though with varying TCP retransmission rate. One possible explanation is that in Verizon and Sprint's 3G networks, shared queues would drop packets once the queue length exceeds a threshold. This design will restrict the variation of RTT but incur more packet loss.

We further analyze the time-of-day pattern using the MobiPerf data set collected across locations. Figure 3.6 shows the aggregate time of day analysis of TCP downlink through-



(a) Downlink throughput

(b) Downlink RTT

(c) Downlink RTT standard deviation

Figure 3.6: Time of day pattern of downlink performance for major carriers in the U.S. (MobiPerf data set, large packets).

put, RTT, and jitter for all AT&T, T-Mobile, and Verizon 3G users in the U.S. Each data point is median value of all data samples across locations in the U.S. within an hour based on user's local time. This analysis technique avoid potential bias by a specific small group of users or any particular locations.

Figure 3.6 (a) shows that AT&T has the most clear time of day pattern for downlink throughput, confirming previous local controlled experiments. Verizon has less obvious yet still observable diurnal pattern for downlink throughput and even less obvious for T-Mobile. TCP retransmission rate for these carriers stays low consistently at different hours, hence the diurnal pattern of TCP downlink RTT in Figure 3.6 (b) explains the throughput fluctuation, especially for AT&T. The standard deviation for TCP downlink RTT also demonstrates a clear diurnal pattern for AT&T and Verizon, while less obvious for T-Mobile shown in Figure 3.6 (c).

These observations suggest that applications with intensive network download require-

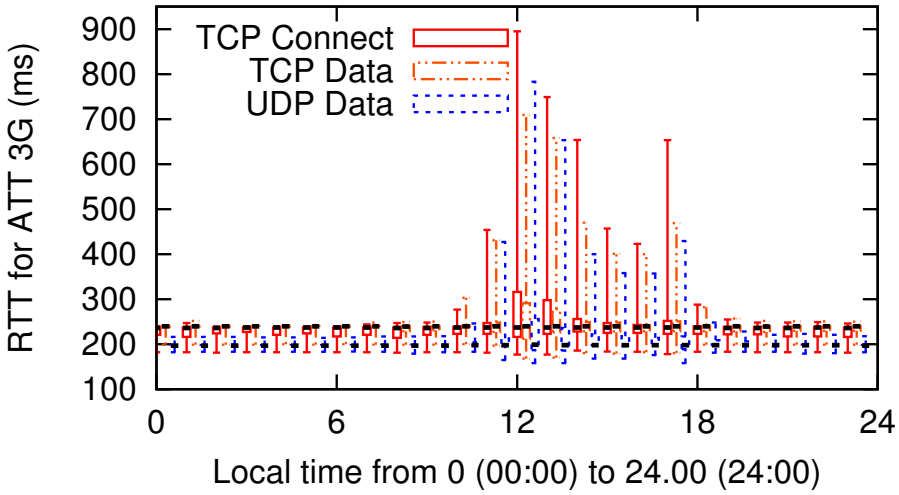


Figure 3.7: Delay in AT&T 3G network (small packets).

ment and little tolerance on RTT jitter may experience worse performance during peak hours. Uplink and LDNS performance are relatively consistent across different hours of day, indicating that the infrastructure support is capable of handling peak hours for such operations. However, for downlink, network resource at peak hours becomes a bottleneck.

Given that the downlink RTT in Figure 3.6 (b) is for large packets, mostly with a size of MTU, we also study the diurnal pattern of small packets. For T-Mobile, similar to Figure 3.6 (b), we do not observe any time of day effect on RTT. For AT&T, in Figure 3.7 showing RTT of small packets (100 bytes) with boxplot, at the median level, there is no diurnal pattern for RTT. At the 75^{th} percentile, 12:00PM and 1:00PM have larger RTT values, and at the 95^{th} percentile, hours between 10:00 AM and 6:00 PM clearly have much larger RTTs. This indicates that during peak hours, most small packets do not experience longer delay, but some (at least 5%) experience much longer RTTs. By comparing with large packets, our local experiment suggests that small packets have less obvious, yet still observable diurnal pattern for AT&T.

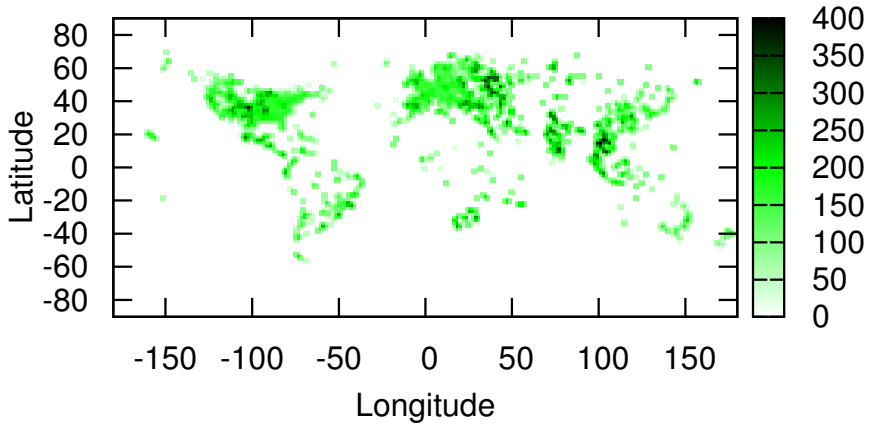
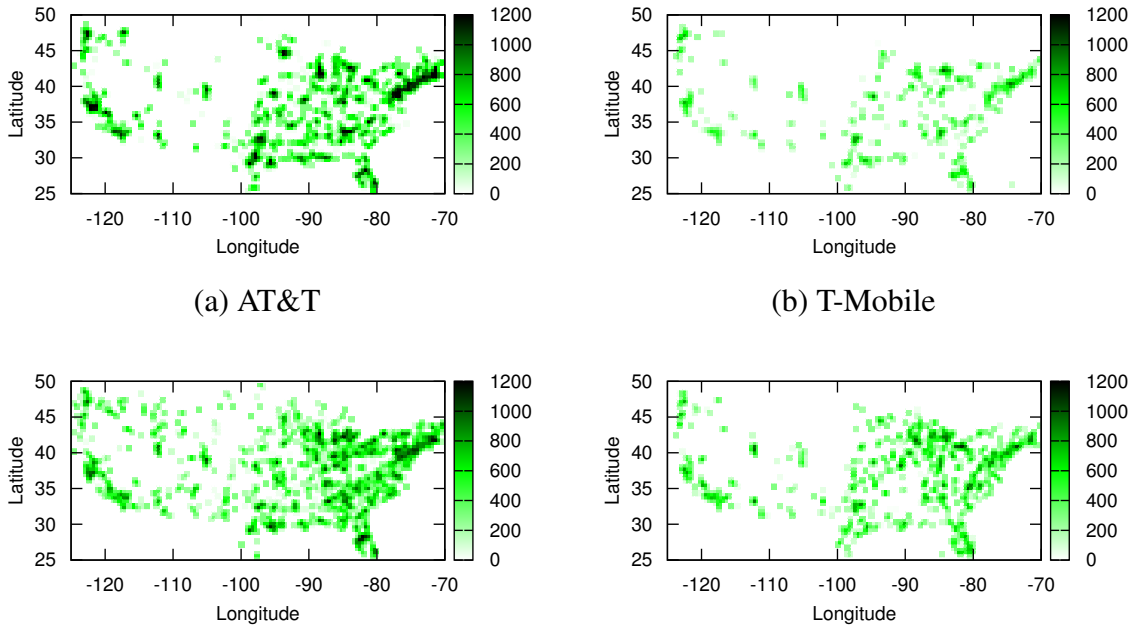


Figure 3.8: Worldwide local DNS lookup time in 3G networks (ms).

3.3.4 Location and Performance Correlation

We only look at worldwide local DNS lookup time in the MobiPerf data set. In Figure 3.8, for each cell with a size of 50 kilometers×50 kilometers, the median DNS lookup time of all 3G users within this region is selected. We can see that for most parts of the world, the DNS lookup time is around 150 ms to 250 ms. Regions including South Asia, Middle East, Eastern Europe and some regions in the middle of the U.S. experience higher DNS delays.

For cellular networks, given that the infrastructure support differs across locations for different carriers, we intend to study how performance correlates with location. We compare TCP downlink performance of major U.S. carriers in Figure 3.9. The plotted cell size is 50 kilometers $\times$ 50 kilometers, excluding those without enough data points to show statistically meaningful results. The coverage of each carrier shown in this figure is clearly affected by the popularity of our app among its customers. Since we focus on the median performance at different locations, where we do have data, our measurements can provide a fairly representative sampled view of the cellular network performance across different locations. Comparing across carriers shows large variation in performance across locations, with a few locations clearly having better downlink performance. Second, these locations



(a) AT&T (b) T-Mobile

(c) Verizon (d) Sprint

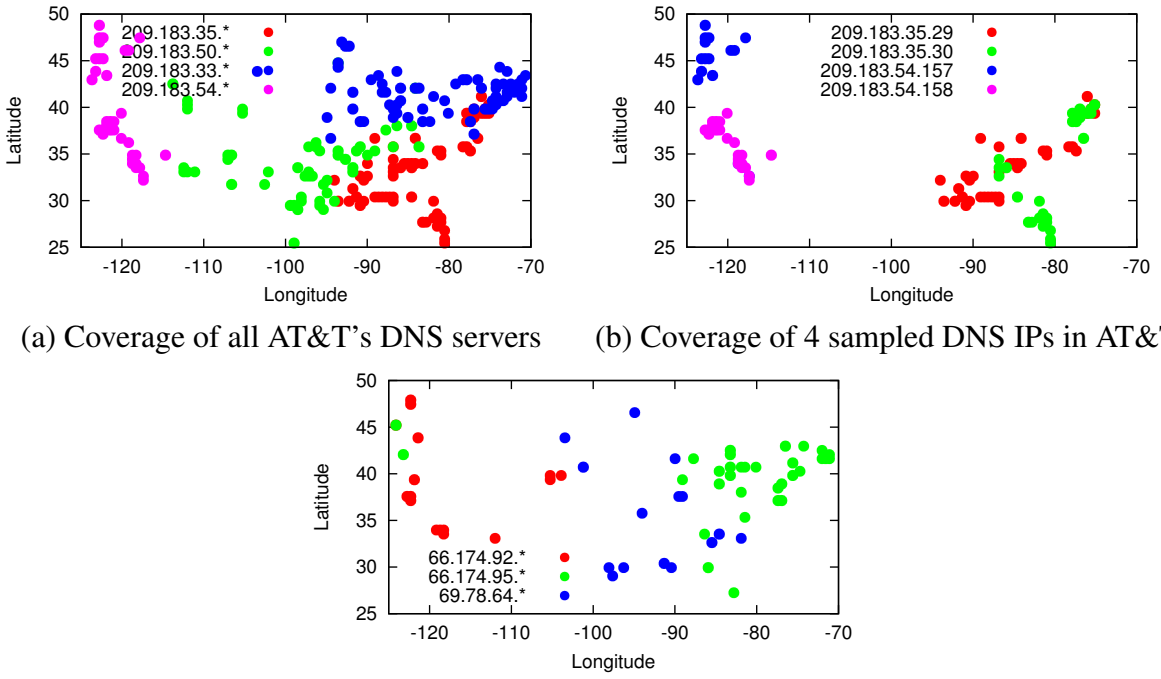
Figure 3.9: Downlink throughput of major carriers in the U.S. (kbps).

of different carriers are typically large cities, such as New York, San Francisco, and Los Angeles. Third, for each carrier, the eastern part of the U.S. has higher user coverage than the western part, except for the state of California and Washington. These observations suggest that for regions with larger user base, carriers may have provided better infrastructure support and newer technology, compared with rural areas with fewer users.

3.3.5 Cellular Network Infrastructure and Implications

In this section, we study LDNS assignment and quantify the effectiveness of CDN service in cellular networks.

LDNS Servers: How LDNS servers are assigned to customers is an interesting and important design decision made by carriers. It reflects how loading balancing is done and affects LDNS lookup performance. It also impacts the effectiveness of DNS-based server selection. With a representative data set for the major American carriers, we study how they



(a) Coverage of all AT&T's DNS servers

(b) Coverage of 4 sampled DNS IPs in AT&T

(c) Coverage of Verizon's DNS servers

Figure 3.10: Coverage of local DNS servers.

assign LDNS servers to their customers. We observe 12 different LDNS IPs for AT&T in 4 different /24 address blocks, each consisting of 3 IPs. For T-Mobile, 11 LDNS IPs are observed in 5 /24 blocks, and for Verizon, 12 LDNS IPs in 3 /24 blocks are detected. In Figure 3.10, we present the correlation between users' location and the assigned LDNS IPs. For both AT&T and Verizon, we can observe that clients for each LDNS address block tend to cluster geographically. This suggests that both carriers assign LDNS servers to clients based on geographic regions. However, for T-Mobile, all identified LDNS IPs are used across the country. These results confirm the unsupervised clustering results of our previous work [43].

To further understand how different individual IPs among a /24 address block are used, we show in Figure 3.10 the geographic distributions of four representative LDNS IP addresses. We can observe that even at the individual IP level, there exists clear correlation between location and LDNS assignment, despite some overlapping regions. Through local experiments for AT&T, we confirm that the assigned LDNS IP remain constant for over 48

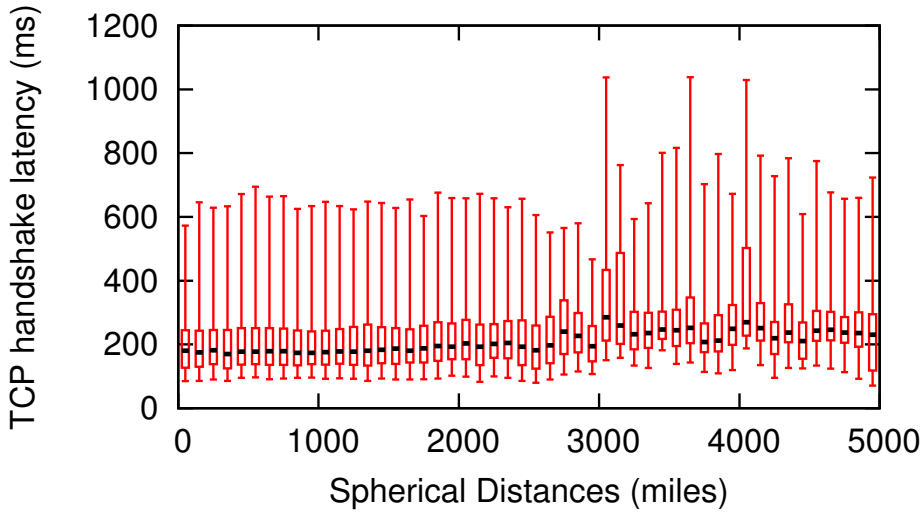


Figure 3.11: Handshake RTT vs. spherical distances.

hours. Having one LDNS IP for a region allows the flexibility to customize DNS caches according to the users' interests within the region. The comparison of the LDNS latency distribution for AT&T and T-Mobile shows no clear performance difference when querying LDNS servers at different locations. As we discuss further below, we conjecture that the bottleneck of network latency is along the wireless hop, making the server location less important. Note that DNS-based server selection cannot effectively choose nearby servers if the LDNS server assignment is not based on geographic regions.

CDN Service for Cellular Networks: Using landmark test results, we study the effectiveness of CDN service in cellular networks. First, we select the users with GPS information and calculate the physical spherical distance [1] to the list of 20 landmark servers. The distribution of TCP handshake latency is show in Figure 3.11. We observe high variation in RTT between users and landmark servers with no clear correlation with physical distance. This suggests that the latency in the 3G wireless hop is dominant in determining the end-to-end latency, rather than the Internet latency.

To quantify the effectiveness of CDN services in cellular networks, we assume the 20

[1] Spherical is the accurate distance in the great circle of the earth based on latitude and longitude.

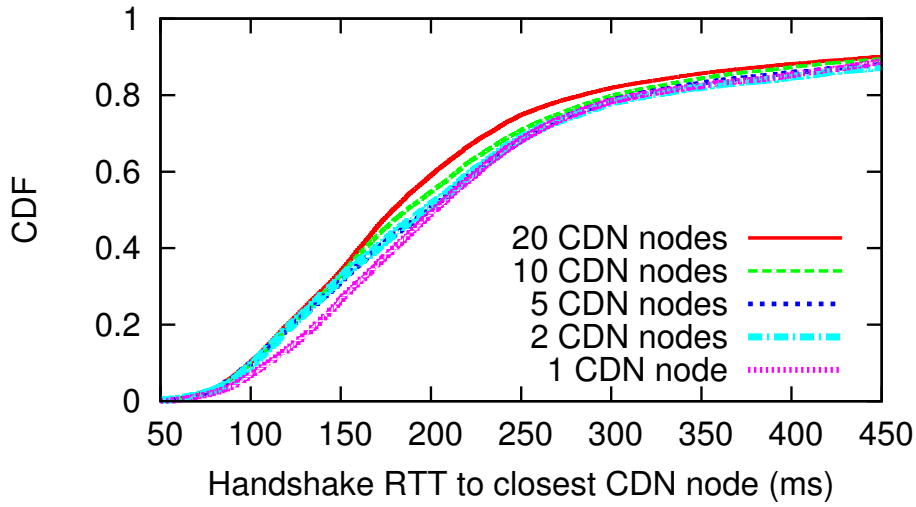


Figure 3.12: RTT to closest synthetic CDN node.

landmark servers to be CDN servers. Then we study 5 different scenarios, assuming 20, 10, 5, 2 and 1 CDN servers are used, respectively. We select the CDN servers to maximize the geographic spread. For each scenario, each user chooses the physically closest CDN server based on its GPS information, and the latency to this server is regarded as the user perceived latency. Figure 3.12 shows the distribution of the user perceived latency in all studied scenarios. Additional CDN servers have very limited effect on reducing user perceived latency. Comparing the best and worst scenario with 20 and 1 CDN servers each, we found the median RTT difference to be 20 ms, only 10% of the median RTT of 200 ms. However, the equivalent RTT saving can be more significant in LTE 4G networks with a much shorter latency at the wireless hop. Given that today's cellular network routing is heavily restricted due to limited number of gateways to the Internet which are the closest locations to the clients for deploying CDN servers, caches can be pushed closer to the user to reduce latency in LTE 4G network, which uses a flatter architecture [33].

3.3.6 Signal Strength Effects

Signal strength is an important factor that affects 3G network performance, since higher signal-to-noise ratio (SNR) allows higher bit rate. We therefore also carried out experi-

ments to understand this correlation. Since it is not easy for us to control the signal strength, we continuously monitor signal strength and TCP downlink throughput for a week. We highlight our major observations here. When the signal strength is too weak, TCP connections will disconnect. When signal strength is at some middle range, we observe clear correlation between signal strength and TCP downlink throughput. TCP downlink throughput is not affected by the signal strength if the latter is above some threshold. Given these observations, we exclude the data points corresponding to poor signal strength from the MobiPerf data set and controlled experiment data set.

3.3.7 Smartphone v.s. Laptop

To understand whether the computation capability of a smartphone limits its 3G network performance, we set up a controlled experiment to compare a smartphone (iPhone 3G) with a laptop (ThinkPad T42). The laptop can access AT&T's 3G network via a wireless data card, while the iPhone measurement is conducted at the same location and the same time. We found that the distribution of downlink throughput is similar, implying that the performance bottleneck is within the 3G network instead of on the phone. However, for other compute-intensive applications, the performance difference is more pronounced. We will study this in more details in Section 4.2.4.

Summary: The main observations of 3G network performance are the following:

- Typical values for 3G throughput range from 500 kbps to 1 Mbps for downlink, and 200 kbps to 400 kbps for uplink, both lower than the advertised rates.
- Network performance differs across all carriers. For downlink RTT and throughput, the differences among carriers are evident.
- Some carriers show clear time of day patterns on weekdays, especially for AT&T's downlink throughput.

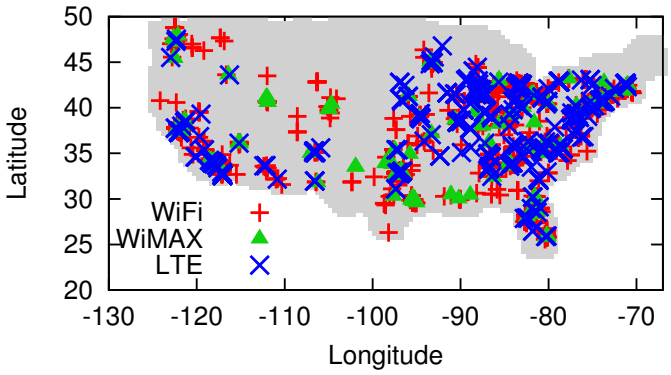


Figure 3.13: MobiPerf user coverage in the U.S between October 2011 and December 2011.

- For simple TCP downloading/uploading, the performance bottleneck is within the 3G network.
- 3G wireless delay dominates the end-to-end RTT.

Our observations suggest that the low uplink throughput and large RTT of current 3G networks raise challenges for offloading computation into the cloud. Network application designers should avoid chatty protocols and minimize total bytes to transfer. 3G operators need to examine their queueing and link layer retransmission policies to reduce latency in the wireless links.

3.4 LTE 4G Network Characterization

During the time of study for the previous section, *i.e.,* in 2009, LTE 4G network has not yet come into commercial market. In 2011, with growing popularity for the commercial LTE network, we decided to study it and compare it with existing cellular network technologies and WiFi technology.

A special version of MobiPerf, branded as 4GTest (in the following, we do not distinguish 4GTest from MobiPerf) was initially released on Android Market on 10/15/2011. Till 12/15/2011, there has been altogether 3294 global users and 13768 runs. In the analysis, we focus on 2292 users and 7063 runs inside the U.S., among which 380 are LTE users

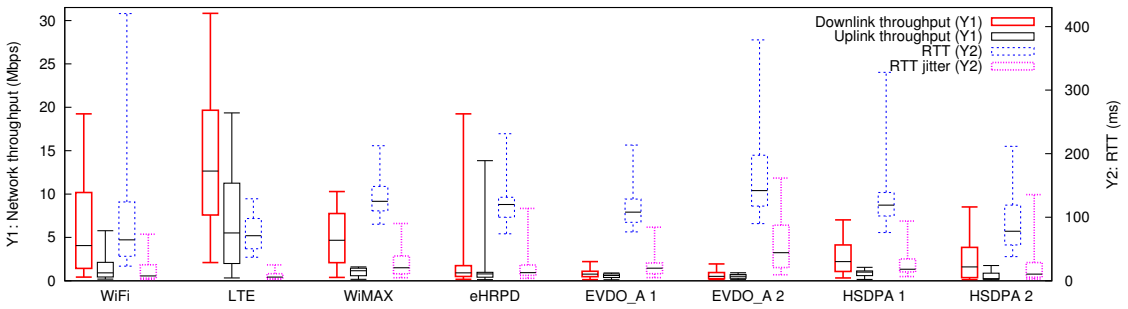


Figure 3.14: MobiPerf result analysis based on network type*.

*Each network type corresponds to the data of one cellular ISP, except for WiFi which has data of various ISPs.

with 982 runs. We check the distribution of MobiPerf users for different network types. For example, in Figure 3.13, the coverage of LTE, WiMAX, and WiFi are mostly similar, covering 39, 37 and 44 states in the U.S., respectively. We also observe that other networks have similar coverage across the U.S. This indicates that MobiPerf data set enables fair comparison on the distribution of performance metrics for different mobile networks in the U.S.

Similar to our previous study for 3G networks in 2009, we complement MobiPerf with local controlled experiments. We use an HTC phone with LTE data plan from a large cellular ISP for controlled experiments. It has 768 MB RAM memory and 1 GHz Qualcomm MSM8655 CPU, running Android 2.2.1. We also use a laptop equipped with LTE USB Modem. It has 8GB memory and 2.53 GHz Intel Core Duo CPU, running Mac OS X 10.7.2.

3.4.1 Comparing LTE to Other Mobile Networks

Figure 3.14 summarizes the performance comparison among various mobile networks based on 4GTest data set. We present anonymized data based on the technology type. The box plot for each metric shows the 95th, 75th, 50th, 25th and 5th percentile among all measurements for each network type. We observe that LTE network has a high downlink and uplink throughput, with the median to be 12.74Mbps and 5.64Mbps, respectively,

which is much higher than WiFi's 4.12Mbps (downlink) and 0.94Mbps (uplink), as well as 4.67Mbps (downlink) and 1.16Mbps (uplink) for WiMAX. The 3G family, including eHRPD, EVDO_A and HSDPA clearly lag behind LTE. We also observe relatively high variation on LTE throughput for different users at different locations, and even for the same user at the same location across different runs.

In terms of RTT and RTT jitter, LTE with the median RTT 69.5ms and RTT jitter 5.6ms is comparable to WiFi's 64.5ms (RTT) and 7.9ms (RTT jitter). Similar to throughput comparison, WiMAX also lags behind LTE with median RTT to be 125.0ms and RTT jitter to be 21.0ms. For 3G networks, compared with our previous results obtained in 2009, median RTT has been reduced from a median of 400ms to below 200ms, some even has a median of 78ms (HSDPA 2), close to LTE RTT. RTT jitter for 3G networks are all larger than 10ms.

To summarize, Figure 3.14 provides a fair comparison on performance distribution among different mobile networks, from which we observe that LTE significantly improves network throughput as well as RTT and jitter, making it comparable to or even better than WiFi for throughput. WiMAX lags behind LTE, followed by HSDPA, eHRPD and EVDO_A networks.

3.4.2 Network-based LTE Parameter Inference

Network-based approach for LTE parameter inference is used to validate the power-based approach (Section 6.1). We use the following experiment to infer LTE state machine and **RRC_IDLE** DRX parameters. The LTE phone maintains a long lived TCP connection with a test server. For each experiment, server first sends a packet P_1 to trigger UE's promotion to **RRC_CONNECTED**, and after X seconds, server sends another packet P_2 to the phone to measure RTT. Assume the RTT at **RRC_CONNECTED** is RTT_b ignoring the minor impact of DRX in **RRC_CONNECTED**. When $X \leq T_{tail}$, $RTT(X) = RTT_b$. Otherwise, $RTT(X) = RTT_b + T_{pro} + T_{drx}$, where $T_{drx} = T_n - X$ and T_n is the start time of the next

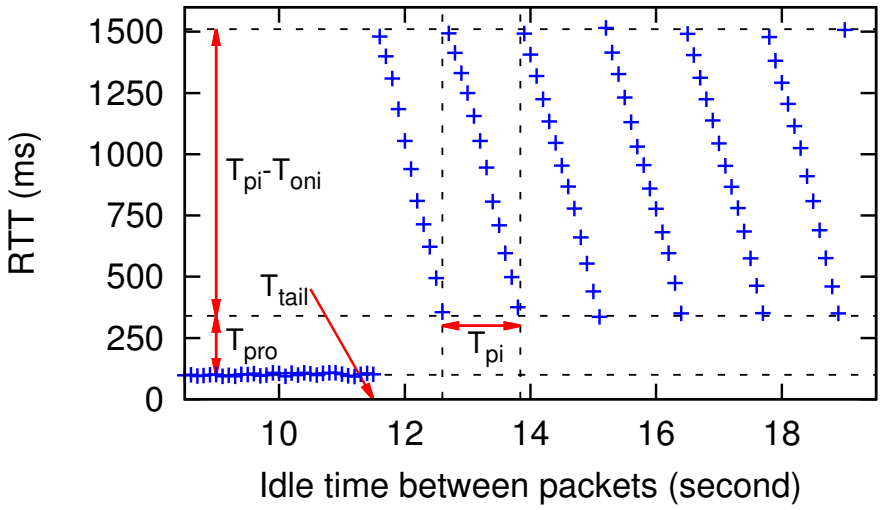


Figure 3.15: LTE RTT v.s. inter-packet idle time.

DRX on duration after P_2's arrival. If P_2's arrival is inside any DRX on duration, $T_{drx} = 0$.

Figure 3.15 summarizes the network-based approach for LTE state machine and DRX parameter measurement using the LTE phone. Each point is the median of five runs and for each run, RTT is measured after making the device idle for a specific time. The pattern of RTT is expected as inter-packet idle time changes. Based on these results, we can infer the values of the following parameters in Table 2.1: T_{tail} is around 11.5s, T_{pi} is 1.28s and T_{oni} is close to 45ms. In addition, we observe T_{pro} is around 260ms. These measured values agree with those measured from the power-based approach in Table 6.1, cross-validating our measurement methodology.

3.4.3 One-way Delay and Impact of Packet Size

To understand the **impact of packet size on one-way delay (OWD)**, uplink and downlink OWD is measured with varying packet size. For each packet size, 100 samples are measured, and we make sure that packets are not fragmented. OWD is measured between the test laptop and the test server. A GPS Receiver [44] is connected to the test laptop, which uses the received GPS messages for time synchronization. The test server is also time-synced with another GPS clock. The maximum error of our setup is less than 1ms.

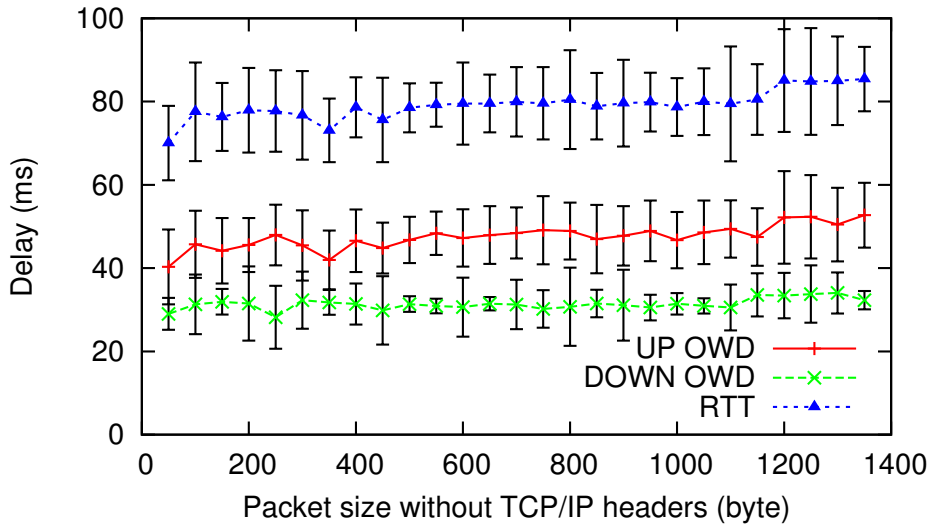


Figure 3.16: OWD/RTT v.s. packet size in LTE network (controlled experiments).

With this setup, we measure uplink/downlink OWD for both LTE and WiFi networks. Experiment in WiFi shows that both uplink and downlink OWD are around 30ms with little correlation with packet size, and RTT is stable around 60ms. However, for LTE, Figure 3.16 shows that uplink OWD is clearly larger than downlink OWD. We also observe that uplink OWD for LTE slightly increases as packet size grows, while the median of downlink OWD stays more stable around 30ms. The median RTT ranges from 70ms to 86ms as packet size increases. In summary, RTT in LTE is more sensitive to packet size than WiFi, mainly due to uplink OWD.

3.4.4 Mobility

Requirements for LTE [11] specifies that mobility across the cellular network should be supported and optimized for low mobile speed from 0∼15 km/h. Higher mobile speed of 15∼120 km/h should be supported with high performance. In this paper, we carry out one of the first studies to quantify the impact of mobile speed for a commercial LTE network.

As shown in Figure 3.17, we measure RTT and downlink/uplink throughput with MobiPerf at three different mobile speeds: stationary, 35mph and 70mph, highway speed limit. All experiments are done in the same car for fair comparison. Given that MobiPerf takes less

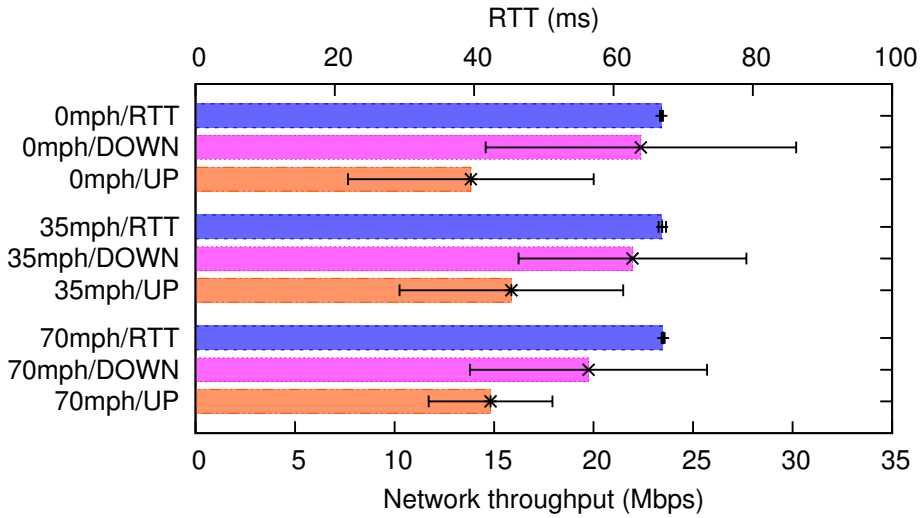


Figure 3.17: LTE performance at different speeds.

than 2 minutes to finish, handover seldom happens during a single test and we filter out samples with cell ID changed during the test. Previous work [45] has shown that signal strength varies across locations and has a non-negligible impact on performance, so we only select sample points with the similar signal strength for fair comparison.

We observe that RTT remains stable at different speeds, with small variation. Uplink and downlink throughput both have high variation of 3∼8Mbps. Comparing throughput at different speeds, there is no clear difference, given the high variation. Our experiments show that at least at our test location, there is no major performance downgrade at high speed for LTE, satisfying the requirements in [11].

In addition, we also study the correlation between LTE performance and **time of day**. We periodically run MobiPerf for 48 hours in two randomly selected days. RTT's median value remain stable at 68ms across different hours. For downlink and uplink throughput, variation across different hours is observed; however, there is no strong correlation with time of day. This may imply that, at our location, LTE currently does not have too many users and has well-provisioned resources available.

To summarize, with both global experiments, via MobiPerf, and local controlled experiments, we observe that LTE has significantly improved network performance over 3G,

making it comparable to, if not better than, WiFi.

CHAPTER IV

Anatomizing Smartphone Application Performance

Unlike traditional Internet-based applications, whose performance is mostly constrained by the wired network, network application performance on smartphones with limited physical resources also heavily depends on factors including hardware and software on the phone as well as the quality and load of wireless link. Understanding the application performance on smartphones is important for the purpose of assisting consumers in choosing carriers and phones and guiding application developers in designing improved software. Moreover, cellular network operators and smartphone hardware and software vendors can use this knowledge to optimize networks and phones for better end-user experiences. Similarly, content providers can leverage this knowledge to better customize content for mobile users. However, this task is quite challenging since the performance of network applications on smartphones is poorly understood before our study, due to a lack of a systematic approach for controlled experiments and comparative analysis. We believe this work fills this gap.

We focus on developing systematic methodology for measuring and analyzing smartphone application performance. We make it relevant to end users by studying real applications directly on the phone platforms. Our approach differs inherently from most previous work of using laptops equipped with 3G data cards in three ways: (1) We measure the performance of applications rather than that of the low-level protocols. Prior work has shown

that application performance often significantly deviates from protocol performance [46]. We target the pervasive web browsing, streaming video, and VoIP applications that most end-users care about; (2) We measure application performance on several common mobile devices. Application performance varies widely across devices due to differences in hardware and software, necessitating direct experimentation on smartphones instead of on laptops with wireless cards; (3) We study the application performance under real-world scenarios and quantify the performance of web browsing by evaluating commercial websites in addition to locally-constructed ones with replicated, real web content under our control. The latter setup helps dissect and analyze the individual factors that contribute to the overall web browsing performance.

In addition to shedding light on the overall application performance, we perform detailed analysis to identify and isolate factors that impact user-perceived performance to help carriers, phone vendors, content providers, and application developers gain insight. For example, for carriers, we infer various network-level problems, *e.g.,* high latency or high loss rate, which they can directly take action on. For phone vendors, we identify performance bottlenecks on the devices or issues associated with the content. These issues can be resolved either independently or by cooperating with content providers. And for application developers, we evaluate factors such as the overhead of HTML rendering and JavaScript execution given a particular software configuration.

We comprehensively study the smartphone application performance for all four major U.S. wireless carriers including AT&T, Sprint, Verizon, and T-Mobile. We choose popular devices including iPhone, Android G2 from HTC, and Windows Mobile phones from Palm, HTC, and Samsung for carrying out experiments. Our results show that their performance varies significantly across network applications. In fact, even for the same network application such as web browsing, certain types of phones consistently outperform others due to the differences in factors such as downloading behavior, customized contents, and page rendering. The application performance also heavily depends on properties of carriers

including DNS lookup, RTT, and loss rate.

We summarize our main observations from extensive experimentation:

1. Besides networks, devices heavily influence application performance. Given the same content and network condition, different devices exhibit vastly different webpage loading time, *e.g.,* the page loading time of Samsung SCHi760 is consistently twice that of iPhone.

2. Mobile devices can benefit from new content optimization techniques like the data URL scheme, *e.g.,* page loading time for GPhone can improve by 20% in our experiments, despite its already good performance compared to other devices.

4.1 Methodology and Setup for Measuring Web Browsing Performance

In this section, we present our methodology and setup for measuring web browsing application performance over 3G networks. By analyzing the performance of web applications, we examine the effects of various factors on the overall application performance.

Unlike most previous works, we directly measure application performance on devices that consumers really use with 3G service provided by four major cellular carriers in the U.S., this helps us understand the client side factors and their impact on application performance. The novelty of our measurement methodology stems from our approach of approximately replicating the 3G network condition for controlled experiments using WiFi to enable reproducibility, and isolating the impact of each factor. These techniques are non-trivial given the complexity of mobile devices and network environment, and essential for eliminating interaction across factors.

Web browsing is one of the most popular smartphone applications. The process of visiting a webpage can be quite complex given the dynamic nature of the content, often generated from JavaScript, resulting in multiple concurrent TCP connections. Content can also be customized based on mobile device and carrier network.

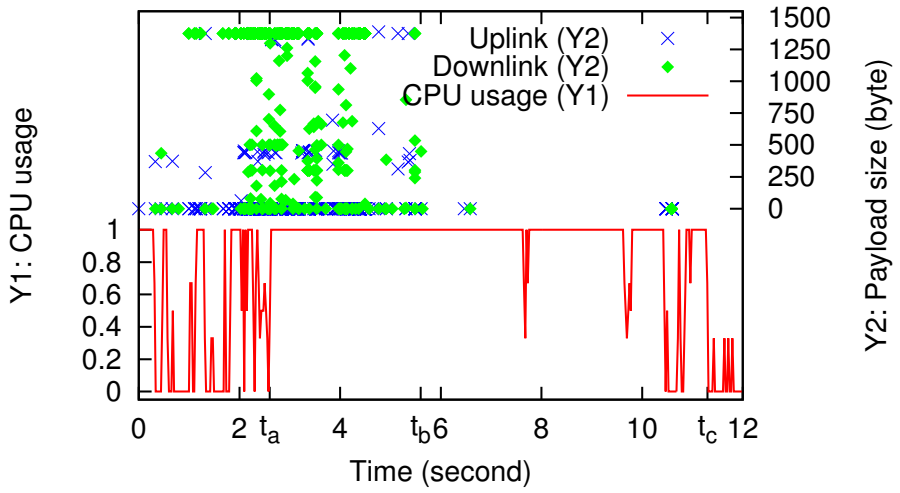


Figure 4.1: Network and CPU trace for Website Y in LTE.

Web browsing performance depends on various factors, *e.g.,* DNS lookup time, TCP handshake time, TCP transfer time, JavaScript execution time, and content size. To study the effect of these factors, we carefully design controlled experiments to modify a single factor at a time while keeping others the same. We first describe the metrics used to evaluate web browsing performance, followed by the controlled experiments to measure these metrics.

4.1.1 Web Browsing Performance Metrics

Application loading time: To measure application loading time, we use CPU usage as an indicator. Figure 4.1 shows co-located network and CPU trace of an Android device visiting popular website in its default browser in the LTE network. At time 0, when GO button is clicked, loading starts. Before t_a, CPU usage stays low most of the time given that UE has not finished downloading the HTML or JavaScript objects. Starting from t_a, CPU usage jumps to 100% and remains a high average usage until t_c. We notice that network activity nearly stops after t_b, with only a few TCP FIN packets for closing TCP connections afterwards, some even come seconds later, *e.g.,* at time 10.5 seconds. During the time between t_a and t_c, UE is rendering HTML pages or executing JavaScript, and

during the time between t_a and t_b, UE is also downloading web objects in parallel.

We define t_c to be the *application loading time* instead of t_b, since at t_b, UE has not fully rendered the contents for the user, though the download process is complete. We validate this by collecting video traces with a camcorder facing the screen of the phone. We replay the video at normal speed and manually mark the start time and the end time, when the website gets fully loaded and all UI activity indicators stop. We verify that t_c is an accurate estimate for application loading time. We also define the average CPU usage between time 0 and t_c to be the *CPU usage* for this application.

Compared with a previous approach for measuring web browsing performance by modifying and instrumenting WebKit [47], our approach is more lightweight and easily extensible to other applications other then web browsers, with reasonable accuracy. One limitation of our approach is that it is not applicable for applications, *e.g.,* streaming video/audio and some game applications, whose CPU usage remains high after initial loading.

JavaScript execution speed: Many webpages contain JavaScript, and hence JavaScript execution speed has significant impact on page rendering time.

Page size: The total number of unique bytes downloaded. It can be used to compute *average throughput* and to detect content variation and customization. We found that in typical web browsing, even the same URL can have different page sizes when accessed from different platforms. We cope with this effect by taking snapshots of URLs and replicate their content on our local web server.

Browser concurrency: Most modern browsers support concurrent TCP connections to a single web domain. The maximum number of concurrent TCP connections to a domain varies across different browsers. Usually, higher concurrency enables better bandwidth utilization which in turn leads to shorter page loading time.

DNS lookup time: A browser sometimes needs to look up the IP address of a domain name before establishing a TCP connection with the web server. Since the content of a webpage can be hosted in multiple domains, a browser may have to perform a DNS lookup for each domain.

TCP handshake time: Each TCP connection starts with a three-way handshake during which no data is transferred. More TCP handshakes for a single page loading often lead to longer page loading time.

TCP idle time & transfer time: Given a TCP connection, an *idle period* is defined to be a period of at least T second with no network activity. The remaining time periods within the connection are *transfer periods*. An idle period usually corresponds to the local processing delay or server processing delay. Given the limited CPU power and memory on smartphones, the TCP idle time is likely to be dominated by local processing delay, *e.g.,* between the receipt of a response and transmission of the next request, often caused by HTML rendering and JavaScript execution.

4.1.2 Web Browsing Controlled Experiment Setup

We select a list of 20 popular and representative URLs (based on Alexa Top 500 Global Sites [38]) including search engines, emails, online maps, social networking websites, *etc.* For most of the URLs, we use their mobile versions, specially optimized for smartphone users, as opposed to desktop versions. To facilitate repeated experiments, we write a program to invoke browser to visit each URL in turn with an interval of 120 seconds. Such interval is expected to be large enough to complete the page download. We used *Apache* 2.0 HTTP server for hosting the replicated websites. We collect packet traces on iOS and Android using *tcpdump* and on Windows Mobile phones using *netlog*. We verify that the CPU utilization caused by trace collection is under 5%. All the experiments are repeated at least 10 times.

These websites are visited using smartphones via 3G networks. From the collected packet trace, we infer various metrics such as page loading time. To study the effect of each factor influencing the web browsing performance, we host static copies of these popular URLs on our local web server. The content is replicated to ensure that all the phones download the same content and all HTTP requests are sent to the local server. To control the network conditions, we uniformly use WiFi across all phones while varying one factor at a time. The WiFi link is lightly loaded and has stable throughput and RTT. To produce network conditions comparable to 3G, we artificially introduce delay and packet loss at our server. We study the impact of the following factors on web browsing performance:

Impact of network: To study the effect of network conditions on page loading time, we vary the RTT and loss rate on our server. To introduce artificial delay and loss, we ran a user-level program on the server. This program intercepts all packets destined to a particular IP address and injects delay and random packet loss. We controlled loss rate values from 0% to 10% and RTT values from 0 ms to 800 ms. These values cover the major range of loss rate and RTT values observed in 3G networks from our previous analysis.

Impact of concurrency: To study the effect of concurrency, we control the maximum number of concurrent TCP connections to a web domain on the server side by configuring the *Apache* server with the help of the *mpm_prefork_module*.

Because a phone also limits the maximum number of concurrent connections per domain, we create a special webpage with 30 embedded objects in which each web object is hosted in a unique domain (a DNS alias) on the same web server. This effectively allows us to bypass the per-domain concurrency limit imposed by the phones. Note that this is necessary as we do not have the permission to directly modify the concurrency limit on the phone. For concurrency experiments, we use the RTT of 400 ms and the loss rate of 0%.

Impact of compression: To study the tradeoff between network overhead and computation overhead, we configure our web server into two modes, one uses compression, while the other does not. We compare the page loading time under these two modes. Specifically, we use *SetOutputFilter*and *BrowserMatch* directives to specify whether compression is enabled for a specific type of browser, *i.e.,* compression is bypassed by adding the following lines to the Apache config file:

```
<Location />
      SetOutputFilter DEFLATE
      BrowserMatch Mozilla no-gzip
</Location>
```

We fix the loss rate at 0% and vary the RTT from 0 ms to 800 ms. Our goal is to understand whether compression is beneficial under different network conditions.

Impact of JavaScript execution speed: To evaluate JavaScript execution speed on different phones, we use a benchmark [48] consisting of 26 different JavaScripts. The benchmark is hosted on our web server and accessed by phones via WiFi so that the downloading time is negligibly small. We measure the total execution time of these JavaScripts.

Impact of the data URL scheme: We also study the effect of the data URL scheme [49], a recently-proposed mobile webpage design technique. We compare the time to load a webpage constructed using and without using the data URL scheme. Specifically, we construct a webpage with 20 images; 10 of them are of size 18KB and the remaining 10 are of size 0.241KB. We create two versions of this webpage, one with links to download the images and the other with the images embedded in the webpage itself. We could not carry out this experiment for Windows Mobile phones since IE does not support the data URL scheme in 2009.

4.1.3 Analysis Methodology

We next describe how to analyze the traces collected from controlled experiments to compute the desired metrics. We calculate the page loading time of each URL as defined in Section 4.1.1 and the average page loading time of all the selected URLs. To measure JavaScript execution time, we modify the JavaScripts to display their execution time when their execution finishes. We use the *average concurrency* as a measure of browser concurrency. The average concurrency of a page loading is calculated by dividing the total duration of all the TCP connections by the page loading time.

For each TCP connection, TCP handshake time is calculated as the time between the first *SYN* and *SYN-ACK* packets. TCP idle time is measured by scanning the connection for durations of more than *T* seconds of no network activity. *T* should be larger than the maximum RTT values. In our analysis, we choose *T = 1* second, as we have seen that in cellular networks, even for 3G networks, RTT is smaller than 1 second in 99% of the cases. Thus, if there are no network activities for 1 second or more, the phone should be busy with some local processing, *e.g.,* rendering HTML pages or executing JavaScripts. TCP transfer time is what remains for the connection excluding handshake and idle time. We also calculate the response time of all the DNS lookups T_{dns}.

Since each web browsing session often consists of multiple concurrent TCP connections, to estimate the contribution of each factor to the overall performance, we logically serialize all DNS lookups and TCP connections. This is possible for mobile web browsing since no HTTP pipelining is observed on any phones. After serialization, we get a total time T_{total} which is the sum of each connection's duration. Assuming the actual page loading time is T^*_{total}, the normalized DNS lookup time T^*_{dns} is calculated as

$$T^*_{dns} = \frac{T_{dns}}{T_{total}} T^*_{total}$$

This metric shows the overall weight of DNS lookup in the actual page loading time. The

normalized TCP handshake time, TCP idle time and TCP transfer time are calculated in a similar way.

4.2 Web Browsing Performance Study

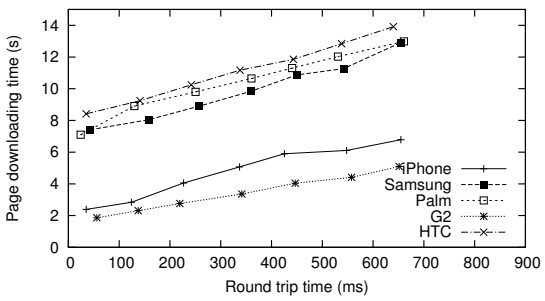
Given the previous discussions on the performance of 3G networks, we now examine one of the most popular applications on smartphone, namely web browsing, in addition to two other popular mobile applications, streaming video and VoIP in §4.3. Note that many factors jointly determine user perceived performance, as an application may not fully utilize available network bandwidth due to limited processing power or memory on the phone [46].

Our study shows that the available 3G bandwidth is often not fully utilized for web browsing, and several modifications can be applied to current web browsers on smartphones to make better use of available network resources, *e.g.,* increasing the limit on concurrent TCP connections per domain, optimizing JavaScript engines *etc.* We also evaluate the effectiveness of a few content optimization techniques, including compression and the recently-proposed data URL scheme [49].

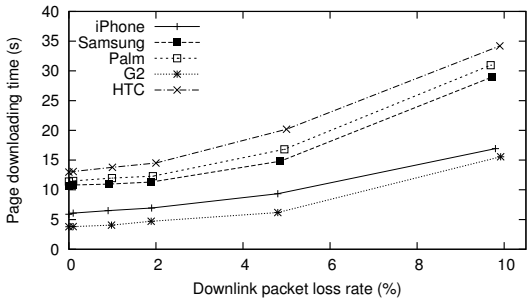
In the following, we study the impact of network condition, browser concurrency, compression, and JavaScript execution speed on web performance. We then break down the page loading time into several major components and identify the performance bottleneck for some mobile devices (§4.2.6).

4.2.1 Network Effects on Web Browsing

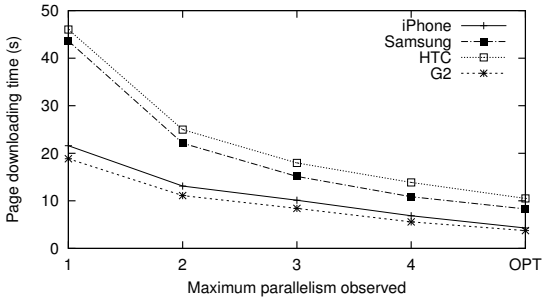
To understand how network condition affects web browsing, we fix the web content, server configurations, browser concurrency and only vary the network condition. We emulate the 3G network condition by injecting packet delay and loss on the WiFi network path as described in §7.3.



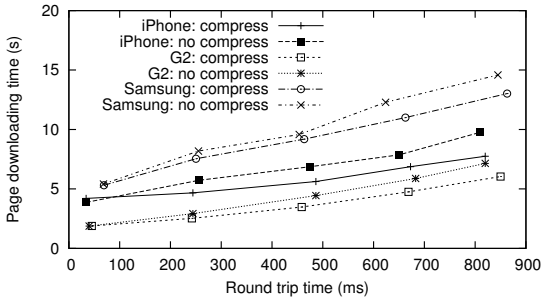
(a) Effect of round trip time on web browsing



(b) Effect of downlink packet loss rate on web browsing



(c) Effect of parallelism on web browsing



(d) Effect of compression on web browsing

Figure 4.2: Factors impacting web browsing performance.

Figure 4.2(a) shows that page downloading time increases linearly with the RTT between smartphone and our local web server. The downloading time is computed by averaging across 20 replicated URLs, with each URL visited 3 times. This is expected as throughput is inversely proportional to RTT. No additional packet loss is introduced since packet loss is observed to be rare in 3G networks, as shown in the previous section. The base RTT in our WiFi network is between 30 ms and 50 ms. The x-axis in Figure 4.2(a) shows the actual RTT after injecting extra delay. We can observe that under the same network condition, downloading time varies across phones though the relative ranking remains consistent. Note that web browsing cannot fully utilize the available network bandwidth, due to the overhead imposed by page execution and rendering on the phone.

Figure 4.2(b) shows the effect of varying downlink packet loss rate for a fixed RTT value of 400 ms. Again the ranking in downloading time across phones is consistent. For small packet loss rate, *e.g.,* 2%, there is little performance degradation. However, with 10% loss rate, the page downloading time increases up to 35 seconds. In summary, smartphone

web browsing performance heavily depends on network delay and loss conditions.

4.2.2 Concurrent TCP Connections

3G network's downlink throughput as measured normally ranges from 500 kbps to 1 Mbps for the carriers we studied (Figure 3.2(a)). We used the phones to visit the chosen URLs and found that the average throughput is only between 20 kbps and 50 kbps, indicating that more concurrent TCP connections can potentially improve web browsing performance.

Current web browsers on smartphones already allow concurrent connections. In browser's settings, there is a parameter specifying the maximum number of concurrent TCP connections per domain. On Windows Mobile phones, it is a registry value named *MaxConnectionsPerServer* with a default value of 4. When we set the value to be smaller than 4, we observe decreased concurrency. However, when we increase the value to be larger than 4, the concurrency does not increase accordingly. This implies there exists another setting on maximum allowed concurrency per domain, which we cannot configure. For iPhone and GPhone, we are unable to set this parameter either. We design controlled experiments to measure the default concurrency setting on different platforms and found it to be 4 for all the phones studied.

We also found that no HTTP pipelining support is present on these platforms. Web objects are fetched sequentially within a persistent connection, and browser will not send a new HTTP request before data transfer of the previous request completes. We analyzed the 20 popular URLs and found that there are 10.8 images embedded in each page on average, along with several other types of embedded objects, such as JavaScript, CSS files, *etc.* Those websites which do not have a mobile version, tend to have even more objects.

To understand how concurrency affects web browsing performance, we devised a set of experiments, with results shown in Figure 4.2(c). We first vary the maximum concurrent connections allowed at the server side from 1 to 4. We observe a significant performance

URL	Text[1]	Image	Size(KB)	Original(KB)[2]	Compress[3]	GZIP	Lines[4]	Redirect	Sever IPs
www.google.com	4	1	79.2	77.6	2.56	✓	14	-	2
m.bing.com	4	3	42.9	218.1	1.46	-	2	-	1
maps.google.com	6	10	479.8	656.0	2.78	✓	8	-	4
mapquest.com	6	13	135.1	1326.35	1.96	✓	752	2	6
xhtml.weather.com	22	9	41.4	977.3	2.53	-	70	4	2
m.youtube.com	5	3	77.6	490.1	2.34	✓	231	-	3
m.ebay.com	4	3	58.6	484.0	2.17	-	1	-	1
m.facebook.com	4	1	19.7	399.1	2.81	✓	7	2	2
m.myspace.com	3	2	14.6	600.2	2.6	✓	98	1	2
m.fox.com	4	26	306.6	2083.0	1.16	✓	297	-	4
mobile.craigslist.org	3	0	113.8	113.8	3.58	✓	652	-	1

[1]This column shows the number of text objects including HTML, JavaScript and CSS files
[2]This column shows the total size of the original website for each mobile URL, for example, www.bing.com for the row of m.bing.com
[3]This column shows the compression ratio for mobile URLs, total size in no compression mode / total size in compression mode
[4]This column shows the total number of lines in the index page indicating whether minification is used

Table 4.1: Characteristics of today's popular mobile websites.

degradation across all platforms with more restricted concurrency. Under the restriction of a single connection per domain, web browsing is 3.5 to 4.5 times slower compared to that under the default setting. This indicates that today's mobile browsers already benefit much from concurrency.

To understand whether further increasing concurrency will improve performance, we make use of DNS aliasing (Section 4.1.2) to bypass the concurrency limit on the phone since we are unable to change this setting directly. Figure 4.2(c) shows that the phones can indeed attain a higher level of concurrency. For example, iPhone and G2 can establish up to 9 concurrent connections for some content-rich URLs. The concurrency for other phones are slightly lower (6 to 7), likely due to their slower rendering and execution speed. Generally, an improvement of 30% is observed when concurrency limit on the phone is removed. This means that given the selected popular URLs, and given current network condition (with RTT of 400 ms), the default concurrency setting on mobile browsers appears to be too conservative. Allowing higher concurrency can help phones make better use of available bandwidth and decrease page downloading time.

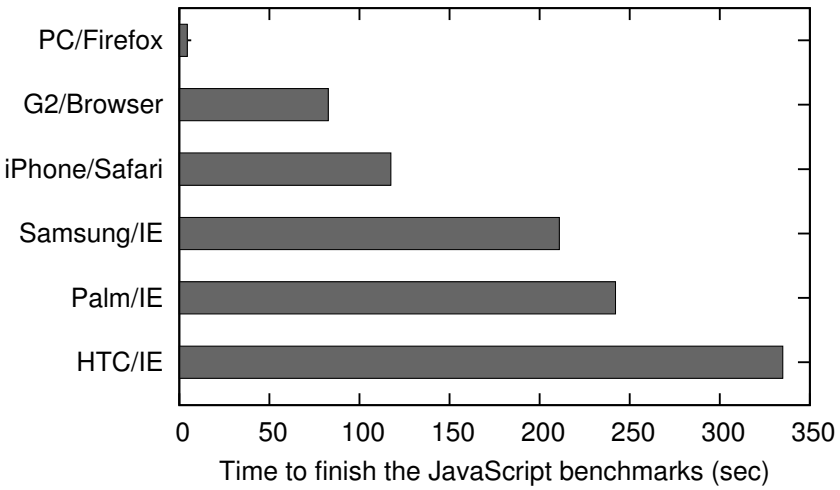


Figure 4.3: Script execution speed for different platforms in 2009.

4.2.3 Content Compression

Compression can dramatically reduce web content size. For text objects, such as HTML, CSS, JavaScript, PHP, *etc.*, the object size can be reduced by around 70%. Usually, a web server does not compress image objects. We calculate the compression ratio for the popular URLs in column *Compress* of Table 4.1, showing that the content size can be reduced by more than 50% for most of the URLs we studied.

While compression reduces the bytes transferred over the network, decomression will increase computation overhead on the phone. To understand this tradeoff, we vary RTT covering the measured range and compare the web browsing performance in compressed and uncompressed modes. In Figure 4.2(d), we exclude the results for HTC and Palm phones as they show similar trends. We observe that compression consistently helps to improve web performance, irrespective of the RTT values. It is especially helpful under poor network condition. For example, it reduces iPhone's page downloading time by 30% when RTT is 800ms.

4.2.4 JavaScript Execution

Given the limited processing power on smartphones, HTML rendering and JavaScript execution may become the bottleneck for web browsing. Several factors jointly determine the page processing speed, including CPU frequency, memory, OS, and browser. Even for the same OS, such as Windows Mobile 6.1, phone vendors can have different builds for different models of phones.

We measure JavaScript execution time on different phones using a benchmark consisting of 26 different JavaScripts [48]. Figure 4.3 shows the total time taken to execute the benchmark on different phones measured in 2009. The results demonstrate that execution time is 20$\sim$80 times longer on smartphones than on desktop computers. Among the smartphones, G2 has the best performance followed by iPhone. For example, G2 is 3 times faster than the HTC phone. Such performance gap helps explain the differences in the page loading time of G2 and iPhone compared to that of the Samsung and Palm phones under the same network conditions in Figure 4.2. Large JavaScript execution time leads to more TCP idle time and under-utilization of network bandwidth.

This experiment shows that network is not the only bottleneck for web browsing performance. Phone itself also plays a major role, underscoring the necessity of measuring application performance on real smartphones. Web designers should avoid using complex JavaScripts when building mobile versions of their websites.

We revisited the JavaScript execution experiments in 2011, as shown in Figure 4.4. We consistently use the same benchmark [48] (version 0.9) to quantify the JavaScript execution speed. In Figure 4.4, the test results on the same G1 phone in 2009 and in 2011 are very close, validating that the benchmark of that version has not changed since 2009.

From 2009 to 2011, iOS has a speedup of 29.88 for iPhone 4 and 51.95 for iPhone 4S, while 21.64 for Android and 22.30 for Windows Phone. And the gap between smartphone and computer has dropped to 5.5$\sim$23.1 times in JavaScript execution speed. Possible reasons for this improvement include fast CPU, larger memory, and better OS and application

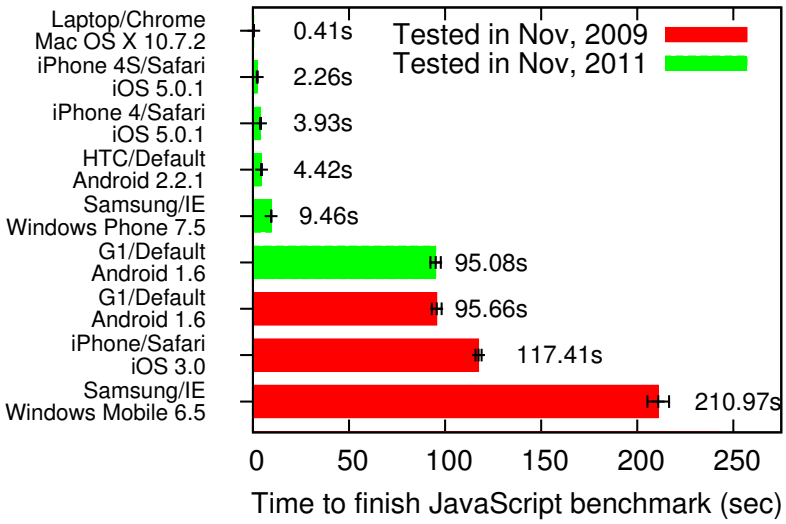


Figure 4.4: JavaScript execution speed comparison.

software for smartphones.

4.2.5 Server Configuration and Content Optimization

Server configurations and content optimization are important factors for web browsing performance. One type of server configuration is the maximum concurrent connections with a client. In §4.2.2, we found that mobile browsers set a default concurrency limit of 4 per domain. However, we did not observe any web servers limit the concurrency per client to be smaller than 4, likely because servers have the incentives to attain good web browsing experience. The compression configuration is similarly important, with the identified setting of the URLs studied shown in the *GZIP* column in Table 4.1. Despite the fact that compression almost always improves web browsing performance (§4.2.3), we found some websites do not enable it by default.

Various content optimization techniques also help improve web browsing performance on smartphones. Most popular websites already customize their contents for mobile users, with more concise texts and fewer and smaller images, *e.g.,* via code minification [50], image scaling, *etc.* We study in particular code minification which refers to the elimination

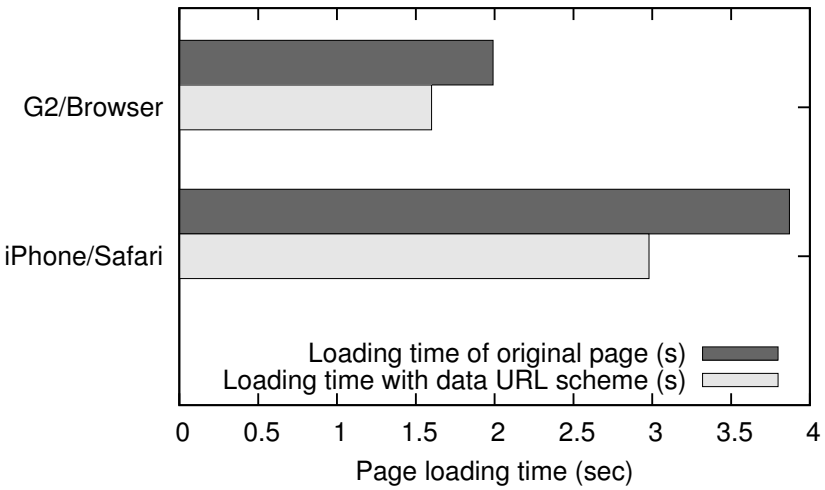


Figure 4.5: Evaluation for data URL scheme.

of redundant characters, such as spaces, tabs, line breaks, *etc.* The size reduction varies from 5% to 25% for the URLs studied. Column *Lines/index* in Table 4.1 shows the number of lines in the index page of a website, providing a hint for whether minification is used. The number of lines will be small for index pages treated with minification. It seems that half of the URLs use this technique to optimize their contents.

Another type of optimization helps reduce the number of HTTP requests used to fetch contents, including the data URL scheme [49], CSS3, *etc.* The general idea is to eliminate TCP connections and HTTP requests for small objects, such as the corner image of a page. We set up a controlled experiment to demonstrate the effectiveness of the data URL scheme, under which small images are integrated with the main HTML page rather than linked as separate objects (§4.1.2). In our experiment, we found that the images are actually 1.3–1.5 times of its original size under the data URL scheme. Figure 4.5 shows that it cuts page loading time by about 20%.

The data URL scheme has not been ubiquitously supported. In fact, only the browser of iPhone and G2 supports it. We also did not observe any URLs we studied adopt this technique, possibly due to the concern of a lack of browser support. Without browser support, the image represented by the data URL scheme will be displayed as a default error

image.

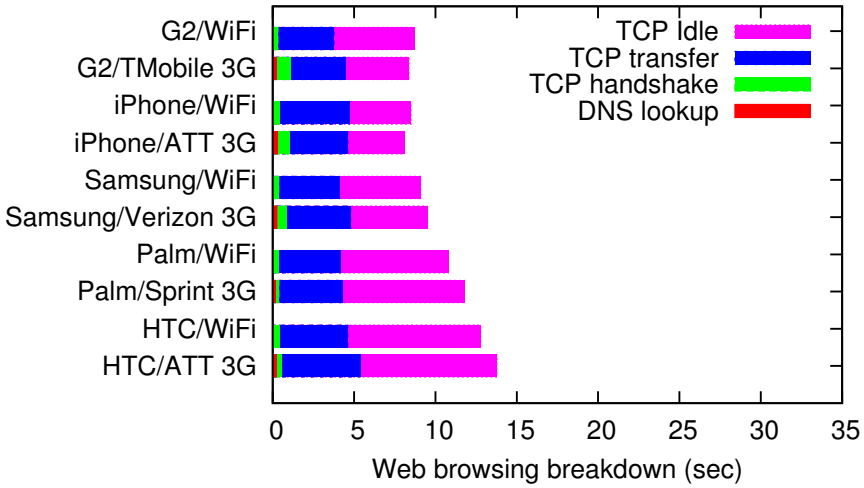
Redirection (HTTP response code 301 and 302) is another issue which may adversely impact web browsing performance. For mobile web browsing, this issue becomes more pronounced given the large RTTs in 3G networks. In column *Redirect* of Table 4.1, we found that some websites have multiple levels redirections. For example, `m.weather.com` will be redirected to `xhtml.weather.com` and then to `mw.weather.com`. In some cases, users are redirected to another URL which is quite similar to the original one. In other cases, web objects have been moved to a new location, and the original URL simply redirects the incoming requests to the new location. We think some of these redirections are unnecessary and can be eliminated with better webpage design.

4.2.6 Understanding Bottleneck Factors for Web Browsing

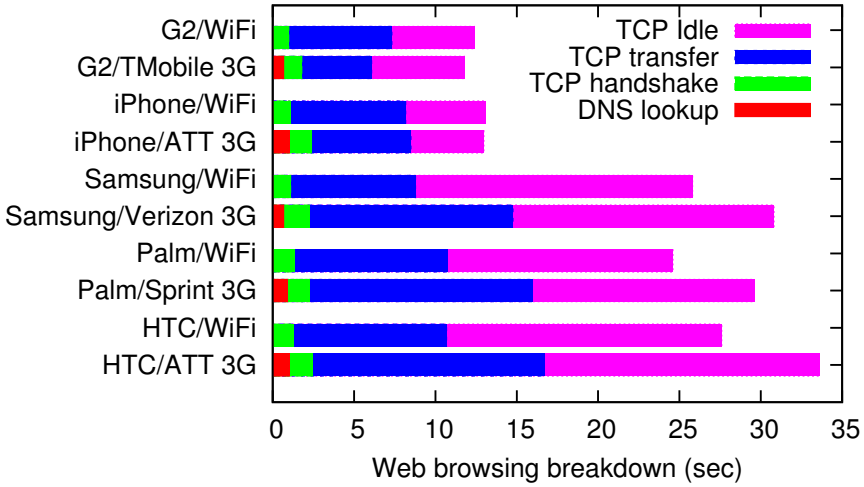
Figure 4.6 shows the case study for two groups of URLs listed in Table 4.1. *Group A* corresponds to the URLs that have concise and simple contents, *e.g.,* `m.ebay.com` (*URL a*) contains 7 objects with a total size of 58.6 KB. Many of these websites are search engines or portals to social networking sites, including `www.google.com`, `m.bing.com`, `m.myspace.com`, and `m.facebook.com`. *Group B* consists of websites with rich contents, *e.g.,* `mapquest.com` (*URL b*) has 19 objects with a total size of 135.1 KB. Other websites in the group include online map (`maps.google.com`), information exchange (`mobile.craigslist.org`), and news (`m.fox.com` and `m.cnn.com`).

There are two sets of data in Figure 4.6. One set is collected when each smartphone visits the real URLs via 3G networks. To eliminate the differences in downloaded contents and network conditions, each phone also visits the replicated URLs via WiFi with an RTT of 400 ms to emulate the typical 3G network conditions.

It is clear that all smartphones experience smaller page loading time for the simple URL in Figure 4.6(a) compared with that for the content-rich URL in Figure 4.6(b). We break down the page loading time into four parts: TCP transfer, TCP idle, DNS lookup,



(a) Web browsing breakdown for `m.ebay.com` (*URL a*)

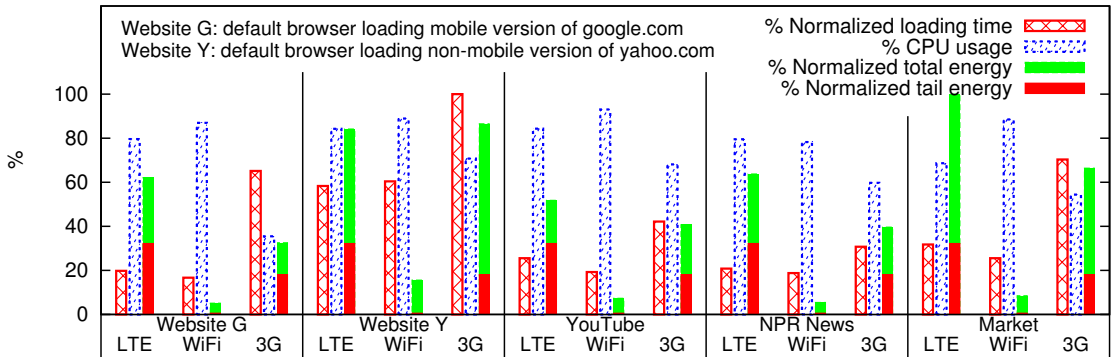


(b) Web browsing breakdown for `mapquest.com` (*URL b*)

Figure 4.6: Web browsing anatomy for two popular mobile websites.

and TCP handshake. The size of `mapquest.com` is larger than that of `m.ebay.com`, resulting in longer TCP transfer time. Moreover, `mapquest.com` contains more contents to render and more complex JavaScripts to execute, leading to longer TCP idle time. The DNS lookup time and TCP handshake time contribute to less than 10% of page loading time, which are negligible.

We further observe that the Palm (Sprint), Samsung (Verizon), and HTC (AT&T) phones experience much longer page loading time for `mapquest.com` compare to iPhone (AT&T) and G2 (T-Mobile). This is likely due to their slower JavaScript execution speed, as shown



⋆ The loading time is normalized by the maximum loading time observed in this figure, and similarly for total energy.

Figure 4.7: Loading time, CPU usage and energy consumption analysis on mobile applications.

in Figure 4.3.

In the WiFi experiments, all the phones download the same contents and experience the same network conditions. As a result, the TCP transfer time differences among all phones are small. However, we can still observe significant page loading time differences, mostly due to the gap in TCP idle times. We further note that their relative ranking is consistent with the ranking of JavaScript execution speed in Figure 4.3.

The previous part of the study was carried out in 2009. Along with improvements in mobile client processing power, LTE technology significantly increased network speed. So we perform another case study of a few popular applications to understand the impact of network speed and processing power enhancement on user-perceived performance in 2011.

With the measurement methodology discussed in Section 4.1.1, we compare loading time, CPU usage, and energy consumption for several popular applications in Figure 4.7. We collect both network and CPU traces on an Android device with LTE data access. Specifically, we select the default browser, YouTube, NPR News and Android Market as our sampled applications given their popularity. For the default browser, we choose two different usage scenarios, *i.e.,* visiting mobile version of `google.com` representing a simple website and visiting non-mobile version of `yahoo.com` representing a content-rich website. We name the two browser usage scenarios as Website G and Website Y, respectively.

Application	Total payload	Maximum object size	# objects	Average size
Website G	160.7KB	87.6KB	11	14.6KB
Website Y	509.9KB	124.2KB	59	8.6KB
YouTube	449.1KB	109.4KB	33	13.6KB
NPR News	306.2KB	169.5KB	9	34.0KB
Market	599.7KB	145.4KB	17	35.3KB

Table 4.2: HTTP object statistics.

While for the other applications, we click the application icon until the initial page is fully presented to the user. We make sure the cache is cleared before each experiment. These usage scenarios are representative for web browsing and other similar web-based applications, different from audio and video streaming applications.

First, loading time comparison shows that LTE and WiFi have comparable user-perceived delay for the studied scenarios, and 3G lags behind with 50%$\sim$200% larger response time. The application loading time for LTE slightly lags behind WiFi. On one hand, given that promotion delay is counted into the loading time, LTE has larger T_{pro}. On the other hand, though LTE has faster throughput than WiFi with throughput tests, its RTT is slightly larger. To understand why RTT matters more than throughput for the usage scenarios in Figure 4.7, we analyze the object size statistics for different applications in Table 4.2.

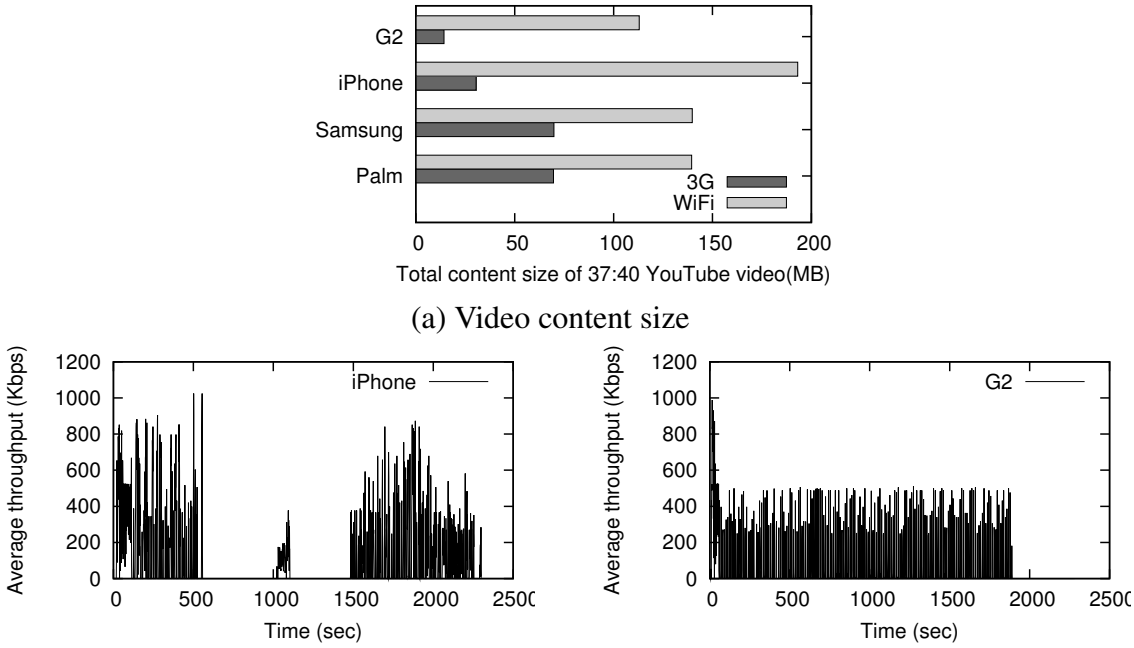
In Table 4.2, across all scenarios, the maximum object size is 169.5KB for NPR News and the average object size is at most 35.3KB for Android market. Especially for Website Y, there are 59 HTTP objects with an average size of only 8.6KB. Due to TCP slow start, these objects are far from saturating the link capacity for either LTE or WiFi. Persistent TCP connections are observed to transfer multiple web objects; however, no HTTP pipelining is observed. We also observe up to 5 parallel TCP connections to different remote servers and even to the same remote server; however the average low throughput, *e.g.,* 364.2kbps for Website Y, indicates that the RTT is more critical than available bandwidth for these web-based applications in LTE and WiFi.

We observe that average CPU usage for 3G network is relatively low ranging from

35.5% to 70.8%, with an average of 57.7%. CPU usage for LTE is between 68.8% and 84.3%, averaging at 79.3%, compared with WiFi's 78.2%∼93.0% and 87.1% as average. This comparison implies that the gap between WiFi and cellular network has narrowed because of LTE's better network performance. For 3G network, network performance appears to be the bottleneck, resulting in underutilized processing power. Given that the average CPU usage for LTE and WiFi is already 80%∼90%, further performance optimization relies more on processing speed improvement and OS/application software enhancement, compared with network improvement. Work such as CloneCloud [51] on offloading computation into the cloud is one such promising direction.

In terms of energy usage, WiFi clearly has significantly higher energy efficiency for the usage scenarios studied, similar to previous discussions. 3G consumes 5.5∼7.8 times the energy of WiFi, with 20.9%∼55.6% of the total energy due to tail. While for LTE, energy usage is 5.4∼12.0 times of WiFi, with 32.2%∼62.2% tail energy. We notice that, most of these applications keep some of the existing TCP connections open even after the initial loading; so that if user generates subsequent requests, these connections could be reused. Similar to previous work [26], we observe that applications remain idle after the initial loading and a few TCP control packets are generated by UE to close the existing applications a few seconds later, such as the packets seen at time 10.5 seconds in Figure 4.1, which reset T_{tail} and further increase the high-power tail length.

Summary: First, we find that higher browser concurrency enables the phones to better utilize available network bandwidth, hence reducing page loading time. Second, server configurations and content optimization play a major role in web browsing performance. Compression tends to always help under typical 3G network conditions. However, a few popular websites are employing sub-optimal server configurations and page designs. Third, we observe that contemporary smartphones have reduced gap with desktop computers in terms of processing power. However, for web-based applications, downloading mostly



(a) Video content size

(b) iPhone timeline for 2260-sec video

(c) G2 timeline for 2260-sec video

Figure 4.8: Streaming video performance: content size, download strategy.

small sized objects, the performance bottleneck is still at the UE processing side, given the significant improvement in network speed for LTE.

4.3 Other Mobile Applications

In this section, we study two other popular mobile applications, streaming video and VoIP.

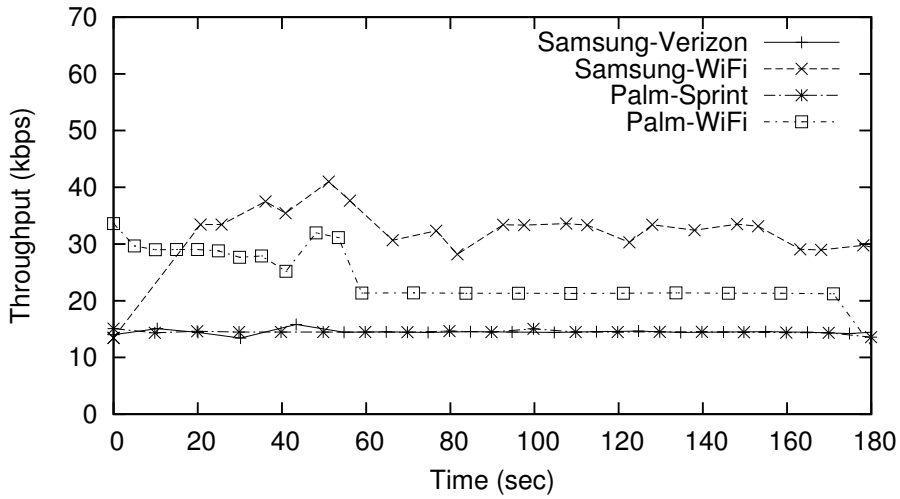
Streaming video is another popular application on smartphones. We measure streaming video performance by playing a 37:40 minute video on the phones using the YouTube application. From the collected packet trace, we calculate the downloading size of the video by adding up the payloads for all the packets from the server to phone while excluding the retransmitted packets.

4.3.1 Streaming video

We downloaded a 37-minute long video using a YouTube player on each phone. Figure 4.8(a) shows the size of the video downloaded using TCP via WiFi and 3G on each phone. As expected, the video size is smaller for 3G than for WiFi, because both the video server and 3G carrier can customize video based on network conditions to ensure good user experience. Interestingly, the video size for 3G also varies across carriers: it is the smallest for T-Mobile, followed by AT&T, Verizon, and Sprint.

Figures 4.8(b)(c) show the representative time series of video download throughput for iPhone and G2 via 3G networks. The timeline of iPhone exhibits a distinct pattern with clear pauses. It initially downloads a portion of the video at a high rate, then stops before downloading the remaining portions. We conjecture that the download stops when the buffered content exceeds a certain threshold, and resumes after the buffered content falls below another threshold. The purpose is likely to accommodate the limited phone memory and to save energy usage associated with the 3G interface. Another observation is that iPhone always terminates the TCP connection every 10–20 seconds and then establishes a new one on demand. We conjecture that iPhone attempts to put the 3G interface into low power state to save energy.

In contrast, G2 shows a different behavior by periodically downloading small chunks of the video every 10 seconds. The Samsung and Palm phones behave similarly with a slightly longer interval of 20 seconds between downloads. This is likely motivated by the fact that users sometimes do not watch the entire video and may skim through certain parts of the video. Such downloading patterns can also help conserve energy. Our initial study shows that the video players on different phones employ different policies to fetch video. This merits more detailed future study.



Downlink throughput timeline for Skype

Figure 4.9: VoIP performance.

4.3.2 VoIP

We carry out a simple VoIP experiment on the Samsung (Verizon) and Palm (Sprint) phones given their uniform support for Skype. During the experiment, the same 3-minute music file is played on both the phone and the desktop, when the two are in a Skype call. The volume is kept the same to have similar voice input. Figure 4.9 shows that the throughput for both phones via 3G is nearly identical, as the same coding rate is used. The throughput is higher under WiFi than under 3G, as different amount of data is transferred depending on the network being used. This reflects how Skype tries to vary the encoding rate according to the network condition to achieve good perceived voice quality.

CHAPTER V

An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance

4G LTE is the latest deployed cellular network technology that provides high-speed data services for mobile devices with advertised bandwidths matching and even exceeding the home broadband network speeds. Recent work [12] has demonstrated the power model of the LTE network compared to 3G provides the promise of higher energy efficiency as a result of the new RRC state machine design and higher achievable throughput. However, this new technology has not been extensively studied empirically in a deployed commercial network setting to understand how network resources are utilized across different protocol layers for real users. It is important to evaluate the benefits of increased bandwidth for popular mobile applications and essential network protocols such as TCP to identify their limitations for needed improvements. Intuitively, network protocol overheads can be significant enough to prevent efficient usage of available network resources [46]. This has been shown in network settings with high network capacity but potentially unpredictable network conditions [52].

We are motivated by the fact that LTE uses unique backhaul and radio network technologies, and has unique features distinguishing it from other access technologies (*e.g.,* much higher available bandwidth and lower RTT), requiring some existing topics to be revisited. Also, the *prevalence* of these problems in commercial LTE networks is very

important for both academia and industry. In this chapter, we evaluate the usage of LTE network resources by analyzing an extensive data trace collected in a large geographic region of a commercial LTE network. As far as we know, this is the first in-depth analysis of deployed LTE technology in a commercial setting. We systematically complement the data analysis with local experiments using controlled traffic patterns to confirm or further investigate our observations based on data traces. Given the prevalence of proxy deployment in cellular networks for improving user perceived performance due to inherently limited radio network resources, we also study the impact of such middleboxes on performance. No previous work has performed any detailed evaluation of such impact.

Our approach to characterizing the usage of a commercial LTE network starts with a careful analysis of basic network characteristics in terms of TCP flow properties, network latency, followed by the congestion control statistics of observed TCP flows. To answer the question whether application traffic is effectively utilizing available network resources, we devise a lightweight method to estimate the available network bandwidth based on the fine-grained TCP data packet and ACK packet exchange close in time, while making use of the TCP Timestamps option. We validate the accuracy of our bandwidth estimation algorithm using controlled experiments. We expect this algorithm to be helpful in identifying protocol level and application level inefficiencies even in the presence of sufficiently available network resources. Besides performance overhead, network usage efficiency has direct impact on the energy usage of mobile devices. We highlight the potential energy waste due to ineffective use of available network resources. Given the prevalence of video and audio applications in cellular networks and their significant contribution to the network resource usage, we perform a case study on popular multimedia applications from the perspectives of network resource usage.

In summary, we make the following contributions:

- Using the TCP Timestamps option, we devise a *passive* method to estimate the available bandwidth by observing the TCP packet streams between the mobile device and

the server.

- We develop a set of pragmatic techniques for passively capturing TCP flow characteristics such as flow size, flow duration, flow rate, loss rate, queuing delay, LTE promotion delay from a monitor placed between the LTE Radio Access Network (RAN) and the Serving Gateway (SGW) or Packet Data Network Gateway (PGW).

- To evaluate performance of TCP flows, we design simple heuristics to identify abnormal TCP behavior based on duplicate ACKs, out of order packets, and slow start through the analysis of packet traces and congestion window size.

Besides these methodological contributions, we make the following insightful observations about LTE network usage.

- For large TCP flows, queueing delay may increase RTT to a few times the normal value. However, as TCP does not use duplicate ACKs to update RTT estimate (thus retransmission timeout RTO), undesired slow start may occur in the middle of a flow upon a single packet loss or reordering, and this phenomenon is observed for 12.3% of all large TCP flows.

- We observe that 52.6% of all downlink TCP flows have experienced full TCP receive window or even zero receive window, limiting the sending rate.

- Overall, with the bandwidth estimation algorithm, we observe that for 71.3% of the large flows, the bandwidth utilization ratio is below 50%. And on average, data transfer takes 52.9% longer than if the bandwidth was fully utilized, incurring additional radio energy overhead.

Based on these observations, we make several recommendations on protocol and application design to more effectively take advantage of the available network resources. We believe our findings apply to other LTE networks given the extensive coverage of the data set and independent controlled experiments carried out locally.

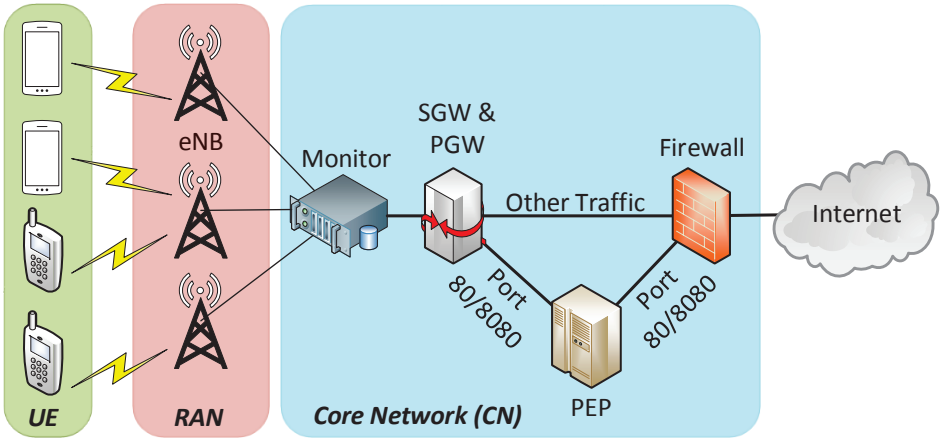


Figure 5.1: Simplified network topology of the large LTE carrier from which we obtained our measurement data.

Here is the roadmap for this chapter. Section 5.1 describes the data set studied and setup for controlled experiments. We then characterize the LTE network characteristics in Section 5.2 and discuss a newly identified TCP performance issue in LTE networks in Section 5.3. We investigate the network resource usage efficiency with a devised bandwidth estimation algorithm in Section 5.4, followed by exploring the network application behaviors that cause network inefficiency in Section 5.5.

5.1 LTE Data and Local Testbed

We give an overview of the LTE network topology before describing our measurement data. We then describe how we perform controlled experiments for validating our findings.

5.1.1 The LTE Measurement Data

As depicted in Figure 5.1, an LTE network consists of three subsystems: user equipment (UE), the radio access network (RAN), and the core network (CN). UEs are essentially mobile handsets carried by end users. The RAN allows connectivity between a UE and the CN. It consists of multiple base stations called Evolved Node B (eNB). The centralized CN is the backbone of the cellular network. It connects to the Internet. In Figure 5.1,

within the CN, "Monitor" is our data collection point. "SGW" and "PGW" refer to the serving gateway and the packet data network gateway, respectively. "PEP" corresponds to the performance enhancing proxy to be described shortly. From the perspective of UEs, we define *downlink* as the network path from the Internet to UEs, and *uplink* as the path in the reverse direction. Similarly, we also use the terms *downstream* and *upstream* from the perspective of the Monitor to indicate the relative locations of network elements, *e.g., downstream* refers to the path between monitor and UEs.

The Performance Enhancing Proxy (PEP). The data collection point is located within the core network of the studied LTE network. TCP traffic from or to server port 80 or 8080 traverses the PEP on the upstream side of the monitor. The PEP splits the end-to-end TCP connection into two, one between the UE and the PEP and the other between the PEP and the server. It can potentially improve the Web performance by, for example, performing compression and caching. Also the PEP makes the split transparent to UEs by spoofing its IP address to be the server's IP address. We will show later how PEP impacts our measurement results.

Data Collection. Our measurement data is a large packet header trace covering a fixed set of 22 eNBs at a large metropolitan area in the U.S. The data collection was started on October 12 2012 and lasted for 240 hours. We record IP and transport-layer headers, as well as a 64-bit timestamp for each packet. No payload data is captured except for headers of HTTP, the dominant application-layer protocol for today's smartphones [53]. No user, protocol, or flow-based sampling is performed. During the 10 days, we obtained 3.84 billion packets, corresponding to 2.9 TB of LTE traffic (324 GB of packet header data, including HTTP headers). To our knowledge, this is the first large real-world LTE packet trace studied in the research community.

Subscriber Identification. Due to concerns of user privacy, we do not collect any subscriber ID or phone numbers. We instead use private IP addresses (anonymized using a consistent hash function) as approximated subscriber IDs, since private IPs of the carrier

are very stable. They change only at the interval of several hours. In contrast, public IP addresses observed by servers may change rapidly [54]. Private IPs can also be reused. We take this into account by using a timing gap threshold of one hour in our analysis. If a private IP has not been seen for one hour, we assume its corresponding user session has terminated. This potentially overestimates the user base, but its impact on our subsequent analyses is expected to be small since changing this threshold to 30 minutes or 2 hours does not qualitatively affect the measurement results in Section 5.2, Section 5.4, and Section 5.5. In total, we observe about 379K anonymized client IPs and 719K server IPs.

Flow Extraction. From the dataset, we extract flows based on a 5-tuple of src/dst IP, src/dst port numbers, and protocol (TCP or UDP). We conservatively use a threshold of 1 hour to determine that a flow has terminated if no flow termination packets are observed. We found that similar to the idle period threshold for subscriber identification, the impact of this value on subsequent analysis results is negligible. Overall, 47.06 million flows are extracted from the trace.

We emphasize here that no customer private information is used in our analysis and all customer identities are anonymized before any analysis is conducted. Similarly, to adhere to the confidentiality under which we had access to the data, in subsequent sections, we present normalized views of our results while retaining the scientifically relevant bits.

5.1.2 Controlled Local Experiments

We also set up a measurement testbed in our lab for controlled experiments. The UE used is a fairly new smartphone model — Samsung Galaxy S III (SGH-I747) running Android 4.0.4 (IceCream Sandwich, Linux kernel version 3.0.8). It connects to a commercial LTE network in the U.S. We configure a server with 2GB memory and 2.40GHz Intel Core 2 CPU, running Ubuntu 12.04 with `3.2.0-36-generic` Linux kernel. Both the UE and the server use TCP CUBIC as their TCP implementation.

Note that the purpose of using local experiments from a potentially different LTE carrier

at locations that may not match where our studied dataset comes from is to provide a different perspective and also evaluate whether observations from analyzing the dataset can be empirically observed.

When measuring TCP throughput and RTT (Figures 5.11, 5.19, and 5.20), the UE establishes a TCP connection to the server, which then transfers randomized data without any interruption. For throughput measurement, we ignore the first 10 seconds of the TCP connection (skip the slow start phase), and calculate the throughput every 500 ms from the continuously transferred data. The RTT is measured by computing the gap between timestamps of transmitting a data packet and receiving the corresponding ACK from the sender-side trace collected by the `tcpdump` tool.

5.2 LTE Networks Characteristics

We study LTE traffic characteristics using the aforementioned 10-day packet trace collected from the large commercial LTE service provider. We also compare our results with two previous measurement studies of cellular and Wi-Fi performance on mobile devices (Section 5.2.4).

5.2.1 Flow Size, Duration, Rate, Concurrency

We begin by showing the protocol breakdown of the dataset. For the transport-layer protocol, TCP dominates the dataset (95.25% flow-wise and 97.24% byte-wise), with majority of the remaining traffic in UDP. Within TCP, as the dominant application-layer protocol, HTTP (port 80/8080) contributes 76.55% and 50.13% of all TCP bytes and TCP flows, respectively. We also notice the popularity of HTTPS (port 443), which account for 14.83% and 42.11% of TCP bytes and flows, respectively. We present a more detailed app-layer content analysis and compare the findings with those for 3G networks in §5.5.1.

Following previous measurement studies of wired and Wi-Fi networks [55, 56, 57], we are interested in three characteristics of LTE TCP flows: size, duration, and rate. Size is

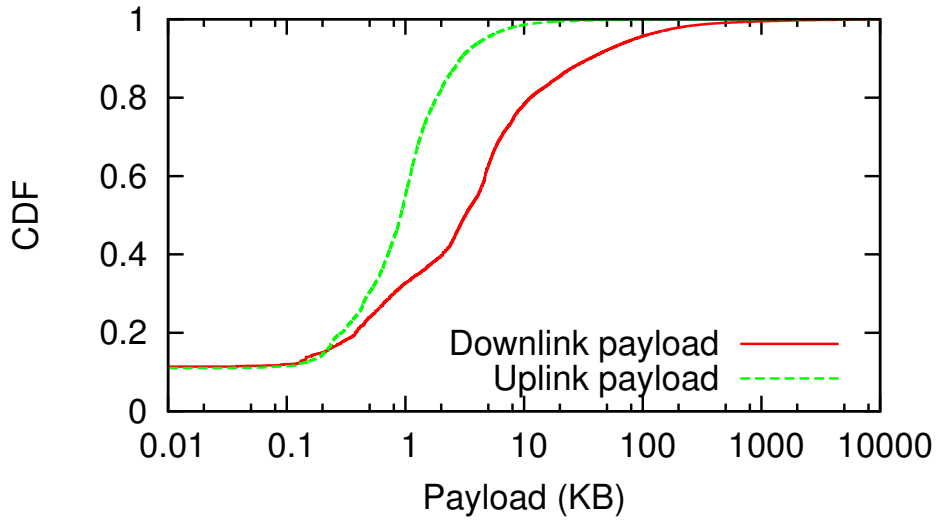


Figure 5.2: Distribution of TCP flow sizes.

the total number of payload bytes within the flow (excluding IP/transport layer headers). Duration is the time span between the first and last packet of a flow. Flow rate is calculated by dividing flow size by flow duration. Understanding these characteristics is vital to many aspects in cellular networks such as eNB scheduling, usage-based billing policy, and RAN resource balancing and optimization. Our focus is TCP since its accounts for the vast majority of the traffic (95.25% of flows and 97.24% of bytes).

TCP Flow Size. Figure 5.2 plots the CDF of uplink and downlink payload sizes, both exhibiting strong heavy-tail distributions. Most flows are small: 90% of flows have less than 2.93 KB uplink payload and 90% of flows carry no more than 35.88 KB downlink payload. In particular, 11.26% (10.86%) of flows do not have any downlink (uplink) payload as they only contain complete or incomplete TCP handshakes. On the other hand, a very small fraction of large flows, which are known as "heavy-hitter" flows [56], contribute to the majority of the traffic volume. For downlink, the top 0.57% of flows ranked by payload sizes, each with over 1 MB of downlink payload, account for 61.73% of the total downlink bytes. For uplink, the top 0.13% of flows, each with over 100 KB of uplink payload, consist of 63.86% of the overall uplink bytes. Such a distribution is as skewed as that in wired networks [56].

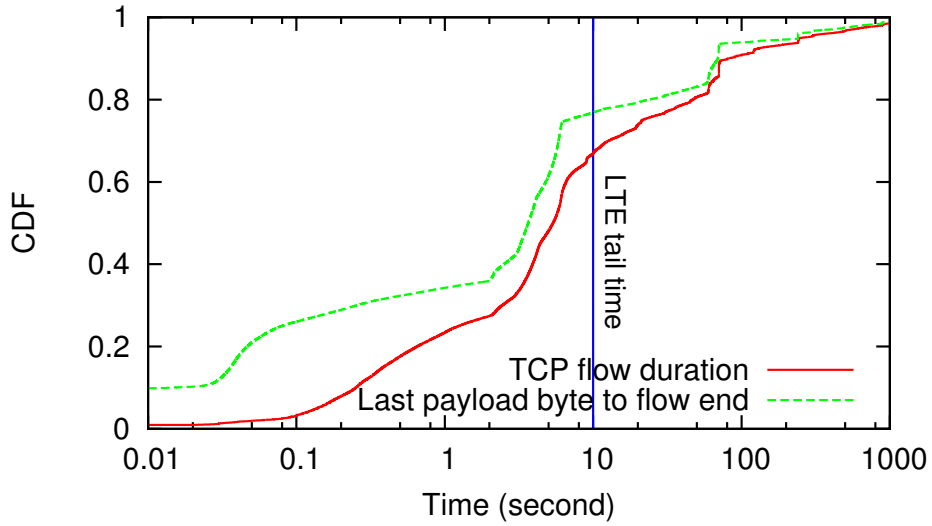


Figure 5.3: Distribution of flow duration and the duration between the last payload byte to the end of the flow.

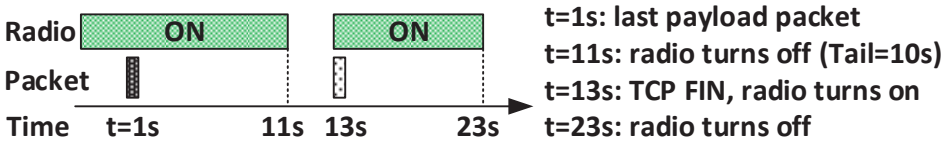


Figure 5.4: An example of delayed FIN packet and its impact on radio resource management.

We next examined the top 5% of downlink flows ranked by their downlink payload sizes. Each of them contains at least 85.9KB of downlink payload data and 80.29% of them use HTTP. By examining the HTTP headers (if exist) of the top 5% downlink flows, we found that 74.35% of their contents (in bytes) are video or audio. Regarding to the top 5% uplink flows, 73.56% of their bytes are images. Most of such traffic corresponds to users uploading photos to social networks such as Instagram.

TCP Flow Duration. Figure 5.3 shows the distribution of TCP flow duration (the solid line), defined to be the time span between the first and the last packets of a flow. Most flows are short: 48.07% are less than 5 seconds. 8.49% of the TCP flows are not even established successfully and they only consist of SYN packets. For the long-tailed part, 6.80% of the flows last at least 3 minutes and 2.77% are longer than 10 minutes.

The dotted curve in Figure 5.3 denotes the timing gap between the packet carrying

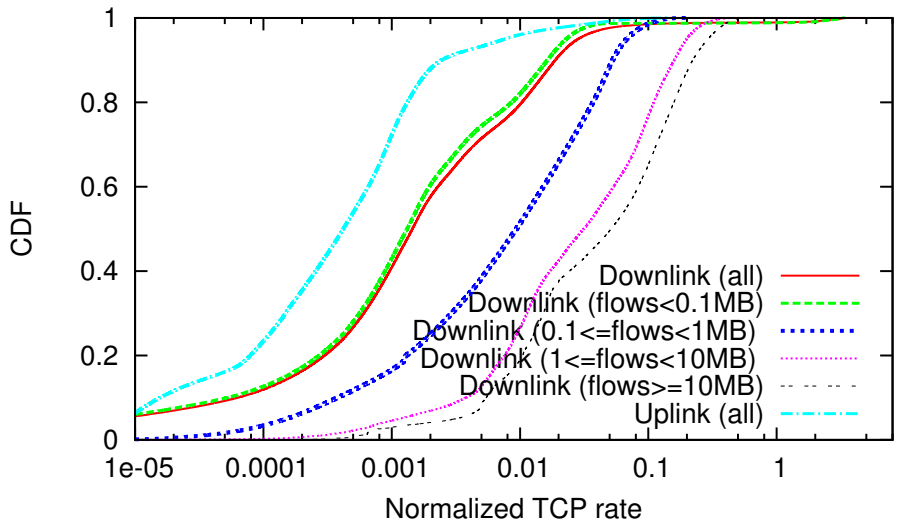


Figure 5.5: Distributions of normalized TCP flow rates.

the last payload byte and the last packet of a flow. Note that most flows in the dataset are properly terminated by either FIN (86.16% of flows) or RESET (5.35%), and for the remaining flows, they consist of only one or more SYN packets (8.49%). One example of the cause of the aforementioned timing gap is persistent HTTP that tries to reuse the same TCP connection for transferring multiple web objects so there is a timeout before the connection is closed. This does not cause any issue in wired and Wi-Fi networks. However, in LTE networks, there exists a timeout for shutting down the radio interface after a data transfer. Such a timeout, which is called *tail time*, saves energy by taking the device to the idle state once it finishes, and prevents frequent radio state switches [12]. We measured the timeout (*i.e.,* the tail time) to be 10 seconds for the studied LTE network. A delayed FIN or RESET packet will incur additional radio-on time of 10 seconds and one additional off-on switch if the delay is longer than 10 seconds, leading to waste of device energy [26]. Figure 5.4 shows one such example, which is found to be prevalent: delaying FIN or RESET for longer than 10 seconds occurs in 23.14% of the flows in our dataset as shown in Figure 5.3.

TCP Flow Rate. Figure 5.5 measures the flow rate. We observe a huge disparity between uplink and downlink rates, due to *(i)* mobile devices usually do not perform bulk

data uploading (*e.g.,* FTP and P2P upload), and *(ii)* cellular uplink channel is significantly slower than the downlink channel, even in LTE networks [33]. The four downlink throughput distributions for flows with different sizes in Figure 5.5 indicate that larger flows tend to be faster. Previous measurements for wired networks also suggest that for Internet flows, there exist correlations among their size, duration, and rate [55, 56]. We quantitatively confirm that similar behaviors also hold for LTE flows. Let S, D, and R be downlink flow size, duration, and rate, respectively, and (X, Y) be the correlation coefficient between X and Y. We calculate the values of $(logS, logD)$, $(logD, logR)$, and $(logR, logS)$ to be 0.196, -0.885, and 0.392, respectively. For uplink flows, the values of $(logS, logD)$, $(logD, logR)$, and $(logR, logS)$ are 0.030, -0.986, and 0.445, respectively. We found the flow duration and the rate are much more negatively correlated, compared with Internet flows studied in [56], whose correlation coefficients are between -0.60 and -0.69 for Internet backbone, VPN, and DSL flows. This is worth further investigation to confirm if the sessions are terminated early due to bad performance.

Concurrent TCP Flows. We explore the concurrency of TCP flows per user in the LTE data set, as shown in Figure 5.6. Specifically, we use 1 second as a threshold to determine the concurrency, *i.e.,* for the sampled time point, we count the number of TCP flows that have the downlink data transfers within the last 1 second. We observe that for 72.14% of the time, there is only one TCP flow actively downloading data, and this percentage might be even larger for smartphone users, considering that our data set also consists of a small share of users that uses LTE data cards on their laptops, which may have high TCP flow concurrency.

5.2.2 Network Latency

Figure 5.7 measures distributions of TCP handshake RTT. “C”, “M”, “P”, and “S” correspond to the client (UE), the monitor (the data collection point), the PEP, and the remote server, respectively. Since the monitor lies in the LTE core network, we can break down

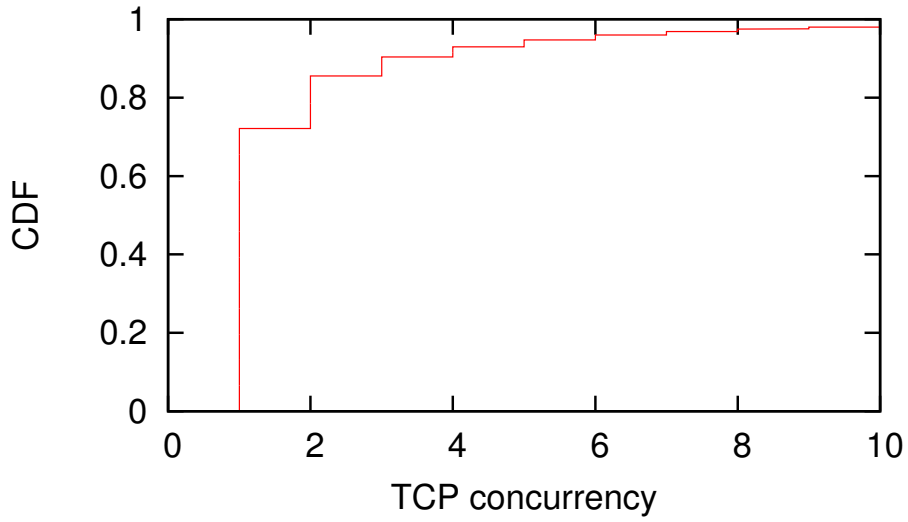


Figure 5.6: Concurrency for TCP flows per user uniformly sampled by time.

the overall RTT into two components: the *downstream* RTT between a client and the monitor ("C-M", for all traffic), and the *upstream* RTT between either the monitor and the PEP ("M-P", for TCP port 80/8080 traffic) or server ("M-S", for other traffic). The downstream RTT is an estimation of the latency in the RAN (Figure 5.1). In a TCP three-way handshake, let the monitor's reception time of SYN (uplink), SYNACK (downlink), and ACK (uplink) be t_1, t_2, and t_3, respectively. Then the upstream RTT is computed as $t_2 - t_1$, and the downstream RTT is $t_3 - t_2$. The "C-S" curve combines both the "C-M" and the "M-S" components (for non-PEP traffic only).

It is well known that in 2G/3G data networks, usually the RAN latency dominates the overall end-to-end delay [43]. This is no longer the case in LTE networks. Figure 5.7 shows that the upstream RTT to a remote server ("M-S") has a higher variance, and is usually larger than the downstream RTT ("C-M"). This is further confirmed by Figure 5.8, which plots the distribution of ratios between the upstream RTT and the downstream RTT for non-PEP ("C-S") flows. For 55% of the non-PEP flows, their upstream RTTs are larger than the corresponding downstream RTT, whose reduction (*i.e.,* the reduction of the RAN latency) is mostly attributed to the flattened network topology in the LTE RAN. For example, the two-layered RAN architecture (NodeB and the Radio Network Controller, RNC)

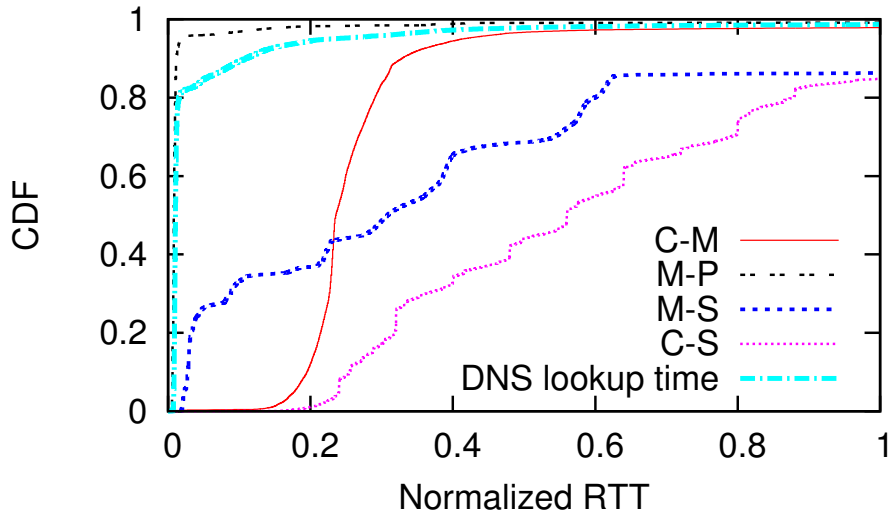


Figure 5.7: Distributions of normalized handshake RTT and DNS lookup time.

in 3G UMTS/HSPA networks is reduced into the single-layered eNB architecture in LTE, helping significantly reducing the RAN latency [33] (See Section 5.2.4 for quantitative comparisons). Further, the "M-P" curve in Figure 5.7 indicates the latency between the monitor and the PEP is very small.

LTE Promotion Delay. In cellular networks, the end-to-end latency of a packet that triggers a UE's radio interface to turn on is significantly long. Such a packet incurs a radio resource control (RRC) promotion delay during which multiple control messages are exchanged between a UE and the RAN for resource allocation. The promotion delay can be as long as 2 seconds in 3G networks [20], and it also exists in LTE networks [12]. The promotion delay is not included in either the upstream RTT or the downstream RTT in Figure 5.7, since the promotion (if any) has already finished when the monitor observes a SYN packet, as illustrated in Figure 5.9. However, we are able to infer the promotion delay using the TCP timestamp embedded into a TCP packet when the packet is about to leave the UE. In a three-way handshake, let the TCP timestamp of the SYN and the ACK packet be TS_b and TS_a, respectively. Then the round-trip time (including the promotion delay) experienced by the UE is $G(TS_b - TS_a)$ where G is the inverse of the ticking frequency of UE's clock generating the TCP timestamp. Note that the TCP timestamps are not wall-

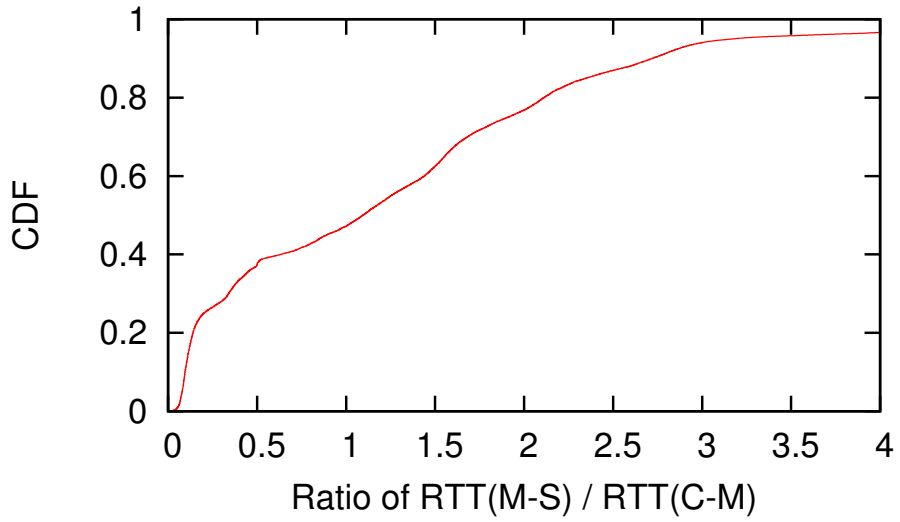


Figure 5.8: Distribution of the radio between uplink and downlink RTT (for non-PEP traffic).

clock times. Their units depend on the ticking frequency of the UE. We detail how to compute G in Section 5.4.1. Finally the promotion delay (if exists) could be derived by subtracting the RTT between the UE and the server/PEP (estimated in Figure 5.7) from $G(TS_b - TS_a)$, as shown in Figure 5.9.

We calculated promotion delays using the aforementioned method, by examining TCP handshakes with the following property: the user does not send or receive a packet within the time window $(t - T, t)$ where t is the reception time of SYN and T is the window size. We conservatively choose $T = 13$ seconds which is larger than the 10-second timeout of the studied LTE network. This restriction ensures the UE is in the idle state when the handshake is initiated. Therefore, the SYN packet must trigger a state promotion. The 25%, 50%, and 75% percentiles of the promotion delay are 319 ms, 435 ms, and 558 ms, respectively. We found they are significantly shorter than the 3G promotion delays (around 2 seconds from idle to high-power state, and around 1.5 seconds from low-power to high-power state [20]), possibly due to the simplified signaling protocol in LTE networks [33].

DNS Lookup. The "DNS" curve in Figure 5.7 measures the DNS lookup delay, computed as the delta between the reception time of a DNS request and its response at the

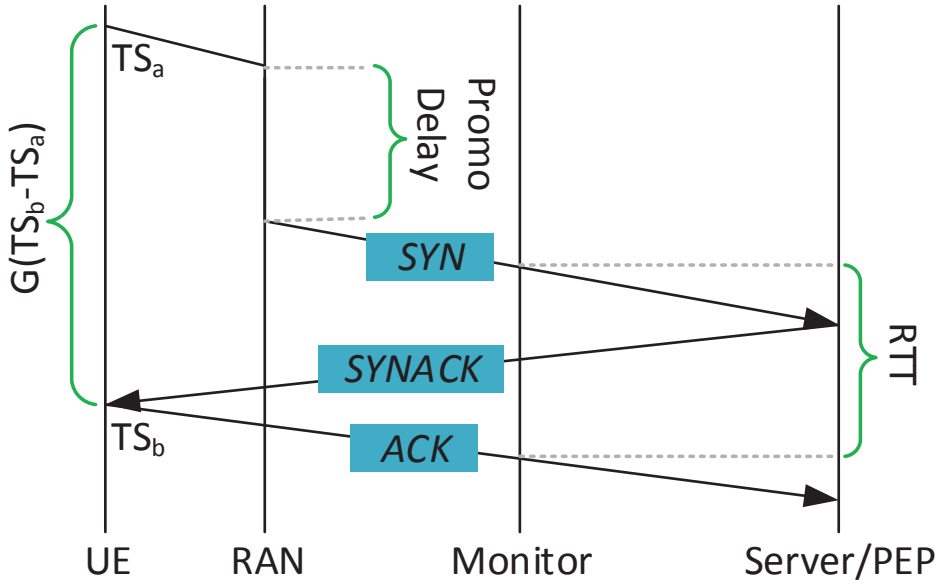


Figure 5.9: Estimating the promotion delay.

monitor. Note this is the latency between monitor and the DNS server, and we are not able to measure the downstream latency since DNS messages are transferred over UDP. We found that the upstream latency is usually very short *i.e.,* less than 10 ms for 87.28% of request-response pairs. Since the studied LTE network (Figure 5.1) has its own DNS server, the short lookup delay indicates the desired effectiveness of the DNS server, which caches most DNS responses so their domain names are effectively resolved locally within the LTE core network.

5.2.3 Queuing Delay and Retransmission Rate

Section 5.2.2 focuses on the RTT of TCP connection establishment during which the small TCP handshake packets are usually unlikely to be buffered by the network. During the data transfer phase, a TCP sender will increase its congestion window, allowing the number of unacknowledged packets to grow. Such "in-flight" packets can potentially be buffered by routers and middleboxes on their network paths, incurring queueing delays. In LTE networks, buffers are extensively used to accommodate the varying cellular network conditions and to conceal packet losses [33].

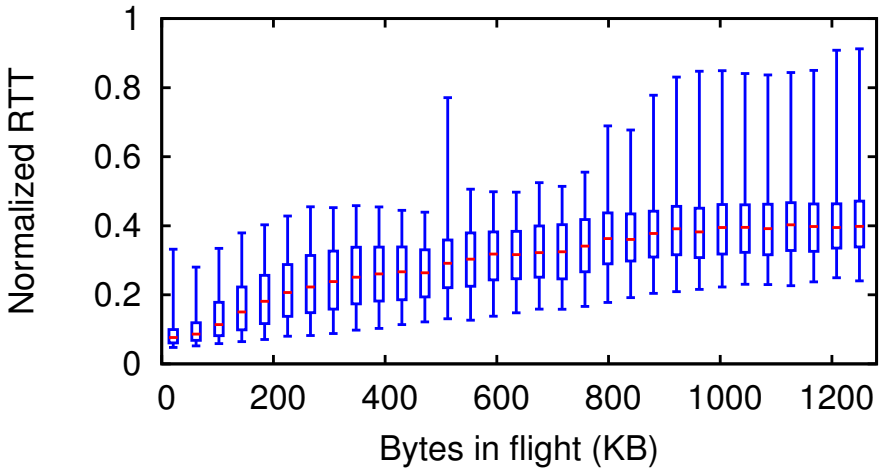


Figure 5.10: Downlink bytes in flight vs. downstream RTT.

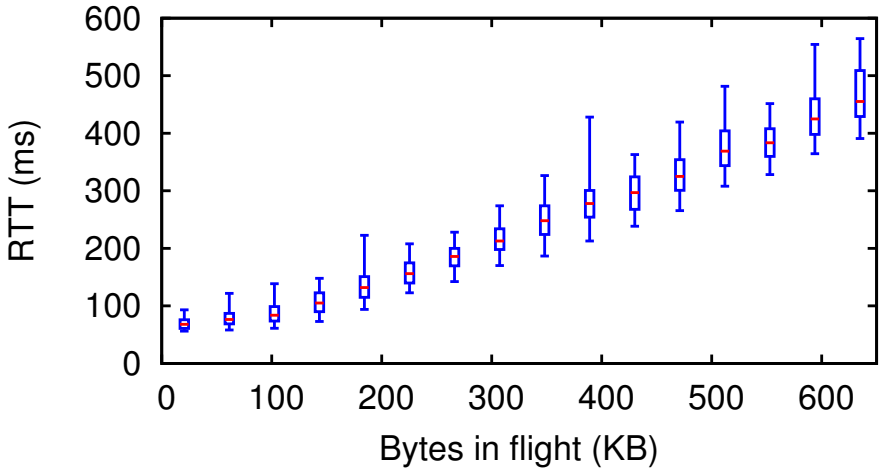


Figure 5.11: Downlink bytes in flight vs. downstream RTT (controlled lab experiments with LTE Carrier A).

Figure 5.10 shows the relationship between the downstream RTT and the number of downlink in-flight bytes, which is computed by counting the unacknowledged bytes. As shown in Figure 5.10, the downstream RTT tends to inflate as the number of in-flight bytes increases. The in-flight bytes in our studied LTE network can be larger than 1200 KB, causing high latency due to the queuing delay. We very this in local experiments (Section 5.1.2) where we measure both the RTT and the bytes in flight at UE for two large commercial LTE networks. As shown in Figure 5.11 and 5.12, the trend that RTT grows with the number of in-flight packets is obvious. Our observation is also consistent with a recent study [58]

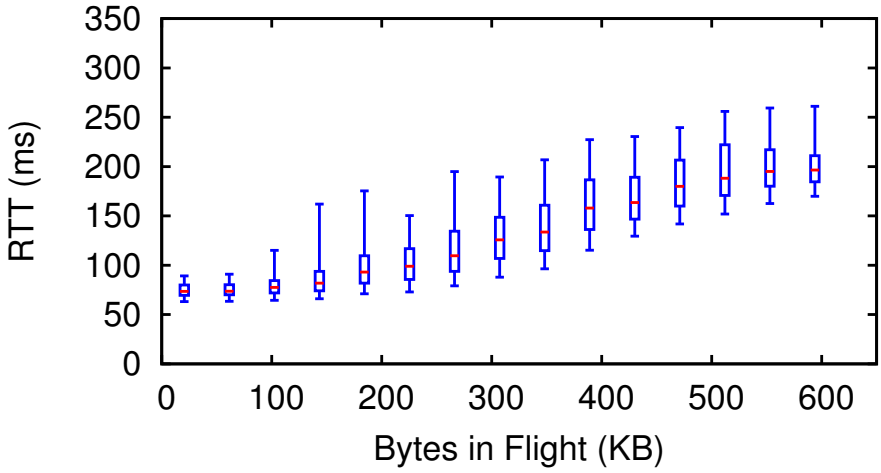


Figure 5.12: Downlink bytes in flight vs. downstream RTT (controlled lab experiments with LTE Carrier B).

that shows the usage of large buffers in today's cellular networks may cause high queuing delays. In addition to that, we further demonstrate its prevalence in today's LTE networks: as shown in Figure 5.13, which plots the distribution of downlink in-flight bytes for large flows ($>$ 1MB), about 10% of measured instances have in-flight bytes greater than 200 KB, potentially leading to long queuing delays.

Clearly, for short flows or traffic triggered by user interactions (*e.g.,* web browsing), queues are not likely to build up. For long-lived flows, usually it is the throughput instead of latency that matters. However, when short-lived and long-lived flows coexist (*e.g.,* performing browsing while streaming in the background), queuing delay may severely deteriorate user experience by introducing unacceptable delays for short flows. Moreover, as a new observation, we found that a high downstream queuing delay may often cause TCP's congestion window to collapse upon a single packet loss. We discuss this newly identified and rather severe issue in Section 5.3.

Retransmission Rate. We study TCP downlink retransmission rate, defined as the number of retransmitted packets divided by all packets, across all downlink flows in our data set. 38.1% of the flows have retransmission rates of zero, and the median value is only 0.06%. Such low retransmission rates are comparable to those in wired networks [56].

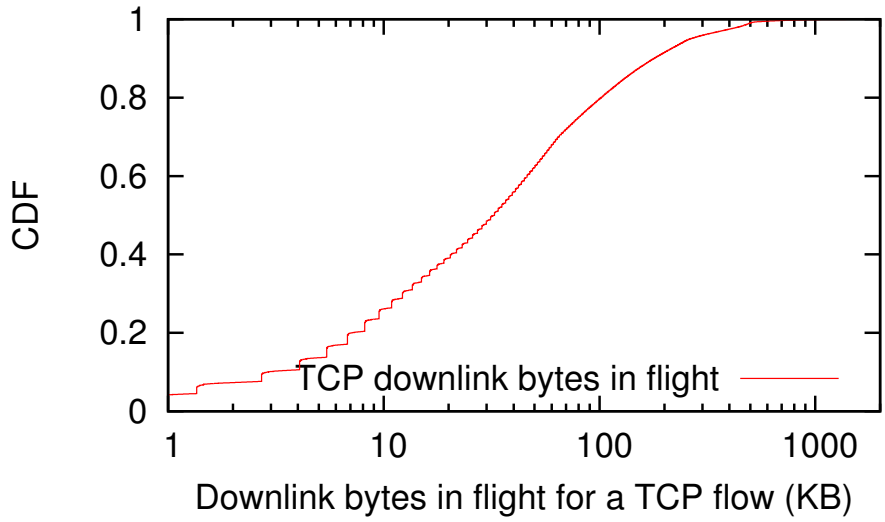


Figure 5.13: Distribution of downlink bytes in flight for large flows ($>$ 1 MB).

Study	Our Results	3GTest [1]	4GTest [12]		SpeedTest [59]					
Time	Oct 2012	Aug-Dec 2009	Oct-Dec 2011		Feb 21 - Jun 5 2011 (15 weeks)					
Location	1 US City	Across U.S.	Across U.S.		New York City		Madison WI		Manchester UK	
Type	LTE Only	Four 3G ISPs	LTE	WiMAX	Cellular	WiFi	Cell'	WiFi	Cell'	WiFi
5% TCP DL*	569	74 - 222†	2112	431	108	404	99	347	28	267
50% TCP DL	9185	556 - 970	12740	4670	1678	7040	895	5742	1077	4717
95% TCP DL	24229	1921 - 2943	30812	10344	12922	17617	3485	14173	3842	15635
5% TCP UL	38	24 - 52	387	172	52	177	55	168	25	180
50% TCP UL	2286	207 - 331	5640	1160	772	2020	478	1064	396	745
95% TCP UL	8361	434 - 664	19358	1595	5428	10094	1389	5251	1659	5589
5% HS RTT	30	125 - 182	37	89	68	21	99	24	98	34
50% HS RTT	70	160 - 200	70	125	159	54	184	69	221	92
95% HS RTT	467	645 - 809	127	213	786	336	773	343	912	313

* TCP DL: downlink throughput (kbps). TCP UL: uplink throughput (kbps). HS RTT: TCP handshake RTT (ms). 5%, 50%, 95% are percentiles.

† For a range $x - y$, x and y are the result of the worst and the best carriers, respectively, for that particular test.

Table 5.1: Comparing with previous measurement studies.

There are even fewer packet losses since the retransmission rate is an upper bound of the packet loss rate in the downstream (*i.e.,* between UE and the monitor, note we are not able to capture losses occurring on the upstream side of the monitor). In fact, in cellular networks, most transport-layer losses are concealed by the physical/MAC-layer retransmission and reduced by buffering. In particular, buffers in LTE networks upstream from the airmile can play an important in absorbing the burstiness of the traffic transmitted over the lossy wireless link, helping achieve a low loss rate.

5.2.4 Comparison to Previous Studies

We compare our results with three previous measurement studies, focusing on three important metrics: TCP downlink throughput, TCP uplink throughput, and TCP handshake RTT. The 3GTest study [1] deployed an app that measures network performance metrics on users' handsets. Their data consisted of 35K cellular (3G only) tests from customers of four large U.S. cellular carriers in late 2009. The 4GTest study [12] adopts a similar approach while focusing on LTE users. Its data comprises of about 1K LTE tests and a few WiMAX tests across the U.S. in late 2011. A recent study [59] examined a 15-week dataset from `speedtest.net` in 2011. Table 5.1 shows their reported performance metrics for handheld device users from three locations: New York City (246K Wi-Fi tests / 79K cellular tests), Madison Wisconsin U.S. (24K Wi-Fi / 4K cellular), and Manchester U.K. (291K / 31K). The cellular technology ranges from 2G EDGE to 4G LTE, but is dominated by 3G (UMTS/EvDO/HSPA).

We discuss three major issues that may affect the comparison. First, all three previous studies perform throughput measurement using bulk data transfer of a large file without any pause while our flows may consist of idle time periods (*e.g.,* due to user think time), leading to a lower throughput. To obtain more fair comparison, here we only consider large non-PEP flows in our dataset (with at least 200 KB for uplink and 1 MB for downlink) with no visible idle time period (with maximum inter-packet time of less than 1 second, which is larger than 99.9^{th} percentile of RTT). Second, in our case, remote servers may impose rate limit [60] while all previous studies perform active probing using dedicated test servers without any limitation on throughput. Third, we infer performance metrics from traces of real Internet servers, while 3GTest, 4GTest, and SpeedTest employ different server selection policies: 3GTest uses a single server located in U.S. while SpeedTest picks a server geographically close to the UE. This in particular affects the latency estimation.

The comparison results are shown in Table 5.1. Despite aforementioned differences among diverse measurement approaches, we believe the comparison can still demonstrate

the advantage of LTE over other types of cellular access technology, since their performance difference is quite significant: the median downlink throughput, uplink throughput, and handshake RTT are 9.5x, 6.9x, and 0.43x compared with the median values of the best U.S. 3G carrier in 2009, respectively. Compared with the 2011 New York City cellular results, the ratios are 5.5x, 3.0x, and 0.44x for DL throughput, UL throughput, and RTT, respectively. Moreover, on mobile devices, LTE also outperforms Wi-Fi in many cases. Specifically, for the 5^{th}/50^{th}/95^{th} percentiles of downlink throughput and the median uplink throughput shown in Table 5.1, LTE performs better handheld Wi-Fi. Based on Table 5.1, LTE's latency appears higher than that of Wi-Fi. However, recall that Speedtest always picks a nearby test server while we are measuring the RTT between UE and real servers that may be far away. This may lead to an unfair RTT comparison. Furthermore, our performance values are consistently lower than those reported by LTE tests of 4GTest, very likely due to the rate limiting imposed by remote servers as mentioned before. A recent study [60] indicates such rate limiting is prevalent across today's Internet servers. We also observe that LTE significantly outperforms WiMAX in all three metrics.

5.3 Abnormal TCP behavior

Due to their resource usage, we focus on *large flows* defined to be relatively long flows, with more than 5 seconds data transfer time, and total downlink payload exceeding 1MB. These large flows account for only 0.3% of all TCP flows in our data set, but their total downlink payload contributes to 47.7% of all downlink payload.

As a background, upon receiving an out-of-order unacknowledged segment, a TCP receiver sends an immediate duplicate ACK [61]. From the sender's perspective, duplicate ACKs can be caused by reordering or loss. Therefore, when there is a large amount of bytes in flight and one data segment S is lost, each data segment with sequence number higher than that of S triggers a duplicate ACK, before a retransmission of S is successfully received. So a long sequence of duplicate ACKs strongly suggests a packet loss. When

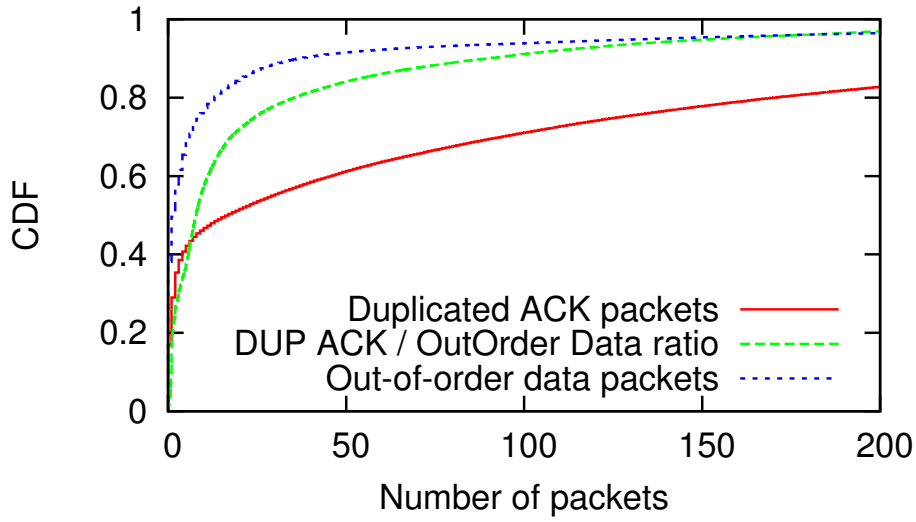


Figure 5.14: Observed duplicate ACKs and packet reordering in large TCP flows.

TCP detects 3 duplicate ACKs, it infers a data packet loss and retransmits it according to the fast retransmit [61]. In the monitor traces, we detect this behavior as the data packet sent by fast retransmit is out-of-order relative to other packets.

Figure 5.14 summarizes the observation of duplicate ACKs and packet reordering in the large TCP flows. Although the median of duplicate ACKs in large flows is 17, for over 29.0% of large flows, there are over 100 duplicate ACKs. We observe that the number of out-of-order data packets in large flows is substantially smaller than that of duplicate ACKs, with a median value of only 2. By studying the ratio between duplicate ACKs and out-of-order data packets, 24.7% of flows have a ratio of over 25, and for some flows, this ratio can reach up to 5,000. This indicates that even a single out-of-order data packet can trigger a large number of duplicate ACKs when the bytes-in-flight are large, using up more uplink bandwidth.

Fast retransmission allows TCP to directly send the lost segment to the receiver possibly preventing retransmission timeout (RTO). If so, TCP would resume data transfer with the congestion window size reduced by half. However, as shown earlier, we identified significant queuing build up between UE and monitor. Such large in-network queues capable of holding up to a few megabytes data could delay the receipt of the retransmitted data packet.

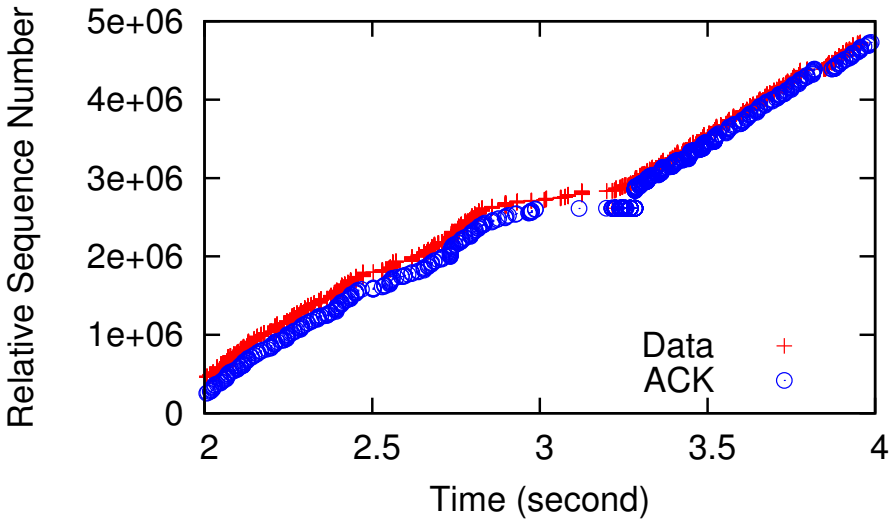


Figure 5.15: Duplicate ACKs not triggering a slow start.

In the meanwhile, if TCP does not use duplicate ACKs to update RTO, a timeout is likely to happen. If the corresponding ACK does not arrive at the server within the RTO (timeout), the congestion window would drop to 1 segment, triggering slow start, significantly hurting TCP performance. We refer to this as the *undesired slow start* problem.

Figures 5.15 and 5.16 demonstrate two sample cases in the data set, where Figure 5.15 shows that the train of duplicate ACKs does not trigger slow start and Figure 5.16 includes a case that slow start is triggered. One key difference is that Figure 5.16 has about 500KB bytes in flight before the first duplicate ACK, while Figure 5.15 has much fewer bytes in flight.

In TCP, RTO is computed by the sender using smoothed round-trip time and round-trip time variation [62]. However, using duplicate ACKs to update RTO, which may be beneficial by allowing more accurate RTT estimation, is not standardized. In Figure 5.16, between 1.1s and 1.5s, the sender receives many duplicate ACKs. Due to the growing queueing size, RTT grows from 262ms (the last RTT sample before the first duplicate ACK) to 356ms, the RTT for the retransmitted packet. The sender's TCP implementation apparently ignores these duplicate ACKs for updating RTO and RTO remains the same without considering the duplicate ACKs. Following the method for calculating RTO [62],

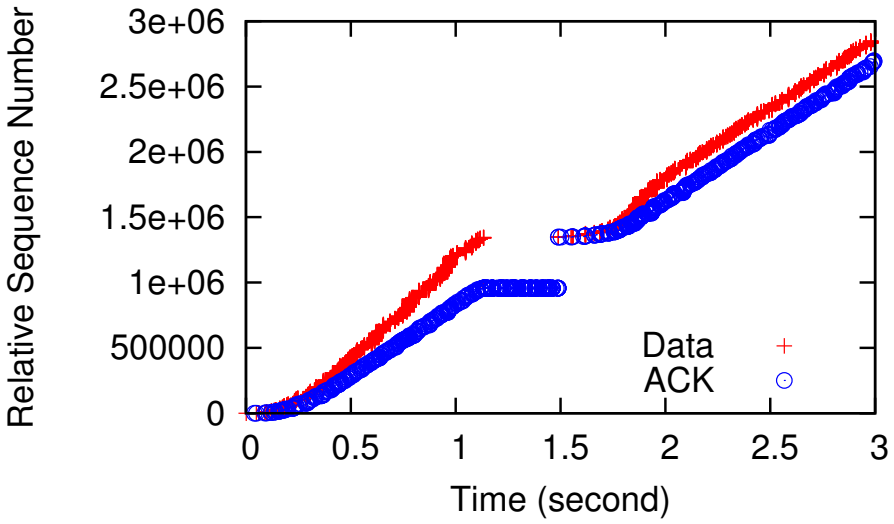


Figure 5.16: Duplicate ACKs triggering a slow start.

we observe that RTO is around 290ms before the first duplicate ACK, which is smaller than the RTT of the retransmitted packet (356ms). This problem does not happen in Figure 5.15, because the RTT before the first duplicate ACK is close to that after the last duplicate ACK, due to the small number of bytes in flight. Although it is recommended that the RTO should be at least 1 second [62], depending on the operating systems, different minimum values are used, *e.g.,* Linux's minimum RTO is 200ms [63]. Such small values of RTO can exacerbate the undesired slow start problem demonstrated in Figure 5.16.

We also study the prevalence of the congestion window collapse problem. To tell whether there is a slow start following a long list of duplicate ACKs, we use a heuristic metric R_{ss}, ratio of slow start: $R_{ss} = \frac{\theta_{[100,200]}}{\theta_{[0,100]}}$, where $\theta_{[t_1,t_2]}$ is the average downlink throughput from t_1 ms to t_2 ms after the last duplicate ACK. We choose 200 ms empirically as it is observed to be shorter than a typical slow start in the LTE networks. During slow start, R_{ss} is expected to be larger than that when there is no slow start. For example, the R_{ss} is 1.0 for Figure 5.15 and R_{ss} is 3.7 for Figure 5.16. In practice, we observe that 1.5 is a good threshold for R_{ss} in determining slow start. Using this threshold, we have determined that for all the large TCP flows with at least one lost data packet, 20.1% of them suffer from the slow start problem, which consists of 12.3% of all large TCP flows. In one

case, a 153-second flow even experience 50 slow starts, resulting in an average throughput of only 2.8Mbps, while the estimated bandwidth actually larger than 10Mbps.

There are different ways to mitigate this problem. One approach is to update RTO with the help of duplicate ACKs with TCP Selective Acknowledgment options (SACK) [64]. Assuming there is no packet reordering, by taking the difference between the SACK window of two consecutive duplicate ACKs, we can identify the exact data packets corresponding to these ACKs. If there is ambiguity, either due to ACK reordering or additional packet loss, we simply ignore this sample. In our data sets, packet reordering rate is less than 1% and SACK is enabled in 82.3% of all duplicate ACKs, making this approach promising.

If SACK is disabled, we can use a fall-back approach to estimate RTT based on duplicate ACKs by assuming that they are in response to the data packets sent out in order. This assumption holds in most cases as the packet reordering rate is low. Using these approaches, we can obtain RTT estimations for duplicate ACKs and update RTO accordingly, which effectively prevents the timeout of retransmitted packet due to queueing delay. Note that the RTT estimation method used in TCP Vegas [65] with help of the TCP Timestamps option is not applicable to duplicate ACKs, since the echo timestamps of all duplicate ACKs are all the same, which is the timestamp of the segment before the lost segment, rather than the timestamp triggering the duplicate ACK. Our initial analysis shows that these two approaches are able to prevent more than 95% of the slow starts. From the mobile network operators' perspective, one simple solution for this problem could be prioritizing the retransmitted packet. The retransmitted packet could be inferred by tracking the TCP flow status. However, the security and performance implications of this approach are yet to be studied.

5.4 Bandwidth Estimation

In order to understand the network utilization efficiency of existing applications in the LTE networks, we first need to know the available bandwidth for each user. Previous

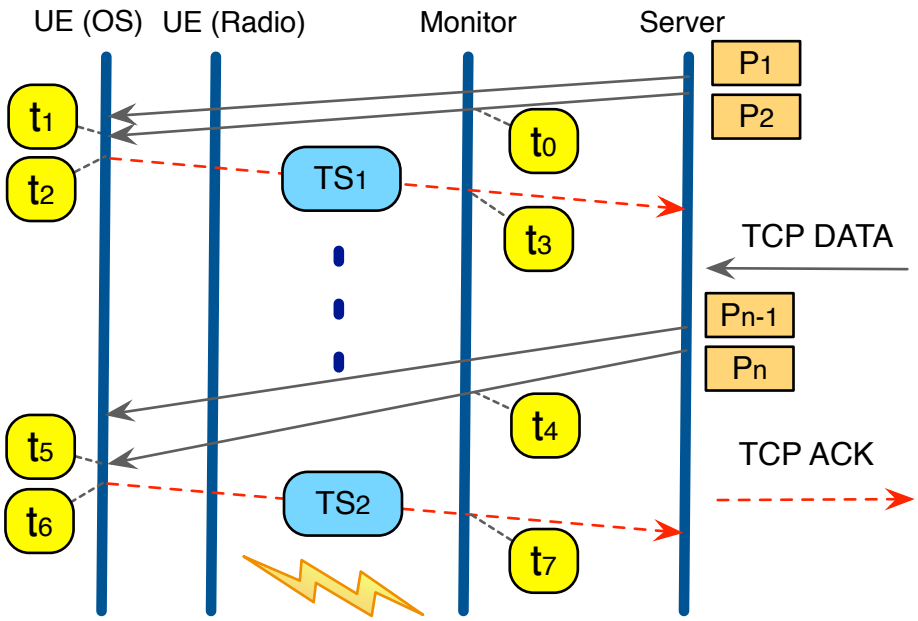


Figure 5.17: Typical TCP data transfer.

work on active bandwidth measurement methodology to estimate available bandwidth, *e.g.,* using packet pairs, packet trains, and parallel TCP connections [66, 67, 68], do not apply here. As existing studies have shown that network condition is highly variable in cellular networks [1], active probing would require us launch measurements for each user in our data set at the time of trace collection. Using packet traces collected at the monitor, we instead devise a *passive* bandwidth estimation algorithm to capture the available bandwidth for each user using TCP flows that may *not* fully utilize the bandwidth.

5.4.1 Bandwidth Estimation Algorithm

Figure 5.17 illustrates a typical TCP data transfer. Our monitor lies between the server and the UE, and we only use packet traces collected at the monitor for the analysis. The high-level idea for our bandwidth estimation algorithm is to select a time window during which the sending rate is fast enough (to be discussed later) and calculate the UE receiving rate. Note that the UE receiving rate is not always equal to the server sending rate, *e.g.,* when the in-network buffers fill up, the sending rate might exceed the receiving rate. The

idea behind our bandwidth estimation algorithm is to use the TCP Timestamps option to calculate receiving rate.

We use Figure 5.17 to illustrate our bandwidth estimation algorithm. At t_2, UE sends an ACK in response to the two data packets P_1 and P_2. And similarly, at t_6, the ACK for P_{n-1} and P_n is sent. From the monitor's traces, we observe that $n-2$ data packets ($P_3 \cdots P_n$) sent to the UE in a time window between t_0 and t_4. Assuming the average payload size of these $n-2$ packets is S, the sending rate between t_0 and t_4 is

$$R_{snd} = \frac{S(n-2)}{t_4 - t_0} \tag{5.1}$$

And from the UE's perspective, the receiving rate for these $n-2$ packets is

$$R_{rcv} = \frac{S(n-2)}{t_5 - t_1}$$

Typically, t_2 is very close to t_1 and similarly $t_5 \approx t_6$. In our controlled lab experiments, for a 30-minute continuous trace, the median value of the delay between a data packet and the corresponding ACK is negligible: 0.3ms. However, such delay, *e.g.,* $t_2 - t_1$, could be large in some rare cases. Typically, one ACK in TCP is for two data packets and when there is only one data packet pending acknowledgement, the receiver may delay sending the ACK by up to 500 ms, which is known as the delayed acknowledgement mechanism [69]. In our example, if P_{n-1} has already been acknowledged by another ACK and after t_5 there are no more data packets arriving at the UE side, the ACK for P_n could be delayed. For simplicity, we ignore the cases when the last ACK is acknowledging only one data packet, indicating it might be a delayed ACK. We also do not consider cases with out-of-order data packets or duplicate ACKs in the time window for bandwidth estimation, as there may be ambiguity in packet timing. Then we know

$$R_{rcv} \approx \frac{S(n-2)}{t_6 - t_2}$$

If the uplink delay from the UE to monitor is stable, we can assume $t_6 - t_2 = t_7 - t_3$. However, as shown previously that RTT could be significantly affected by the bytes in flight and thus the assumption of $t_6 - t_2 = t_7 - t_3$ may not hold. Instead, we use TCP Timestamps option [52] to calculate $t_6 - t_2$. Specifically, the ACKs from the UE to the server may contain the Timestamp Value field (*TSval*), which contains the current value of the UE's timestamp clock. The unit for *TSval* is clock tick and for different devices, the actual time per clock tick could be different. Assume for each clock tick, the corresponding time is G milliseconds and G can be treated as a constant for the same device. Assuming G is known, we can estimate R_{rcv} as

$$R_{rcv} \approx \frac{S(n-2)}{G(TS_2 - TS_1)} \tag{5.2}$$

where TS_1, TS_2 are the *TSval* in the two corresponding ACKs.

Our bandwidth estimation algorithm only requires ACKs from the UE having TCP Timestamps option enabled, with no requirement on servers and in our data set, for 92.61% of the TCP flows, this requirement is satisfied.

We infer the G value using a similar method from previous work [70]. Using the example in Figure 5.17, we have

$$G \approx \frac{TS_2 - TS_1}{t_7 - t_3} \tag{5.3}$$

We require $t_7 - t_3$ to be large enough, because clock tick is discrete, not continuous, and if it is too small, *e.g.,* when $t_7 - t_3 = 0.6G$ and $TS_2 - TS_1 = 1$, the calculated value is actually $0.6G$. Also, there might be variations in uplink one-way delay from UE to monitor, and such variation is only negligible in calculating G when $t_7 - t_3$ is large enough. Assuming the threshold for $t_7 - t_3$ to calculate G with small error rate is δ_G, we present the error curve for calculating G value of two UEs in Figure 5.18 based on controlled experiments. The actual G values for device 1 and 2 are measured at the UE side using 30-minute long traces. We observe that the error for G inference drops drastically as δ_G

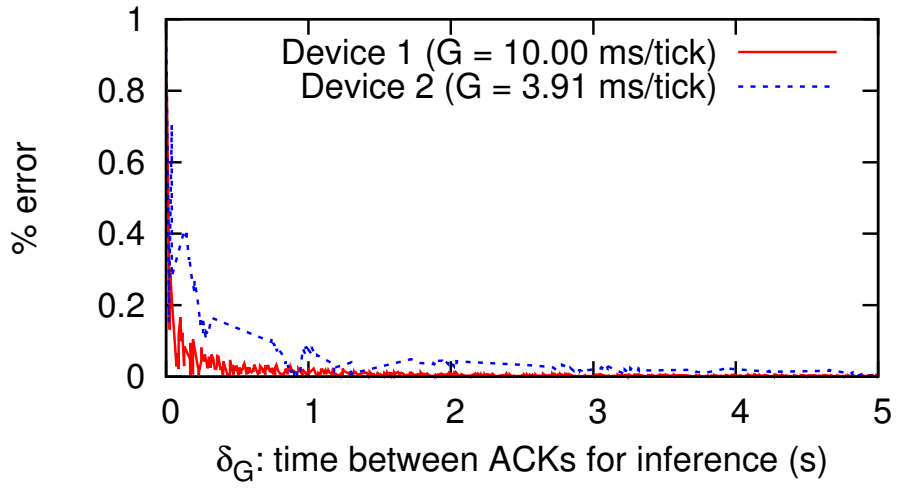


Figure 5.18: G inference and the selection of δ_G.

increases and we conservatively select $\delta_G = 3$ seconds, incurring less than 0.1% error. Note that the error curve may be different for different G values. In our data set, among all large flows, except for 5.93% of them without UE Timestamps option enabled, 57.33% have $G \approx 1$ms/tick, 36.37% have $G \approx 10$ms/tick, and the rest 0.37% have other values, *e.g.,* $G \approx 100$ms/tick. With $\delta_G = 3$ seconds, we estimate the inferred G has less than 0.1% error for majority of the flows.

Summary: for a target TCP flow, if G value is not known, the initial $\delta_G = 3$ seconds is used to infer G using Formula 5.3, with two sample packets more than δ_G apart selected. Flows without UE TCP Timestamps option support are ignored. Then the algorithm scans for a window with high sending rate R_{snd} calculated by Formula 5.1. If $R_{snd} \geq C$, a pre-known constant of the maximum possible available bandwidth in the studied network, and there is no out-of-order data packets or duplicate ACKs within the time window for estimation, and the last ACK of this window is not a delayed ACK, one bandwidth estimation sample is obtained using Formula 5.2. The selection of C is important, *i.e.,* if C is too small, the bandwidth may be underestimated, as small R_{snd} may be chosen; if C is too large, we may not be able obtain many estimation samples. We conservatively choose

C = 30Mbps, which is verified to be higher than the rate of most flows, and in the meanwhile allows us to predict bandwidth for over 90% of the large downlink flows during our analysis.

In addition to downlink bandwidth, our algorithm is also applicable to uplink bandwidth estimation, by interchanging the UE and the server in Figure 5.17. However, some tuning is required, *e.g.,* the choice of C and Δ_G, *etc.* Similarly, our bandwidth estimation algorithm also works in other network types, such as 3G, WiFi and even wired networks, with proper parameter settings.

Although the described algorithm is based on one single TCP flow per user, a similar idea can be applied to multiple concurrent flows per user by summing up the predicted bandwidth for different flows. As long as we ensure that the total sending rate for all concurrent flows are larger than C, the total receiving rate would be the accurate estimation of the available bandwidth. In this work, we focus on the application of our bandwidth algorithm for the downlink traffic (UEs downloading contents from servers) in the LTE networks for single TCP flows, *i.e.,* with no competing downlink flows for the same user.

5.4.2 Validation with Local Experiments

To validate the bandwidth estimation algorithm, we use controlled experiments with the setup described in Section 5.1.2.

Figure 5.19 shows the UE-perceived throughput over 30-minute duration, as well as the absolute error for the estimated bandwidth. The actual throughput fluctuates frequently around 10Mbps and error fluctuates within $\pm$2Mbps in most cases. In Figure 5.20, we compare the distribution of estimated bandwidth calculated from the server-side packet traces with actual used bandwidth calculated from UE-side packet traces. We can see that the estimated bandwidth curve is very close to the actual throughput curve. Based on UE traces, we can select different window size to calculate actual throughput. For each window, we also get one bandwidth estimation sample whose timestamp is closest to the center of

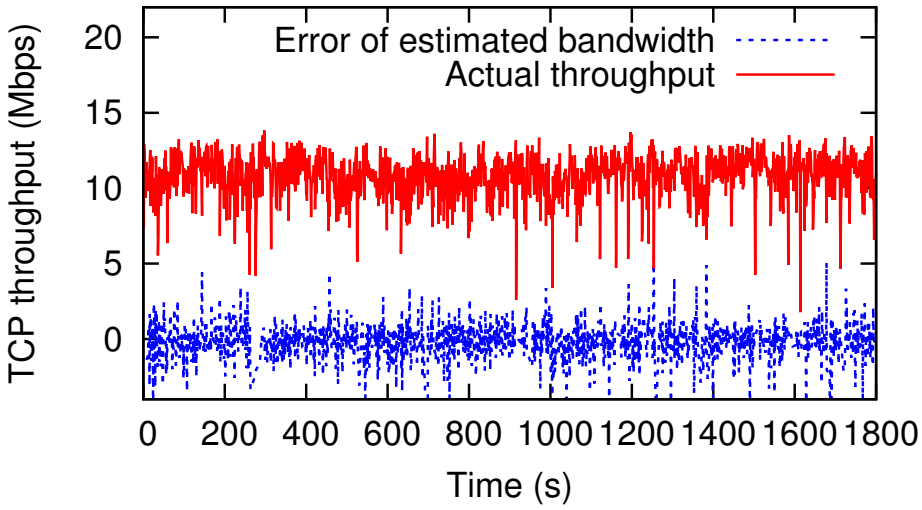


Figure 5.19: Time series of bandwidth estimation for LTE network (controlled lab experiments).

that window, and we compare this sample with the average actual throughput to obtain an error sample. The error distribution for two window sizes, *i.e.,* 1.0s and 0.1s, are shown in Figure 5.20. For 1.0-second window, the average error is 7.92% and for 0.1s window, the UE throughput has higher variation and the average error is slightly higher. Note that the term "error" here is relative to the actual throughput observed from UE-side traces, which itself might not be the actual available bandwidth, and the true error rate for our estimation algorithm could be even smaller.

5.4.3 Bandwidth Utilization by TCP Flows

In this section, we analyze the LTE traffic data set to understand network utilization efficiency of TCP flows. As shown in Figure 5.6 (§5.2.1), most users have only one TCP flow actively downloading data. This allows us to focus on applying our bandwidth estimation algorithm to single TCP flows with no competing downlink flows from the same user.

We then apply the bandwidth estimation algorithm to the large TCP downlink flows that are not concurrent with other large flows and summarize the results in Figure 5.21. We use a time window size of 250ms and for each window, we take one bandwidth estimation

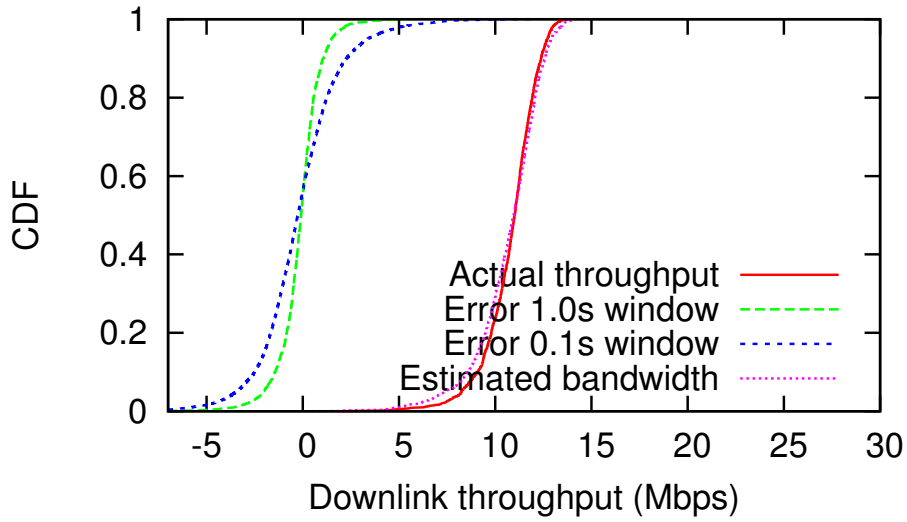


Figure 5.20: CDF of bandwidth estimation results for LTE network (controlled lab experiments).

sample that is closest to the center of the window. For some flows, there exist windows that do not contain any valid bandwidth estimation samples and we simply assume the bandwidth distribution for these unknown windows is the same for the known windows. The possible bias will not likely significantly affect our analysis as such unknown duration accounts for less than 20% of the total flow duration. For each flow, we use the average value of all bandwidth estimation samples as the estimated bandwidth and compare it with the actual utilized bandwidth.

For Figure 5.21, we plot the ratio of used bandwidth to estimated bandwidth per flow. The median value is only 19.78%. For 71.26% of the large flows, the bandwidth utilization ratio is below 50%. We notice that for 6.41% of the flows, the used bandwidth is slightly larger than the estimated bandwidth and this is possibly due to estimation error. On average, the utilization ratio is 34.60%, meaning that the data transfer for a TCP flow takes 52.91% longer than if the bandwidth is fully utilized, while keeping the ratio interface in the high-power state, incurring significant additional radio energy overhead [12].

Figure 5.22 shows two sample large TCP flows and their estimated bandwidth in the LTE data set. Note that these flows belong to two separate users at different time and the

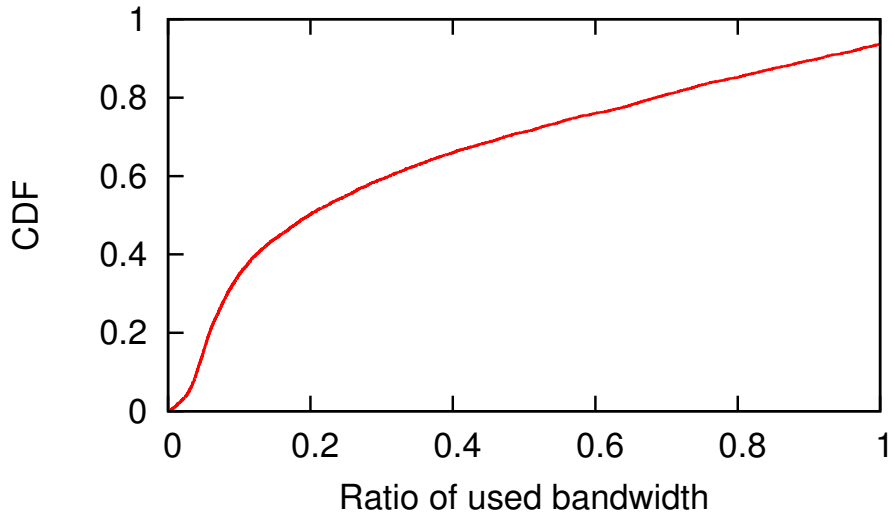


Figure 5.21: Bandwidth utilization ratio for large downlink TCP flows.

time is aligned only for presentation purposes. We can see that the available bandwidth varies significantly over time and even on the scale of seconds. This could be due to network condition changes, such as signal strength, as well as the changes of the network load in the associated eNB. In order to dissect the root cause of such variability, more information, *e.g.,* load information of eNB, is needed.

To understand how well TCP performs under highly variable available bandwidth, we use `iptables` to redirect packets to a packet scheduler we designed, which controls the variation of available bandwidth following the variations observed in LTE networks. The packet scheduler also adds different delays to each packet allowing us to understand the impact of RTT. Intuitively, TCP would adapt slower to the fast varying available bandwidth under large RTTs during congestion avoidance, as the congestion window is updated only once per RTT. We measure the bandwidth utilization ratio with the packet scheduler changing the available bandwidth every 500ms. We observe that under small RTTs, TCP can utilize over 95% of the available bandwidth. However, when RTT exceeds 400$\sim$600ms, the utilization ratio drops to below 50%. We also observe that for the same RTT, higher variation leads to lower utilization. These observations further suggest that long RTTs can degrade TCP performance in the LTE networks, which have inherently varying available

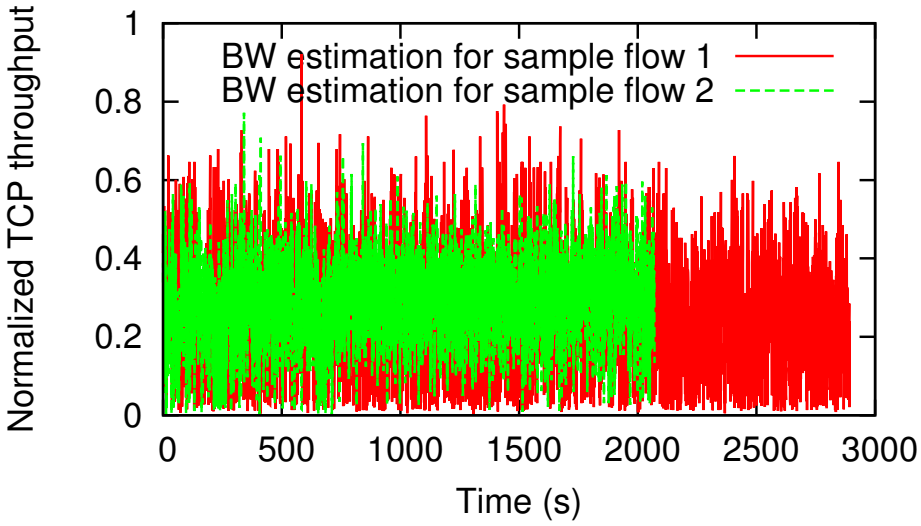


Figure 5.22: Bandwidth estimation timeline for two sample large TCP flows.

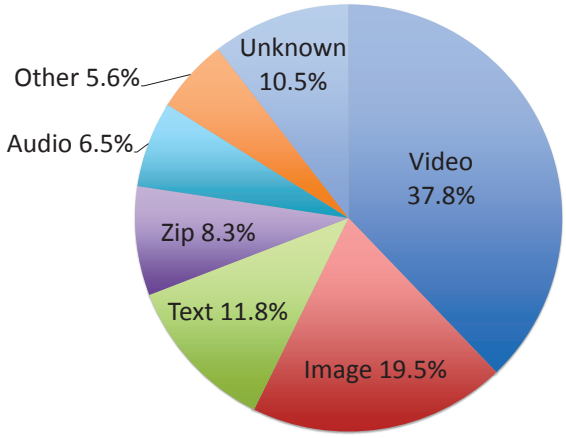


Figure 5.23: Breakdown of content type for all HTTP traffic based on total content size.

bandwidth (likely caused by changes in load and channel conditions).

5.5 Network Applications in LTE

In this section, we characterize the network applications and traffic patterns in the LTE data set. Specifically, we study the causes of inefficient bandwidth usage observed in Section 5.4.

5.5.1 HTTP Content Characterization

We break down the content type based on total bytes for HTTP traffic in Figure 5.23. About 37.8% is video, followed by 19.5% of image and 11.8% of text. Zip contributes to 8.3% of the HTTP traffic and we observe that these mostly correspond to file downloads, such as application updates, and audio contents consume 6.5%. The other files consist of 5.6% , and the remaining 10.5% is unknown. Within video contents, we observe 12.9% to be *octet-stream* type, *i.e.,* byte stream in binary format, and most of them generated by video players via byte-range requests.

Previous studies show that the multimedia content (video and audio) correspond to 40% of the traffic generated by mobile hand-held devices in DSL networks [71], and video contributes to 30% of the 3G cellular traffic [72]. Although we observe slightly higher percentage of multimedia traffic in this LTE network, the difference is insignificant. Overall, we observe multimedia content is still dominant in the LTE network studied followed by image content.

5.5.2 Inefficient Network Usage

We investigate the large flows with under-utilized network bandwidth and observe that the TCP receive window size [52] has become the bottleneck in many cases.

Figure 5.24 shows one such example: an iOS user uses the popular Shazam application [73] to download a 30-second music file of 1MB size. Initially, the data transfer speed is fast and between time 0s and 2s the average downlink throughput is over 3Mbps. However, between 2s and 9s, the average throughput decreases to less than 300Kbps. The total download time is 9 seconds and as indicated by the *ideal case* curve, the download could have been completed within 2.5 seconds, based on our estimation of the available bandwidth. In addition, we notice that the TCP connection is not immediately closed after the file transfer is complete, although the HTTP request specifies the connection type to be `Connection: close`. Instead, the connection is only torn down at 30s, after the mu-

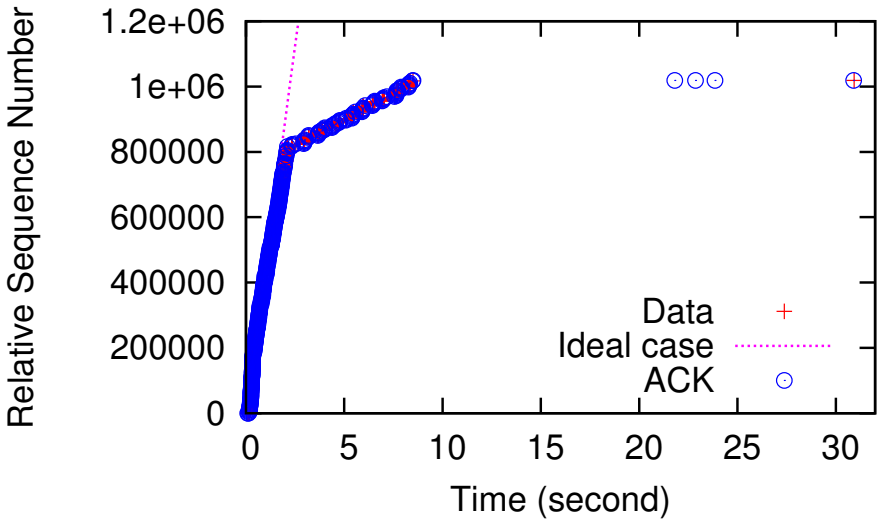


Figure 5.24: Full receive window slows Shazam player (a popular app) in downloading a 30-second music file.

sic clip has finished playing, and the client sends some TCP receive window updates to the server between 20s and 25s with increasing receive window size. Ideally, the connection is closed immediately after the file transfer completion. Overall, the total download process keeps the radio interface of the device active for around 38 seconds. Assuming a tail time of 10 seconds, in the ideal case, the active radio time is only 12.5 seconds.

The performance drop at 2s in Figure 5.24 is due to the full TCP receive window, *i.e.,* byte in-flight fully occupy the window size. Between 0s and 2s, the window size has gradually dropped to a small value, *e.g.,* at the turning point around 2s, the window size is 816 bytes, even smaller than the maximum payload size (1358 bytes in our traces). As TCP rate is controlled by both congestion window and receive window jointly, the full receive window would prevent the server from sending more data, regardless of the congestion window, leaving the network resource underutilized.

The reason for the full TCP receive window is two-fold. First, the initial receive window size is not large, *e.g.,* 131.76KB in the case of Figure 5.24, much smaller than the file size. We explore the initial advertised receive window size in all TCP flows, and observe that the values fall in 131,712$\pm$600 bytes for over 99% of the flows, for iOS, Android and

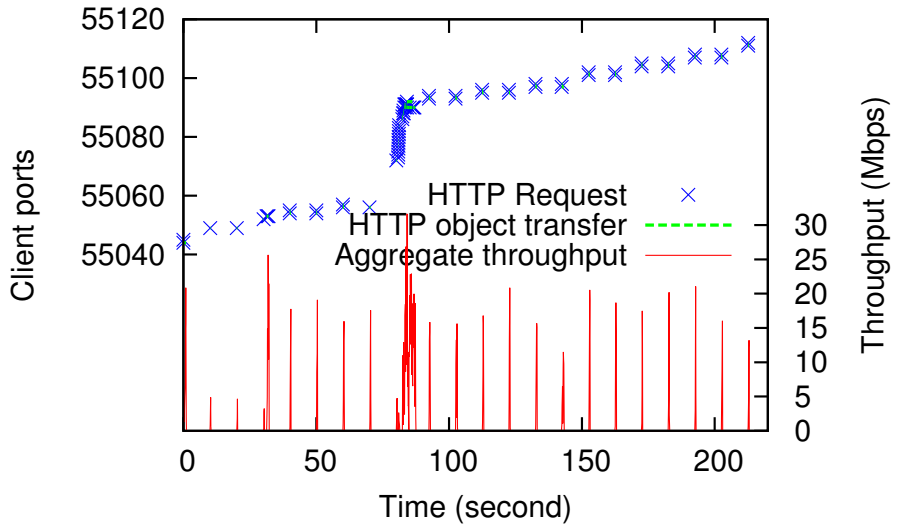


Figure 5.25: The periodic request behavior of Netflix player limiting its overall throughput.

Windows Phone devices. Second, the application is not consuming the data fast enough from the receiver's buffer, otherwise, even if the initial receive window is small, the receive window size should not drop to close to 0 afterwards.

We further study the prevalence of such TCP performance throttling by the receive window. We find that for all the downlink TCP flows, 52.61% of them experienced full receive window or even zero receive window. And for 91.24% of these affected flows, throttling of receive window happens in the initial 10% of the data transfer duration. These observations suggest that about half of the TCP flows in the LTE network studied are experiencing lower network performance limited by receive window size.

We also observe that some applications under-utilize the bandwidth due to the application design. Figure 5.25 shows the network behavior of the popular Netflix application [74] on iOS. The upper half of the figure shows the HTTP requests and the HTTP object transfers on different client ports. At around 70s, the user browses through a list of video thumbnails and switches to another video. We observe that all HTTP requests for video download are HTTP byte-range requests and the corresponding HTTP object transfers are mostly short in duration, *i.e.,* smaller than 1s. The content size of the requests ranges from 1MB to 4MB. The client periodically requests chucks of data every 10s, with each TCP

connection typically reused by two consecutive HTTP requests. The bottom half of the figure shows the aggregate downlink throughput, showing a clear periodic pattern corresponding to the periodic requests. While the peak throughput can reach up to 30Mbps, for most of the time, the network interface is idle. This type of traffic pattern is known for incurring high tail energy [30]. Especially in this case, we know that the tail timer for the studied network is 10s, and based on the radio resource control RRC state machine of the LTE networks [12], the 10-second request cycle would keep the radio interface of the device always in the high-power state, incurring high energy overheads.

5.5.3 Discussions

We have shown that multimedia traffic is dominant in the LTE network studied and the available bandwidth is far from effectively utilized by many applications. Hence optimizing the network utilization efficiency for these applications is critical for better user experiences, in terms of both responsiveness and energy consumption.

About the TCP receive window problem, existing studies [58] have shown that smartphone venders may have been reducing maximum receive window size to mitigate the buffer bloat problem, resulting in TCP performance degradation. Dynamic receive window adjustment (DRWA) [58] is proposed towards this problem. However, such proposals require changes to TCP stacks, making the deployment potentially challenging. Also, the potential drawbacks for such TCP modifications remain to be studied in the real LTE networks. Alternatively, a more lightweight and yet still effective solution could be that applications have their own buffers, or increase the buffer size if the existing buffer size is small, to consume the bytes in the TCP's receiver buffer quickly. For example, the ideal behavior of the Shazam player described in Figure 5.24 is that it downloads the file as fast as possible and stores the contents in an application-layer buffer, no matter how much of the music has been played, and closes the connection immediately when the file transfer is complete. The application-layer buffer is beneficial for both network and energy efficiency.

As for the periodical behavior of the Netflix player in Figure 5.25, in addition to the application-layer buffer, it is also recommended that it sends fewer requests and downloads more content for each request. We have shown that transferring data in a large batch significantly reduces the radio energy than otherwise [12]. Bulk data transfers also allow TCP to make better use of the available bandwidth.

5.6 Summary

In this chapter, we use a large-scale LTE data set to study the impact of network protocol and application behaviors on network performance. We show that the RTT in the LTE network studied grows with the TCP bytes in flight due to long queueing delay. Such high RTTs have caused performance problems for TCP upon packet losses. We have also shown that the available bandwidth for the LTE networks has high variation and long RTTs prevent TCP from fully utilizing the bandwidth as the congestion window cannot adapt fast enough. We further notice that the limited TCP receive window size has constrained TCP data rate for 52.61% of the downlink flows. By devising a bandwidth estimation algorithm, we observe that for 71.26% of the large flows, the bandwidth utilization ratio is below 50%. In addition, we also observe that the application design may result in underutilized bandwidth. We have seen that about 44.3% of the HTTP content is multimedia; therefore, increasing the network efficiency of the corresponding applications is important in improving the application responsiveness and energy efficiency.

CHAPTER VI

Characterizing Radio Energy Usage of Smartphones in Cellular Networks

Smartphones with cellular data access have become increasingly popular across the globe, with the wide deployment of 3G and emerging LTE [2] networks, and a plethora of applications of all kinds. Cellular networks are typically characterized by limited radio resources and significant device power consumption for network communications. The battery capacity of smartphones cannot be easily improved due to physical constraints in size and weight. Hence, battery life remains a key determinant of end-user experience. Given the limited radio resources in these networks and device battery capacity constraints, optimizing the usage of these resources is critical for cellular carriers and application developers. Specifically, radio power is reported to be 1/3 to 1/2 of the total device power [26], and in this chapter, we first devise a systematic way to characterize smartphone radio energy usage given packet traces as input and then we discuss our analysis results with real user traces.

6.1 Power Measurement Methodology

Similar to previous studies [20, 75], we use Monsoon power monitor [76] as power input for our device measuring power traces at the same time. The power trace contains

two fields, timestamp and average instant power, and the sampling rate is 5000Hz. The test device is an HTC phone with LTE data plan from a cellular ISP. It has 768 MB RAM memory and 1 GHz Qualcomm MSM8655 CPU, running Android 2.2.1. We remove the battery and connect $\oplus$ and $\ominus$ pins of the power monitor to device's $\oplus$ and $\ominus$ pins, respectively. By enabling V_{out} with the voltage of 3.7V, the device boots properly and the power monitor records the total power traces consumed by the device. Another tricky part for our setup is that, the back cover of our test device, HTC Thunderbolt, must remain attached firmly, otherwise, if it is pried off the device, LTE session is terminated and 1xRTT session is started. This is because part of the LTE antenna circuit lies inside the back cover [77]. In the end, we let the wires going out of the back cover through the hole of earplugs.

We share the same observation with previous study [75] that screen plays an important role in device power consumption, *i.e.,* with screen 100% on, the UE idle power is 847.15mW compared with 11.36mW with screen off. For all measurements, we keep the test application running in the background with screen completely off to minimize power noise, unless UI interactions are required and screen should be kept on, *i.e.,* measuring power for browser. In this case, we subtract screen power from the total, with slightly increased noise. All experiments are repeated at least 5 times to reduce measurement error.

To measure state transition power levels, UE keeps a long-lived TCP connection with the server and packet traces are collected to make sure there is no background traffic. In order to trigger state promotions, we make the device idle for sufficient time, *e.g.,* 30 seconds, and then send a packet from server to client. UE remains idle afterwards and demotes to idle state in the end, and the power trace covers the full tail.

6.2 Smartphone Power Model

In Section 2.1, we have discussed the radio resource control mechanisms in cellular network. In fact, RRC state machine is the key factor for determining the UE power consumption for both 3G and LTE 4G networks.

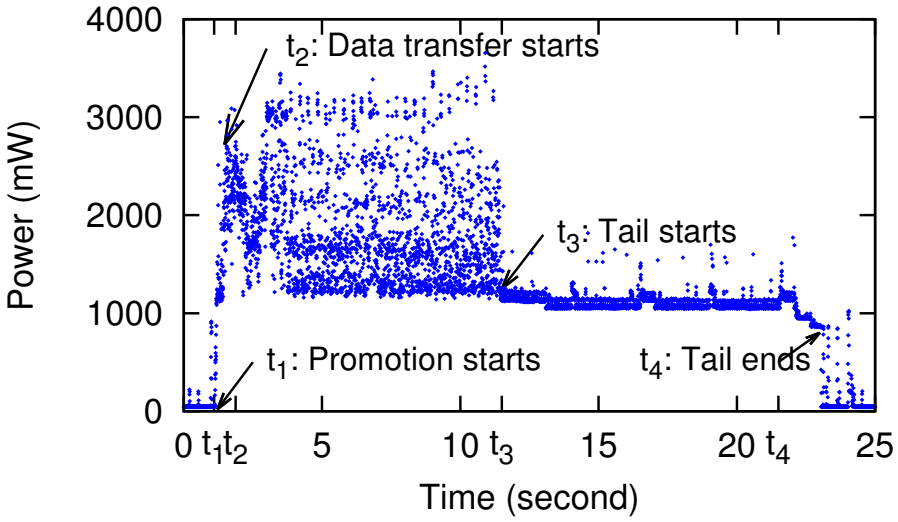


Figure 6.1: Power states of LTE.

In this section, we take LTE network as a sample and illustrate the power traces of our test Android smartphone in a commercial LTE network based on local experiments. We observe that network activities match the corresponding state transitions indicated by different power levels.

Figure 6.1 shows the power trace of uploading at the speed of 1Mpbs for 10 seconds. With screen off, the energy is mostly consumed by the radio interfaces, as the power level is less than 20mW before t_1. At t_1, the application sends a TCP SYN packet triggering **RRC_IDLE** to **RRC_CONNECTED** promotion, and the application waits for T_{pro} until starting data transfer at t_2. Between t_2 and t_3, depending on the instant data rate, the power level fluctuates. We notice the power level during fast data transfer is significantly higher than the base power in **RRC_CONNECTED**, which motivates us to incorporate data rates into our LTE power model. After the data transfer completes at t_3, the device remains in **RRC_CONNECTED** for a fixed tail time T_{tail}, until t_4, when the device goes back to **RRC_IDLE**. The periodicity of DRX between t_3 and t_4 is not obvious due to limited sample rate.

Figure 6.2 is a 125× zoom-in view of Figure 6.1's tail, which clearly illustrates DRX activity in **RRC_CONNECTED** mode. The device activates its receivers to listen to downlink

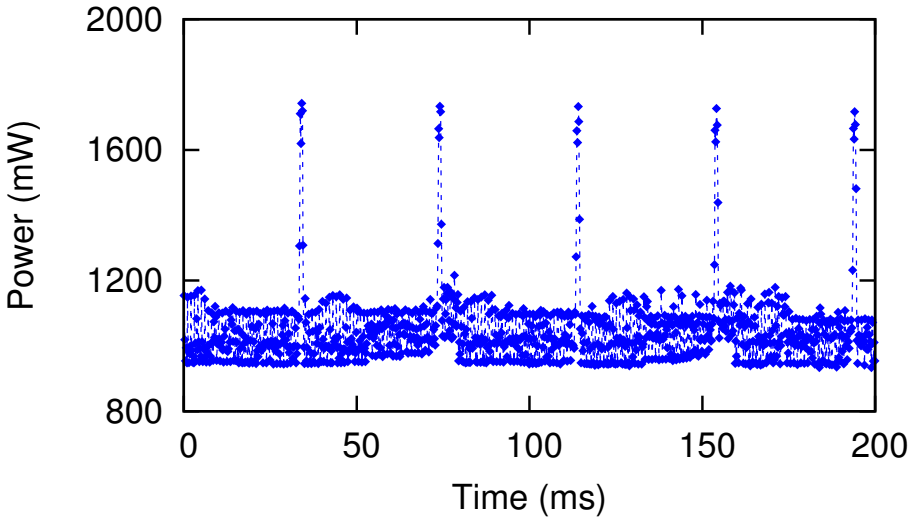


Figure 6.2: Zoom-in view in **RRC_CONNECTED**.

control channel. If downlink traffic is found awaiting, the device goes into Continuous Reception model; otherwise, the device goes into a dormant state without continuously checking downlink control channel, though still remaining in **RRC_CONNECTED** mode. The spikes appearing every 40 ms in Figure 6.2 match the on period of DRX in **RRC_CONNECTED**.

6.3 Power Model Construction

This section summarizes the construction of the new LTE power model, as well as the 3G and WiFi model measured from the same LTE phone. We then compare energy efficiency in bulk data transfer for different networks and validate LTE power model in the end.

6.3.1 Power model for RRC and DRX

With the experimental setup described in Section 6.1, we measure power model for LTE, 3G, and WiFi on the LTE phone, summarized in Table 6.1. The LTE parameter values are validated by the network-based measurement in Section 3.4.2. For simplicity, we ignore the WiFi AP scanning and association, assuming UE is already connected with

	Power* (mW)	Duration (ms)	Periodicity (ms)
Screen off (base)	11.4±0.4	N/A	N/A
Screen 100% on	847.2±2.7	N/A	N/A
LTE promotion	1210.7±85.6	T_{pro}: 260.1±15.8	N/A
LTE Short DRX On **RRC_CONNECTED**	1680.2±15.7	T_{on}: 1.0±0.1	T_{ps}: 20.0±0.1
LTE Long DRX On **RRC_CONNECTED**	1680.1±14.3	T_{on}: 1.0±0.1	T_{pl}: 40.1±0.1
LTE tail base	1060.0±3.3	T_{tail}: 11576.0±26.1	N/A
LTE DRX On **RRC_IDLE**	594.3±8.7	T_{oni}: 43.2±1.5	T_{pi}: 1280.2±7.1
3G promotion	659.4±40.4	582.1±79.5	N/A
3G DCH tail base	803.9±5.9	8088.2±149.6	N/A
3G FACH tail base	601.3±6.4	824.2±148.1	N/A
3G DRX (idle)	374.2±13.7	55.4±1.5	5112.4±37.7
WiFi promotion	124.4±2.6	79.1±15.1	N/A
WiFi tail base	119.3±2.5	238.1±9.2	N/A
WiFi beacon (idle)	77.2±1.1	7.6±0.1	308.2±1.0

Table 6.1: LTE, 3G, and WiFi power model.

*All power readings in this table include the base power (screen off), which has negligible impact on total energy.

an AP.

First, we observe that LTE reduces the promotion delay (T_{pro}) from 3G's 582.06ms to 260.13ms. However, the power level is almost doubled, *i.e.,* 1210.74mW (LTE) v.s. 659.43mW (3G). WiFi has the most lightweight state promotion with smaller T_{pro} and much lower power level.

Secondly, LTE appears to have longest tail (11.576 seconds) with highest tail base power (1060.04 mW). Summing up DCH and FACH tail, 3G's total tail time (8.9 seconds) is smaller than LTE's T_{tail} of 11.6 seconds. Even 3G DCH's tail base power is 24.17% lower than LTE's tail base power, and the gap becomes 25.25% if we consider LTE DRX in **RRC_CONNECTED** with a high on duration power (1680.20mW). WiFi is much more power

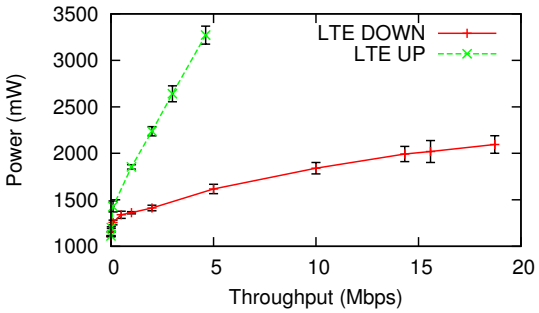


Figure 6.3: Power-throughput curve for LTE network.

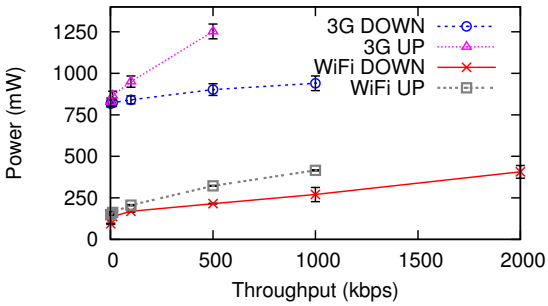


Figure 6.4: Power-throughput curve for 3G and WiFi.

efficient, with shorter tail and much lower base power.

We also compare LTE DRX in **`RRC_IDLE`** with 3G DRX and WiFi beacon in the idle state. LTE has the highest on power and slightly smaller On Duration than 3G, while WiFi has smallest on power and On Duration. The cycle of LTE (1.28 seconds) is in between 3G and WiFi.

Based on these observations, LTE is less energy efficient during idle state and for transferring smaller amount of data. For example, if only one packet is transferred, the energy usage considering both promotion and tail energy for LTE, 3G and WiFi is 12.76J, 7.38J and 0.04J, respectively. One possible reason for LTE's higher power states is that devices must incorporate *multiple-input and multiple-output* (MIMO) to support LTE network, *e.g.,* the test device we use has 1 transmit antenna and 2 receive antennas, which contributes to higher power consumption.

6.3.2 Power Model for Data Transfer

Previous work on 3G UMTS power modeling either treats DCH power state to have a fixed power value [75, 26], or assumes energy per bit to be the same constant for both uplink and downlink [21]. These assumptions might be reasonable given that 3G has relatively low data rates. However, for LTE, we observe that device power is much higher during high speed data transmission (up to 3300mW for uplink) relative to the base power (1060mW)

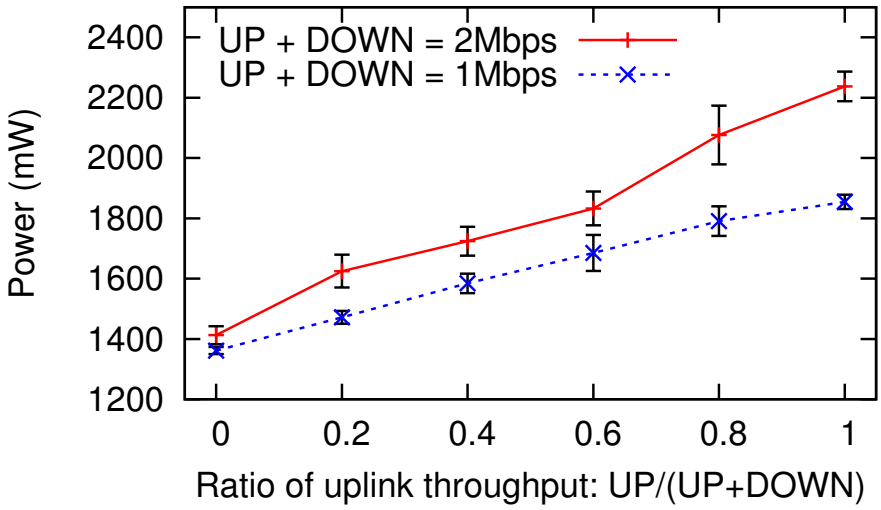


Figure 6.5: Power of simultaneous uplink and downlink transfers.

	α_u (mW/Mbps)	α_d (mW/Mbps)	β (mW)	α_u/α_d
LTE	438.39	51.97	1288.04	8.44
3G	868.98	122.12	817.88	7.12
WiFi	283.17	137.01	132.86	2.07

Table 6.2: Data transfer power model.

in **RRC_CONNECTED**, and there is significant difference between downlink and uplink power levels at the same data rate. In this paper, we propose a new comprehensive power model for LTE empirically derived in a commercial LTE network.

We start with measuring device power states with controlled uplink or downlink throughput. The impact of TCP ACK packets, which are small in size, is minor, thus ignored.

Figures 6.3 and 6.4 present the power-throughput curve for LTE, 3G, and WiFi. The curves are limited by the peak data rate we can achieve at the test location. We observe that for all networks, a linear model fits well for both uplink and downlink. Assume uplink throughput is t_u (Mbps) and downlink throughput is t_d (Mbps), the power level (mW) for uplink is $P_u = \alpha_u t_u + \beta$ and for downlink $P_d = \alpha_d t_d + \beta$. The best fit parameters are listed in Table 6.2.

By looking at α_u/α_d, we notice that uplink power increases faster than downlink for

all three networks types. This is expected because sending data requires more power than receiving data for wireless data access [78]. LTE has the largest gap of $\alpha_u/\alpha_d = 8.44$ among three network types. This is largely because α_d for LTE is quite small. For 3G, both α_u and α_d are larger than LTE. β is the base power when throughput is 0, with the ranking of LTE $>$ 3G $>$ WiFi. This is consistent with the tail base power comparison in Table 6.1. We notice that β is slightly higher than the tail base for all networks types. This is possibly because of the overhead of switching transmitters or receivers into high speed mode.

For simultaneous uplink and downlink transfers, given that transmitters and receivers are separate, we conjecture that the power level (mW) is given by the following formula:

$$P = \alpha_u t_u + \alpha_d t_d + \beta$$

To validate this conjecture, we measure the power levels for concurrent uplink and downlink transfers in Figure 6.5. Assume total throughput $t = t_u + t_d$ and the ratio of uplink throughput $\epsilon = t_u/t$:

$$P = \alpha_u t_u + \alpha_d t_d + \beta = (\alpha_u - \alpha_d)t\epsilon + \alpha_d t + \beta$$

When t is a constant, P grows linearly with ϵ and the slope is $(\alpha_u - \alpha_d)t$. Figure 6.5 shows two curves of $t = 1$Mbps and $t = 2$Mbps, both having a strong linear pattern and the slope is less than 5% off the expected value.

6.3.3 Energy Efficiency for Bulk Data Transfer

To compare the power efficiency of different networks in the wild, we use bulk data transfer experiments to measure energy per bit. Perrucci *et al.* [79] measure energy per bit for 3G and WiFi with a fixed bulk size. In addition to taking LTE into consideration, we vary the bulk size to cover more possible network usage scenarios. Figure 6.6 shows the measured energy per bit in $\mu J/bit$ ($10^{-6} Joule/bit$) with different bulk data size. All data

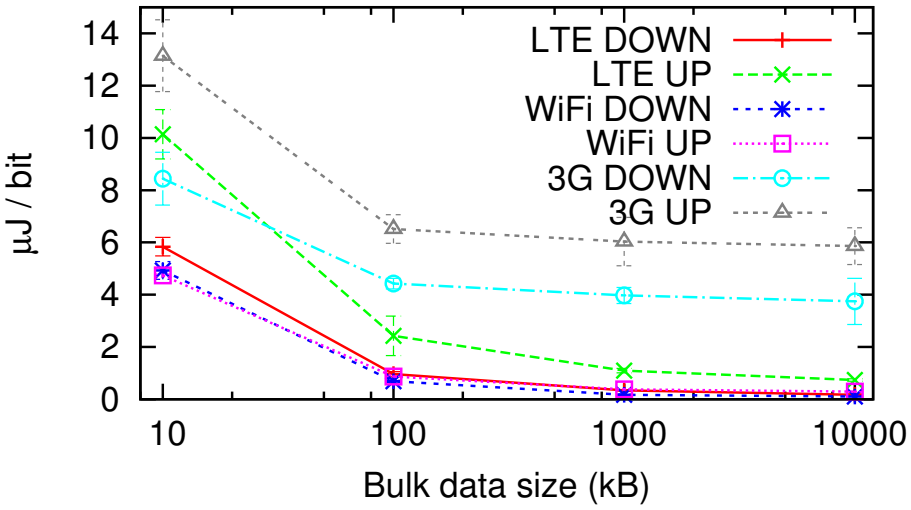


Figure 6.6: Energy per bit for bulk data transfers.

is randomly generated so that there is no chance for caching. We do not include promotion or tail energy but instead focus on data transfer energy. Given that signal strength and peak data rate on wireless networks fluctuates, both affecting energy per bit, our measurement only serves as a sampled view for the energy efficiency of different networks.

First, energy per bit decreases as bulk data size increases, largely because with a small data size, throughput does not reach link capacity due to TCP slow start. We also observe that LTE's energy per bit in downlink is comparable with WiFi, although the absolute power level of LTE is much higher than WiFi. This is due to high downlink throughput for LTE at our test location, even compared with the WiFi network. Similarly, for LTE uplink, it drops from $10\mu J/bit$ to less than $1\mu J/bit$ as bulk data size increases. With bulk data size of 10MB, LTE consumes 1.62 times the energy of WiFi for downlink and 2.53 for uplink. With lowest throughput, 3G has the worst energy efficiency for large data transfer, *e.g.,* for downloading 10MB data, 3G requires 21.50 times the energy of LTE and 34.77 times the energy of WiFi, and for uplink, 7.97 times of LTE and 20.16 times of WiFi.

App	Measured energy (J)[1]	Simulated energy (J)[1]	Error
Website G[3]	24.77	24.37	-1.61% (-2.06%[2])
Website Y[4]	31.84	30.08	-5.53% (-7.04%)
YouTube	21.81	21.14	-3.07% (-4.17%)
NPR News	24.65	24.37	-1.12% (-1.70%)
Market	38.64	38.03	-1.58% (-3.03%)

[1]Both measured and simulated energy include tail energy.
[2]This error is for simulated energy assuming $\alpha_u = \alpha_d = 0$.
[3]Website G is the mobile version of `google.com`.
[4]Website Y is the non-mobile version of `yahoo.com`.

Table 6.3: LTE power model validation.

6.3.4 Power Model Validation

To validate the LTE power model and the trace-driven simulation (§6.4.3), we compare measured energy (measured from the LTE phone) with simulated energy for case study applications. Table 6.3 contains the sample application usage scenarios described in §4.2.6. The error rate is consistently less than 6%, with the largest error rate from Website Y, which includes heavy JavaScript execution and HTML rendering. Since our power model focuses on radio power and ignores the impact of CPU, for Website Y, the total energy usage is slightly underestimated.

The error rate is increased if the impact of downlink and uplink throughput is ignored, *i.e.,* assuming $\alpha_u = \alpha_d = 0$. However, the increase is not significant, at most 1.5%. This is because for these web-based applications, network throughput is low due to small object size. For other applications, such as video/audio streaming and file download, we expect to see a larger gap in error rate if the impact of downlink/uplink is ignored.

In this section, in addition to comparing energy per bit in bulk data transfer for different networks, we construct a new LTE power model and validate its accuracy, which is the basis for the following analysis.

6.4 Trace-driven Analysis Methodology

To compare energy consumption for different networks using real user traces and evaluate the impact of setting LTE parameters, we devise a systematic methodology for trace-driven analysis, which is applied to a comprehensive user trace data set, named UMICH.

6.4.1 UMICH Data Set for Analysis

UMICH data set is collected from 20 smartphone users for seven months, from May 9 2011 to December 5 2011. These participants consist of undergraduate and graduate students from 8 departments at University of Michigan. The 20 participants are given Motorola Atrix (11 of them) or Samsung Galaxy S smartphones (9 of them) with unlimited voice, text and data plans, all running Android 2.2. They are encouraged to take advantage of all the features and services of the phones. As stated in our study consent form, we keep collected data and users' identities strictly confidential [1].

We develop custom data collection software and deploy it on the 20 smartphones. It continuously runs in the background and collects four types of data:

- Full packet traces in `tcpdump` format including both headers and payload. The detailed packet traces allows us to keep detailed track of traffic patterns generated by each individual user for a long period.

- The process name responsible for sending or receiving each packet. This packet-to-process correspondence is derived by efficiently correlating three mappings in Android OS in realtime [26]: `/proc/PID/fd` (inode of each TCP/UDP socket→Process ID), `/proc/net/tcp(udp)` (socket→inode), and `/proc/PID/cmdline` (Process ID → Process name).

- User input events such as pressing a button and tapping the screen. This is collected by reading `/dev/input/event*` in the Android system.

[1] This user study has been approved by the University of Michigan IRB-HSBS #HUM00044666.

- Screen on/off status data with a sampling rate of $1Hz$. In order to associate individual packets with their screen status, we define a time window $[t_1, t_2]$ to be a screen-on (or screen-off, respectively) window if "all" the screen samples in this window have screen-on (or screen-off) status. Then we classify a packet as either screen-on or off if its timestamp falls respectively into any screen-on or screen-off window, and unknown otherwise. One reason for the occurrence of the unknown category is data collection errors.

Both cellular and Wi-Fi traces are collected without any sampling performed. The data collector incurs no more than 15% of CPU overhead, although the overhead is much lower when the throughput is low (*e.g.,* less than 200 kbps).

We also build a data uploader that uploads the data to our server when the phone is idle as indicated by low network throughput. The upload is suspended by increased network activity of user applications. Since the data collector needs to be shut down when the data is being uploaded, we set the interval of consecutive uploading attempts to 6 hours to minimize the impact of uploader on data collection. The entire data collection and uploading process is transparent to the users, although we do advise the users to keep their phones powered on as often as possible.

During the deployment period, we have collected 152GB data (150 GB `tcpdump` traces). Although both cellular and Wi-Fi traces were collected, in this study, we consistently use 3G traces only, which contribute 57.82% of the total traffic volume, in the UMICH dataset for further analysis. Comparing with the Wi-Fi traces, 3G traces better represent the user and application behavioral patterns in cellular networks.

6.4.2 Burst Analysis Methodology

In order to understand the screen-off traffic pattern and its impact on radio resource and device energy, we use the following traffic model for burst analysis. Intuitively, a burst is a continuous data transfer with preceding and succeeding idle times. For each user, the

traffic trace is a sequence of network packets, $P_i (1 \leq i \leq n)$. Notice that P_i could be either downlink or uplink. If the timestamp of P_i is defined to be t_i, we have $t_i \leq t_j$ for any $i < j$. Using a burst threshold **BT**, the packets are divided into ***bursts***, *i.e.,* $\{P_p, P_{p+1}, \cdots, P_q\}$ belongs to a burst B, if and only if: *i)* $t_{k+1} - t_k \leq$ **BT** for any $k \in \{p, \cdots, q-1\}$, *ii)* $t_{q+1} - t_q >$ **BT** and *iii)* $t_p - t_{p-1} >$ **BT**. We define the inter-burst time **IBT** for burst B to be the time gap following this burst, *i.e.,* $t_{q+1} - t_q$. In this paper, we empirically choose to use **BT** = 2 seconds, which is validated to be larger than most packet gaps for 3G/4G networks within a continuous data transfer, such as downloading a web object.

6.4.3 Trace-driven Modeling Methodology

We build a network model and power model analysis framework for the trace-driven analysis, totaling about 8,000 lines of code in C++.

Network model simulator takes the binary packet trace files in libpcap [80] format and preprocesses the data following the same methodology as previous study [20], for the purpose of removing existing delays imposed by state promotions and extracting the actual traffic patterns, *e.g.,* for the purposing of simulating the trace in another network. It then applies a specific network model to the processed traces and adjusts timestamps for some packets, since different network models have different delay behavior due to state machine differences. There are two reasons for a packet's timestamp to be adjusted, promotion delay and DRX waiting time. We use the same methodology to adjust timestamp for promotion delay as previous study [20]. In LTE network, if there is no packet for over T_{tail} time, when the next packet P arrives, an extra delay T_{pro} is inserted before P. In terms of DRX waiting time, if a downlink packet P' comes at time $T_{P'}$ and UE is in DRX mode, either in **RRC_CONNECTED** or **RRC_IDLE**, there is no DRX waiting time if P' arrives inside any DRX on duration; otherwise, P' would experience an extra delay of $T_n - T_{P'}$, where T_n is the start time of the next DRX on duration after $T_{P'}$. The parameters for different net-

work models are listed in Table 6.1, obtained from local experiments. We also empirically assume that UE goes into complete sleep with 0 power usage after being idle for T_{tail2} (1 minute). Notice that T_{tail2} is not actually observed in our experiments and is only used to bound the total idle energy and simplify our analysis.

The output of network model simulator is an array of packets with adjusted timestamps, in ascending order, as well as the RRC and DRX states of UE at any time.

Power model simulator takes the output of network model simulator and calculates the energy consumption based on the power model, detailed in Section 6.3. We break down the total energy into four components: *promotion*, *data transfer*, *tail*, and *idle*.

For promotion energy, both promotion delay T_{pro} and average promotion power P_{pro} are fixed. If a trace includes N promotions, promotion energy is given by $NT_{pro}P_{pro}$.

For idle energy in **RRC_IDLE**, assume that DRX cycle is T_{pi} (with base power P_b) and on duration is T_{oni} (with on power P_{oni}), energy consumption of an duration of t_{idle} is given by

$$\lfloor \frac{t_{idle}}{T_{pi}} \rfloor \Big(T_{oni}P_{oni} + (T_{pi} - T_{oni})P_b \Big) + E_{res}$$

where the remaining energy E_{res} can be calculated as

$$E_{res} = \begin{cases} T_{res}P_{oni} & \text{if} \quad T_{res} \leq T_{oni} \\ T_{oni}P_{oni} + (T_{res} - T_{oni})P_b & \text{if} \quad T_{res} > T_{oni} \end{cases}$$

with

$$T_{res} = t_{idle} - T_{pi} \lfloor \frac{t_{idle}}{T_{pi}} \rfloor$$

Tail energy is more complex for LTE since **RRC_CONNECTED** has three modes as in Figure 2.2, while 3G and WiFi do not have DRX in connected mode. We simulate the transitions among Continuous Reception, Short DRX and Long DRX to calculate tail energy, in a similar way of idle energy calculation. Note that we only count tail energy when

the tail is complete, otherwise, the energy is considered as data transfer energy, *i.e.,* if one packet P is observed before UE demotes to **RRC_IDLE** at the end of the tail, the energy between P and the previous packet is still considered as data transfer energy.

For data transfer energy, we propose a novel model. Assume uplink throughput is t_u (Mbps) and downlink throughput is t_d (Mbps), the instant power level (mW) of UE is

$$P = \alpha_u t_u + \alpha_d t_d + \beta$$

Validation of this formula is in §6.3.2. We simply consider payload throughput in our power model, without considering header length and ACK packets due their small impact. Since TCP instant throughput changes frequently, in order to accurately estimate the instant power levels, we divide the trace into small time windows with size W seconds and within each window, throughput t_u and t_d are calculated, which determine the power P. W is set to be 1 second empirically in our setup and the total energy is not sensitive to the choice of W as long as W is larger than a few RTTs and smaller than most flow durations.

6.5 User Trace based Tradeoff Analysis

In this section, we apply the LTE power model to UMICH data set and compare energy efficiency with 3G and WiFi, with detailed break down of the total energy. We then study the tradeoff of configuring different LTE parameters via our analysis framework.

6.5.1 Energy Efficiency Comparison

We use the UMICH data set to simulate the LTE, WiFi and 3G model. Assume that the simulated energy usage for LTE, WiFi and 3G power model is E_{lte}, E_{wifi} and E_{3g}, respectively, the energy ratio of LTE/WiFi is defined as E_{lte}/E_{wifi}, and that for 3G/WiFi is calculated as E_{3g}/E_{wifi}. With the same traces, we can make fair comparison among different power models to understand their energy efficiency.

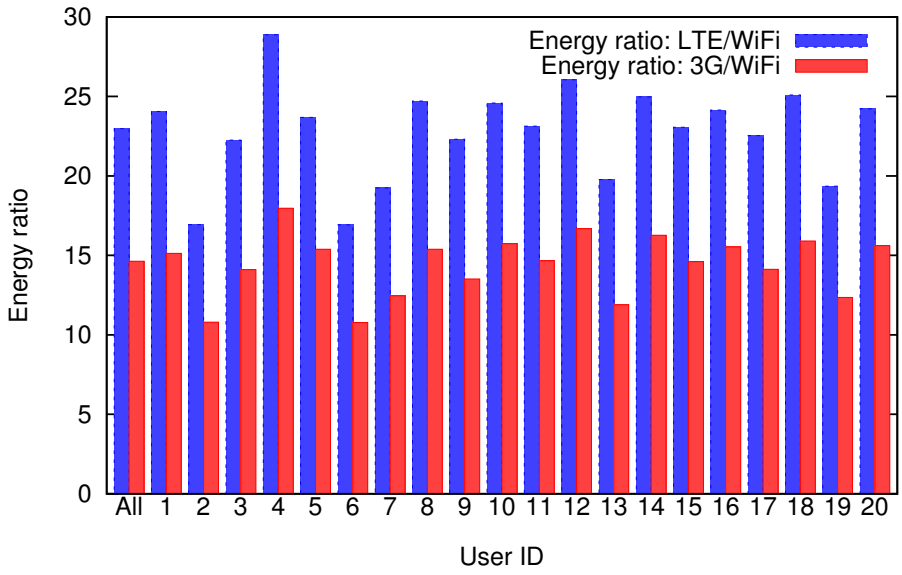


Figure 6.7: Power model simulation: energy ratio.

In Figure 6.7, we compare the energy efficiency of different networks for the 20 users both individually and in aggregate (summing up for all users). We first observe that LTE power model consumes significantly more energy than WiFi. The ratio of LTE/WiFi ranges from 16.9 to 28.9 and the aggregate ratio for all users is 23.0. Notice that the gap between LTE and WiFi is larger compared with the bulk data transfer experiments in §6.3.3. This is because, for bulk data transfer, LTE's high throughput could compensate the low energy efficiency, compared with real traces, which do not saturate the link capacity. Second, for 3G/WiFi ratio, the range is between 10.8 and 18.0, and in aggregate, 3G/WiFi ratio is 14.6, lower than LTE. In summary, the energy efficiency for LTE is lower than 3G, with WiFi having a much higher energy efficiency. Notice that in this work, we do not consider AP association energy for WiFi networks due to the lack of such information, so E_{wifi} is an underestimate. However, we believe that AP association does not happen very often and the impact is not significant.

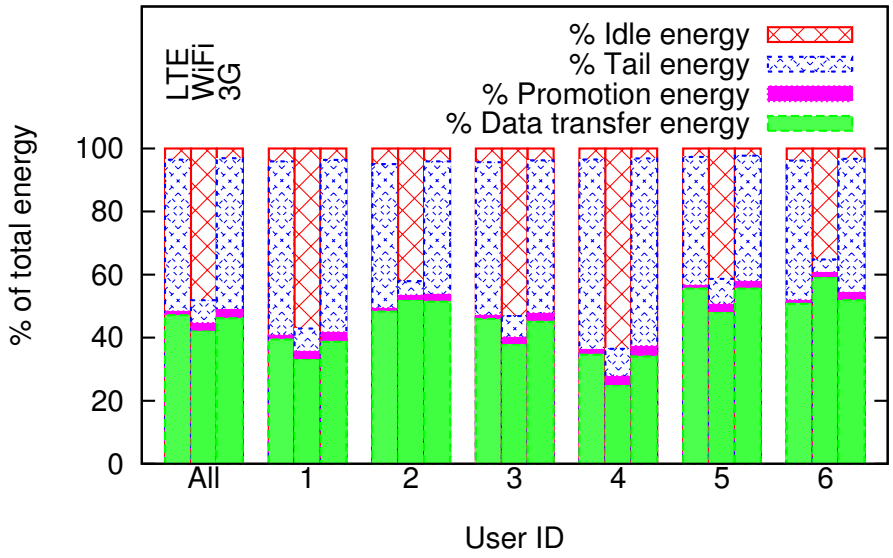


Figure 6.8: Break down of energy usage in analysis.

6.5.2 Energy Consumption Break Down

To further understand the energy consumption, with the methodology discussed in §6.4.3, it is decomposed into promotion energy, data transfer energy, tail energy, and idle energy in Figure 6.8 with results for aggregate analysis and 6 sample users. These energy values are calculated including the UE base power.

Promotion energy contributes to a small portion ($< 4\%$) of the total energy for all networks, *i.e.,* in aggregate, its contribution is only 1.2%, 2.5% and 2.6% for LTE, WiFi, and 3G, respectively. In terms of idle energy, WiFi has significantly higher percentage than LTE and 3G, despite small average power difference at idle state across networks, *i.e.,* 31.1mW [2] for LTE, 13.0mW for WiFi and 15.3mW for 3G. This is explained by WiFi's smaller total energy, making its idle energy contribution relatively higher.

Aggregate data transfer energy percentage is 47.1%, 42.1% and 46.2% for LTE, WiFi and 3G, respectively. The variation across users is high, *e.g.,* for LTE network, it ranges from 22.0% to 62.3%, due to traffic pattern differences across users.

[2]The idle power for LTE network is calculated as the average power of a DRX cycle in **RRC_IDLE**, and similarly for WiFi and 3G networks.

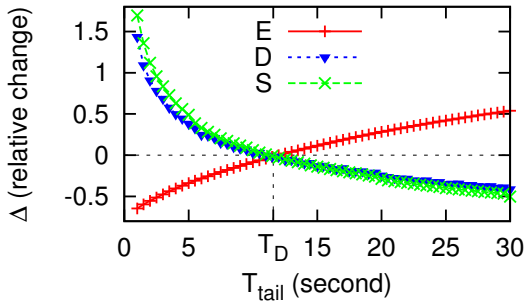


Figure 6.9: Impact of the LTE tail timer T_{tail}.

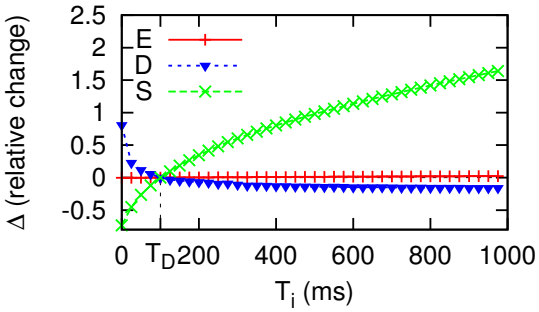


Figure 6.10: Impact of DRX inactivity timer T_i.

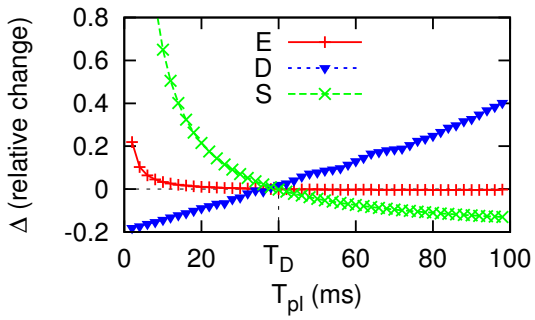


Figure 6.11: Impact of the DRX cycle in **RRC_CONNECTED** (T_{pl}).

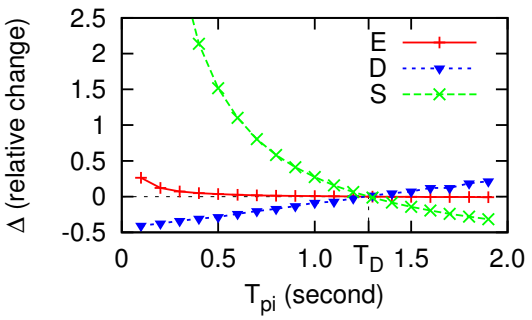


Figure 6.12: Impact of DRX cycle in **RRC_IDLE** (T_{pi}).

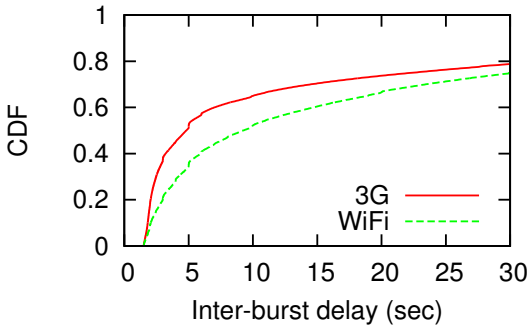


Figure 6.13: CDF of Inter-burst delay for UMICH data set.

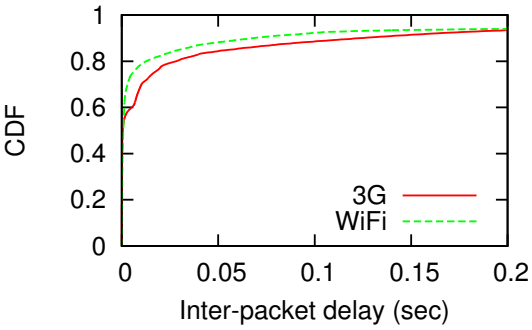


Figure 6.14: CDF of Inter-packet delay for UMICH data set.

Surprisingly, the biggest energy component for LTE network is tail, rather than data transfer. The average tail energy for LTE and 3G is 48.2% and 48.1% respectively compared to 7.2% for WiFi. Our observation for 3G is consistent with previous study [21]. Combined with high data transfer energy due to the higher power levels, tail energy lowers the energy efficiency of LTE and 3G compared to WiFi.

6.5.3 Impact of LTE Parameters

Similar to Section 6.5.1, we use WiFi traces in the UMICH data set to study the impact of LTE parameter configuration on radio energy E, channel scheduling delay D, and signaling overhead S. E is the simulated total energy consumed by UE. Since most energy is consumed by radio interfaces, with display energy excluded, we approximate this total energy as radio energy. D is the sum of scheduling delay for all packets, resulting from two factors: waiting for the next DRX cycle's on duration for a downlink packet and waiting for T_{pro} during **RRC_IDLE** to **RRC_CONNECTED** promotion. S is the overhead of the LTE network for serving this UE. S has different definitions in our study given that different parameters studied affect signaling load in different ways, *e.g.,* T_{tail} affects mostly the state promotions, and T_{pi} affects the total on duration in **RRC_IDLE**. D and E directly affect end users via application delay and battery life time; S is of interest to cellular ISPs for supporting large user base at low cost.

Impact of LTE tail timer T_{tail}: In Figure 6.9, we study the impact of LTE tail timer T_{tail}, with S defined to be the total number of **RRC_IDLE** to **RRC_CONNECTED** promotions. T_{tail} varies from 1 second to 30 seconds. T_D is the default configuration of 11.6 seconds for T_{tail} in the measured LTE network with the corresponding values of E, D and S calculated as E_D, D_D and S_D, respectively. When T_{tail} is set to be T', similarly, we get E', D' and S'. The change relative to the default setting is calculated as $\Delta(E) = (E' - E_D)/E_D$, and similarly for $\Delta(D)$ and $\Delta(S)$. Δs for each T_{tail} values are plotted in Figure 6.9 and they

are all 0 at T_D.

As expected, a larger T_{tail} value reduces both D and S, at the cost of E. We observe that $\Delta(S)$ and $\Delta(D)$ curves are almost identical, since most channel scheduling delay results from idle to active promotions. In addition, the impact of T_{tail} on the radio energy E is significant. For example, when T_{tail} is set to be 30 seconds, total radio energy E increases by 55%, while for $T_{tail} = 1$ second, E decreases by 65%, at the cost of 143% $\Delta(D)$ and 169% $\Delta(S)$. This indicates that T_{tail} is a very important parameter for LTE network.

In previous study [26], traffic bursts attribute to low efficiencies of radio resource and energy utilization in 3G UMTS network, due to tail time. Given that LTE has similar tail time, we analyze the bursts in the UMICH data set. We follow the same definition of a burst as in [26], which is a sequence of consecutive packets whose inter-arrival time is less than a threshold δ, with δ set to be 1.5 seconds, since it is longer than common cellular RTTs [1]. From ISP's perspective, T_{tail} should not be smaller than majority of the inter-burst delays; otherwise, a larger number of promotions would be triggered.

We study the inter-burst delay in the 3G and WiFi traces of UMICH data set in Figure 6.13. For 3G traces, 67.3% inter-burst delays are smaller than the default value of T_{tail} (11.58 seconds), and 55.4% for WiFi traces. Compared with WiFi traces, 3G traces have smaller inter-burst delays. We also observe the inter-burst delay distribution for different users have non-negligible differences. This makes us believe that a per-UE based dynamic T_{tail} configuration mechanism, adapting to traffic patterns, provides a better chance for optimizing radio resource efficiency and balancing energy utilization. Notice that this approach is inside the radio access network and transparent to UE. Previous study on 3G UMTS network also points out two possible approaches to deal with the tail effect with support from UE, *i.e.,* applications proactively alter traffic patterns based on the state machine behavior without interaction with the network, or cooperate with the radio access network in allocating radio resources by fast dormancy [31, 32]. These approaches remain to be applicable solutions for the tail problem in LTE.

Impact of DRX inactivity timer T_i: Figure 6.10 shows the tradeoff of setting DRX inactivity timer T_i. Signaling overhead S is defined as the sum of the continuous reception time and DRX on durations in **RRC_CONNECTED**, since during these time slots, UE exchanges control messages with eNB (base station of LTE network) [33]. A larger T_i keeps UE in continuous reception longer and reduces the scheduling delay for downlink packets, *i.e.,* no DRX waiting time, at the cost of higher energy usage, since continuous reception has higher power level than DRX idle state. T_D is the default value 100ms.

We observe that T_i has negligible impact on E, with only 2% $\Delta(E)$ as T_i is set to be 1 second. Different from E, S is significantly affected by T_i. For D, when T_i is set to be small than 100ms, $\Delta(D)$ grows up to 80%. However, even when T_i is set to be as large as 1 second, D is only reduced by 16%. This can be partly explained by Figure 6.14 that 88.6% packet pairs has ≤ 100ms inter-packet delay for 3G traces and 92.3% for WiFi traces in the UMICH data set. Decreasing T_i causes more packets arrive outside of continuous reception, which may experience DRX delay. Hence similar to T_{tail}, the setting of T_i is also affected by the traffic pattern, *i.e.,* inter-packet delay distribution, which may differ across users.

Impact of DRX cycle in RRC_CONNECTED: In **RRC_CONNECTED**, Short DRX cycle timer T_{is} determines the transition from Short DRX to Long DRX. Given that we already vary the DRX cycle to understand its impact, for simplicity, we assume T_{is} to be 0, *i.e.,* the DRX cycle in **RRC_CONNECTED** refers to Long DRX cycle T_{pl}.

In Figure 6.11, S is defined similarly as in Figure 6.10, *i.e.,* the sum of continuous reception time and DRX on durations. T_D is 40ms, the default setting for T_{pl}. First, we notice that compared with T_D, when T_{pl} is set to be 100ms, energy saving $\Delta(E)$ is minor (0.2%) with 13.0% reduction in S, at the cost of 40% increase in D. So T_{pl} is not recommended to be set much higher than T_D unless ISP prioritizes S reduction. When T_{pl} is set to be a small value, for example 2ms, E increases by 21.9% and D reduces by 18.1%, with a significant increase in S of 412.9%. Especially for ISPs with fast growing LTE user volume and con-

gested radio access network, DRX cycle in **RRC_CONNECTED** is not recommend to be set to too small values. We omit the simulation results for Short DRX cycle T_{ps} and Short DRX timer T_{is}, since reducing T_{ps} or increasing T_{is} has similar effect of decreasing T_{pl}, *i.e.,* resulting in a shorter average DRX cycle, and their impact is similar as in Figure 6.11.

Impact of DRX cycle in RRC_IDLE: Similarly to **RRC_CONNECTED**, we study the impact of DRX cycle in **RRC_IDLE**. Under this setting, signaling overhead S is defined to be the total on duration of DRX cycles in **RRC_IDLE**, during which control messages are exchanged between UE and eNB [33]. In Figure 6.12, the default setting 1.28s is shown as T_D. The impact of T_{pi} on E is not significant when T_{pi} is larger than T_D, with 0.7% energy saving when T_{pi} set to be 2.0s, which also brings in 21.3% increase in D and 31.7% reduction in S. When T_{pi} is set to be 100ms, the energy overhead is 26.2%, with 40.7% reduction in D and up to 1147.2% increase in S.

Similar to T_{pl}, T_{pi} causes significant signaling overhead when set to be too small. Default values for both T_{pl} and T_{pi} are reasonable settings and adjustment around the default configurations results in moderate impact on channel scheduling delay and negligible impact on radio energy.

In this section, we observe that LTE consumes up to 23 times the total energy of WiFi in the simulation, and the tail problem in 3G UMTS network is still a key contributor for the low energy efficiency for LTE. Similar to 3G network, T_{tail} remains a key parameter for LTE. In addition to the UE-participated approaches including proactive traffic shaping and fast dormancy proposed by previous study [20], we discuss a possible approach to solve the tail problem in LTE network, which dynamically adapts T_{tail} to traffic pattern at a per-UE base, being transparent to UE. Similar observations are shared by the other LTE parameters and we leave the feasibility study of such dynamic parameter configuration framework for LTE as future work.

	# of packets (million)	%
Total	131.49	100%
Screen on	72.50	55.13%
Screen off	47.14	35.84%
Unknown	11.85	9.02%

Table 6.4: Packet statistics of the UMICH data set.

Traffic type	Payload (GB) / %[a]	% of dl payload	# of packets ($\times 10^6$) / %[b]	% of dl packets	Avg dl packet payload (B)	Avg ul packet payload (B)
Screen-on	51.47 / 64.31%	96.31%	72.50 / 55.13%	60.71%	1126	67
Screen-off	21.82 / 27.26%	93.52%	47.14 / 35.84%	52.60%	823	63
Process name	**Off payload (GB) / %[c]**	**% of dl off payload[d]**	**# of off packets ($\times 10^6$) / %[e]**	**% of dl off packets[f]**	**Avg dl off packet payload (B)**	**Avg ul off packet payload (B)**
Genie Widget	1.76 / 72.21%	97.01%	3.80 / 73.16%	49.97%	901	28
Google Music	3.13 / 57.14%	99.91%	3.30 / 57.02%	68.60%	1384	3
Epicurious Recipe	1.65 / 70.05%	99.22%	2.69 / 69.29%	50.46%	1212	10
/system/bin/mediaserver	2.39 / 10.09%	99.77%	2.66 / 11.05%	66.95%	1342	6
android.process.media	2.35 / 28.42%	99.98%	2.37 / 29.06%	71.55%	1388	1
Skypekit[g]	0.04 / 25.54%	48.44%	2.07 / 46.73%	48.32%	22	22
Facebook	0.46 / 32.96%	86.13%	1.95 / 40.67%	42.55%	487	58
Yahoo! Sportacular	0.23 / 80.45%	83.53%	1.94 / 81.05%	41.98%	238	34
Gmail	0.39 / 46.00%	63.65%	1.33 / 54.46%	47.70%	400	208

[a] Payload refers to the total screen-on/off payload, and % is relative to the total payload of all traffic.
[b] % relative to the total number of packets of all traffic.
[c] Off payload refers to the screen-off payload of the specific application, and % is relative to the total payload of this application.
[d] % of downlink screen-off payload of the specific application relative to the total screen-off payload of that application.
[f] % of downlink screen-off packet count of the specific application relative to the total screen-off packet count of that application.
[g] Full process name: /data/data/com.skype.raider/files/skypekit, which is not the actual Skype application (com.skype.raider).

Table 6.5: Packet characteristics of screen-on/off traffic and top processes for screen-off traffic.

6.6 Characteristics of Screen-off Traffic

In this section, we present packet characteristics and burst analysis of screen-off traffic in the UMICH data set, which is described in Section 6.4.1. Then we compare screen-on and screen-off traffic and scrutinize the top applications generating screen-off traffic.

6.6.1 Packet Characteristics of Screen-off Traffic

Using the methodology described in Section 6.4.1, we classify all packets to be screen-on, screen-off or unknown. Table 6.4 lists the number of packets in different categories. Among the total 131.49 million packets, 55.12% of them are screen-on packets and 35.85% are screen-off packets, with 9.02% unknown. The possible reasons for unknown packets are multifold, including that voluntary users may have accidentally killed the data collector. For the unknown category, we conservatively choose not optimize it for either screen-on or

screen-off traffic optimization.

The top section of Table 6.5 lists the packet characteristics of both screen-on and screen-off traffic. Packet payload size refers to the size in bytes of an IP packet excluding the TCP/UDP and IP headers, and payload for a given process is the sum of the packet payload sizes of all packets corresponding to that process. We notice that screen-off traffic has far less packets (35.84% of total) than screen-on traffic (55.13% of total), much smaller total payload (27.26% for screen-off and 64.31% for screen-on traffic), and smaller average downlink packet payload size.

To understand individual process behavior within screen-off traffic, we scrutinize the top processes sorted by the number of screen-off packets, in the bottom section of Table 6.5. In the second column, titled "Off payload", we observe that some processes have most of their payload transferred during screen-off sessions, *e.g.,* Genie Widget, Epicurious Recipe, *etc.* Especially for Yahoo! Sportacular, 80.45% of all its payload is transferred when the screen is off. This is possibly due to the background behaviors of these processes, involving either periodically pulling updates from servers or traffic triggered by server-initiated push notifications. In terms of the % of downlink payload, compared to other processes, Skypekit, Gmail, Sportacular and Facebook have smaller proportions of their respective total downlink payloads associated with the screen-off states. These processes also have a smaller average downlink packet size ($<$500B). In contrast, a process like **`android.process.media`** has an average size of 1388B, indicating that most packets have a size of MTU (maximum transmission unit, $\sim$1500B). This shows that for screen-off traffic, application behavioral diversity still exists similar to that of screen-on traffic.

A group of processes share quite similar behavior patterns, including Google Music, **`/system/bin/mediaserver`** and
`android.process.media`, which have larger payload ($>$2GB), and $>$99% of the payload is downlink. Their average downlink packet sizes are close to the MTU and average

Traffic type	# of bursts	Avg[a] # of ul packets	Avg[a] # of dl packets	Avg[a] ul payload (B)	Avg[a] dl payload (KB)	Avg[a] burst length (sec)	Avg[a] **IBT** following (sec)
Screen-on	650.9K	43.75	67.62	2910.44	76.17	2.92	335.13
Screen-off	1910.9K	11.69	12.98	739.78	10.68	1.37	113.60
Process name	# of bursts	Avg[a] # of ul packets	Avg[a] # of dl packets	Avg[a] ul payload (B)	Avg[a] dl payload (KB)	Avg[a] burst length (sec)	Avg[a] **IBT** following (sec)
Genie Widget	6.0K	319.73	319.36	8852.48	287.88	17.87	3,892.87
Google Music	5.3K	195.69	427.56	505.54	591.92	4.53	5,111.50
Epicurious Recipe	63.2K	21.07	21.46	202.22	26.01	0.67	159.34
/system/bin/mediaserver	8.2K	106.44	215.53	669.82	289.35	5.01	14,451.70
android.process.media	1.4K	461.88	1,156.93	246.99	1,605.84	19.83	123,565.00
Skypekit	42.7K	25.08	23.46	555.38	0.52	1.93	832.79
Facebook	203.5K	5.49	4.07	318.83	1.98	0.86	547.23
Yahoo! Sportacular	133.8K	8.39	6.07	285.44	1.45	1.52	261.78
Gmail	105.5K	6.60	6.02	1375.30	2.41	1.17	2,002.60

[a] Each "avg" in this table stands for the average value per burst.

Table 6.6: Burst analysis of screen-on/off traffic and top processes for screen-off traffic.

uplink packet sizes are close to 0. Also, their ratio of downlink packets is close to 2/3. This is because TCP's delayed ACK would generate one uplink ACK packet for every two downlink data packets, resulting in a ratio of 2/3 of downlink packets. These observations suggest that Google Music is downloading large amount of data, as it can run in the background allowing users to listen to the music with the screen off. Similarly, although **`/system/bin/mediaserver`** and **`android.process.media`** are not actual applications, they are used by other applications, such as Pandora, to download contents for users while the screen is off. However, this group of processes does not necessarily have higher energy consumption compared with the remaining processes, and we explore this in more detail in latter sections.

6.6.2 Burst Analysis of Screen-off Traffic

Following the methodology in §6.4.2, Table 6.6 lists the results of the burst analysis for screen-off traffic, with that of screen-on traffic listed for comparison purposes.

We observe that screen-off traffic contains much more bursts than screen-on traffic, although the total number of packets for screen-off traffic is smaller. For screen-off traffic, bursts are smaller in terms of the number of downlink/uplink packets and payload. Especially, for average downlink payload per burst, screen-on traffic is 7 times that for screen-off traffic. In addition, the average burst length and the **`IBT`** following bursts for screen-off

traffic are both shorter than those of screen-on traffic. The above observations indicate that screen-off bursts are smaller in size and duration and appear more often — such behavior is likely to cause longer channel occupation time in the high energy RRC state and therefore incur significant battery usage.

By studying the screen-off burst behavior of individual processes, we classify them into two separate groups. The first group, which we call *Gathered* group, includes Genie Widget, Google Music, **`/system/bin/mediaserver`** and **`android.process.media`**. These processes have a small number of larger bursts in terms of the number of uplink/-downlink packets per burst and the average downlink payload. Notice that the uplink payload for these bursts is not necessarily large, since a small uplink payload of HTTP request can result in a large file download. The *Gathered* group also has longer bursts and longer trailing **`IBT`** in average, indicating a less frequent appearance.

The rest of the processes fall into the second group, called the *Scattered* group, which generate significantly more bursts and on average, these bursts contain less packets and smaller downlink payload. In addition, these bursts are shorter in duration and appear more frequently. A representative process from this group is Facebook, which includes over 200,000 bursts, and the major reason for this behavior of Facebook is the periodic keep-alive transfers [30].

Based on this comparison, we believe that it is both easy and important for mobile application developers to optimize their application behaviors during the screen-off stage, *e.g.,* for delay-insensitive traffic, they can batch the data into larger bursts, or even eliminate the screen-off data transfers if they are not necessary.

Traffic type	ΔE %[a]	Min $\lvert\Delta E\rvert$ %[c]	Max $\lvert\Delta E\rvert$ %[c]	ΔS %[a]	ΔD %[a]
Screen-on	-22.18%	3.78%	38.11%	-14.87%	-17.57%
Screen-off	-58.55%	12.39%	73.53%	-58.03%	-54.46%
Process name	ΔE %[b]	Min $\lvert\Delta E\rvert$ %[c]	Max $\lvert\Delta E\rvert$ %[c]	ΔS %[b]	ΔD %[b]
Genie Widget	-0.34%	0%	2.46%	0.11%	-0.81%
Google Music	-0.12%	0%	1.68%	-0.02%	-0.08%
Epicurious Recipe	-1.63%	0%	26.06%	-1.78%	-0.94%
/system/bin/mediaserver	-0.13%	0%	1.08%	0.02%	-0.16%
android.process.media	-0.08%	0.01%	0.52%	0.02%	-0.05%
Skypekit	-0.96%	0%	7.32%	-0.24%	-0.51%
Facebook	-5.25%	0%	34.68%	-5.82%	-4.15%
Yahoo! Sportacular	-3.01%	0%	20.04%	-1.88%	-1.44%
Gmail	-1.18%	0.04%	4.59%	-0.69%	-1.47%

[a] ΔE, ΔS and ΔD are calculated by removing all screen-on/off traffic from the original traces
[b] ΔE, ΔS and ΔD are calculated by removing the screen-off traffic of one process from the original traces
[c] Min and max refer to the minimum and maximum energy saving $|\Delta E|$ across all users, respectively.

Table 6.7: Radio resource and energy impact of screen-on/off traffic and top processes for screen-off traffic.

6.7 Radio Resource, Energy Impact and Optimization of Screen-off Traffic

Using the network and power model simulation (Section 6.4.3), we now evaluate the radio resource and energy impact of screen-off traffic and evaluate some optimization approaches.

6.7.1 Radio Resource and Energy Impact of Screen-off Traffic

Table 6.7 presents highlights of the simulation results using the LTE network and power model derived previously. ΔE, ΔS and ΔD represents the change of network energy, signaling overhead and channel scheduling delay after removing the traffic of an application or type, and a negative value indicates a reduction. The results indicate that, compared to screen-on traffic, screen-off traffic clearly has larger impact on the network energy E, as well as S and D. For example, removing all screen-off traffic reduces the total network energy by 58.55%, and for one user, this reduction is as high as 73.53%.

Comparing the *Gathered* group and *Scattered* group discussed in §6.6.2, the former has very small impact on E, S and D, while the later has a substantial impact. This is because,

Process name	Optimization	Settings	ΔE^a	ΔS^a	ΔD^a
All applications	Fast dormancy	$T_{i,on}$ [b] = 8s, $T_{i,off}$ [b] = 8s	-16.39%	16.95%	13.14%
		$T_{i,on}$ = 4s, $T_{i,off}$ = 8s	-20.60%	28.29%	21.26%
		$T_{i,on}$ = 8s, $T_{i,off}$ = 4s	-34.44%	47.04%	35.21%
		$T_{i,on}$ = 4s, $T_{i,off}$ = 4s	-38.66%	58.38%	43.31%
	Batching	Only for screen-off, α = 50s, β = 10s	-22.33%	-6.24%	-11.27%
		Only for screen-off, α = 50s, β = 5s	-27.15%	-6.24%	-10.67%
		Only for screen-off, α = 100s, β = 10s	-36.72%	-30.00%	-33.43%
		Only for screen-off, α = 100s, β = 5s	-40.79%	-30.00%	-34.25%
	Fast dormancy + Batching	$T_{i,on}$ = 8s, $T_{i,off}$ = 4s, batching only for screen-off traffic, α = 100s, β = 5s	-60.92%	-25.33%	-30.59%
Facebook[c]	Fast dormancy + Batching	$T_{i,on}$ = 8s, $T_{i,off}$ = 4s, batching only for screen-off traffic, α = 100s, β = 5s	-60.19%	-36.27%	-34.93%
Google Music[c]	Fast dormancy + Batching	$T_{i,on}$ = 8s, $T_{i,off}$ = 4s, batching only for screen-off traffic, α = 100s, β = 5s	-57.30%	7.12%	-21.11%

[a] $\Delta E, S, D$ are relative to the E, S, D of all traffic for the specific application.
[b] $T_{i,on}$ is the inactivity threshold of fast dormancy for screen-on traffic, and $T_{i,off}$ is for screen-off traffic.
[c] For these two application rows, we consider the traffic of *only* one specific application, excluding that from other applications.

Table 6.8: Traffic optimization with fast dormancy and batching.

for the *Scattered* group, a large number of small bursts could result in a large number of RRC tails if the **IBT**s among these bursts are larger than the tail time and a long channel occupation time otherwise. For example, although the Facebook process does not generate the most screen-off traffic, it has the largest energy impact among all processes, *i.e.,* 5.25% of the total network energy can be saved by only removing Facebook's screen-off traffic, and for some users this number could be as high as 34.68%.

6.7.2 Traffic Optimization

Based on the above analysis, we find that screen-off traffic has a clearly different pattern compared to screen-on traffic, and accounts for a huge proportion of the UE network energy E, signaling overhead S and channel scheduling delay D. Intuitively, since screen-off traffic is more tolerant to delays as users are typically not actively interacting with the device, and hence more amenable to more aggressive optimization efforts, a traffic optimization approach that is appropriately tuned to the two different traffic categories would yield significant efficiencies. To verify this intuition, we study two common optimization techniques, fast dormancy and batching:

(i) *Fast dormancy (FD)* [31, 32] is a mechanism in 3G networks for reducing the amount of tail time incurred by a device by quickly demoting it to a low energy RRC state with-

out waiting for the tail timer to expire. In our simulations, we explore fast dormancy in an LTE setting, motivated by the fact that LTE, like 3G, also suffers from a serious tail problem [12]. Our fast dormancy-based optimization works as follows: when the UE has not observed any network activity for some idle time T_i, it sends a special RRC message to the network to make the allocated radio resource be released earlier, instead of occupying it for the whole RRC tail. After the radio resource is released, the UE switches to the low power idle state (**RRC_IDLE** for LTE networks), saving energy. The setting of T_i is important for balancing the tradeoffs among UE energy saving (ΔE), signaling overhead (ΔS) and channel scheduling delay (ΔD), *i.e.,* a smaller T_i would results in a larger ΔE at the cost of larger ΔS and ΔD, and *vice versa*.

(ii) *Batching* is a widely used traffic shaping technique, which has been discussed in previous studies [21, 30]. In this study, batching uses two parameters, source window size α (seconds) and target window size β (seconds), and $\alpha > \beta$. For each α seconds long time window $[t, t+\alpha]$, packets within $[t, t+\alpha-\beta]$ are delayed and batched with those inside $[t+\alpha-\beta, t+\alpha]$. Notice that if α/β is too large, the limited bandwidth of the cellular network could become the bottleneck, making the batching impossible. So in this study, we make sure that our choice of α and β results in acceptable bandwidth usage for LTE networks.

We first compare applying fast dormancy to screen-on and screen-off traffic with a separate T_i settings. The default setting of the **RRC_CONNECTED** inactivity timer is 11.58 seconds for a major LTE ISP [12]. In Figure 6.15, we vary the setting of T_i from 10 seconds to 2 seconds with a step-size of 2 seconds (from left to right in Figure 6.15) and calculate the $\Delta(E, S, D)$ relative to when fast dormancy is not applied. We observe that for the same T_i setting, there is a higher energy saving ΔE and a lower ΔS and ΔD for screen-off traffic, compared with screen-on traffic, and these gaps are larger when T_i is smaller, *i.e.,* applying fast dormancy more aggressively. So in order to achieve the same energy saving for screen-off and screen-on traffic, the T_i should be set to a much smaller value for screen-

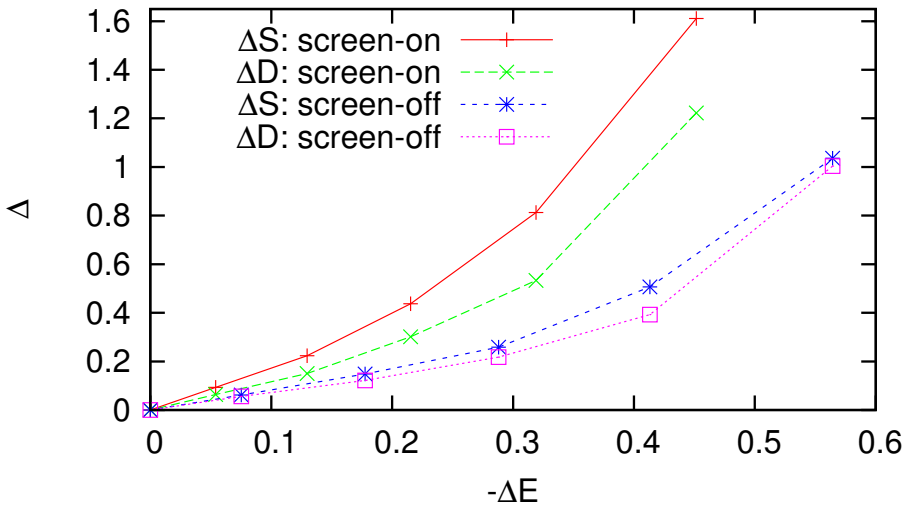


Figure 6.15: Effectiveness comparison of fast dormancy.

on traffic, incurring much larger ΔS and ΔD. However, the responsiveness requirement when the screen is on is actually higher, hence a higher ΔD for screen-on traffic is not acceptable. A better strategy would be being more aggressive (a smaller T_i) for screen-off traffic, which produces significant energy savings, and more conservative (a relatively larger T_i) for screen-on traffic, which limits the negative impact on user experience, though with less energy savings. For example, when $T_i = 8s$, there is 42% energy savings with 52% ΔS and 40% ΔD for screen-off traffic, and when $T_i = 4s$, for screen-on traffic, though the energy saving is small (24%), ΔS (43%) and ΔD (30%) are also limited. With these two fast dormancy settings, for the whole traffic, we can achieve 34% energy saving, with 47.04% ΔS and 35.21% ΔD. This is a better tradeoff than one single T_i setting for both screen-on and screen-off traffic.

Besides fast dormancy, batching [30, 21] is also proposed for optimizing mobile traffic. In Table 6.8, we compare fast dormancy and batching under different settings for all applications together, and also individually for two applications, Facebook representing the *Scattered* group and Google Music representing the *Gathered* group. For fast dormancy, with reduced E, there is increased S, D, while for batching, all E, S, D are decreased. This is because for fast dormancy, since UE demotes to **`RRC_IDLE`** earlier, there would be

more promotions resulting in increased S, D, while for batching, since the traffic pattern is altered, scattered packets are gathered into groups and hence there are less promotions. Notice that the metric D here does not include the delay of packets incurred by batching. In this work, we only focus on batching for screen-off traffic, since any delay for screen-on traffic is likely to be more perceptible to users.

In Table 6.8, when applying fast dormancy to all applications, we set a different T_i for screen-on/off traffic, *i.e.,* $T_{i,on}$ and $T_{i,off}$. For simplicity, let $<a, b>$ stands for the case when $T_{i,on}$ = a seconds and $T_{i,off}$ = b seconds. Based on Figure 6.15, we empirically select two values for $T_{i,on}$ and $T_{i,off}$, 4s as an aggressive setting and 8s as a conservative setting. Compared with $<8, 8>$, reducing $T_{i,on}$ to 4s, *i.e.,* $<4, 8>$, only saves 4.21% energy additionally. And $<8, 4>$ has 11.34% reduction in S and 8.10% reduction in D, with only 4.22% less energy saving, compared with $<4, 4>$. This verifies that a different setting for $T_{i,on}$ and $T_{i,off}$ balances the tradeoff of saving energy and reducing overhead, and we select $<8, 4>$ as a reasonable setting. Notice that there are other possible parameter settings representing different aggressiveness with different energy saving and overhead, which may be more appropriate for different settings.

In Table 6.8, batching applied to the screen-off traffic is able to reduce all E, S, and D. Notice that we do not apply batching for screen-on traffic since it may affect the user experience, *e.g.,* when the user is waiting for a response at real time. We observe that most of the screen-off traffic (in terms of the energy impact) is less delay-sensitive, *e.g.,* push notification, since user interaction is not involved. However, there are also some exceptions for screen-off applications which requires real-time data transfer, *e.g.,* when the user is making a VoIP call with the screen off. Ideally, traffic from these delay-sensitive applications should not be batched even during screen-off stage. In this study, we do not attempt to completely solve this problem, instead, we show an upper bound of the benefit by batching all screen-off traffic. In reality, we need to prioritize delay-sensitive traffic during screen-off stage, and we leave it to future study. The choice of α and β values is

limited by the available bandwidth for 3G/4G networks. Comparing among the empirically selected candidate settings in Table 6.8, α = 100s and β = 5s is a better setting, which saves up to 40.79% energy, with a 30.00% reduction in S and 34.25% in D. Notice that the α and β settings studied are just example settings that work well in practice. The goal is to demonstrate the benefit of batching for screen-off traffic and the selection of optimal settings is left to future work.

Then we evaluate applying fast dormancy and batching jointly for all applications and for two sample applications, with the settings specified in Table 6.8. For all applications, there is a total network energy saving of 60.19%, with 25.33% reduction in S and 30.59% reduction in D. Facebook has similar energy saving, and due to its more "scattered" traffic pattern for screen-off traffic, the batching optimization results in even more reduction for S and D. However, for Google Music, whose traffic is already "gathered" as large bursts, the impact of fast dormancy is more obvious than batching, hence the reduction for D is smaller and there is even an increase in S, unlike the other two scenarios.

6.7.3 Discussions

We took a first step towards understanding the impact of screen status on cellular application traffic behavior. Our evaluations in the context of LTE cellular networks, show that although the number of packets and total payload for screen-off traffic are much smaller than that for screen-on traffic, the former accounts for a disproportionate majority (58.55%) of the total network energy consumed by a device. Exploration of resource optimization techniques like fast dormancy and batching indicate that the strategy of optimizing the screen-off traffic more aggressively than screen-on traffic can realize substantial resource savings, without adversely impacting user experience.

We are pursuing this research further, first to explore screen-off traffic in greater detail and explore optimization strategies tailored to the potentially different delay-requirements of subsets of that traffic. Second, in this paper, our optimization strategies imposed the

strict constraint that screen-on traffic should not suffer any additional delays that come with traffic shaping approaches. In reality, some limited delay jitter would be tolerable depending on the application and traffic semantics - this is be an additional source of resource optimization beyond the savings we have shown in this paper. Finally, logistics permitting, it would be nice to extend the study to a larger user group.

CHAPTER VII

RadioProphet: Optimizing Smartphone Energy and Radio Resource in Cellular Networks

Achieving energy efficiency for mobile devices when connected to cellular networks without incurring excessive network signaling overhead, even despite diverse application and user behavior, still remains a rather difficult and yet important challenge to tackle. Energy use due to network access, particularly cellular networks, is becoming increasingly dominant due to numerous network-based smartphone applications. In many cases, achieving network energy savings must reside on the mobile device's OS to effectively and centrally manage the data scheduling decisions transparent to applications and with minimal changes to the network.

The key mechanism that determines the energy consumed by cellular network interface is the radio resource control (RRC) state machine [20] pre-defined by carriers (Figure 2.1) that governs when radio resources are acquired and released. Previous studies [21, 20, 22, 12] have shown that the origins of low resource efficiency comes from the way radio resources are *released*. To avoid unnecessary state transitions, radio resources are only released after an idle time (also known as the "tail time") controlled by a statically configured inactivity timer. During the tail time, radio energy is essentially wasted. Values as large as 11.6 seconds are configured [12] in current networks, contributing to about half of the total radio energy on user handsets (UEs) spent in idle times for common usage

scenarios.

Without knowing when network traffic will occur, large tail timer settings are essentially a conservative way to ensure low signaling overhead due to state transitions, as signaling is known to be a bottleneck for cellular networks. Furthermore, they also help minimize performance impact experienced by users caused by state promotion delays incurred whenever radio resource is acquired. Given that application and user behavior are not random, using a statically configured inactivity timer is clearly suboptimal. Smaller static timer values would help reduce radio energy, but is not an option due to the risk of overloading cellular networks caused by signaling load increase.

An attractive alternative is to configure the timer dynamically — adaptively performing radio resource release either signaled by the UE or triggered by the network itself by monitoring the UE traffic, accommodating different traffic patterns, improving the overall resource efficiency. But the key challenge is determining *when* to release resources, which essentially comes down to accurate and efficient prediction of the idle time period. Clearly, the best time to do so is when the UE is about to experience a long idle time period, otherwise the incurred resource allocation overhead (*i.e.,* signaling load) is wasteful due to unnecessary radio state transitions, and the achieved resource savings are very small. Therefore, accurate and efficient prediction of the idle time period is a critical prerequisite for dynamic timer schemes.

This paper proposes RadioProphet (RP), a practical system that makes dynamic decisions to deallocate radio resources based on accurate and efficient prediction of network idle times. It makes the following contributions.

First, RP utilizes standard online machine learning (ML) algorithms to accurately predict the network idle time, and performs resource deallocation only when the idle time is sufficiently long. We explored various ML algorithms and prediction models with tunable parameters, with the main contribution of finding robust and easy-to-measure features (not relying on packet payload), whose complex interaction with the network idle time can

be automatically discovered by the ML algorithms. The model is validated using seven-month-long traces collected from real users (§7.4).

Second, we implement the full RP system on the Android platform, and it incurs negligible energy and CPU overhead on our test device demonstrating its practicality. To reduce the runtime overhead, RP strategically performs *binary* prediction (*i.e.,* whether the idle time is short or long) at the granularity of a traffic burst consisting of a packet train sent or received in a batch. Compared to fine-grained prediction of the precise value of packet inter-arrival time, our proposed approach is much more efficient while yielding similar optimization results.

Third, we overcome critical limitations of previously proposed approaches, *i.e.,* RadioJockey [81] and MakeIdle / MakeActive [82] are only applicable to background applications without user interaction, with the ideal usage scenario of RadioJockey for a single application only. With multiple concurrent applications, it suffers from low prediction accuracy with increased overhead. By design, RP applies to both foreground and background traffic, maximizing energy savings. Since its prediction is based on the aggregate traffic of all applications, RP incurs no additional performance degradation nor overhead for supporting concurrent apps.

Fourth, we evaluate RP using real-world smartphone traces. The overall prediction accuracy is 85.9% for all traffic. RP achieves radio energy saving by 59.1%, at the cost of 91.0% additional signaling overhead in LTE networks, significantly outperforming previous proposals. In order to achieve such energy saving (59%), the additional signaling overheads incurred by MakeIdle and naïve fast dormancy [31, 32] are as high as 305% and 215%, respectively. The maximal energy saving achieved by RadioJockey is 27% since it is only applicable to background traffic.

In this chapter, we present the design of RP in Section 7.1. Then we explore feature selection in Section 7.2. With the methodology discussed in Section 7.3, we evaluate the performance of RP in Section 7.4, before summarizing in Section 7.5.

7.1 The Design of the RadioProphet System

As described in Chapter II, the static tail times are the root cause of low resource efficiency in cellular networks. Previous work points out that around 50% energy is incurred by the long high-power RRC tail for 3G [20] and 4G networks [12]. Our proposed RadioProphet (RP) system leverages the fast dormancy feature to *dynamically and intelligently determine when to release radio resources*.

Challenge 1: the tradeoff between resource saving and signaling load. Clearly, the best time to perform resource deallocation is when the UE is going to experience a long idle time period t. Specifically, if t is longer than the tail time, deallocating resources immediately after a data transfer saves resources without any penalty of signaling load (*i.e.,* state promotions). If t is shorter than the tail time, performing deallocation ahead of time trades off resource saving with signaling load, because doing so will incur an additional state promotion. Such a critical tradeoff presents the key challenge of *predicting the idle time* between data transfers so that fast dormancy is only invoked when the idle time is sufficiently long.

Challenge 2: the tradeoff between prediction accuracy and system performance. RP is a service running on the UE with limited computational capabilities and even more importantly, limited battery life, so we need to minimize the runtime overhead without sacrificing much of the prediction accuracy. Otherwise the resource utilized by RP itself may overpower its benefits.

Challenge 3: the requirement to handle both foreground traffic and background traffic. Idle time prediction is particularly difficult for applications involving user interactions. Previous systems, such as RadioJockey [81] and MakeActive [82], simply avoid this by only handling traffic generated by applications running in the background. To maximize the energy saving, the proposed system should be able to handle foreground traffic well, in addition to background traffic.

To address **Challenge 1**, we established a novel machine-learning-based framework

for idle time prediction. We explored a wide range of ML algorithms with tunable parameters, and evaluated their effectiveness and efficiency. More importantly, our primary focus is addressing the hard problem of selecting discriminating features that are relevant to our specific problem. We find that strategically using a few simple features (*e.g.,* packet direction, size, and application name, *etc.*) leads to a high prediction accuracy of 85.9%.

To address **Challenge 2**, we perform *binary* prediction at the granularity of a traffic *burst* consisting of a train of packets sent or received in a batch. In other words, we find that the knowledge of whether the inter-burst time (**IBT**) is short or long (determined by a threshold) is already accurate enough for guiding the resource deallocation. Such an approach is much more efficient while yielding similar optimization results compared to more fine-grained and more expensive prediction for the precise value of packet inter-arrival times.

Further, by design, since the prediction of RP is based on the aggregate traffic of all applications, it incurs no additional performance degradation nor overhead for supporting concurrent apps. In contrast, previous systems such as RadioJockey have the ideal usage case for a single app.

To address **Challenge 3**, we choose robust features for idle time prediction for all traffic. We also customize the prediction settings for different types of traffic, either background traffic without user interaction or foreground traffic likely to be triggered by users. We use smartphone screen status as a hint to indicate whether a user is interacting with the device, *i.e.,* screen-on traffic belongs to foreground applications. We then apply different settings for prediction based on the screen status and evaluate RP with all traffic.

We illustrate the design of RadioProphet (RP) in Figure 7.1. In summary, a data collector collects packet traces (only headers are required) and some other auxiliary information, *e.g.,* process association for all packets. The data collected is fed into the **IBT** prediction framework, which trains models to predict the **IBT** for the current burst. Then, based on the prediction result, the fast dormancy scheduler makes decision on whether to invoke fast

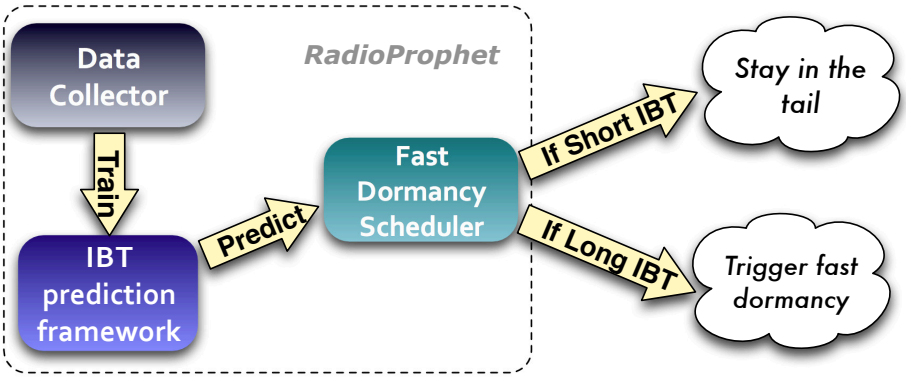


Figure 7.1: Working flow of RadioProphet (RP).

dormancy.

For **IBT** prediction, we define the traffic model as follows. For each user, the traffic trace is a sequence of network packets, $P_i (1 \leq i \leq n)$. P_i could either be a downlink packet or an uplink one. Assume the timestamp of P_i is t_i, $t_i \leq t_j$ for any $i < j$. Using a burst threshold **BT**, the packets are divided into ***bursts***, *i.e.,* $\{P_p, P_{p+1}, \cdots, P_q\}$ belongs to a burst B, if and only if:

1. $t_{k+1} - t_k \leq$ **BT** for any $k \in \{p, \cdots, q-1\}$
2. $t_{q+1} - t_q >$ **BT**
3. $t_p - t_{p-1} >$ **BT**

We define the inter-burst time **IBT** for burst B to be the time gap following this burst, *i.e.,* $t_{q+1} - t_q$. We also use a short **IBT** threshold **SBT** to classify an **IBT**, *i.e.,* if **IBT** $\leq$ **SBT**, it is *short*, otherwise, it is *long*.

The **IBT** prediction framework trains the prediction model based on historical traffic information collected by the data collector. The information collected is an array of bursts $\{B_1, \cdots, B_m\}$. Each B_i is a vector

$$B_i : \quad < f_1, f_2, \cdots, f_t, ibt_i >$$

where $\{f_1, \cdots, f_t\}$ is the list of features of B_i and ibt_i is the **IBT** following burst B_i. With the prediction model, RP monitors the UE traffic in real-time and whenever there is an idle time of **BT**, *i.e.,* the last packet's arrival time was **BT** time ago, the prediction process starts. The feature vector of the current burst $\{f_1, \cdots, f_t\}$ is generated and fed to the prediction framework, which predicts whether the **IBT** following the current burst is short or long. If short, as shown in Figure 7.1, no change is made and UE stays in the tail, since the prediction framework suggests that a packet would appear soon. Otherwise, the prediction framework triggers fast dormancy to save energy.

7.2 Feature Selection

In this section, we study the traffic burst characteristics in the UMICH data set. Then we look at a few burst features for **IBT** prediction.

7.2.1 The UMICH Dataset

The measurement data used in this study, which we call the UMICH data set, is collected from 20 smartphone users for five months. The participants consisted of undergraduate and graduate students from University of Michigan. The 20 participants are given Motorola Atrix (11 of them) or Samsung Galaxy S smartphones (9 of them) with unlimited voice, text and data plans.

We develop custom data collection software and deploy it on the 20 smartphones. It continuously runs in the background and collects three types of data. *(1)* Full packet traces in `tcpdump` format including both headers and payload. The detailed packet traces allows us to keep detailed track of traffic patterns generated by each individual user for a long period. *(2)* The process name responsible for sending or receiving each packet. *(3)* User input events such as pressing a button and tapping the screen. This is collected by reading `/dev/input/event*` in the Android system. We also build a data uploader that uploads the data to our server when the phone is idle as indicated by low network throughput.

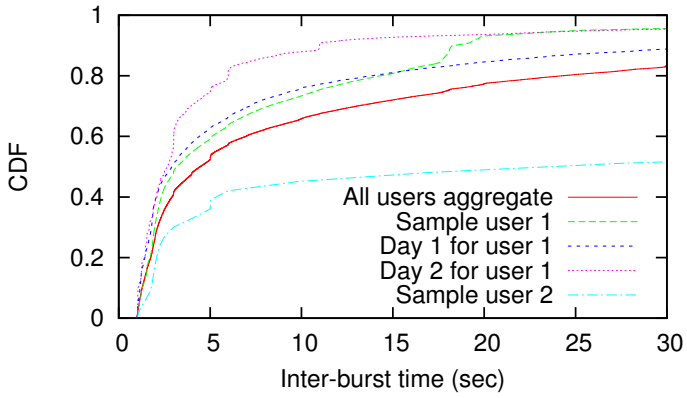


Figure 7.2: CDF of IBT for sampled/all users.

We deployed the user study in May 2011. From May 9 2011 to December 5 2011, we collect 152GB data (150 GB `tcpdump` traces). Although both cellular and Wi-Fi traces were collected, in this study, we consistently use 3G traces only, which contribute 57.8% of the total traffic volume, in the UMICH dataset for further analysis. Comparing with the Wi-Fi traces, 3G traces better represent the user and application behavioral patterns in cellular networks.

7.2.2 Characteristics of Bursts

For burst characterization, we first need to pick a value for **BT**. Obviously, **BT** should be larger than typical RTTs in cellular networks, which is around hundreds of milliseconds [1, 12], so as not to break a single continuous TCP flow into separate bursts. In the meanwhile, **BT** should not be set to be too large; otherwise, there could be a large gap inside a burst, during which, fast dormancy could have been triggered to save energy. We start our analysis with **BT** $= 1s$, resulting in 5.44 million bursts in total, and later we would explore how different **BT** values affect our results.

By analyzing all 3G traces in the UMICH data set, we plot the CDF of **IBT**s for all users (**IBT**s are calculated separately for each user before aggregate analysis) and two sample users in Figure 7.2. The median of **IBT** for all users is 4.46 seconds, and 68.24% of **IBT**s are smaller than the tail timer T_{tail} in our studied network.

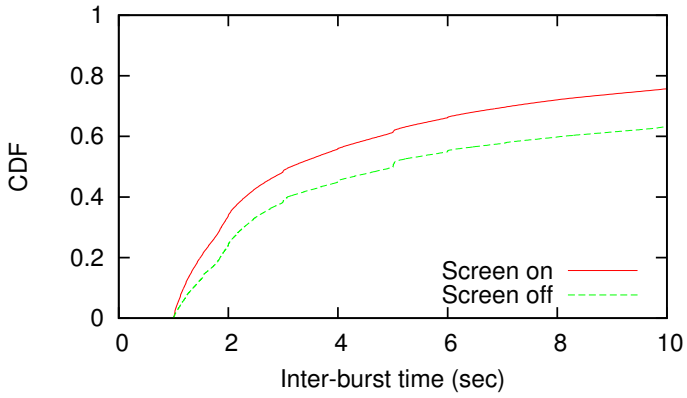


Figure 7.3: CDF of IBT for different screen status.

In Figure 7.2, we also compare the aggregate curve of all users with those of two sample users. The difference for **IBT** distribution is mainly due to the user preference over different applications, *e.g.,* sample user 1's curve has a big jump around 20 seconds and this is verified to be correlated with the heavy use of the Facebook application as shown in Figure 7.9. Therefore, RP predicts **IBT** using a per-user model instead of a global model for all users. Even for the same user, *i.e.,* user 1, at day 1 and day 2, which are months apart, we also observe a clear difference. We observe that user 1 uses a quite different list of applications for these two days, though not completely different. Even for the same application, different versions may also result in traffic pattern changes. So we think it is not practical to build a static model for each user. Instead, RP uses a dynamic model based on the most recent traffic history for each user.

Choosing a proper value for **SBT** is challenging, *i.e.,* a large **SBT** reduces the chance of RP for saving energy and a small **SBT** would result in excessive signaling load, since RP would trigger fast dormancy for long **IBT**s. Previous work [83] indicates that traffic pattern is different between screen-on state and screen-off state. In Figure 7.3, we also observe that screen-off traffic has higher **IBT**s than screen-on traffic. The median **IBT** for screen-off and screen-on traffic is 5.00s and 3.14s, respectively. This motivates us to select different **SBT** values based on the different screen states, *i.e.,* we could use an more aggressive (smaller) value of **SBT** to trigger more fast dormancy for screen-off traffic.

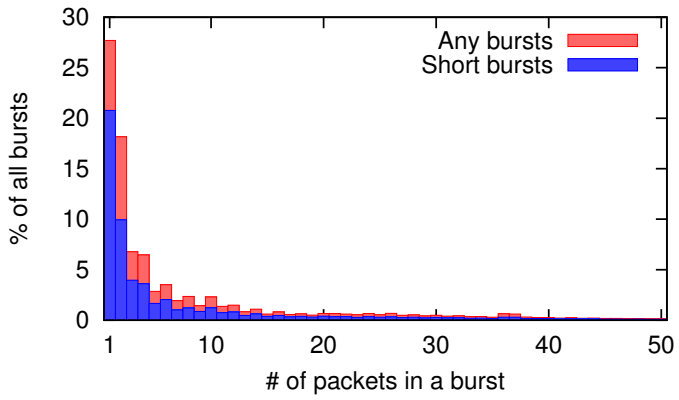


Figure 7.4: Distribution of bursts grouped by # of packets.

Figure 7.4 shows the distribution of the number of packets of a burst. We observe that the distribution is heavy-tailed. In particular, 27.69% of all bursts only contain 1 single packet, and 75.00% of these 1-packet bursts are short bursts. About 52.63% of all bursts consist of no more than 3 packets and 73.47% bursts contain no more than 10 packets, indicating that small bursts dominate the user traffic.

7.2.3 Selecting Features for Burst Classification

To predict whether an **IBT** is larger or smaller than **SBT**, we naturally choose to look at the features of the burst preceding the **IBT**. Based on our observation, the correlation between **IBT** and the bursts further beyond the preceding burst is small, hence we do not use those bursts for training features. For the preceding burst, we look at the detailed features of the last three packets. The choice of three packets is because in most cases, bursts are small, and even for large bursts, we can tell the nature of a burst based on the last three packets, *e.g.,* TCP three-way handshake for connection establishment and termination. For each of the last three packets, we only look at header information, not the payload, since many of these packet either do not have any payload, or use encryption. The features of the last three packets [1] used for training are listed in Table 7.2.3 and we study some of them as follows. These features are selected empirically that are most relevant to **IBT**.

[1] If a burst contains less than three packets, all features for the missing packet(s) have a value of 0.

Name	Symbol	Description
Packet direction	*Dir(i)**	Downlink or uplink
Server port number	*Port(i)*	The remote port number
Total packet length	*Len(i)*	Including headers
Protocol	*Prot(i)*	Protocol field in the IP header
TCP flags	*Flag(i)*	TCP flags field in the TCP header and 0 if not TCP
Application ID	*AppID(i)*	The global application ID which generates this packet

* *Dir(i)* (i = 1, 2 or 3) refers to the i-th to last packet, *e.g.,* when i = 1, it refers to the last packet. Similar rule applies to all other features.

Table 7.1: Features for the last three packets of a burst.

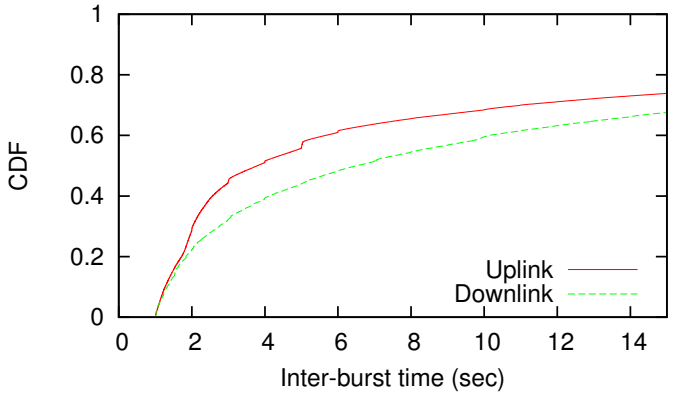


Figure 7.5: CDF of **IBT** v.s. direction of the last packet.

In Figure 7.5, CDF of **IBT** for the two possible directions of the last packet is shown. In general, **IBT** following a burst whose last packet is downlink is larger than that for uplink, *e.g.,* the median **IBT** is 6.49 seconds for downlink and 3.79 seconds for uplink.

Figure 7.6 shows the CDF of **IBT**s of all users for the top 6 ports ordered from top to bottom in the legend, *i.e.,* port 80 is the top 1 port and port 993 ranks 6-th. **IBT** distribution for different ports have clear differences, especially for port 53. By further investigation on the traffic traces, we know that the sudden jump for port 53 at **IBT** $= 5$ seconds is related with the DNS request retransmission behavior, *i.e.,* when UE fails to get DNS response within 5 seconds, either due to poor signal strength or slow DNS response, it resends the DNS request and the initial DNS request itself forms a burst. We also observe that the

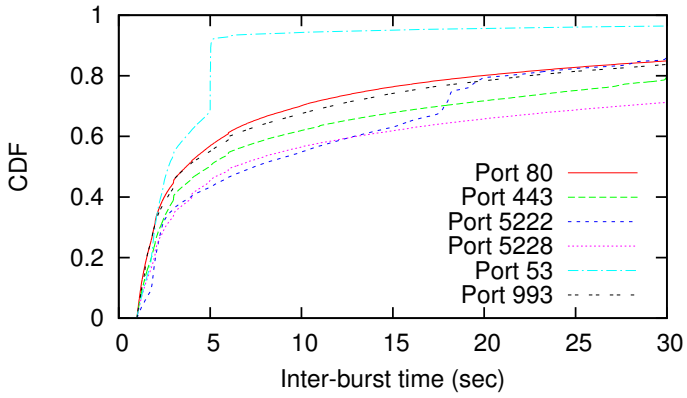


Figure 7.6: CDF of **IBT** v.s. port of the last packet.

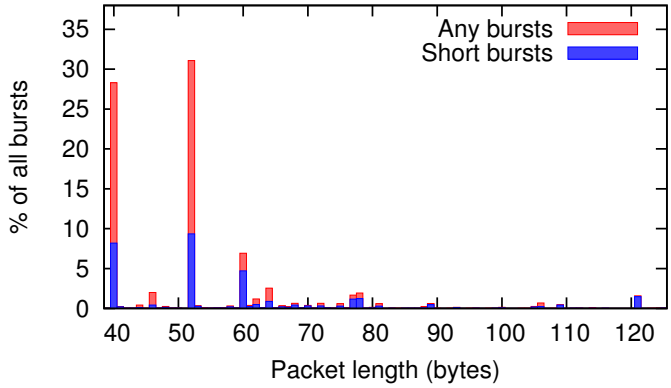


Figure 7.7: Distribution of bursts grouped by packet length of the last packet.

jump for port 5222 around 20 seconds corresponds to the periodic keep-alive messages for Facebook app.

Figure 7.7 summarizes the number of bursts for different packet length of the last packet. Packet length outside the plotted range has much fewer bursts. As expected, we observe that most bursts ends with small packets, *i.e.,* 84.59% of all bursts have the last packet $\leq$ 100 bytes, since large packet typically indicates being in the middle of data transfer, thus not the end of a burst. For some other values, *e.g.,* 121 bytes, 93.04% bursts are short bursts indicating high correlation. Machine learning algorithms could learn such rules and make prediction. Although there is less correlation between **IBT** and packet length for the two major spikes at 40 bytes (71.05% short bursts) and 52 bytes (70.00% short bursts), machine learning algorithms would consider other features for prediction automatically for higher

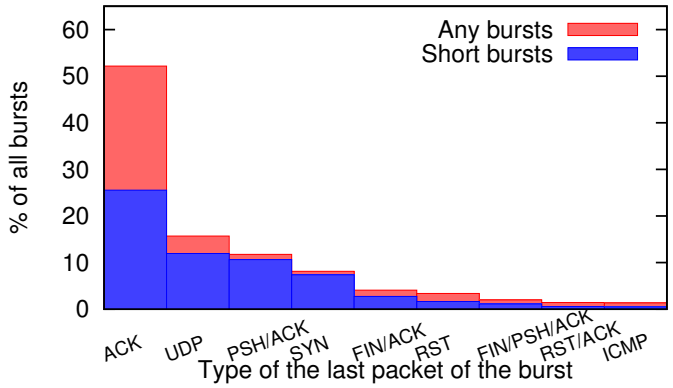


Figure 7.8: Distribution of **IBT** grouped by the type of the last packet.

accuracy.

In Figure 7.8, the number of bursts for different *Flag(1)* is presented, sorted in a descending order from left to right. Specifically, for *Flag(1) == 0*, such bursts fall into two groups, UDP packets and ICMP packets. Besides UDP packets, we also observe strong correlation between *Flag(1)* and short **IBT**s for SYN and PUSH-ACK packets, with 90.44% and 90.72% confidence, respectively.

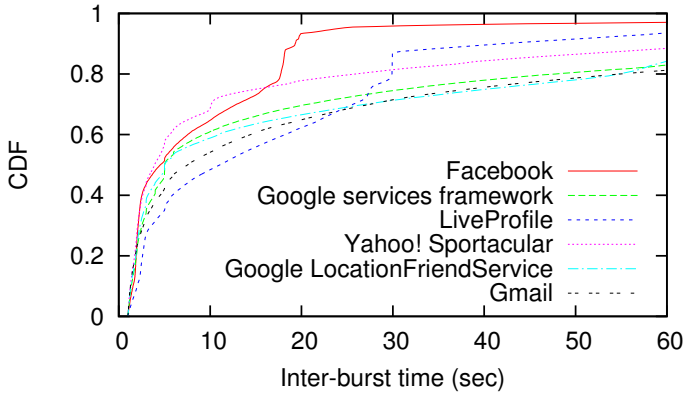


Figure 7.9: CDF of **IBT** v.s. the application generating the last packet.

The last feature we look at for each packet is the application ID. Notice that this feature is not directly obtainable from packet traces and only available at the client side [2]. In Figure 7.9, the legend shows the sorted top list of applications contributing to most bursts

[2] At the network side, using the **User-Agent** field in each TCP flow, the application name may be inferred, however, it is not 100% accurate and for non-HTTP traffic, no such information is available.

with Facebook as the top 1 app. The application-wise differences of **IBT** is obvious which necessities the choice of this feature. We also observe that for some applications, periodic transfer behavior contributes to clear jumps in **IBT** distribution, *e.g.,* Faceboook and LiveProfile.

7.3 Evaluation Methodology

We present the metrics and methodology for evaluating RP in this section.

7.3.1 Evaluation Metrics

Although there are some differences between 3G and 4G RRC mechanisms as described in Chapter II, the overall tail mechanisms are similar:

1. There is a fixed promotion overhead, which incurs both energy and signaling overhead.

2. There is a fixed tail which incurs UE energy overhead, but useful for preventing excessive promotions.

With the help of accurate **IBT** predictions, RP can make decisions on fast dormancy for each burst in a similar way for both 3G and 4G networks, although fast dormancy was initially proposed for 3G networks. In this study, we select the 4G LTE network as a representative for simplicity and the conclusions we draw if we use the 3G network and power model should only have the quantitative differences from the 4G model, rather than qualitative ones.

We consistently use the same terms described below. Specifically, assuming a UE is used for some period t. We define the total radio energy usage of the UE in t to be E. Notice that E is one of the most significant components for UE's total energy usage, along with screen energy and CPU energy, *etc.* [26]. We then define the total duration of UE

being in **RRC_CONNECTED** to be R, to quantify the channel occupation time, since radio resource is only allocated to UE in **RRC_CONNECTED** in the LTE network. E and R are approximately proportional, because the power level in **RRC_CONNECTED** is much higher than that in **RRC_IDLE**, and the radio energy E energy is dominated by that in **RRC_CONNECTED**: $E \approx PR$, where P is the average power level in **RRC_CONNECTED**, with small variation in most cases. For the rest of the paper, we only study E, and assume the channel occupation time R is proportional to E.

To quantify the state promotion overhead, we define signaling overhead S as the number of state promotions, since one promotion corresponds to fixed number of signaling messages [81]. To save the radio energy E, UE would spend less time in **RRC_CONNECTED** and hence there is a higher probability for RRC state promotions, resulting in increased S. The design goal of our system is to reduce E, while minimizing the increase of S. A state promotion also incurs a fixed delay (T_{pro}). Since the total of such delays is proportional to S, we do not use a separate metric to quantify the introduced delay by promotions. In addition, a state promotion requires fixed energy ($T_{pro}P_{pro}$), which is taken into consideration in calculating E.

Assume a specific user trace is fed to the LTE network and power model simulator with all default settings, the values for E and S are E_d and S_d, respectively (the subscript $_d$ stands for **d**efault). Then assume after the energy saving scheme is applied and the resulting E, S values are E', S'. We define

$$\Delta(T) = \frac{T' - T_d}{T_d}$$

where $T = E, S$. $\Delta(S)$ is positive, while $\Delta(E)$ is always negative. For simplicity, we redefine

$$\Delta(E) = \frac{|E' - E_d|}{E_d}$$

making it positive, representing the ratio of energy saved. In sum, the goal is to maximize $\Delta(E)$ while minimizing $\Delta(S)$.

7.3.2 RRC and Radio Power Simulator

We build a 4G LTE RRC state machine simulator with the LTE power model for the trace-driven analysis.

The LTE state machine simulator takes as input the 3G packet trace. It first preprocesses the trace using the methodology introduced in previous work [20] by removing existing state promotion delays. The purpose is to decouple promotion delays introduced by the existing 3G state machine from the actual traffic patterns. Therefore the 3G trace can be simulated in a different (4G LTE) network. Subsequently, the simulator injects new promotion delays into the trace, then infers the RRC states experienced by the UE based on the timing, size, and direction of each packet, according to the 4G LTE state transition model depicted in Figure 2.1(b). The output of network model simulator is an array of packets with adjusted timestamps, as well as the RRC states of the UE at any given time.

The power model simulator takes as input the output of the state machine simulator and calculates the energy consumed by the 4G radio interface. The total energy consumption consists of four components: idle, promotion, tail, data transfer. In idle mode, the radio power consumption is very low (almost zero). For promotion energy, both the promotion delay and the average promotion radio power are fixed. The radio power consumed on the tail is also fixed, based on our measurement using a real LTE phone (detailed below).

The parameters of the model are measured by us using an HTC ThunderBolt smartphone with an LTE data plan of a large cellular ISP in the U.S. The phone has 768MB RAM memory and 1 GHz Qualcomm MSM8655 CPU, running Android 2.2.1. We use the Monsoon power monitor [76] to measure the UE power consumption. The measured

	Power[a] (mW)	Duration (ms)
Screen off	P_{base}: 11.4±0.4	N/A
Idle average[b]	P_{idle}: 31.1±0.4	N/A
Promotion	P_{pro}: 1210.7±85.6	T_{pro}: 260.1±15.8
Tail average[b]	P_{tail}: 1075.5±3.3	T_{tail}: 11576.0±26.1

[a] All power values include the base power with screen off.
[b] The average power is measured with DRX considered.

Table 7.2: The LTE power model parameters measured from a real cellular ISP.

parameters are summarized in Table 7.2. We validate the LTE state machine and the power model simulator by comparing the radio energy measured by power monitor with simulated radio energy for four popular Android applications. The error rate is less than 6%.

7.3.3 Prediction Framework Evaluation Methodology

To evaluate the performance of the prediction framework, we need to calculate the prediction accuracy and the resulting $\Delta(E)$ and $\Delta(S)$.

It is straightforward to calculate the prediction accuracy. Assume n bursts are predicted and n_s of them are followed by short **IBT**s and n_l are followed by long **IBT**s ($n = n_s + n_l$). Then assume the prediction framework correctly predicts n_1 out of the n_s **IBT**s to be short and n_2 out of the n_l **IBT**s to be long, the prediction accuracy is $(n_1 + n_2)/n$.

For calculating $\Delta(E)$ and $\Delta(S)$, if an **IBT** is predicted to be short, the prediction framework would have no impact since it keeps UE stay in the high power tail and there is no change to the RRC state machine. When an **IBT** is predicted to be long, the prediction framework would make UE demote to **RRC_IDLE** to save energy.

Assume the current burst B_i ends at time t_0 and the next burst B_{i+1} comes at time t_1. At time t_0+**BT**, when the existence of B_i is detected by the prediction framework, it collects information about B_i and predicts the **IBT** following it, denoted as ibt_i, whose actual value is $t_1 - t_0$. We also assume that the short burst threshold **SBT** $\leq T_{tail}$, the tail timer. Otherwise, if **SBT** $> T_{tail}$, the energy saving opportunity would be much less,

Algorithm 1 Update $\Delta(E), \Delta(S)$ for burst B_i

$P_{tail} \leftarrow$ power level at **RRC_CONNECTED** tail
$P_{idle} \leftarrow$ power level at **RRC_IDLE** state
$P_{pro} \leftarrow$ power level during promotion
$T_{pro} \leftarrow$ promotion delay
if ibt_i is predicted to be $>$ **SBT then**
  if $ibt_i \leq T_{tail}$ **then**
    //Demote to **RRC_IDLE** at time **BT** $+ t_0$, save energy
    $\Delta(E)$ += $(P_{tail} - P_{idle})(ibt_i - \textbf{BT})/E_D$
    //Extra idle to active promotion overhead
    $\Delta(E)$ −= $P_{pro}T_{pro}/E_D$
    $\Delta(S)$ += $1/S_D$
  else
    //Demote to **RRC_IDLE** at time **BT** $+ t_0$, save energy
    $\Delta(E)$ += $(P_{tail} - P_{idle})(T_{tail} - \textbf{BT})/E_D$
    //No extra promotion overhead
  end if
else
  // ibt_i predicted to be short, remain in the tail, no impact
end if

since when an **IBT** is predicted to be short, it is still possible that **IBT** $> T_{tail}$, and it is an indication that the tail could be ended earlier to further save energy. Based on these definitions, we provide the detailed pseudocode of updating $\Delta(E)$ and $\Delta(S)$ for burst B_i in Algorithm 1. If $ibt_i > T_{tail}$ and ibt_i is predicted to be long, for the original state machine, the arrival of B_{i+1} already triggers a promotion, so the prediction framework does not incur any extra promotion overhead. Otherwise, if $ibt_i \leq T_{tail}$, an extra promotion would occur.

7.4 Implementation and Evaluation

In this section, we first talk about the implementation of RP. Then we evaluate different prediction models for **IBT** prediction using various machine learning algorithms, and analyze the impact on energy saving and signaling overhead.

7.4.1 Implementation

We implement the full RP system on Android to evaluate its running overhead and practicality. In addition, we implement the IBT prediction framework described in Section 7.1 in MATLAB, which allows us to evaluate a broader set of machine learning algorithms.

Android implementation : the test device we use is Samsung Galaxy S3 phone running Android 4.0.4 (Ice Cream Sandwich), with 1.0GB RAM and Quad-core 1.4 GHz Cortex-A9 CPU. We modify the `tcpdump` source code and cross-compile it for Android platform to be our data collector. Specifically, our data collector aggregates the traffic information into bursts in real time. For each burst, the features listed in Table 7.2.3 are collected. Whenever an idle gap of **BT** is observed, indicating the previous packet marks the end of the current burst, the data collector summarize all features of the current burst and feed them to the IBT prediction framework. Currently, the data collector and IBT prediction framework works in separate processes and the inter-process communication (IPC) is via file read/write for simplicity, and better IPC mechanisms can be further investigated.

The IBT prediction framework is implemented as an Android application which runs in the background and only periodically wakes up to check whether there is any burst pending prediction. If there is a burst for prediction, the framework would predict the **IBT** based on the features of the current burst and the prediction model based on previous burst history. The purpose for the periodic wake-ups is to prevent the IBT prediction always acquiring the CPU lock, thus wasting computational resources. The periodic cycle represents a tradeoff between responsiveness and CPU overhead. In practice, we choose the cycle to be 0.2**BT**, which at most incurs 0.2**BT** delay in making prediction and in the same while incus less than 1% CPU overhead, as the checking for a new burst is quite lightweight, *i.e.,* checking the last-modified time in our implementation. Our IBT prediction framework uses the machine learning algorithm implementation included in the Weka library [84].

In order to implement the fast dormancy scheduler, we need both client and network

support for actively invoking fast dormancy. However, we do not find any programming interface for this purpose and we bypass this technical challenge, using the fact that fast dormancy is available on our device and the timer for triggering fast dormancy can be changed after the device is rooted. Specifically, we change the fast dormancy timer to be **BT**, *i.e.,* by default, fast dormancy is invoked after each idle gap of **BT**. When fast dormancy scheduler decides to invoke fast dormancy, the default fast dormancy settings helps achieve this purpose, and when the opposite, the fast dormancy scheduler sends small UDP packets to extend the tail time. Assuming the default tail timer (without fast dormancy) is T_{tail} and the timer for fast dormancy is T_{fd} ($T_{fd} =$ **BT** in our implementation), fast dormancy scheduler needs to keep sending the small UDP packets for $T_{tail} - T_{fd}$ time, and the gaps between such UDP packets should be slightly less than T_{fd} to ensure that no extra promotion is triggered by these additional packets. Notice that this trick is a necessity with fast dormancy invocation support, and we only use it for evaluation purpose.

MATLAB implementation: in order to easily evaluate a broader set of machine learning algorithms, we implement the IBT prediction framework in MATLAB using its statistics toolbox [85]. This implementation is intended to evaluate the accuracy and effectiveness of different machine learning algorithms and parameter settings for the UMICH data set traces. Specifically, we look at Naïve Bayes, discriminant analysis, SVM, classification tree, regression tree, Ensemble (RobustBoost) and Ensemble (bag). We only report the representative ones with better performance in our following analysis.

We use MATLAB R2011b (7.13.0.564) running on a desktop with Intel Xeon CPU (3.16GHz) and 16GB memory, running Linux 2.6.38. With the same C++ benchmark programs, we observe that the running time on the studied Android device is around 4 times of the desktop. We use this ratio as a simple heuristic to estimate the training time and prediction time of different algorithms. Notice that we only use this ratio to estimate the running time of MATLAB machine learning algorithms on Android. For actual evaluation

of the running overhead, we rely on the Android implementation and the experiments on real Android devices.

7.4.2 Accuracy v.s. Performance Metrics

We start by discussing the impact of **IBT** prediction accuracy on the energy saving $\Delta(E)$ and signaling load change $\Delta(S)$ when RP is deployed.

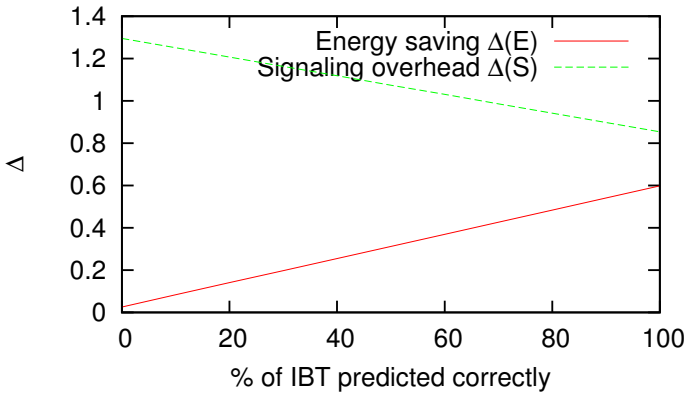


Figure 7.10: $\Delta(E), \Delta(S)$ v.s. prediction accuracy.

Given a sequence of bursts $B_1, B_2, \cdots, B_n$, let the **IBT** after B_i be t_i. Ideally, the prediction framework could accurately predict whether $t_i \leq$ **SBT**. We calculate $\Delta(E)$ and $\Delta(S)$ as described in §7.3.3 by varying the binary prediction accuracy. Figure 7.10 shows that $\Delta(E)$ and $\Delta(S)$ have a linear pattern as accuracy changes. When the accuracy is 0, namely all **IBT**s are predicted to be wrong, there is little energy saving of 2.6% with $\Delta(S) = 129.5\%$. The energy saving is so small because a large number of promotions also incurs significant promotion energy. As accuracy increases, RP saves more energy with reduced $\Delta(S)$. In the optimal case, when the accuracy achieves 100%, 59.8% of the total energy could be saved while the signaling overhead increases by 85.4%. The signaling overhead is not 0, because for **IBT**s smaller than T_{tail} but larger than **SBT**, even if the prediction is correct, triggering fast dormancy would still incur an extra state promotion. Such overhead is inherent for any optimization technique, such as a smaller static T_{tail} setting and RadioJockey. The design goal of RP is to approach this optimal case.

$\alpha =$	100	500	1000	2000	5000
$\beta = 1$	81.5%	83.7%	**84.2%**	82.4%	80.2%
$\beta = 2$	80.1%	81.4%	82.9%	82.0%	80.0%
$\beta = 5$	79.8%	80.9%	81.4%	81.0%	79.3%
$\beta = 10$	79.4%	80.0%	80.9%	80.0%	79.0%
$\beta = 20$	78.9%	79.6%	80.2%	79.5%	78.7%

Table 7.3: Tuning α, β for **`MostRecent`** prediction model.

Name	Description	Accuracy
`MostRecent`	Use most recent α bursts to predict next β bursts (per user)	84.2%
`PerUser`	Use n historical bursts of a user to train a fixed model (per user)	80.8%
`AllUsers`	Use k historical bursts of all users train a fixed model (all users)	77.5%

Table 7.4: Summary of prediction models.

7.4.3 Prediction Model Comparison

For **`IBT`** prediction, historical **`IBT`** information is needed to train a prediction model. For RP, we propose to use the most recent historical information to train a model for each user, denoted as **`MostRecent`**. Specifically, for each user, the most recent α bursts are used to predict the next β bursts. We study the impact of changing α, β in Table 7.3, using the Ensemble *bagging* [86] learning algorithm as an example (number of trees set to 20). We calculate the prediction accuracy for different (α, β) settings. We observe that if α is too small, there is not enough history information for learning; in the opposite, if α is too large, the user is more likely to switch to new applications so old rules do not apply. Based on Table 7.3, we choose $\alpha = 1000$ and $\beta = 1$ that maximize the accuracy in the experimental measurement. In practice, α and β could also be dynamically adjusted based on prediction history.

In Table 7.4, we compare the **`MostRecent`** model with two other models, **`PerUser`** and **`AllUsers`**. For fair comparison, we use the same machine learning algorithm (Ensemble *bagging*) as in Table 7.3. We set $\alpha = 1000$ and $\beta = 1$ for the **`MostRecent`** model

Settings (unit: sec)	Accuracy	$\Delta(E)$	$\Delta(S)$	$\frac{\Delta(E)}{\Delta(S)}$
S_0 **BT**: 1 **SBT**: 3	82.65%	52.10%	101.64%	0.513
S_1 **BT**: 1 **SBT**: 2	84.80%	56.69%	158.99%	0.357
S_2 **BT**: 1 **SBT**: 4	81.94%	49.07%	83.34%	0.589
S_3 **BT**: 0.5 **SBT**: 3	84.71%	53.74%	100.36%	0.535
S_4 **BT**: 1.5 **SBT**: 3	85.39%	58.85%	93.75%	0.628
S_5 **BT**: 1/1.5 off/on **SBT**: 2.5/3 off/on	85.88%	59.07%	91.01%	0.649

⋆ All evaluations in this table for RP are using *Classification Tree* with *MostRecent* prediction model ($\alpha = 1000$, $\beta = 1$).

Table 7.5: Comparison of different parameter settings.

as discussed previously. Similarly, we set $n = 10,000$ for **PerUser** and $k = 10,000$ for **AllUsers** (n and k are defined in Table 7.4). The values of n and k here are empirically selected, which yield good prediction accuracy for each prediction model. We observe that **MostRecent** has higher prediction accuracy than the other two modes, and **PerUser** is more accurate than **AllUsers**. This suggests that it is necessary to have a separate model based on the most recent history for each individual user.

7.4.4 Selecting Burst Thresholds

In the previous subsections, we only examine one setting of **BT** and **SBT**, *i.e.,* **BT** = 1s and **SBT** = 3s. Here we study how different settings of these parameters affect the performance of RP. In the top half of Table 7.5, we select some combinations of **BT** and **SBT** values and compare them with the setting of S_0. We notice that S_1 sets **SBT** to be 2s, which significantly increases $\Delta(S)$. Among all settings explored, S_4 yields the highest $\Delta(E)/\Delta(S)$ [3].

As suggested by previous work [83], a different parameter setting distinguishing screen-on and screen-off could yield better optimization results given their traffic pattern difference, because screen-off traffic is very likely to be generated by background applications

[3] $\Delta(E)/\Delta(S)$ is the average energy saving per unit signaling overhead, and we use it to quantify the performance of a setting.

without user interaction. A more aggressive setting could be applied to screen-off traffic to save energy without incurring much signaling overhead. Specifically, here a more aggressive setting means a smaller **BT**, which gives RP an opportunity to predict **IBT** earlier, and a smaller **SBT**, which classifies more **IBT**s into the long category hence resulting in more fast dormancy invocations. In Table 7.5, S_5 is an example of such screen-aware setting that yields good results. Compared with S_4, S_5 saves more energy with less signaling overhead incurred. In fact, S_5 achieves comparable optimization results as the optimal case shown in Table 7.6 does.

In practice, given the training data, we can periodically search within a large range of **BT** and **SBT** values for an optimal setting. The setting can also be made dynamic and customized for each user.

To summarize, RP achieves radio energy saving by 59.1%, at the cost of 91.0% additional signaling overhead in LTE networks, outperforming previous proposals. Specifically, as shown in Table 7.6, in order to achieve such energy saving (59%), the additional signaling overheads incurred by MakeIdle and naïve fast dormancy are as high as 305% and 215%, respectively. The maximal energy saving achieved by RadioJockey is 27% since it is only applicable to background traffic, ideally generated by a single application running in the background.

7.4.5 Comparing Resource Optimization Approaches

In Table 7.6, we compare various radio resource optimization techniques, including the basic fast dormancy with static timers, RadioJockey [81], MakeIdle [82] and RP using different machine learning algorithms. Using the network and power model simulators (§7.3.2), we apply the above optimization techniques to the UMICH data set. We ignore the overhead of invoking fast dormancy, since invoking fast dormancy from the client only requires sending one special message called *Signaling Connection Release Indication* (SCRI). In contrast, a state promotion from the idle state to the connected state involves 12

Name	Description & Configuration	$\Delta(E)$	$\Delta(S)$	$\Delta(E)/\Delta(S)^a$
Fast dormancy 1s	Invoke fast dormancy after 1s idle time	62.7%	214.9%	0.29
Fast dormancy 3s	Invoke fast dormancy after 3s idle time	40.9%	95.8%	0.43
Screen-aware FD 3s on, 1s off	Invoke FD after 3s idle time at screen on; invoke FD after 1s idle time at screen off	52.9%	115.1%	0.46
RadioJockey Assuming 100% accuracy	RadioJockey applied to background traffic[d]	30.1%	51.7%	0.58 (background[b])
RadioJockey Assuming 90% accuracy	RadioJockey applied to background traffic	27.2%	52.0%	0.52 (background)
MakeIdle M:1000, N:100	MakeIdle: based on previous M packets, predict next N packets	64.9%	305.2%	0.21
MakeIdle M:10, N:10	MakeIdle: based on previous M packets, predict next N packets	44.9%	195.2%	0.23
RP: Naïve Bayes	Naïve Bayes classification with *mvmn*: multivariate multinomial distribution	53.0%	107.9%	0.49
RP: Classification Tree	Binary decision tree for classification	59.1%	91.0%	0.65
RP: Ensemble (Bag)	Method: *Bag*; type: classification weak leaner: decision tree; # of trees: 20	59.3%	90.2%	0.66
RP: Optimal	Predict all IBTs correctly	59.8%	85.4%	0.70

* All evaluations are using *MostRecent* prediction model with $\alpha = 1000$ and $\beta = 1$, with setting S_5 in Table 7.5.
[a] $\Delta(E)/\Delta(S)$ shows the average energy saving per unit signaling overhead. The higher this metric, the better.
[b] We use smartphone screen status to distinguish between foreground and background traffic.

Table 7.6: Comparison of different optimization algorithms.

or 30 signaling messages [81].

Basic fast dormancy : we simply change T_{tail} to be a constant smaller than its original value.

Screen-aware fast dormancy : we use different T_{tail} settings for fast dormancy based on screen status [83].

RadioJockey [81]: it uses system call traces to predict the *end-of-session* (EOS) for background application with no user interaction, with the ideal usage scenario for a single application. For multiple concurrent applications, it invokes fast dormancy only if *i)* EOS of one application is predicted, and *ii)* there is no active session of any application. RadioJockey does not handle foreground traffic, because user interactions may violate the correlation between system calls and EOS [81]. Given that we do not have system call traces in the UMICH data set, we assume that RadioJockey could predict the EOS for **each** of the applications, even when multiple applications are running concurrently. This is a

strong assumption, because the authors admit that the accuracy for prediction would be lower for concurrent applications, and RadioJockey does not use real traces to evaluate concurrent applications.

MakeIdle [82]: it relies purely on packet timing without considering signaling overhead, *i.e.,* it tries to find a wait time T_{wait} which maximizes the energy saving if T_{tail} is set to be T_{wait} for the previous M packets, it then applies this T_{wait} for the next N packets. The range we search for an optimal T_{wait} is $[0.5, 11.5]$, as suggested by the author. No recommendations have been made for the values of M and N, so we empirically select their values. MakeActive, proposed by the same set of authors tries to reduce the signaling overhead of MakeIdle. However, as we discuss in Chapter VIII, MakeActive is not a perfect fix without causing bad user experiences, especially for foreground traffic.

RadioProphet : we explore various machine learning algorithms with the **`MostRecent`** prediction model ($\alpha = 1000$ and $\beta = 1$). Specifically, we look at Naïve Bayes, classification tree, and Ensemble (bag) algorithms in the MATLAB implementation. All of them are off-the-shelf machine learning algorithms used widely. The IBT prediction accuracy and performance are listed in Table 7.7. We observe that Classification Tree has better accuracy than Naïve Bayes, while Ensemble (bag) has slightly better accuracy than Classification tree. In Table 7.7, the training time and prediction is the estimated time running these algorithms on Android. We use a suite of C benchmark programs that can run both on Android and on the server to measure the speed difference of the two platforms. Assume the total running time of the benchmark suite is T_a for Android and T_s for server, and assume the training/prediction time of the MATLAB implementation measured on the server is T, the estimated training/prediction time on Android is

$$T' = \frac{T_a}{T_s} T$$

Name	Training/prediction time (ms)	Accuracy
Naïve Bayes	25.7 / 9.8	76.1%
Classification Tree	547.7 / 23.7	85.9%
Ensemble (bag)	2504.2 / 426.2	87.4%

Table 7.7: Prediction performance and accuracy of different machine learning algorithms.

Notice that such estimation is only used for us to rough estimation and it may be inaccurate. Later, we will talk about our evaluation by running machine learning algorithms on real Android device. Based on Table 7.7, the running time overhead for Ensemble (bag) is unacceptable and we choose to use Classification Tree, which has slightly lower accuracy, but much lower running overhead.

In Table 7.6, we observe that "fast dormancy 1s" is an aggressive approach, saving 62.7% of energy while incurring 214.9% of signaling overhead. "Fast dormancy 3s" significantly reduces $\Delta(S)$ to 85.4%, with less energy saving of 40.9% as expected. The screen-are fast dormancy achieve a better balance between energy saving and signaling overhead, and the $\Delta(E)/\Delta(S)$ is 0.46. For RadioJockey, by assuming the accuracy for predicting EOS for each background application to be 90%, it saves 27.2% of energy with 52% of $\Delta(S)$. There is slight improvement when the accuracy increases to 100%. However, we are not able to evaluate RadioJockey for foreground traffic on which the prediction accuracy is unknown. For MakeIdle, we pick two sample (M, N) settings. For $(1000, 100)$, the energy saving is significant, but the additionally incurred signaling overhead has an unacceptable value of 305.2%. Even for $(10, 10)$, $\Delta(S)$ is as high as 195.2%. This is because MakeIdle does not consider balancing between energy saving and signaling overhead [4].

For RP, in the optimal case assuming there is an oracle with complete knowledge of traffic, it saves 59.8% of energy with 85.4% of extra signaling overhead incurred. Note that this signaling overhead depends on the choice of **SBT** and is inherent for any fast dormancy based optimization. For example, if there are two packets P_1 and P_2, such that

[4] Another orthogonal approach proposed by [82] to reduce the signaling overhead (called MakeActive) is not applicable, as it requires traffic shifting that could result in unacceptably negative impact on user experience for foreground traffic.

P_2 arrives 11.5s after P_1 and there is no other packet in between, in the default setting where $T_{tail} = 11.6s$ (Figure 2.1), there is no state promotion between P_1 and P_2. However, if we want to eliminate the long radio-on period after P_1, we have to invoke fast dormancy at some time between P_1 and P_2, and then the extra state promotion at P_2 is inevitable. Among all explored machine learning algorithms, *ensemble (bag)* performs best, achieving 87.4% of accuracy, 59.3% of $\Delta(E)$ and 90.2% of $\Delta(S)$. Classification Tree, which is more practical, achieve comparable performance

For the metric $\Delta(E)/\Delta(S)$, which quantifies how well the tradeoff between saving resources and incurring signaling load is handled, RP outperforms other algorithms. RadioJockey has comparable performance for background traffic, however, RP has good results for both background and foreground traffic from concurrent applications.

7.4.6 Running Overhead on Real Phone

We implement the prototype RP system on Android as discussed in §7.4.1, in order to demonstrate its practicality for today's smartphones. We breakdown the running overhead into different components and evaluate each of them. Specifically, the overhead includes data collection, model training and prediction, and invoking fast dormancy. As discussed in previous work [81], invoking fast dormancy incurs negligible overhead compared with state promotion. For "fast dormancy 3s", the total promotion energy is only 2.3% of the total energy. Hence, it is reasonable to ignore the fast dormancy invocation overhead.

Data collector : unlike RadioJockey requiring system call traces for all applications, RP only needs to monitor packet traces, which is also required by RadioJockey. Our data collector runs in the background and does not interfere with other user applications. It incurs no more than 1% of CPU overhead when parsing packet headers and producing output about burst features, although the overhead is much lower when the throughput is low (*e.g.,* less than 200 kbps). The additional power to run the data collector is less than

17mW most of the time, which is only 1.6% of the tail power for the studied LTE network.

Model training and prediction : we use *Classification Tree* implementation from the Weka [84] library for evaluation. We measure that the average model training time is 200ms and the average prediction time is 0.1ms. In terms of energy overhead, the incurred power overhead is also very small, mostly coming from the CPU computation, and the measured energy overhead is less than 1% of the tail energy.

In terms of the **storage cost**, for the **`MostRecent`** model, information of the α = 1000 historical bursts needs to be stored at the UE side, and according to our implementation, the storage of one burst is less than 200 bytes. So the total storage cost is only 200KB, negligible for modern smartphones.

7.5 Summary

We propose a novel, practical, and effective system called RadioProphet to address the radio resource deallocation problem for cellular networks. RP takes as input the most recent history information of UE traffic to build a prediction framework for traffic inter-burst time (**`IBT`**), leveraging machine learning to predict **`IBT`** based on a list of lightweight and carefully selected features. It performs resource deallocation only when **`IBT`** is sufficiently long to ensure low signaling overhead. We demonstrate that RP can be implemented at the UE transparently to applications, with negligible energy overhead and fast response times. Using 7-month-data collected from 20 real users with accurate RRC and radio power simulator for LTE 4G networks, we evaluated the performance of RP. With 85.9% of accuracy in predicting **`IBT`**s, RP provides the option of saving radio energy by up to 59.1%, at the cost of 91.0% of additional signaling overhead. We believe the RP system represents a significant step in systematically optimizing energy and radio resource for 3G and 4G LTE networks.

CHAPTER VIII

Related Work

In this chapter, we summarize related work in different categories.

8.1 Characterizing Mobile Network Usage and Performance

Netdiff system [87] establishes a benchmark for comparing performance of different ISPs. In our research, we attempt to establish an equivalent benchmark for comparing network application performance on smartphones. Although there are some existing tools available for such comparison, *e.g.,* `speedtest.net` [35] and FCC's broadband test [36], which measure the throughput and latency in 3G networks, MobiPerf covers a more comprehensive set of metrics, including DNS lookup, Ping to the first hop, TCP handshake, and HTTP request to landmark servers, *etc.* Existing studies have compared 3G and WiFi performance on smartphones [88] and studied the influence of the packet size on the delay in 3G networks [89]. Jiang *et al.* examine how large buffers in cellular networks contribute to significant TCP queuing delay [58]. Tan *et al.* carried out a similar measurement study on multiple commercial 3G networks [90]. However, their study is limited to one location (Hong Kong) and a few devices. Compared with their study, our work covers significantly more users from many different locations. Netalyzr [37] carries out network measurement for Internet users, not for smartphone users.

Regarding cellular network infrastructure, Xu *et al.* characterized 3G data network

infrastructures, leading to an observation that the routing of cellular data traffic is quite restricted as traffic must traverse through a small number of gateway nodes [43]. Wang *et al.* unveiled cellular carriers' NAT and firewall policies [91]. Balakrishnan *et al.* investigated IP address dynamics in 3G networks. They found that cellular IPs embed much less geographical information than wired IPs do [54].

Existing work has built tools to infer traffic differentiation policies from either local experiments or public deployment [92, 93, 94]. And our work is among the first attempts to systematically uncover the previously unknown network policy in cellular networks.

Prior efforts [95, 96, 97] deploy smartphone user studies and collect data from tens to hundreds of participating users. Those studies investigate various aspects including the diversity of smartphone users, the popularity of mobile applications, and the effectiveness of compression techniques on cellular traffic *etc.* Our study features a much larger user base of around 300K customers using the LTE networks whose characteristics are far from being well understood. Some previous studies also perform large-scale measurement of mobile networks and smartphones. Sommers *et al.* compare cellular and Wi-Fi performance using crowd-sourced data from `speedtest.net` covering 15 metro areas, focusing on throughput and latency [59]. Xu *et al.* profile diverse usage behaviors of smartphone applications [53]. Qian *et al.* perform network-wide measurement studies of cellular periodic transfers [30]. In contrast, our investigated spectrum is more diverse, covering traffic characteristics, network performance, protocol interaction, bandwidth utilization, and application usage in the increasingly popular LTE networks. We also compare our results with those presented in [59]. Some previous studies [98, 57] also examined mobile handsets using Wi-Fi networks.

8.2 Characterizing Smartphone Application Performance

There have been several studies focusing on mobile users from the perspective of applications, such as a study no the bottlenecks of web browsers' performance [47], a study [99]

which characterizes the relationship between users' application interests and mobility, and another study [54] which examines the possibility of geolocating IP address in 3G networks. Other related measurement works of cellular data networks include a study of traffic characteristics on mobile devices [100], and performance analysis of TCP/IP over 3G network with rate and delay variation [101].

Our work is inspired by numerous network and application measurement studies [102, 46, 99], *e.g.,* Trestian *et al.* characterize the relationship between users' application interests and mobility [99], Zhuang *et al.* investigate application-aware acceleration to improve application performance [46], and Liu *et al.* study the interaction between the wireless channels and applications [103]. Our work complements these works with different focus and methodology.

While many applications are built on TCP, their actual user-level performance does not match that of TCP alone due to various adaptation strategies and overheads induced by the applications. Realizing this problem, Chesterfield *et al.* [104] evaluated the performance of streaming media application over a WWAN (Wireless Wide Area Network). But their work is limited to a customized streaming application named *vorbistreamer*. In a separate study, Chakravorty *et al.* [102] measured the performance of TCP and web browsing over WWANs. However, that study is also different from ours because they focus only on comparing the overall throughput across different link layers.

Unlike previous studies, *e.g.,* [103, 102, 105], which perform measurements on desktop or laptop systems, relying on cellular network data cards or phones tethered through USB as a modem, in this study, the application performance data is collected directly from end-users' devices, thus more accurately reflecting the user-perceived performance.

8.3 Radio and Energy Optimization for Cellular Networks

In cellular networks, there exists a radio resource control (RRC) state machine that manages the handset radio interface. It is the key coupling factor bridging the application

traffic patterns and the lower-layer protocol behaviors. Previous studies [20] and [12] examine the RRC state machine and its interaction with cellular traffic, for 3G UMTS and 4G LTE networks, respectively. We study for hundreds of thousands of users their state transition delay and transport-layer idle time, two key factors incurring signaling load and energy overhead, respectively, due to the LTE RRC state machine.

DRX in 4G LTE networks. Zhou *et al.* [16] model the DRX mechanism as a semi-Markov process, with which they analytically study the effects of DRX parameters on the performance, as well as the tradeoff between the power saving and wake-up delay. Kolding *et al.* [18] also investigate the balance between throughput and power saving by changing the configuration of DRX parameters, using a web-browsing traffic model. Bontu *et al.* [17] further considers the impact of DRX parameters on different applications with various delay sensitivities. Wigard *et al.* compares a long-DRX-only scenario and a scenario with both long and short DRX, in terms of throughput and power consumption. All above studies employing analytical models suffer from an inherent limitation: the expressiveness of an analytical model is quite limited and is unlikely to capture the characteristics of real-world traffic patterns using a statistical distribution with a few parameters. The existence of concurrent applications accessing the network further increases the difficulty of modeling the packet dynamics. In contrast, our work is the first empirical study to investigate the impact of the configurations of DRX parameters and tail timer. We overcome the above limitation by employing network traces from real smartphone users, thus more realistically revealing the complex tradeoffs incurred by various DRX parameters.

Smartphone power modeling has also been investigated by previous empirical measurements. The Bartendr project [45] studies the relationship between signal strength and smartphone radio power usage. PowerTutor [75] collects power traces for individual hardware components under controlled experiments then uses multi-variable regression to compute the coefficients for a linear power model considering all hardware components. ARO [26] employs an approach similar to [75] but focusing only on radio power usage. It

performs more fine-grained simulation of transmission queues to capture state transitions. Our LTE power model is also empirically derived, but it differs from all aforementioned models in two aspects. First, it considers DRX in **RRC_CONNECTED**, a new power management mechanism in LTE networks. Second, it further takes into account both uplink and downlink data rates, resulting in a more accurate estimation of the radio power consumption when the throughput is high, as is a common case in LTE networks.

Investigation of the inactivity timers in cellular networks. Inactivity timers, which determine the tail times, are the most important parameters in cellular radio resource management. Radio Resource Control (RRC) tail [21], is necessary and important for cellular networks to prevent frequent state promotions (resource allocation), which can cause unacceptably long delays for the UE, as well as additional processing overheads for the radio access network [23, 24]. Today's cellular carriers use a static and conservative setting of the tail time in the order of many seconds, and previous studies have revealed this tail time to be the root cause of energy and radio resource inefficiencies in both 3G [20, 21, 106, 107]. Recent work [108] investigates impact of traffic patterns on radio power management policy. The impact of inactivity timers on the UMTS network capacity [107] and the UE energy consumption [109, 110] has also been studied. Another study [20] makes a further step by characterizing the impact of operational state machine settings with real traces and conclude that the fundamental limitation of the current cellular resource management mechanism is treating all traffic according to the same RRC state machine *statically and globally* configured for all users, therefore, it is difficult to balance various tradeoffs of resource utilization.

Adaptive resource release. Distinct from using static timers, Liers *et al.* [111] propose to determine inactivity timers dynamically, based on the current load, radio resources, and processing capability. However, they only address the problem from the perspective of network capacity as to reducing the call blocking and dropping rate, and the same timer values are applied globally to all UEs at any given time. In comparison, RP is based on fast

dormancy where resource release is requested by each individual UE. Such fine-grained control leads to much more significant savings of the radio resource and the UE energy consumption.

Fast dormancy (FD) [31, 32] is a mechanism in 3G networks for reducing the amount of tail time incurred by a device by quickly demoting it to a low energy RRC state without waiting for the tail timer to expire. TOP [22] proposes to leverage fast dormancy to eliminate the tail whenever possible. It **assumes** each individual application can predict an imminent long **`IBT`** with reasonable accuracy, and fast dormancy is only invoked when the predicted aggregate idle time across all concurrent applications is long enough. While TOP provides framework and mechanism for optimization, it does *not* solve the problem of prediction. In this study, we solve the open research problem of predicting **`IBT`**s. Unlike TOP being a proposed framework with unsolved assumptions, our system RP is a practical working system.

MakeIdle [82] uses packet timestamp information to calculate the optimal idle time before fast dormancy should be invoked that maximizes the energy saving. However, MakeIdle does not look at the application context and other useful features used in RP, resulting in worse performance as shown later. Moreover, MakeIdle algorithm does not consider the balance of energy saving and signaling overhead, and leaves the job of reducing the signaling overhead to another algorithm called MakeActive [82], based on shifting packets (batching). MakeActive does not work for foreground traffic, and even for background traffic, there is no guarantee that it would not affect user experience, since it may incur up to seconds extra delay. In this study, to minimize such negative impact on user experience, RP does not rely on any traffic shifting techniques, especially because the goal for the design of RP is to be working for all traffic, both foreground and background.

RadioJockey[81] is another closely related study. It uses system call traces to determine the *end-of-session* (EOS) for each application and triggers fast dormancy if there is no active session for any of the applications. However, RadioJockey only works for back-

ground application with no user interaction, and the authors point out that the prediction accuracy would be lower for foreground traffic since user interactions may violate the correlation between system call patterns and EOS; while for RP, there is no such limitation and we achieve even better performance for all traffic than RadioJockey for screen-off traffic. Also, RadioJockey treats different applications separately and could not predict the *start-of-session* (SOS), hence when there are multiple concurrent applications, the prediction accuracy would be jeopardized, with higher instrumentation and prediction overhead. Our approach uses aggregate traffic from all applications and does not suffer from these problems. Further, RadioJockey requires complete system call traces for all applications in addition to the network traces, which incurs higher overhead than RP. In the evaluation section later, we compare both MakeIdle and RadioJockey with RP and ignore MakeActive because it alters user traffic resulting in negative impact on user experience, which is not acceptable especially for foreground traffic.

Traffic scheduling There exists various traffic scheduling algorithms (*e.g.,* piggyback, batching [30, 112], TailEnder [21], and Intentional Networking [113]) for optimizing resource consumption in cellular networks. For example, in piggyback [30], transfers can be shifted earlier, or be postponed till later, so that they can potentially be overlapped with non-target transfers, thus reducing the total tail time. TailEnder [21] schedules transfers to minimize the energy consumption while meeting user-specified deadlines by delaying transfers and transmitting them together. In addition, specialized energy saving techniques for mobile applications have been proposed for specific applications [114, 115] and for specific protocols [116]. The major limitations of these approaches is that they are only applicable to delay-tolerant traffic (RSS feeds, push notification *etc.*). RP can be applied to any type of traffic (in particular, delay-sensitive traffic triggered by users), and can be used together with the aforementioned scheduling techniques.

Traffic modeling and prediction has be explored by various previous works. A linear prediction-based time series forecasting technique is used to predict future no-data intervals

for multimedia data streaming [117]. Hidden Markov Models are also used for modeling and predicting traffic of several Internet applications [118]. In our work, we look at the aggregate traffic, including all different applications, and predict whether inter burst time is longer than a threshold, which is a joint effect of packets generated by all applications, and hence is more challenging and better for practical use. This specific, yet important problem for energy and network resource saving has not been studied before in mobile platforms with a large real-world data set as ours.

CHAPTER IX

Conclusion and Future Work

My dissertation is dedicated to understand and optimize the performance and energy footprint of smartphone applications, given their wide popularity. These issues are important for smartphone users, application developers, smartphone vendors, content providers and cellular network operators because improving the end-user experiences is their common interest.

Given that smartphones and 3G/4G cellular networks are relatively new and such research topics are not well-studied, we have to solve many challenges and take some of the first steps in this area. For example, in order to characterize application performance, we need to understand the network characteristics of the underlying cellular networks and MobiPerf tool is built for this purpose. Also, in order to optimize the energy footprint of smartphone applications, we have devised a smartphone power model for 3G and LTE networks with real devices. The MobiPerf tool and the smartphone power model, along with the network and power characterization results, can potentially help other researchers in both industry and academia for their related research projects.

With controlled experiments separating individual factors on application performance, we are able to narrow down the bottleneck factors for web browsing applications and understand the effect of various performance optimization techniques. In addition, we study the interplay between TCP, application behavior and LTE network with a data set consisting

of over 300,000 real LTE users and spot serious performance problems while identifying their root causes.

With the knowledge of smartphone power model and cellular radio resource management, we design and implement the RP system that makes intelligent decisions to dynamically release radio resources based on traffic pattern prediction. RP is implemented on Android incurring negligible CPU and energy overhead, while achieving radio energy savings and minimizing additional signaling overhead, significantly outperforming existing proposals. In addition, we also propose screen-aware optimization, which can better balance the key tradeoffs in cellular networks.

Throughout my research study, I make sure that the measurement results are *representative*, *e.g.,* via deploying smartphone applications to hundreds of thousand of real smartphone users and collect data directly from end devices. We also pay special attention to user privacy and only the anonymous measurement results that are beneficial for research are collected and all user identification information has been properly hashed. When designing systematic solutions for either measuring performance or optimizing energy footprint, we keep in mind that the best solution should be general, practical and long-lasting. Whenever possible, we implement and evaluate our system, *e.g.,* RP, on real devices, instead of simulations or synthetical analysis. We also pay special attention for the new LTE network, making sure we have thorough understanding of its network and power characteristics and our solutions are applicable for it, as it is believed to be a long-lasting cellular technology expected to exist for decades [2].

9.1 Future Research Agenda

As future directions, I will continue to extend my previous works in the following aspects:

- *Cellular Network Performance Characterization.* We will continue the development

MobiPerf and make it into a general mobile measurement platform that can benefit the entire research community. In addition, we will make use of the data set collected by *MobiPerf* to make more analysis on cellular network performance and this data set [119] is also available for other researchers.

- *Energy Optimization for Cellular Networks.* We will improve the prediction algorithm of RP to consider TCP-level and application level context, which may further increase the prediction accuracy.

- *Effect of Network Protocol and Application Behavior on Performance for LTE Networks*. We will improve the design of the bandwidth estimation algorithm that we used to quantify the network utilization ratio of existing applications in the LTE networks. We will also explore the predictability of the available bandwidth in the LTE networks and study its implications and applications.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Huang, J., Xu, Q., Tiwana, B., Mao, Z. M., Zhang, M., and Bahl, P., "Anatomizing Application Performance Differences on Smartphones," *MobiSys*, 2010.

[2] "3GPP LTE," http://www.3gpp.org/LTE.

[3] "Canalys Press Release: R2009112," http://www.canalys.com/pr/2009/r2009112.htm.

[4] "Canalys Press Release: R2012111," http://www.canalys.com/newsroom/sony-and-htc-overtake-rim-and-nokia-smart-phones.

[5] "Smartphone 3G Test (3GTest)," http://www.eecs.umich.edu/3gtest.

[6] "4GTest," https://play.google.com/store/apps/details?id=com.mobiperf.lte.

[7] "MobiPerf (3GTest)," http://mobiperf.com.

[8] "FCC Open Internet Apps Challenge," http://openinternetapps.challenge.gov/.

[9] "MobiPerf Source Repository," https://github.com/Mobiperf/MobiPerf.

[10] "Measurement Lab," http://www.measurementlab.net/.

[11] "3GPP TR 25.913: Requirements for Evolved UTRA and Evolved UTRAN (V9.0.0)," 2009.

[12] Huang, J., Qian, F., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O., "A Close Examination of Performance and Power Characteristics of 4G LTE Networks," *MobiSys*, 2012.

[13] "3GPP TS 36.211: Physical Channels and Modulation (V10.3.0)," 2011.

[14] "3GPP TS 25.304: User Equipment (UE) procedures in idle mode and procedures for cell reselection in connected mode (V10.2.0)," 2011.

[15] "3GPP TS 36.321: Medium Access Control (MAC) protocol specification (V10.3.0)," 2011.

[16] Zhou, L., Xu, H., Tian, H., Gao, Y., Du, L., and Chen, L., "Performance Analysis of Power Saving Mechanism with Adjustable DRX Cycles in 3GPP LTE," *Vehicular Technology Conference*, 2008.

[17] Bontu, C. S. and Illidge, E., "DRX Mechanism for Power Saving in LTE," *IEEE Communications Magazine*, 2009.

[18] Kolding, T., Wigard, J., and Dalsgaard, L., "Balancing Power Saving and Single User Experience with Discontinuous Reception in LTE," *ISWCS*, 2008.

[19] Wigard, J., Kolding, T., Dalsgaard, L., and Coletti, C., "On the User Performance of LTE UE Power Savings Schemes with Discontinuous Reception in LTE," *ICC Workshops*, 2009.

[20] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O., "Characterizing Radio Resource Allocation for 3G Networks," *IMC*, 2010.

[21] Balasubramanian, N., Balasubramanian, A., and Venkataramani, A., "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications," *IMC*, 2009.

[22] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O., "TOP: Tail Optimization Protocol for Cellular Radio Resource Allocation," *Proc. ICNP*, 2010.

[23] "System impact of poor proprietary fast dormancy. 3GPP discussion and decision notes RP-090941," 2009.

[24] Lee, P. P. C., Bu, T., and Woo, T., "On the Detection of Signaling DoS Attacks on 3G Wireless Networks," 2007.

[25] Chatterjee, M. and Das, S. K., "Optimal MAC State Switching for CDMA2000 Networks," *INFOCOM*, 2002.

[26] Qian, F., Wang, Z., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O., "Profiling Resource Usage for Mobile Applications: a Cross-layer Approach," *MobiSys*, 2011.

[27] "3GPP TR 25.813: Radio interface protocol aspects (V7.1.0)," 2006.

[28] "3GPP TS 36.331: Radio Resource Control (RRC) (V10.3.0)," 2011.

[29] "System Impact of Poor Proprietary Fast Dormancy," 3GPP discussion and decision notes RP-090941, 2009.

[30] Qian, F., Wang, Z., Gao, Y., Huang, J., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O., "Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization," *World Wide Web*, 2012.

[31] "UE "Fast Dormancy" behavior. 3GPP discussion and decision notes R2-075251," 2007.

[32] "Configuration of fast dormancy in release 8. 3GPP discussion and decision notes RP-090960," 2009.

[33] Sesia, S., Toufik, I., and Baker, M., "LTE: The UMTS Long Term Evolution From Theory to Practice," John Wiley and Sons, Inc., 2009.

[34] "AT&T Admits To Network Overload," http://thedroidguy.com/2011/04/att-admits-to-network-overload/.

[35] "Speedtest.net," http://www.speedtest.net/.

[36] "FCC Consumer Broadband Test," http://broadband.gov/qualitytest/.

[37] "ICSI Netalyzr," http://netalyzr.icsi.berkeley.edu/.

[38] "Alexa Top 500 Global Sites," http://www.alexa.com/topsites.

[39] "MaxMind," http://www.maxmind.com.

[40] Sundaresan, S., de Donato, W., Teixeira, R., Crawford, S., and Pescap, A., "Broadband Internet Performance: A View From the Gateway," *SIGCOMM*, 2011.

[41] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J., "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *SIGCOMM*, 1998.

[42] "Uncovering Cellular Network Characteristics: Performance, Infrastructure, and Policies," *University of Michigan Technical Reports, CSE-TR-578-13*, 2013.

[43] Xu, Q., Huang, J., Wang, Z., Qian, F., Gerber, A., and Mao, Z. M., "Cellular Data Network Infrastructure Characterization and Implication on Mobile Content Placement," *SIGMETRICS*, 2011.

[44] "BU-353 USB GPS Receiver," http://www.usglobalsat.com/p-62-bu-353-w.aspx.

[45] Schulman, A., Navda, V., Ramjee, R., Spring, N., Deshpande, P., Grunewald, C., Jain, K., and Padmanabhan, V. N., "Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling," *MobiCom*, 2010.

[46] Zhuang, Z., Chang, T.-Y., Sivakumar, R., and Velayutham, A., "A3: Application-Aware Acceleration for Wireless Data Networks," *MOBICOM*, 2006.

[47] Wang, Z., Lin, F. X., Zhong, L., and Chishti, M., "Why are Web Browsers Slow on Smartphones?" *HotMobile*, 2011.

[48] "SunSpider JavaScript Benchmark 0.9," http://www.webkit.org/perf/sunspider/sunspider.html.

[49] L. Masinter, "The "data" URL Scheme," RFC 2397, 1998.

[50] "Best Practices for Speeding Up Your Web Site," `http://developer.yahoo.com/performance/rules.html#minify`.

[51] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patt, A., "CloneCloud : Elastic Execution between Mobile Device and Cloud," *Proceedings of 6th European Conference on Computer Systems (EuroSys 2011)*, 2011.

[52] V. Jacobson and R. Braden and D. Borman, "TCP Extensions for High Performance," RFC 1323, 1992.

[53] Xu, Q., Erman, J., Gerber, A., Mao, Z. M., Pang, J., and Venkataraman, S., "Identifying Diverse Usage Behaviors of Smartphone Apps," *IMC*, 2011.

[54] Balakrishnan, M., Mohomed, I., and Ramasubramanian, V., "Where's That Phone?: Geolocating IP Addresses on 3G Networks," *Proceedings of IMC*, 2009.

[55] Zhang, Y., Breslau, L., Paxson, V., and Shenker, S., "On the Characteristics and Origins of Internet Flow Rates," *SIGCOMM*, 2002.

[56] Qian, F., Gerber, A., Mao, Z. M., Sen, S., Spatscheck, O., and Willinger, W., "TCP Revisited: A Fresh Look at TCP in the Wild," *IMC*, 2009.

[57] Chen, X., Jin, R., Suh, K., Wang, B., and Wei, W., "Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network," *IMC*, 2012.

[58] Jiang, H., Wang, Y., Lee, K., and Rhee, I., "Tackling Bufferbloat in 3G/4G Networks," *IMC*, 2012.

[59] Sommers, J. and Barford, P., "Cell vs. WiFi: On the Performance of Metro Area Mobile Connections," *IMC*, 2012.

[60] Gerber, A., Pang, J., Spatscheck, O., and Venkataraman, S., "Speed Testing without Speed Tests: Estimating Achievable Download Speed from Passive Measurements," *IMC*, 2010.

[61] Allman, M., Paxson, V., and Blanton, E., "TCP Congestion Control," RFC 5681, 2009.

[62] Paxson, V., Allman, M., Chu, J., and Sargent, M., "Computing TCP's Retransmission Timer," RFC 6298, 2011.

[63] Sarolahti, P. and Kuznetsov, A., "Congestion Control in Linux TCP," *USENIX Annual Technical Conference*, 2002.

[64] M. Mathis and J. Mahdavi and S. Floyd and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, 1996.

[65] Brakmo, L. and Peterson, L., "TCP Vegas: end to end congestion avoidance on a global Internet," *Selected Areas in Communications, IEEE Journal on*, Vol. 13, No. 8, 1995, pp. 1465 –1480.

[66] Prasad, R., Dovrolis, C., Murray, M., and kc claffy, "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools," *IEEE Network*, 2003.

[67] Jain, M. and Dovrolis, C., "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," *IEEE Network*, 2003.

[68] Hu, N., Li, L. E., Mao, Z. M., Steenkiste, P., and Wang, J., "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," *SIGCOMM*, 2004.

[69] R. Braden, "Requirements for Internet Hosts – Communication Layers," RFC 1122, 1989.

[70] Halepovic, E., Pang, J., and Spatscheck, O., "Can you GET Me Now? Estimating the Time-to-First-Byte of HTTP Transactions with Passive Measurements," *IMC*, 2012.

[71] Maier, G., Schneider, F., and Feldmann, A., "A First Look at Mobile Hand-held Device Traffic," *PAM*, 2010.

[72] Erman, J., Gerber, A., Ramakrishnan, K., Sen, S., and Spatscheck, O., "Over The Top Video: The Gorilla in Cellular Networks," *IMC*, 2011.

[73] "Shazam App," `http://www.shazam.com/`.

[74] "Netflix App," `http://www.netflix.com/`.

[75] Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L., "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," *CODES+ISSS*, 2010.

[76] "Monsoon power monitor," `http://www.msoon.com/LabEquipment/PowerMonitor`.

[77] "HTC Thunderbolt Review: The First Verizon 4G LTE Smartphone," `http://www.anandtech.com/show/4240/htc-thunderbolt-review-first-verizon-4g-lte-smartphone`.

[78] Cui, S., Goldsmith, A. J., and Bahai, A., "Energy-Efciency of MIMO and Cooperative MIMO Techniques in Sensor Networks," *IEEE J. Sel. Areas Commun., vol. 22, no. 6, pp. 10891098*, 2004.

[79] Perrucci, G., Fitzek, F., and Widmer, J., "Survey on Energy Consumption Entities on Smartphone Platform," *Vehicular Technology Conference*, 2009.

[80] "TCPDUMP and LIBPCAP," `http://www.tcpdump.org/`.

[81] Athivarapu, P., Bhagwan, R., Guha, S., Navda, V., Ramjee, R., Arora, D., Padmanabhan, V., and Varghese, G., "RadioJockey: Mining Program Execution to Optimize Cellular Radio Usage," *MobiCom*, 2012.

[82] Deng, S., "Reducing 3G Energy Consumption on Mobile Devices (Master Thesis of MIT)," 2012.

[83] Huang, J., Qian, F., Mao, Z. M., Sen, S., and Spatscheck, O., "Screen-Off Traffic Characterization and Optimization in 3G/4G Networks," *IMC*, 2012.

[84] "Weka library," `http://www.cs.waikato.ac.nz/ml/weka/`.

[85] "MATLAB statistics toolbox," `http://www.mathworks.com/products/statistics/`.

[86] Breiman, L., "Bagging Predictor," *Machine Learning*, Vol. 24, No. 2, 1996.

[87] Mahajan, R., Zhang, M., Poole, L., and Pai, V., "Uncovering Performance Differences in Backbone ISPs with Netdiff," *Proceeding of NSDI*, 2008.

[88] Gass, R. and Diot, C., "An Experimental Performance Comparison of 3G and WiFi," *Proceedings of PAM*, 2010.

[89] Arlos, P. and Fiedler, M., "Influence of the Packet Size on the One-Way Delay in 3G Networks," *Proceedings of PAM*, 2010.

[90] Tan, W. L., Lam, F., and Lau, W.-C., "An Empirical Study on 3G Network Capacity and Performance," *Proc. IEEE INFOCOM*, 2007.

[91] Wang, Z., Qian, Z., Xu, Q., Mao, Z. M., and Zhang, M., "An Untold Story of Middleboxes in Cellular Networks," *SIGCOMM*, 2011.

[92] Zhang, Y., Mao, Z. M., and Zhang, M., "Detecting Trafc Differentiation in Backbone ISPs with NetPolice," *IMC*, 2009.

[93] Dischinger, M., Marcon, M., Guha, S., Gummadi, K. P., Mahajan, R., and Saroiu, S., "Glasnost: Enabling End Users to Detect Trafc Differentiation," *NSDI*, 2010.

[94] "WindRider: A Mobile Network Neutrality Monitoring System," `http://www.cs.northwestern.edu/~ict992/mobile.htm`.

[95] Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., and Estrin, R. G. D., "Diversity in Smartphone Usage," *MobiSys*, 2010.

[96] Shepard, C., Rahmati, A., Tossell, C., Zhong, L., and Kortum, P., "LiveLab: Measuring Wireless Networks and Smartphone Users in the Field," *HotMetrics*, 2010.

[97] Qian, F., Huang, J., Erman, J., Mao, Z. M., Sen, S., and Spatscheck, O., "How to Reduce Smartphone Traffic Volume by 30%?" *PAM*, 2013.

[98] Gember, A., Anand, A., and Akella, A., "A Comparative Study of Handheld and Non-Handheld Traffic in Campus Wi-Fi Networks," *PAM*, 2011.

[99] Trestian, I., Ranjan, S., Kuzmanovic, A., and Nucci, A., "Measuring Serendipity: Connecting People, Locations and Interests in a Mobile 3G Network," *Proceedings of IMC*, 2009.

[100] Maier, G., Schneider, F., and Feldmann, A., "A First Look at Mobile Hand-held Device Traffic," *Proceedings of PAM*, 2010.

[101] Chan, M. C. and Ramjee, R., "TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation," *MOBICOM*, 2002.

[102] Chakravorty, R., Banerjee, S., Rodriguez, P., Chesterfield, J., and Pratt, I., "Performance Optimizations for Wireless Wide-Area Networks: Comparative Study and Experimental Evaluation," *MOBICOM*, 2004.

[103] Liu, X., Sridharan, A., Machiraju, S., Seshadri, M., and Zang, H., "Experiences in a 3G Network: Interplay between the Wireless Channel and Applications," *MOBICOM*, 2008.

[104] Chesterfield, J., Chakravorty, R., Crowcroft, J., Rodriguez, P., and Banerjee, S., "Experiences with Multimedia Streaming over 2.5G and 3G Networks," *Journal ACM/MONET*, 2004.

[105] Jang, K., Han, M., Cho, S., Ryu, H.-K., Lee, J., Lee, Y., and Moon, S. B., "3G and 3.5G Wireless Network Performance Measured from Moving Cars and High-speed Trains," *Proc. of ACM MICNET*, 2009.

[106] Lee, C.-C., Yeh, J.-H., and Chen, J.-C., "Impact of inactivity timer on energy consumption in WCDMA and CDMA2000," *the Third Annual Wireless Telecommunication Symposium (WTS)*, 2004.

[107] Chuah, M., Luo, W., and Zhang, X., "Impacts of Inactivity Timer Values on UMTS System Capacity," *Wireless Communications and Networking Conference*, 2002.

[108] Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., and Estrin, D., "A First Look at Traffic on Smartphones," *IMC*, 2010.

[109] Lee, C.-C., Yeh, J.-H., and Chen, J.-C., "Impact of inactivity timer on energy consumption in WCDMA and cdma2000," *Wireless Telecommunications Symposium*, 2004.

[110] Yeh, J.-H., Chen, J.-C., and Lee, C.-C., "Comparative Analysis of Energy-Saving Techniques in 3GPP and 3GPP2 Systems," *IEEE transactions on vehicular technology*, Vol. 58, No. 1, 2009.

[111] Liers, F. and Mitschele-Thiel, A., "UMTS data capacity improvements employing dynamic RRC timeouts," *PIMRC*, 2005.

[112] Ra, M., Paek, J., Sharma, A., Govindan, R., and a nd M. Neel, M. K., "Energy-delay tradeoffs in smartphone applications," *MobiSys*, 2010.

[113] Higgins, B., Reda, A., Alperovich, T., Flinn, J., Giuli, T., Noble, B., and Watson, D., "Intentional Networking: Opportunistic Exploitation of Mobile Network Diversity," 2010.

[114] Wang, Y., Lin, J., Annavaram, M., Jacobson, Q., Hong, J., Krishnamachari, B., and Sadeh, N., "A framework of energy efficient mobile sensing for automatic user state recognition," *MobiSys*, 2009.

[115] Kang, S., Lee, J., Jang, H., Lee, H., Lee, Y., Park, S., Park, T., and Song, J., "Seemon: scalable and energy-efficient context monitoring framework for sensor-rich mobile environments," *MobiSys*, 2008.

[116] Agarwal, Y., Chandra, R., Wolman, A., Bahl, P., and a nd R. Gupta, K. C., "Wireless wakeups revisited: energy management for VoIP over WiFi smartphones," *MobiSys*, 2007.

[117] Wei, Y., Chandra, S., and Bhandarkar, S., "A Statistical Prediction-based Scheme for Energy-aware Multimedia Data Streaming," 2004.

[118] Dainotti, A., Pescape, A., Rossi, P. S., Palmieri, F., and Ventre, G., "Internet traffic modeling by means of Hidden Markov Models," *Computer Networks*, Vol. 52, No. 14, 2008, pp. 2645 – 2662.

[119] "MobiPerf Open Dataset," `http://openmobiledata.appspot.com/`.