

PHYSICAL LAYER INFORMED MOBILE SERVICE OPTIMIZATION

by

Xiufeng Xie

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

Doctor of Philosophy

(Electrical and Computer Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2017

Date of final oral examination: 07/20/17

The dissertation is approved by the following members of the Final Oral Committee:

Xinyu Zhang, Assistant Professor, Electrical and Computer Engineering

Parmesh Ramanathan, Professor, Electrical and Computer Engineering

Yuhon Hu, Professor, Electrical and Computer Engineering

Aditya Akella, Professor, Computer Science

Eugene Chai, Researcher, NEC Labs America

© Copyright by Xiufeng Xie 2017
All Rights Reserved

To my parents.

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my Ph.D. advisor, Professor Xinyu Zhang, for his guidance and support throughout my graduate study. Professor Zhang has devoted a great amount of time and effort to train me as a confident Ph.D. with solid research skills. He taught me not only knowledge, but also the way to conduct research. By spending lots of time with me discussing about research ideas and providing insightful comments, he helps me to shape my research style. Besides the valuable research skills, his impressive working attitude and dedication to the duties will continue inspiring me in my future works. I would also like to thank Professor Parmesh Ramanathan, Professor Yuheng Hu, Professor Aditya Akella and Dr. Eugene Chai for serving on my thesis committee. I thank them for spending their time to review my thesis and provide valuable feedback.

I am also thankful to my fellow labmates and colleagues at UW-Madison, their help greatly contributes to this thesis. Shilin Zhu, Anfu Zhou, and Srujith Poondla helped me conduct some of the experiments especially the system evaluation under driving mobility. Teng Wei, Sanjib Sur, Chi Zhang, and Jialiang Zhang offered insightful comments to my research works. It was a great experience sharing the lab with them during the last five years.

During my internship in NEC Labs America, Dr. Eugene Chai, Dr. Karthik Sundaresan and Dr. Amir Khojastepour offered valuable guidance and suggestions on my research. I sincerely appreciate their kind help. I would also like to thank my fellow colleagues Ramanujan Sheshadri, Junguk Cho and Sharan Naribole for their help during my internship.

I would also like to thank Dr. Li Erran Li and Professor Swarun Kumar for our collaboration that contributed to the development of this thesis. In addition, I gratefully acknowledge the financial support from the National Science Foundation under grants CNS-1343363, CNS-1350039 and CNS-1318292.

Lastly but most importantly, I wish to express my gratitude to my family members. Without their unconditional support, I would have never reached this far. This dissertation is dedicated to them.

CONTENTS

Contents	iii
List of Tables	vii
List of Figures	viii
List of Abbreviations	xii
Abstract	xiv
1 Introduction	1
1.1 <i>Background & Motivation</i>	1
1.1.1 <i>A Brief Survey on Existing Solutions</i>	2
1.1.2 <i>Fundamental Limitations of Existing Solutions</i>	5
1.2 <i>Problem Statement & Solution Space</i>	6
1.3 <i>Research Contributions</i>	8
1.3.1 <i>PHY-Informed Mobile Video Streaming Optimization</i>	8
1.3.2 <i>Cellular-Link-Aware Mobile Web Loading.</i>	10
1.3.3 <i>Freeze-Precautionary Interactive Video Chat over LTE.</i>	10
1.3.4 <i>360° Panoramic Video Telephony over LTE.</i>	11
1.4 <i>Thesis Organization</i>	12
2 A Primer on the Physical Layer of LTE Cellular Network	13
2.1 <i>LTE Resource Structure</i>	13
2.2 <i>LTE Resource Allocation</i>	13
2.3 <i>LTE Cellular Link Statistics</i>	15
3 PHY-Informed Mobile Video Streaming Optimization	17
3.1 <i>Introduction</i>	17
3.2 <i>Background and Motivation</i>	20

3.3	<i>piStream Design</i>	24
3.3.1	<i>Radio Resource Monitor</i>	24
3.3.2	<i>PHY-Informed Rate Scaling</i>	30
3.3.3	<i>LRD-Based Video Adaptation</i>	33
3.4	<i>piStream Implementation</i>	38
3.5	<i>piStream Evaluation</i>	41
3.5.1	<i>Micro-benchmarks</i>	42
3.5.2	<i>System Level Test</i>	45
3.6	<i>Summary</i>	50
4	Accelerating Mobile Web Using Cellular Link Information	51
4.1	<i>Introduction</i>	51
4.2	<i>Background</i>	54
4.3	<i>Challenges for Mobile Web Over LTE</i>	55
4.4	<i>CLAW Design</i>	58
4.4.1	<i>Cellular Load Analyzer</i>	59
4.4.2	<i>Cellular-Informed Rate Control</i>	64
4.4.3	<i>Network Bottleneck Detector</i>	69
4.5	<i>CLAW Implementation</i>	71
4.6	<i>CLAW Evaluation</i>	73
4.6.1	<i>Micro-benchmarks</i>	75
4.6.2	<i>System Level Test</i>	79
4.7	<i>Discussion</i>	85
4.8	<i>Summary</i>	86
5	Freeze-Precautionary Interactive Video Chat over LTE	89
5.1	<i>Introduction</i>	89
5.2	<i>A Primer on The WebRTC Standard</i>	93
5.3	<i>Motivation</i>	94
5.3.1	<i>Measurement Setup</i>	95
5.3.2	<i>Limitations of Delay/Loss-based Congestion Control</i>	95

5.4	<i>pyChat Design</i>	99
5.4.1	<i>PHY-Assisted Video Freeze Predictor</i>	100
5.4.2	<i>Freeze-Precautionary Rate Control (FPRC)</i>	106
5.5	<i>pyChat Implementation</i>	110
5.6	<i>pyChat Evaluation</i>	112
5.6.1	<i>Micro-benchmarks</i>	112
5.6.2	<i>System Level Test</i>	116
5.7	<i>Discussion</i>	120
5.8	<i>Summary</i>	121
6	360° Panoramic Video Telephony over LTE Cellular Networks	123
6.1	<i>Introduction</i>	123
6.2	<i>Background of 360° Video</i>	126
6.3	<i>Challenges & Solution Space</i>	127
6.3.1	<i>ROI Quality Fluctuation.</i>	127
6.3.2	<i>Video Rate Control Responsiveness</i>	130
6.3.3	<i>Uplink Bandwidth Underutilization.</i>	131
6.4	<i>POI360 Design</i>	132
6.4.1	<i>System Model</i>	133
6.4.2	<i>Adaptive Spatial Compression</i>	135
6.4.3	<i>Firmware Buffer Aware Congestion Control</i>	137
6.5	<i>POI360 Implementation</i>	141
6.6	<i>POI360 Evaluation</i>	143
6.6.1	<i>Micro-benchmarks</i>	144
6.6.2	<i>System Level Test</i>	151
6.7	<i>Discussion</i>	153
6.8	<i>Summary</i>	154
7	Related Work	156
7.1	<i>Transport Layer Rate Control over Wireless</i>	156
7.1.1	<i>Split-Connection Protocols</i>	156

7.1.2	<i>Link-Layer Solutions</i>	157
7.1.3	<i>Cross-Layer Solutions</i>	157
7.1.4	<i>Network Bandwidth Estimation</i>	159
7.1.5	<i>TCP Friendliness</i>	161
7.2	<i>LTE Measurement Profiling</i>	161
7.3	<i>Adaptive Video Streaming</i>	162
7.4	<i>Improving the Mobile Web Performance</i>	163
7.5	<i>Interactive Video Chat over Cellular Networks.</i>	164
7.6	<i>360° Panoramic Video Streaming</i>	165
8	Conclusion	167
8.1	<i>Concluding Remarks</i>	167
8.2	<i>Discussion & Future Works</i>	168
8.2.1	<i>BS-Centric Solutions</i>	168
8.2.2	<i>Physical-Layer Informed Mobile Edge Computing</i>	169
	References	171

LIST OF TABLES

1.1	Summary of the contributions.	9
3.1	Implemented DASH algorithms.	41
4.1	CLAW and benchmark schemes.	74
6.1	Mapping PSNR to Mean Opinion Score (MOS).	145

LIST OF FIGURES

2.1	LTE downlink resource structure.	14
3.1	Throughput measurement depends on the traffic demand.	22
3.2	Throughput-based DASH adaptation converges slowly to bandwidth. .	22
3.3	Existing DASH adaptation cannot follow the bandwidth variation. . .	24
3.4	Poor bandwidth predictions drain out client's buffer and cause video stalls.	24
3.5	piStream system architecture.	25
3.6	LTE resource allocation example for a two antenna BS.	25
3.7	Energy-based resource utilization monitor is feasible but challenging. .	27
3.8	Reference signals can capture frequency selectivity and antenna diversity.	27
3.9	PRB utilization versus bandwidth utilization.	31
3.10	PHY-informed rate scaling example.	31
3.11	A simple example for the bandwidth sojourn time.	35
3.12	A Hurst parameter 0.75 indicates the long range dependency.	35
3.13	Sojourn time distribution fits a Pareto distribution.	35
3.14	K-S test statistic. (difference between the empirical and the fitted CDF)	35
3.15	piStream implementation.	39
3.16	Our resource monitor RMon outputs accurate resource utilization. . .	42
3.17	PIRS component alone improves the performance.	42
3.18	Trace-driven emulation with perfect knowledge of current & historical (but not future) bandwidth.	44
3.19	LVA has the lowest overshoot prob.(18% lower than the runner-up) . .	46
3.20	Average video bitrate & video stall ratio for a static client.	46
3.21	piStream outperforms other DASH algorithms in mobile environments.	48
3.22	Performance with multiple DASH clients.	49
4.1	Page loading latency over LTE is affected by both RTT and packet losses.	56

4.2	A snapshot of TCP sender cwnd for 5 back-to-back loading events visiting the same static HTTP/2 webpage (with browser cache disabled).	58
4.3	CLAW system architecture.	59
4.4	LTE reference signal under different resource allocation status (example shows a 2-antenna BS).	60
4.5	CLAW models LTE downlink resource allocation using a phone's locally sensed RSRQ metric. This figure compares the model with measurement (error bars show std.).	63
4.6	A running instance of the cellular-informed hybrid bandwidth exploration over an LTE channel saturated by existing users.	67
4.7	CLAW detects wireline bottleneck based on resource allocation pattern during bandwidth exploration.	70
4.8	The network bottleneck detector in CLAW.	70
4.9	A microscopic demonstration of how CLAW's cell load analyzer can reliably track the cell load.	76
4.10	(a) CLAW server explores the network bandwidth faster than TCP Cubic (b) CLAW client is thus allocated more LTE PHY resources than TCP Cubic.	77
4.11	Web loading latency for a bare-bone Web page with a 2MB image. . . .	78
4.12	Packet latency (RTT) during loading Web page with a 2MB image . . .	78
4.13	CLAW's bottleneck detector handles the rare wireline bottleneck. . . .	78
4.14	Profiling Alexa top 200 Website sizes.	78
4.15	CLAW over different websites (number in circle is the percentage of latency reduction over Cubic).	80
4.16	PLT under different transport protocols.	81
4.17	CLAW outperforms benchmark schemes under various scenarios. . . .	87
4.18	Scalability test.	88
4.19	TCP coexistence and multi-user scenarios.	88
5.1	Customized testbed for evaluating real-time congestion control protocols for interactive video.	94

5.2	Snapshot of video chat statistics over an LTE downlink (Long-term measurements are in §5.4.1).	96
5.3	When sender streams live video over LTE uplink, every video freeze comes <i>after</i> a cell load surge, which is also observed in the downlink scenario (Figure 5.2).	98
5.4	pyChat workflow.	99
5.5	PHY resource utilization increases with traffic demand.	102
5.6	PHY resource utilization increases under lower channel quality.	102
5.7	Distribution of the time interval between cell load surge and its following video freeze.	102
5.8	Distribution of the time interval between consecutive cell load surges, and that between video freezes.	102
5.9	pyChat's PHY assisted video freeze predictor.	104
5.10	PHY cell load follows the rate increasing.	113
5.11	Accuracy of the video freeze predictor.	113
5.12	FPRC adapts the traffic bitrate following the cell load to prevent video freezes.	115
5.13	FPRC increases bitrate when unlikely to freeze thus achieves higher video quality than GCC.	116
5.14	FPRC reduces bitrate when video tends to freeze thus achieves lower video delay than GCC.	116
5.15	Video freezing ratio in idle channel.	117
5.16	Video freezing ratio in busy channel.	117
5.17	Video quality&delay in idle channel.	117
5.18	Video quality & delay in busy channel.	117
5.19	Video freezing ratio under driving mobility.	119
5.20	Video quality&delay under car driving.	119
5.21	Vanilla WebRTC freezes when coexisting with TCP flow. pyChat does not.	120
5.22	Both pyChat and Vanilla WebRTC do not affect the TCP throughput. .	120

6.1	Example application scenario for portable interactive 360° video telephony over cellular networks.	124
6.2	ROI-based compression in end-to-end 360° video streaming.	127
6.3	Problem for ROI-based compression: viewer ROI change causes user-perceived quality fluctuation.	128
6.4	Trade-off between compression ratio and user-perceived quality stability during ROI update.	129
6.5	(a) Conventional end-to-end rate control. (b) Cellular uplink congestion control for interactive VR.	130
6.6	The relation between uplink firmware buffer occupancy and TBS. . . .	131
6.7	Uplink firmware buffer level under WebRTC default rate control. . . .	131
6.8	POI360 system architecture.	132
6.9	POI360 video compression model.	133
6.10	POI360 rate control model.	133
6.11	ROI mismatch time.	136
6.12	Comparing video quality under POI360 compression and benchmark algorithms.	146
6.13	Short-term ROI compression level variation (std. value of ROI compression level in a 2s window).	147
6.14	360° video frame delay under POI360 compression and benchmark algorithms.	148
6.15	360° video freeze ratio under POI360 compression and benchmark algorithms.	149
6.16	FBCC is allocated higher uplink TBS than GCC as it maintains a proper firmware buffer level.	150
6.17	Comparing POI360 performance with FBCC or GCC (WebRTC's default rate control).	151
6.18	System-level POI360 evaluation.	155

LIST OF ABBREVIATIONS

BS Base Station

UE User Equipment

ACK Acknowledgement

DASH Dynamic Adaptive Streaming over HTTP

RTT Round Trip Time

QoE Quality of Experience

RSSI Receive Strength Signal Indicator

RSRP Reference Signal Received Power

RSRQ Reference Signal Received Quality

MCS Modulation and Coding Scheme

RE Resource Element

PRB Physical Resource Block

TBS Transport Block Size

PDCCI Physical Downlink Control Channel

DCI Downlink Control Information

OFDM Orthogonal Frequency-Division Multiplexing

RLC Radio Link Control

ROI Region of Interest

FoV Field of View

VR Virtual Reality

HMD Head Mounted Display

ECN Explicit Congestion Notification

ABSTRACT

PHYSICAL LAYER INFORMED MOBILE SERVICE OPTIMIZATION

by

Xiufeng Xie

Under the supervision of Professor Xinyu Zhang

At the University of Wisconsin-Madison

In recent years, the proliferation of smartphones and massive deployment of 4G LTE cellular infrastructures have triggered a plethora of mobile network services like video streaming and ubiquitous Web browsing. However, current mobile services still severely underperform their true potential despite the high link bandwidth provided by LTE. Such vast performance gap roots in the poor coordination between the wireless physical layer designed by electrical engineers, and the upper-layer network protocols developed by computer scientists. To bridge this gap, the central theme of our research is the physical layer informed mobile service optimization: In contrast to conventional solutions treating both the cellular links and application scenarios as black-boxes, we propose novel protocols to boost the mobile service performance in dedicated application scenarios by harnessing the LTE cellular link statistics extracted from the physical layer. Following this design principle, our research explores the unique challenges and opportunities in 4 important categories of mobile network services, including video streaming, Web loading, interactive video chat, and 360-degree video telephony, and present novel system designs that substantially boost the performance over conventional solutions. We implement prototypes running in real time over commercial LTE cellular networks to validated the feasibility, and the experiment results demonstrate the superior performance of our proposed systems.

Chair: Xinyu Zhang

1 INTRODUCTION

1.1 Background & Motivation

Mobile services are becoming the essentials of modern life: Mobile Web browsing fills the fragmented time when waiting in the bus stop, train station, bank, or restaurant. It is not surprising that more websites are viewed on mobile devices than desktops nowadays [1]; Video streaming dominates the mobile network, accounting for 70% of the mobile Internet traffic, with a compound annual growth rate of 78% [2]; Meanwhile, mobile video chat is also rapidly gaining popularity — now over 1 billion users chat with Skype over mobile devices [3]. The solid connection between such ubiquitous mobile services and the modern society brings endless thirst for the mobile network performance. However, despite the high-speed wireless access with 12 Mbps average throughput [4] and 600 Mbps peak data rate [5] of 4G LTE, the dominating cellular network technology with more than 80% U.S. market share in 2016, mobile service performance remains frustrating compared to the true potential of the high cellular link speed. For instance, mobile video streaming with only 2~3 Mbps bitrate still freezes frequently [6, 7], while loading a small mobile webpage can easily cost several seconds over LTE cellular networks [8].

The huge gap between the frustrating performance of mobile services and their true potential over the high-speed cellular network roots in the dynamic wireless channel capacity and high cellular link latency. First, the capacity of wireless channel is always varying due to the dynamic wireless environment and user mobility, but the rate control algorithms of existing mobile services are unable to follow such link dynamics. Furthermore, the wireless spectrum sharing and complicated mobile network architecture cause high latency over cellular links, which further degrades the responsiveness of existing mobile services to adapt their traffic to the wireless network variations. As a result, even the cellular link bandwidth is getting close to the wireline network bandwidth, many mobile services directly migrated from their desktop versions still suffer from poor performance, as many crucial design assumptions are made for wireline networks, *e.g.*, low round-

trip latency and stable network bandwidth, which no longer hold over wireless networks. The unique challenges of dynamic bandwidth and high latency call for wireless-specific solutions.

1.1.1 A Brief Survey on Existing Solutions

Although many mobile services gain popularity only in recent years thanks to the high data rate provided by the 4G LTE cellular network, decades of tremendous research efforts have been devoted to improving the data transport over wireless networks, but the wireless performance issues remain as open challenges till now. In what follows, we briefly discuss the rationale and limitation of the related works in this area, and a more detailed survey is provided in Chapter 7.

(i) *Split-connection protocols* like indirect-TCP [9] are proposed dating back to mid-1990s, which break the end-to-end connection into two separated paths at the wireless Base Station (BS). In this way, the remote sender can be shielded from the local wireless channel dynamics and losses, while the intermediate sender at the BS can focus on the wireless loss recovery. However, as revealed in [10], split-connection TCP protocols suffer from several fundamental limitations: It breaks the end-to-end semantics of TCP Acknowledgment (ACK), causes timeout at the intermediate sender thus forces the remote sender to stall, and slows down the handoff procedure between wireless BSs.

(ii) Another way to improve TCP performance over wireless is the *link-layer solutions* [11, 12, 10], which allows the end-to-end TCP connection to pass through the lossy wireless link, and use the retransmissions in the lower link layer to combat the wireless losses. The basic concept of such link-layer solutions is proposed more than 30 years ago, and nowadays the LTE cellular networks already have the Radio Link Control (RLC) layer [13], which uses Automatic Repeat Query (ARQ) in the link layer to handle local retransmission and error correction. Although such solutions have effectively addressed most wireless loss-related problems over cellular links, the performance issues of mobile services nowadays are caused by the dynamic bandwidth and high RTT, but existing link layer solutions are unable

to address such challenges.

(iii) *End-to-end cross-layer rate control.* Existing mobile services are unable to adapt to the highly dynamic wireless network due to the poor coordination between the wireless physical layer designed by electrical engineers, and the upper-layer network protocols and applications developed by computer scientists. As a result, a natural solution is cross-layer rate control which enables better coordination between the different layers of the network stack. The past 30 years witness an enormous amount of research works following this concept. For instance, explicit loss notification (ELN) enables the remote sender to distinguish between congestion related loss and wireless related loss [14]; Other cross-layer works [15] expose the physical and MAC layer knowledge of the wireless medium to higher layers to facilitate efficient network scheduling and resource allocation. In recent years, with the prosperity of 3G/4G cellular networks and data-hungry mobile services, estimating the highly dynamic cellular link bandwidth becomes a key challenge, and the concept of cross-layer wireless protocols proposed decades ago has been revisited. For instance, recent work CQIC [16] estimates the physical-layer cellular channel capacity based on the wireless signal strength and sets it as the sending rate of traffic flow.

(iv) *End-to-end network bandwidth estimation.* The wireless link bandwidth is highly dynamic, and hence adapting the traffic rate to fit the instantaneous link bandwidth is crucial for mobile services. Unfortunately, conventional TCP uses packet loss or delay as the indicator of congestion, thus the corresponding rate control fails to track the varying wireless link bandwidth. Furthermore, the LTE BS is equipped with deep buffer and robust link-layer retransmission mechanisms, thus packet loss becomes rare over the LTE link. Ironically, “no packet loss” triggers a new problem over LTE, as the deep buffer at the BS masks the congestion and neutralizes loss-based TCP congestion control. This problem is also known as bufferbloat [17]. To meet these challenges, TCP Westwood [18] estimates the end-to-end network bandwidth by monitoring the rate of returning ACKs, thus does not rely on the packet losses for rate control. Meanwhile, some other works [19] use packet pairs to probe the wireless link bandwidth, which is typically the bottleneck

on a network path. In recent years, several works propose building stochastic model from historical network statistics to facilitate rate control over cellular networks: Sprout [20] learns a stochastic throughput model based on its historical throughput statistics over LTE networks, and use that to control the traffic bitrate. Similarly, Verus [21] learns a delay-based model for rate control. Since the wireless link is typically the network bottleneck, to achieve accurate bandwidth estimation, most of these related works [16, 20, 21] choose to let the mobile client side estimate the end-to-end bandwidth. In other words, they follow the principle of client-initiated rate control, in which the client needs to feedback the bandwidth estimation to the sender. Unfortunately, the high round-trip latency of cellular network can severely slow down this feedback process. Furthermore, over cellular networks, these bandwidth estimation mechanisms fail due to a fundamental limitation – the unawareness of cellular physical layer rate control. The transport layer TCP rate control and the cellular physical layer rate control are nested in the network stack, which can form vicious cycles in the rate adaptation and cause network bandwidth underutilization. We will further discuss this in Chapter 7.

(v) *DiffServ networks.* An alternative solution to enhance mobile service performance is the DiffServ [22, 23, 24] network infrastructure. For example, video over LTE (ViLTE) [25] can explicitly reserve the LTE physical layer resources for video chat traffic and thus guarantee the Quality of Service (QoS). However, such solutions are unscalable because the wireless spectrum resources are always limited whereas application traffic demand is constantly growing. Novel mobile network services, e.g., mobile virtual reality and mobile drone control, are emerging and will become pervasive in the near future. It is impossible to reserve network resource for all kinds of potential application demands. If we reserve resources for each of them instead of using the shared data channel, the spectrum can easily drain out and it will be impossible to add new services. Therefore, using the shared data channel is the more scalable solution under the spectrum scarcity.

1.1.2 Fundamental Limitations of Existing Solutions

From the above survey, we can observe that existing works still fail to address the practical performance issues of mobile services due to two fundamental limitations: (i) *Ignoring the unique features of the cellular physical layer.* Existing cross-layer works estimate the cellular link bandwidth simply based on the wireless channel quality. Unfortunately, the cellular physical layer is much more complicated in practice. For instance, the physical-layer details like radio resource management and packet queuing in the firmware buffer are ignored in such existing works, although they play vital roles in the performance of mobile services: The radio resource allocation status affects the available network bandwidth of our user as the physical channel is shared by all users in this cell; The firmware buffer level affects both the resource allocation and packet delay. Without such crucial information, existing works only achieve “*wireless channel awareness*”, but mobile service optimization calls for “*physical layer awareness*”.

(ii) *Ignoring the unique features of the mobile service applications.* Different mobile network services can have very different traffic patterns, latency requirements or application scenarios. However, most of the existing works focus on developing a general data transport protocol over cellular networks, without considering the unique features of the specific upper-layer network services to support. As a result, such a general data transport protocol may fit some mobile services but prone to underperform or even fail under some other important services. For instance, both on-demand video streaming and interactive video chat are important network services to be provided to mobile users, but the control algorithms for these two services need to adopt very different strategies: When optimizing on-demand video streaming, we can focus more on the throughput performance or video quality, as the client can have a large playback buffer and pre-buffer minutes of video content ahead. In contrast, end-to-end latency is vital for interactive video chat, as its application scenario naturally does not allow large video buffer, thus we need to sacrifice video quality to guarantee the video smoothness and interactivity when cellular link condition gets worse. In other words, existing

works are “*network-performance-oriented*”, but mobile service optimization should be “*service-performance-oriented*”.

As discussed above, the poor cooperation between application-layer mobile service, network-layer protocol, and cellular physical layer prevents present-day mobile services from exploiting the full potential of the high speed cellular link. Unfortunately, existing solutions lack the awareness of the cellular physical layer operations, and ignore the unique features of the mobile services to support, thus fail to address these challenges. Unless they are resolved in a principled way, mobile services will continue to underperform even if the wireless network capacity can be upgraded in future.

1.2 Problem Statement & Solution Space

As discussed above, the unawareness of the cellular link physical-layer status and the ignorance of the unique features of the mobile service have jointly plagued the mobile service performance over present-day LTE cellular networks.

To break through these fundamental limitations, we propose the research endeavor of this dissertation – *physical layer informed mobile service optimization*. On one hand, we harness the cellular link information extracted from the LTE physical layer to guide the upper-layer network protocols and mobile service applications, hence enables more efficient and timely adaptation to the highly dynamic wireless network environment. On the other hand, we identify and investigate the unique challenges in optimizing different categories of mobile network services, and then address them by leveraging the extra knowledge of cellular link statistics, thus overcomes the limitations of using a general cross-layer data transport solution.

Problem statement: Given the widely deployed 4G LTE cellular network infrastructure, instead of rebuilding the entire cellular network architecture, this dissertation focuses on the following question –

Can we build end-to-end adaptation algorithms to substantially boost the performance of mainstream mobile services by leveraging the cellular link infor-

mation extracted from the LTE physical layer with minimum or no hardware modification to the user equipment?

Solution space: We divide our solution space into two essential parts, *cellular physical layer monitoring*, and *mobile service data rate control*. Specifically, for the cellular physical layer monitoring, in our first research work along this direction, we employ USRP software radio to sniff the cellular channel spectrum in real time thus measure the physical layer resource utilization. Since such a solution requires hardware modification at the client side, in our following works, we leverage the phone’s diagnostic interface to extract the cellular link statistics thus achieves compatibility to commodity phones. However, the diagnostic interface only provides energy-based channel measurements, and does not directly report the crucial information like the resource utilization. To address this problem, by digging deeply in the LTE physical-layer resource structure, we propose the first analytic model that can compute the cell-wide resource utilization using such energy-based channel measurements as input.

For the mobile service data rate control, we adopt a service-oriented solution approach. Specifically, we identify the unique problems when the rate control algorithms of 4 categories of mainstream and potential mobile services operate over 4G LTE cellular networks, including *adaptive video streaming*, *mobile Web loading*, *interactive video chat* and *360° video telephony*, and then develop dedicated cellular-link-informed data rate control to address the unique problem for each of the categories.

It is worth noting that although this dissertation targets at 4G LTE, the most popular cellular network technology for now, the principle of our designs can be potentially applied over future cellular networks. Specifically, our proposed designs focus on addressing the challenge of dynamic network bandwidth, the conflict between the high network latency and delay-sensitive applications, and the poor interaction between end-to-end rate control and cellular physical layer rate control. These challenges will persist in future 5G cellular networks, where our solutions can still be adopted. In the next section, we introduce our research works following

the solution approaches discussed above.

1.3 Research Contributions

Table 1.1 summarizes our works on physical layer informed mobile service optimization. In what follows, we describe the challenges, rationales and contributions of these works in more detail.

1.3.1 PHY-Informed Mobile Video Streaming Optimization

Our research starts by optimizing the dominating mobile service, video streaming, which accounts for 70% of the present-day mobile Internet traffic but suffers from severe underutilization of the LTE link capacity. Through extensive measurements, we found that such bandwidth underutilization roots in a vicious cycle that happens when the widely adopted Dynamic Adaptive Streaming over HTTP (DASH) protocol meets the LTE cellular infrastructure: DASH adapts the server’s video quality or bitrate based on the client-side network throughput measurement. However, the LTE BS refrains from allocating idle physical-layer spectrum resources to the client if current video traffic load is low, thus the video client measures a low throughput and informs the sender to use a low video bitrate accordingly, which in turn enforces a conservative LTE resource allocation and low throughput at client side. This essentially forms a vicious cycle in the DASH adaptation logic, and the video stream is trapped in the region of low quality even the cellular link provides high network bandwidth.

To break out of the vicious cycle, we propose an end-to-end adaptive video streaming system called piStream. By using a software radio to monitor the LTE physical channel, piStream allows a client to monitor the utilization status of LTE radio resources under current video bitrate, and inform the sender to promote the video bitrate/quality accordingly to exploit currently unused resources. Comparison with state-of-the-art adaptive streaming protocols demonstrates that piStream

Chapter	System Name	Mobile Service Type	Unique Challenges	Physical-Layer Information
Chap. 3	piStream	On-demand video streaming	Cellular network bandwidth underutilization caused by the vicious cycle in video adaptation over LTE	Cellular downlink spectrum
Chap. 4	CLAW	Web page loading	Cellular network bandwidth underutilization caused by sluggish TCP bandwidth exploration over LTE	PRB utilization, MCS, TBS, RSRQ, RSRP
Chap. 5	pyChat	Interactive video chat	Responsive adaptation for the network congestion	Cellular downlink spectrum
Chap. 6	POI360	360° video telephony	Cellular uplink congestion & 360° video compression over LTE	Cellular uplink firmware buffer level, TBS

Table 1.1: Summary of the contributions.

can effectively utilize the LTE bandwidth, achieving high video quality with minimal stalling rate.

1.3.2 Cellular-Link-Aware Mobile Web Loading.

In our next piece of work, we focus on accelerating the sluggish mobile Web loading over 4G LTE cellular networks. In contrast to the heavy and continuous video traffic investigated in our first work, Web loading has light and bursty traffic pattern. However, mobile Web also suffers from severe underutilization of the cellular link capacity, but for a very different reason. The root cause lies in the inefficient TCP bandwidth exploration: To avoid congestion, the Web server starts from a low TCP sending rate and gradually speeds up until reaching the network bandwidth. Unfortunately, this sluggish probing procedure always fails to converge to the bandwidth within the short Web loading session (typically below 2 seconds), thus most of the cellular link bandwidth are wasted during Web loading. To overcome the limitation of the sluggish TCP bandwidth exploration, we propose cellular-link-aware Web (CLAW), which harnesses the LTE cellular link statistics readily available on commodity phones to gauge the LTE channel resource utilization, and then directly scales the sending rate to the potential network bandwidth supported by currently available resources. Consequently, it can fully exploit the available LTE bandwidth almost within one RTT, and hence significantly reduces page loading latency. We have prototyped CLAW on commodity LTE phones. Across a wide range of experimental settings, CLAW consistently reduces Web loading latency by more than 30%, compared to classical TCP variants and state-of-the-art congestion controls for cellular networks.

1.3.3 Freeze-Precautionary Interactive Video Chat over LTE.

We then switch our attention to another mainstream mobile service, interactive video chat, which seems like on-demand video streaming at first glance, but in fact bears more challenging application scenarios and strict performance requirements. Specifically, to guarantee the interactivity, the end-to-end video frame latency must

be small, which prevents the usage of large video buffers at the client to absorb the variation of the network conditions. As a result, interactive video chat is more prone to the cellular link dynamics, and responsive rate control is the key to meet this challenge. Unfortunately, despite decades of research, interactive video rate control over cellular networks remains an open challenge. Existing solutions typically use a high packet delay or loss as the indicator of under-performing network status to trigger video bitrate adaptation. However, such strategies suffer from frequent video stalls due to the inability to adapt to the highly dynamic cellular links.

Can we prevent video freeze by adapting in advance, rather than reacting after its detrimental effect occurs? Our work provides an affirmative answer with a new design principle called precautionary adaptation. The key observation is that video freezing events are always preceded by a surge in LTE resource utilization, due to the interplay between channel condition, scheduling and video application. Therefore, freeze-precautionary congestion control algorithm leverages the radio resource utilization as indicator of potential freezing events in video telephony. In this way, our system can achieve close to zero video freezing rate while maintaining similar quality to existing interactive video chat systems.

1.3.4 360° Panoramic Video Telephony over LTE.

Finally, we investigate a timely but challenging problem: can we stream real-time interactive 360° videos across existing LTE cellular networks, so as to trigger new applications such as ubiquitous 360-degree video chat or flying a drone remotely based on real-time panoramic cockpit view feedback? The key challenge for such panoramic video telephony services is the strict latency requirement, which calls for responsive rate control to avoid queue building up in the network and thus minimize the latency. To meet this challenge, we leverage the diagnostic statistics on commodity phones to promptly detect early-stage congestion on the cellular link and throttle the rate correspondingly, hence significantly boosting the rate control responsiveness. In addition, our measurements revealed that conventional 360° video compression algorithms can trigger unexpected severe quality fluctuations in

the viewer’s region-of-interest (ROI) when running over cellular networks with high round-trip time, and our solution to this issue is an adaptive compression scheme, which dynamically adjusts the compression strategy to stabilize the video quality within ROI under various user input and network conditions. Extensive field tests for our real-time prototype validate its effectiveness in enabling panoramic video telephony over the highly dynamic cellular networks.

1.4 Thesis Organization

The remainder of this dissertation is organized as follows. In Chapter 2, we introduce the physical layer resource structure of the LTE cellular network, which is the foundation of all our research works discussed in this dissertation. In Chapter 3, we introduce our PHY-informed mobile video streaming system that fully exploits the LTE cellular link bandwidth. In Chapter 4, we propose cellular-link-aware Web (CLAW) that accelerates mobile Web services using cellular link information. In Chapter 5, we describe our design of freeze-precautionary rate control for interactive video chat over LTE networks, which leverages physical-layer resource utilization as the indicator of potential congestion and video freezes. In Chapter 6, we introduce our mobile 360° video telephony system running over LTE cellular network, which leverages cellular link information to facilitate the real-time smooth streaming of interactive 360° video. Chapter 7 provides a brief survey on related works. Finally, Chapter 8 summarizes the contribution of this dissertation and proposes future work.

2 A PRIMER ON THE PHYSICAL LAYER OF LTE CELLULAR NETWORK

In this chapter, we introduce the basics of the LTE physical-layer resource structure, resource allocation policy, and cellular link statistics, which serve as the foundation of the rest of this dissertation.

2.1 LTE Resource Structure

Figure 2.1 illustrates the LTE downlink radio resource structure. LTE BS allocates radio resources to clients in the unit of time and frequency blocks. In the time domain, the channel is divided into *frames* of 10ms length, each comprising 10 *subframes* of 1ms length. A subframe can be further divided into two 0.5ms *slots*. In the frequency domain, The BS transmits each subframe using OFDM (orthogonal frequency-division multiplexing), which divide an entire spectrum band (5MHz or 10MHz) into small subcarriers. In this way, the available radio resources are divided into grids in both time and frequency domain, where each grid cell is called a *resource element* or RE. As the smallest resource unit, one RE spans one OFDM symbol (66.7 μ s) in time and one OFDM subcarrier (15kHz) in frequency. The BS allocates radio resources at the granularity of *physical resource block* (PRB), which comprises multiple REs. Typically, each PRB contains 7×12 REs as it spans across half-a-subframe (*i.e.*, 0.5 ms or 7 OFDM symbols) in time and 12 OFDM subcarriers (*i.e.*, 180 kHz or 12 subcarriers) in frequency domain. LTE specifies the number of PRBs $N_{\text{PRB}} = 25$ for a 5 MHz band and $N_{\text{PRB}} = 50$ for 10 MHz band.

2.2 LTE Resource Allocation

It is worth noting that not all downlink resource elements are used for data transmission. To name a few relevant ones,

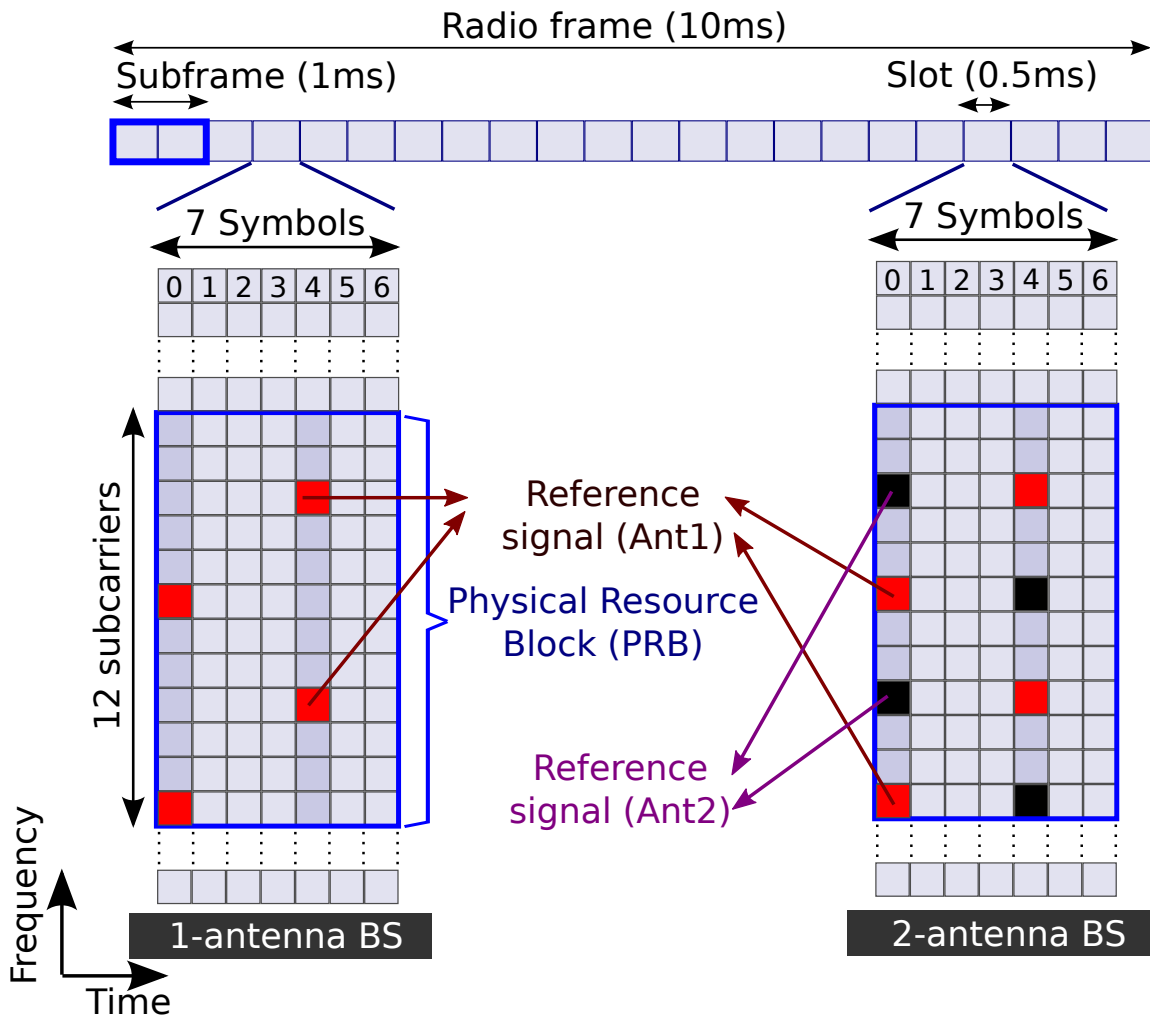


Figure 2.1: LTE downlink resource structure.

- (i) Cell-specific reference signals (*CRS*) are scattered over time and frequency domain, which helps the UE acquire the reference point for the downlink cell power.
- (ii) Physical Downlink Control Channel (*PDCCH*) carries the downlink resource assignment for UEs.
- (iii) Physical Control Format Indicator Channel (*PCFICH*) carries the Control Frame Indicator (*CFI*), which denotes the number of symbols (typically 1 to 3) allocated for control channel in each downlink subframe.

The BS dynamically allocates non-overlapping sets of PRBs to different UEs, depending on their channel conditions and traffic demands. The allocation strategy is vendor-specific. However, typical BS enforces some form of proportional fairness, which ensures a balance between UEs with the best channel quality and those consuming the most resources. The per-UE resource assignment is conveyed to its target UE over a dedicated downlink control channel (PDCCH), and each UE only decodes the control message targeting itself. Each control message, called downlink control information (DCI), contains multiple fields describing the resource allocation, *e.g.*, the number and location of resource blocks allocated and the modulation and coding scheme (MCS).

2.3 LTE Cellular Link Statistics

We then introduce the cellular link statistics that characterize the status of PHY resources. From the baseband chipset's diagnostic interface, the user equipment (UE), *e.g.*, smartphone, can access these statistics. In addition, each client can read the BS's configuration parameters, such as frequency bandwidth and the number of antennas N_{ant} , from a shared control channel.

Receive Strength Signal Indicator (RSSI): the total received power across the entire downlink frequency band, which is measured from the OFDM symbols carrying *reference signals* (symbol 0 & 4 as shown in Fig. 2.1).

Reference Signal Received Power (RSRP): average per-RE received power on the REs carrying reference signals, *i.e.*, the sum power of all reference REs divided by the number of reference REs.

Reference Signal Received Quality (RSRQ): the ratio between RSRP and RSSI, *i.e.*,

$$Q = N_{\text{PRB}} P / I \quad (2.1)$$

where Q , P and I denote RSRQ, RSRP, and RSSI, respectively. N_{PRB} is the number of PRBs across the downlink spectrum band, specified to $N_{\text{PRB}} = 25$ and 50, for 5MHz and 10MHz downlink band, respectively [5].

Modulation and Coding Scheme (MCS): an index to the combination of modulation type, coding rate and number of spatial streams (for MIMO transmission), which can be translated into bitrate through a look-up table.

Transport Block Size (TBS): In LTE system, transport block is the basic unit of data in the physical layer. One transport block is generated in each TTI (Transmission Time Interval), or 1ms. TBS is the number of bits carried in one transport block, which depends on both the number of physical resource blocks (N_{PRB}) assigned to the UE and the MCS.

3 PHY-INFORMED MOBILE VIDEO STREAMING OPTIMIZATION

3.1 Introduction

Mobile video streaming service has witnessed a surge in the past few years, accounting for 70% of the mobile Internet traffic, with a compound annual growth rate of 78% [2]. To match such growing traffic demand, LTE cellular networks have been massively deployed. However, despite the 600 Mbps peak downlink bitrate of LTE, which is almost $10\times$ over 3G, the quality of experience (QoE) of mobile video streaming service over LTE remains unsatisfactory. A recent world-wide measurement survey [6] reveals that, even in regions with wide LTE coverage, LTE only increases video quality by 20% over 3G. On the other hand, the average stalling time remains 7.5 to 12.3 seconds for each minute's mobile video playback [6].

These two effects are seemingly contradictory: the video streaming application seems to severely underutilize the LTE bandwidth, yet stalling occurs due to bandwidth overestimation. However, based on a microscopic measurement study (§3.2), we identify a single root cause behind: the inability of the streaming application to track the network bandwidth which is affected by downlink traffic dynamics at the BS.

Our measurement focuses on the popular HTTP based adaptive streaming protocols (collectively called *DASH*) [6]. A DASH client adaptively requests video segments of certain quality (size) from the server, to ensure the best-quality segment can be downloaded in time for playback. Due to high probing cost, a DASH client needs to infer the *current bandwidth* implicitly from the throughput of its video segments. However, the throughput value may largely *underestimate* the bandwidth unless a video segment can saturate the client's end-to-end bandwidth. On the other hand, DASH uses historical bandwidth records to predict future bandwidth, which occasionally leads to *overestimation*, and hence video stalling, as the LTE network condition fluctuates. In either aspect, the DASH client is left to suffer from the worst impact of both underestimation and overestimation of the end-to-end network bandwidth.

To meet the above challenges, we dig for the network bandwidth utilization from the LTE physical layer. The well organized radio resource structure distinguishes LTE from most other wireless communication systems like WiFi or Bluetooth. Given a particular LTE cell and time duration, the total amount of radio resources available for downlink transmission is always known. Moreover, the end-to-end network bandwidth is typically bottlenecked by the access link bandwidth due to the architectural property of cellular networks [26, 27]. These unique features bring new angles to remedy the bandwidth underutilization problem.

Following the design principle of PHY-informed mobile service optimization discussed in Chapter 1, we propose piStream, a client-centric video adaptation framework that fully exploits the aforementioned features to enhance adaptive video streaming over LTE. piStream is compatible with the MPEG-DASH standard [28], but tailored for LTE clients. From a high level, piStream enables an LTE client to monitor the cell-wide physical layer resource utilization status and map it to the utilization ratio of network bandwidth, the client then correspondingly scales up its current throughput measurement to obtain the potential network bandwidth.

piStream's PHY resource monitor scheme aims to be accurate, efficient, and deployable on LTE user equipment (UE). A straightforward way to obtain the resource allocation status is to decode the messages on the control channel¹[29, 30]. However, this approach falls short of efficiency and scalability. It requires each UE to keep monitoring the control channel and decoding the control messages to all other UEs, for which the error detection mechanism specified in current LTE standard cannot take effect (§3.3.1). piStream addresses this challenge by taking advantage of the well organized LTE resource structure. Instead of decoding the downlink control messages dedicated to all other UEs, a UE only needs to inspect the signal energy on LTE radio resources to assess their occupancy.

After acquiring the amount of surplus radio resources at the physical layer, piStream scales up its measured throughput accordingly to obtain an estimation

¹In LTE, UEs within the same cell share frequency/time resources allocated by the BS in a centralized manner. Per-UE resource assignment is conveyed to each target UE over a dedicated control channel.

of the potential network bandwidth it can leverage, which is then exposed to the application layer to facilitate video rate adaptation. In this way, piStream overcomes a common limitation of a wide range of DASH protocols that are slow at exploring unused bandwidth [31, 32, 33].

From the DASH application perspective, to maximize bandwidth utilization while minimizing video stalling rate, the adaptation logic ideally needs to predict the future bandwidth evolution [34, 35], which is challenging for an LTE UE. Instead of forecasting the elusive bandwidth value, piStream takes advantage of the invariant burstiness of traffic patterns in packet switching networks [36, 37]. It estimates how likely the aggregated downlink traffic (or resource usage), and hence available bandwidth, is to remain at a similar level as the current one. It then makes a probabilistic decision to maximize video quality while minimizing the risk of stalling.

We validate the piStream design by tethering a software-radio that implements the PHY-informed bandwidth estimation mechanisms, to an LTE smartphone that implements the application-layer video adaptation scheme. Our piStream client prototype can directly play video in real-time from any server that follows the industrial MPEG-DASH standard [28]. We benchmark piStream’s performance against a standard DASH player from GPAC², and three state-of-the-art DASH schemes that have demonstrated superior performance over commercial DASH players. These schemes include buffer-based adaptation (BBA [32]), optimization-based adaptation using historical throughput (FESTIVE [33]), and TCP-like bandwidth probing (PANDA[38]). Under a variety of experimental settings including time, location and mobility, piStream outperforms all other DASH schemes by achieving higher video quality and lower/comparable video stalling rate. Under typical static indoor environments, piStream achieves around $1.6\times$ video quality (bitrate) gain over the runner-up (BBA) while maintaining a low video stalling rate close to 0%.

To our knowledge, piStream represents the *first* protocol to facilitate LTE adaptive video streaming using PHY-informed bandwidth estimation, which is evalu-

²GPAC [31] is a popular (over 3300000 homepage visits during 2014) open-source framework to generate and play DASH video data sets following the MPEG-DASH standard [28].

ated via a *real-time* implementation. The specific contributions of piStream can be summarized as follows:

(i) We design a lightweight PHY-layer resource monitor (§3.3.1) and rate scaling mechanism (§3.3.2) that enables an LTE client to efficiently estimate available bandwidth, and facilitate application-layer protocols;

(ii) We propose a video adaptation algorithm that harnesses the PHY-informed bandwidth estimation, while probabilistically balances video quality and stalling rate, taking into account the LTE bandwidth variation (§3.3.3);

(iii) We conduct a real-time implementation of the piStream framework on a software-radio platform tethered to a smartphone (§3.4), and demonstrate its significant performance gains over four state-of-the-art protocols (§3.5).

piStream does not require any changes in existing cellular infrastructure or video streaming servers. Its PHY modules piggyback on the UEs' existing communication hardware, and can potentially be deployed via a firmware upgrade.

3.2 Background and Motivation

In this section, we first provide a brief background of DASH, and then conduct a measurement study of the challenges in running DASH over the highly dynamic LTE channel.

A primer on DASH. DASH refers to the class of protocols (MPEG-DASH[28], Microsoft Smooth Streaming[39], Apple HLS[40], Adobe HDS[41], *etc.*) that adopt HTTP based adaptive video streaming. A DASH server splits a video into multiple segments with uniform playback time (typically 1 to 10 seconds). Each segment is encoded into multiple copies with discrete encoding bitrate levels and thus different sizes. Before a DASH video session starts, a client obtains an available bitrate map from the server. To download each segment, the client needs to send an HTTP request to the server, and specify the bitrate level it prefers for that segment.

DASH has gained broad interest owing to several salient advantages over the traditional server-controlled video transport protocols [42]. It eases deployment as HTTP video traffic can easily bypass middleboxes and firewalls, and can be

supported by commodity web servers and CDNs. The use of stateless servers also simplifies load balancing and fault tolerance. Therefore, DASH is becoming the dominant Internet video streaming technology [6].

The client is fully responsible for executing DASH’s adaptation logic, which should choose the video bitrate on a per-segment basis to maximize the Quality of Experience (QoE). An optimal bitrate choice can be made only if the current and future network bandwidth are known. Many existing DASH algorithms attempt to approximate this objective by estimating/predicting available bandwidth [35, 33, 38], or by balancing the buffer occupancy at a desired level [32]. However, these solutions render unsatisfactory performance in LTE networks as detailed in the following section.

Challenges in estimating the current network bandwidth. Typical DASH protocols [31, 33, 28] use the throughput observed from each video segment to estimate the available end-to-end network bandwidth. However, such estimations stay below the bandwidth unless the segment size (and equivalently the video bitrate) is large enough to saturate the network pipeline. This is rarely the case in LTE, which bears large in-network buffers with hundred-millisecond scale end-to-end latency [43]. Therefore using throughput to guide video adaptation often underutilizes the network bandwidth.

To validate this problem in existing DASH protocols, we conduct DASH video streaming tests over Verizon LTE network (more details about our set up are available in Section 3.4). The DASH server is hosted by Akamai³ and our client is the GPAC player[31]. We run the tests in static environment during late night to ensure a relatively stable bandwidth.

We first disable the DASH adaptation by forcing the client to keep selecting a fixed video bitrate and repeat the tests for all bitrate levels in the server’s DASH data set. Figure 3.1 plots the mean per-segment throughput under each video bitrate level. Error bars represent the *standard deviation* across all segments. As bitrate increases, measured throughput grows until a saturation point where it matches the available bandwidth. As a natural consequence, if the DASH client selects a video

³http://dash.edgesuite.net/akamai/streamroot/050714/Spring_4Ktest.mpd

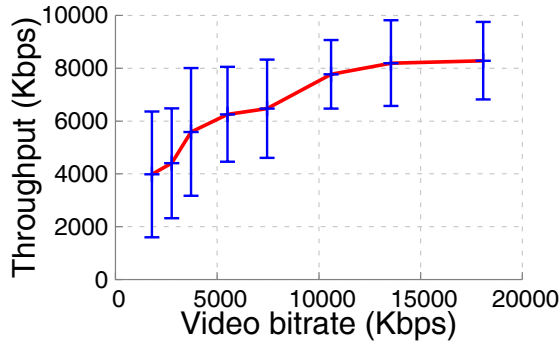


Figure 3.1: Throughput measurement depends on the traffic demand.

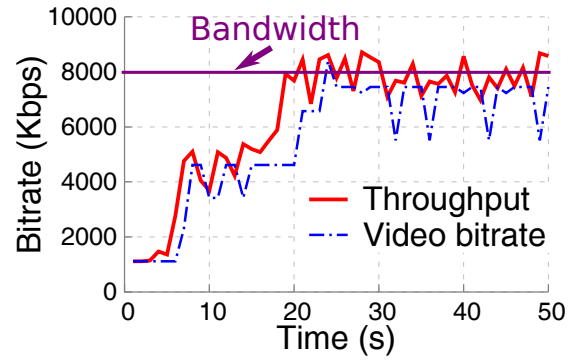


Figure 3.2: Throughput-based DASH adaptation converges slowly to bandwidth.

segment with low bitrate, it may experience a throughput far below the bandwidth. Using this throughput as bandwidth estimation, it proceeds to select a low video bitrate for next segment. This vicious cycle remains until the client opportunistically experiences a higher throughput owing to end-to-end throughput variation, but it will take a long time to eventually converge to the available bandwidth.

Then we enable the client's DASH adaptation under the same setting. Figure 3.2 illustrates the slow converging process as discussed above, where the blue curve shows the client's per-segment bitrate decisions and the red curve plots the resulting throughput. Consequently, the client takes around 25 seconds to converge to the bandwidth. Notably, a similar phenomenon has been observed in existing work [35] through trace-driven simulation.

It is worth noting that (i) The slow convergence is not caused by TCP. In one test, we manually switch the video bitrate from 1 Mbps to 8 Mbps and TCP (Cubic) only takes a few hundred milliseconds to ramp up; (ii) DASH clients usually pick a small *initial* segment size (video bitrate) to reduce the video loading time and build a sufficiently large buffer level to avoid future stalling[44], which exacerbates the slow convergence; (iii) LTE bandwidth of a client typically fluctuates more than shown in Figure 3.2 due to the competing traffic and mobility. In such cases, the slow convergence causes the throughput estimation to keep lagging behind bandwidth,

which severely compromises the performance.

One may consider using TCP-like probing mechanism to explore available bandwidth[38]. However, video adaptation runs at segment-level (second-level) in contrast to TCP's packet-level (millisecond) adaptation. DASH cannot afford the frequent probing that pushes the throughput to the limit but causes frequent video stalling, especially for the highly dynamic LTE link. Besides, using very small video segments seems helpful but eventually causes more problems: Due to the large RTT of LTE networks (100 to 300 ms according to [43] and our measurements), the request delay from client to server will incur formidable overhead unless amortized by large segments [33].

Challenges in predicting future bandwidth. An estimation of *current available bandwidth* helps a client choose the best video quality if the bandwidth is relatively stable. Yet clients suffering from link dynamics, *e.g.*, cellular network clients, ideally needs to look ahead to predict *future bandwidth* [35]. To approximate the future bandwidth, most existing DASH protocols can be classified as two categories: (i) those using the bandwidth estimation of the latest video segment as the bandwidth prediction for the next segment, *e.g.*, GPAC's player[31]; (ii) those using smoothed historical bandwidth (for example, harmonic mean over a time window) as the prediction, *e.g.*, FESTIVE[33]. Such strategies have proven effective in wireline networks, but perform poorly in LTE networks due to frequent bandwidth variations.

We then inspect these two widely used strategies to reveal their ineffectiveness. To isolate the aforementioned bandwidth underestimation problem, we assume current and historical (but not future) bandwidths are known exactly. Specifically, we collect a time series of available LTE bandwidth by measuring a saturated Iperf[45] session, and then perform trace-driven emulation for the above two strategies.

Figure 3.3 plots the time series of bandwidth prediction error at segment-level granularity. The error falls between -20 Mbps to 10 Mbps, and thus the selected video quality can deviate wildly from the optimal one. When a severe overestimation occurs, even though infrequently, the accumulated video buffer can quickly drain off (Figure 3.4), resulting in video stalls. The harmonic-mean based prediction causes less video stalling, but at the cost of severe bandwidth underutilization and

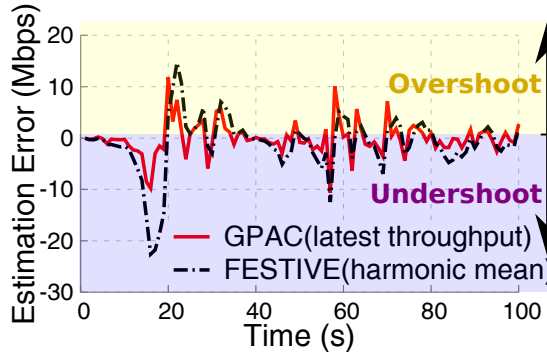


Figure 3.3: Existing DASH adaptation cannot follow the bandwidth variation.

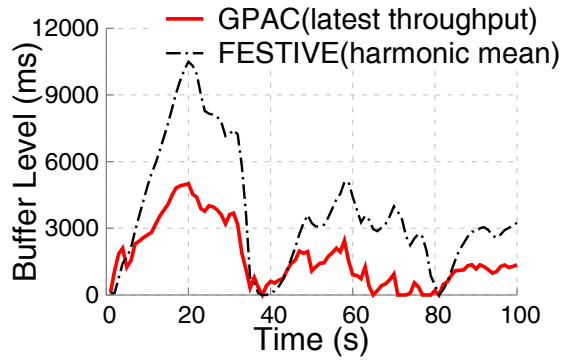


Figure 3.4: Poor bandwidth predictions drain out client's buffer and cause video stalls.

thus video quality degradation.

3.3 piStream Design

piStream is a cross-layer framework to address the above challenges for adaptive video streaming over LTE. It incorporates several client-side innovations and remains compatible with any MPEG-DASH servers [28]. piStream consists of three main design components (Figure 3.5). A *radio resource monitor* (RMon) estimates the amount of unused radio resources by sensing the LTE downlink channel. A *PHY-informed rate scaling* (PIRS) scheme translates the utilization statistics into the current available network bandwidth that can be legitimately exploited. Finally, an *LRD-based video adaptation* (LVA) algorithm estimates how long the current available bandwidth is likely to last, and accordingly selects the bitrate for the next video segment to maximize the QoE.

3.3.1 Radio Resource Monitor

The RMon acts as a PHY-layer daemon in each UE that monitors cell-wide utilization of radio resources (referred to as Physical Resource Blocks, or PRBs). RMon needs to be highly reliable, yet simple enough to be executable in real-time and

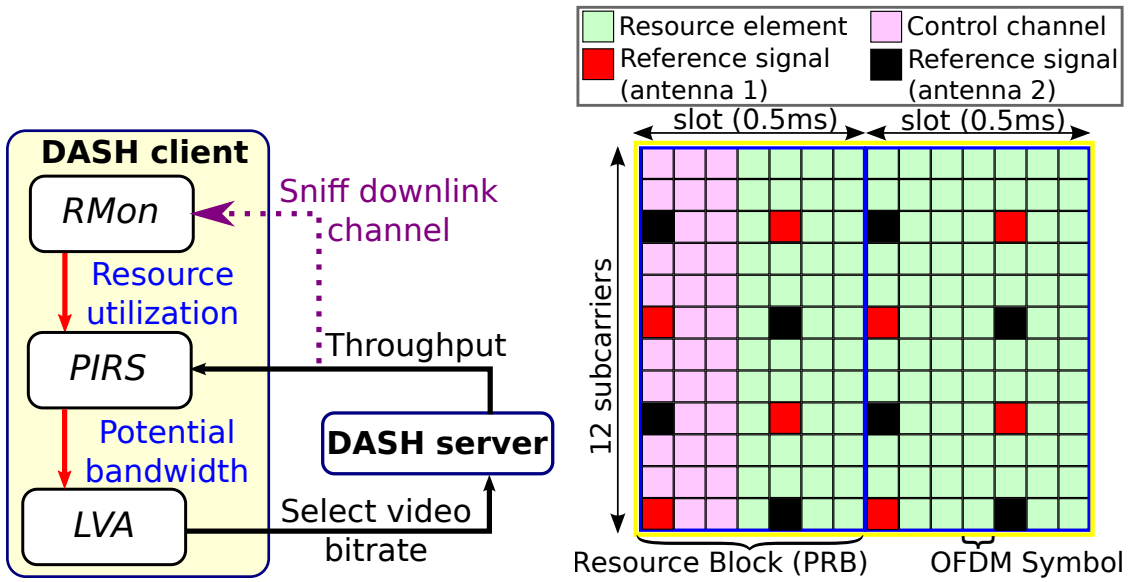


Figure 3.5: piStream system architecture. Figure 3.6: LTE resource allocation example for a two antenna BS.

easily compatible with current UE's hardware. Below we present the background, challenges and our design to meet these goals.

Why Existing LTE Sniffers are Insufficient?

piStream calls for a highly efficient radio resource monitor that is compatible with the UE's hardware. There exist a few platforms that sense the resource allocation by decoding LTE's downlink control channel (§2.2). Yet none of them meet piStream's design goal.

(i) QXDM[46], a monitor available for UEs with a Qualcomm LTE modem, can only analyze the radio resource utilization of a single UE rather than cell-wide information.

(ii) LTEye [29] can provide cell-wide resource utilization by decoding the downlink control information for all UEs. However, it requires a UE to keep monitoring the downlink control information to all other UEs, which is unspecified in LTE standard and incurs significant modifications to the UE's PHY layer. Besides, the

decoding overhead increases with the total number of UEs, hence it is unscalable. Finally, the LTE error detection mechanism⁴ does not work for LTEye since LTE design never considered that a UE will attempt to decode other UE's downlink control information. Thus, LTEye works well only when the wireless channel induces almost zero bit errors and hence the resource allocation can be deciphered without CRC.

(iii) OpenLTE[30] provides a software-radio based LTE downlink demodulator, which has the same problem as LTEye when decoding the resource allocation information.

In summary, to obtain cell-wide resource (PRB) allocation status, decoding the downlink control channels is not a deployable solution as it seems. It is unscalable, entails significant hardware modifications, and its results are unreliable.

Energy-based Radio Resource Monitor

piStream adopts an energy-based spectrum monitor called RMon to assess cell-wide PRB allocation status without decoding the control channel. RMon offers a robust, energy-efficient and readily-deployable solution to expose the PHY-layer downlink resource assignment to a UE.

Despite its conceptual simplicity, the unique features of LTE PHY layer bring several challenges to RMon's design. In particular, frequency selectivity makes it difficult to set up a threshold to examine whether an LTE resource element is allocated. Besides, wide adoption of multi-antenna BS exacerbates the variation of energy levels across frequency/time, since the antennas may have diverse channel gains to the UE.

Figure 3.7 showcases these challenges through one OFDM data symbol over a 10MHz LTE downlink channel with 600 usable subcarriers (resource elements). We obtain the ground truth of subcarrier allocation by running the LTEye sniffer [29] intentionally close to the BS to ensure almost zero bit error in decoding. We see that

⁴ Since the CRC of downlink control information (DCI) is scrambled by each UE's physical layer ID[5] which is transmitted via encrypted upper layer channels and only available to the UE itself, a UE cannot legitimately obtain the CRC of other UE's DCI.

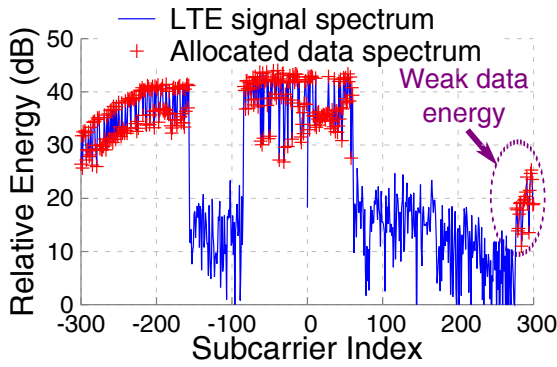


Figure 3.7: Energy-based resource utilization monitor is feasible but challenging.

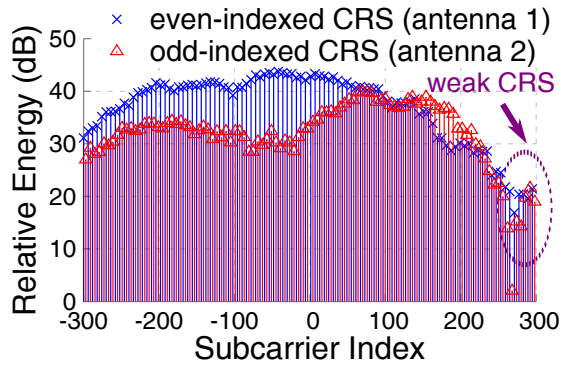


Figure 3.8: Reference signals can capture frequency selectivity and antenna diversity.

the energy on allocated subcarriers is generally 20dB higher than the noise floor. However, the energy levels of different subcarriers vary by up to 33 dB, some even falling below the noise floor. According to LTEye’s decoded control information, the BS enforces transmit diversity using 2 antennas around subcarrier index -300 and 0 , which causes a wild energy variation (about 15dB).

In the rest of this section, we detail the design of our resource monitor RMon to meet such challenges.

Obtaining per-subcarrier energy-level statistics. The resource monitor RMon can be considered as a simple intercepting module to a standard LTE UE which already performs frequency-time synchronization to the BS, and runs an FFT on each OFDM symbol to obtain the energy levels across subcarriers. RMon only requires the UE to expose such per-subcarrier energy-level statistics. Thus, it does not require any additional communication hardware. Although this is not yet feasible in most current LTE hardware (which usually only exposes total energy as signal strength indicator), we believe the potential of cross-layer design provides compelling reason to enable it. In fact, the Qualcomm QXDM already exposes to each UE its per-subcarrier energy level [46]. In many recent WiFi chipsets, per-subcarrier statistics are already available to higher layers [47, 48].

Locating resource elements available for downlink data. When monitoring

the resource assigned to downlink data, the control signals and reference signals from the BS should be excluded. Such signals are scattered across the resource grid (Figure 3.6). Fortunately, they can be located and then isolated owing to LTE's well defined resource structure.

Specifically, RMon first senses the physical channel width based on the frequency domain spectrum, which can be converted to N_{rb}^{DL} , the total number of physical resource blocks (PRBs) available in one LTE time slot. It then locates the OFDM symbols assigned to *control signals* based on the control format indicator (CFI) field specifying the location of control symbols. Finally, RMon locates the resource elements allocated to *reference signals*, which are mapped from the physical cell ID (PCI) following the LTE specification [5]. The PCI is in turn obtained during the UE-to-BS synchronization procedure.

RMon is designed to operate well in cases where LTEye fails (§3.3.1). It only relies on robust control informations: CFI and PCI are available to all UEs, and are designed to be very robust by using 32 bit code to carry 2 bit information (CFI) or signal correlation techniques (PCI).

Evaluating resource element occupancy. Since not all available resource elements are occupied by actual transmission. RMon employs an energy detector to single out active resource elements. Since the resource elements' energy levels depend on link distance, frequency selectivity, and multi-antenna diversity, it is infeasible to use a constant energy threshold to examine resource element occupancy. To remedy this problem, RMon adopts a dynamic threshold customized for each LTE resource element.

Specifically, to combat the signal variation caused by pathloss and frequency selective fading, we note that for each resource element on subcarrier k and OFDM symbol t , there exist two persistent reference signals within the subcarrier range $[k-2, k+2]$ and symbol range $[t-2, t+2]$. When active, the resource element should have similar energy level as the nearby reference signals owing to channel coherence. As an example, Figure 3.8 depicts the reference signal energy corresponding to the data signals in Figure 3.7. We let resource elements in the first half of a subframe refer to the reference signals inside the 1st symbol of this subframe and the 2nd half

refer to the 4th symbol. We use t_r to denote the global index of the nearest symbol containing reference signals. Suppose k^+ is the nearest reference signal subcarrier indexed higher than k , and k^- the one indexed lower than k . If k is an even number, we have $k^+ = k + 1$ and $k^- = k - 2$. Otherwise $k^+ = k + 2$ and $k^- = k - 1$ [5].

Moreover, for a multi-antenna BS, the reference signals from different antennas are perfectly separated in frequency domain (Figure 3.8). Regardless of the BS's transmission mode⁵ (*e.g.*, SISO, transmit diversity, and open-loop MIMO), a subcarrier's energy should be no less than the energy from the transmit antenna with the worst channel gain, which can be approximated by the smallest energy on the two close-by reference signal subcarriers.

Consequently, for each subframe, RMon specifies the energy-detection threshold $\tau(k, t)$ for an LTE resource element (RE) on subcarrier k and in the symbol at time t as:

$$\tau(k, t) = \alpha \min(e(k^+, t_r), e(k^-, t_r)) \quad (3.1)$$

where $e(k, t)$ is the measured energy of the RE on subcarrier k in the symbol at time t . Since the RE to inspect at (k, t) and its nearest reference signal REs at (k^+, t_r) and (k^-, t_r) are very close to each other in both time and frequency domain, there will only be slight channel diversity between them, and hence we can use a constant factor $\alpha = 0.8$ in Eq. (3.1) to safely accommodate such slight channel diversity.

Sanity check based on PRB resolution. The OFDM PHY layer naturally allows energy monitoring on each resource element. However, the smallest resource allocation unit in LTE is one PRB, which spans 12 consecutive subcarriers. RMon leverages this structure to further combat variation of subcarrier energy due to narrow band interference, noise, or deep fading. It computes the harmonic mean energy-level within each 12-subcarrier window, and then compares it with the threshold $\tau(k, t)$ to decide whether the PRB is allocated, thus filtering out outlier

⁵Closed-loop MU-MIMO beamforming is not supported in typical commercial LTE networks yet, and hence we leave the design to handle this particular transmission mode to future work. Even if MU-MIMO frames are used, current piStream UE can at least detect this based on the length of Downlink Control Information (DCI) and then revert to conventional DASH mode.

subcarriers. In this way, we obtain the total number of allocated PRBs.

Output the resource utilization ratio. After all steps above, RMon is ready to compute its output, the resource (PRB) utilization ratio u over a time duration of M subframes, *i.e.*, M ms, by comparing the number of allocated resource elements to the total number of resource elements available for downlink data:

$$u = \frac{N_{\text{alloc}}}{2N_{\text{rb}}^{\text{DL}}M} \quad (3.2)$$

where N_{alloc} is the number of allocated PRBs over duration M , $N_{\text{rb}}^{\text{DL}}$ is the number of PRBs in each 0.5ms time slot. Note that a subframe has 2 time slots. Intuitively, M should be small enough to enable responsive online adaptation algorithm. However, a too small M leads to very unstable average PRB utilization since the traffic load and per-subframe resource allocation are sporadic at a small time scale. In piStream, M is default to 200 ms, which can effectively capture the relationship between PRB utilization and traffic load, while remaining responsive to traffic variation.

The PRB utilization output of RMon is then leveraged to enable the PHY-informed rate scaling (PIRS), which is detailed in the following section.

3.3.2 PHY-Informed Rate Scaling

Given the PRB utilization, instead of using the throughput-based adaptation to underutilize the bandwidth (§3.2), a piStream client can scale up its measured throughput to obtain an estimation of its *potential bandwidth* at this time. In this section, we detail the rationale and operations of such a PHY-Informed Rate Scaling (PIRS) mechanism.

Relation between PRB and bandwidth utilization. PIRS builds on a hypothesis that *the PRB utilization of a client grows linearly proportional with throughput until it saturates network bandwidth*. Behind this hypothesis are two observations: (i) The end-to-end network bandwidth is bottlenecked by the LTE access link, which has been validated in existing measurement [27]; (ii) The access link capacity is determined by the bitrate and the cell-wide PRB utilization. However, since the BS

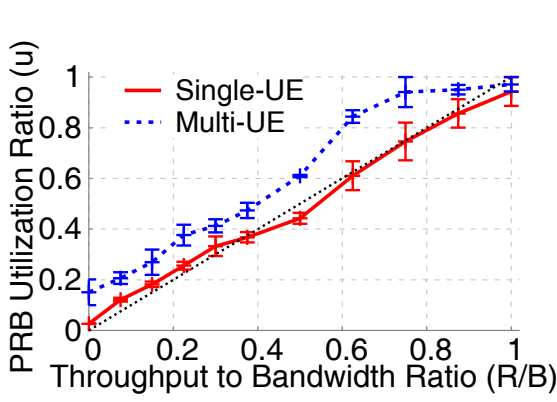


Figure 3.9: PRB utilization versus bandwidth utilization.

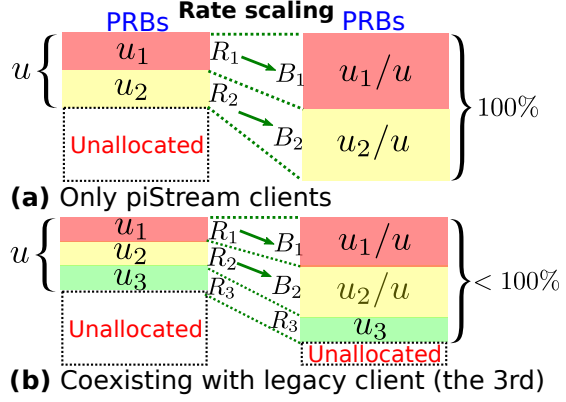


Figure 3.10: PHY-informed rate scaling example.

already performs rate adaptation to maximize the link bitrate, the additional bandwidth available should only depend on the available PRBs. Below we validate this observation through experiments.

Our experiment is conducted in late night with relatively stable network condition. We first measure the end-to-end bandwidth (B) similarly to §3.2, and then initiate a fixed-rate TCP stream from the server to one UE and record its throughput (R). Meanwhile, we use RMon to measure the cell-wide PRB utilization (u) per 200ms, and compute the harmonic mean over an entire 300s Iperf session. We also inspect a multi-UE test case, where the measurements are collected during busy hours so that our UE coexists with multiple UEs in this commercial LTE network.

Figure 3.9 shows that the bandwidth utilization R/B almost equals the PRB utilization u for the single-UE test case. Therefore, *when the downlink is dominated by a single UE, the cell-wide PRB utilization directly reflects the UE's bandwidth utilization*. For the multi-UE case, we see that u tends to be higher than R/B . This is because the PRB utilization comprises the traffic load of not only our target UE, but also other UEs within the same cell. Meanwhile, the target UE's B decreases due to downlink resource sharing. However, a linear relation still holds before u saturates, *i.e., cell-wide PRB utilization increases linearly with a UE's throughput to bandwidth ratio until the utilization approaches 100%*. This essentially validates our hypothesis.

Rate scaling based on PRB utilization. Based on the above observations, we derive the following principle for a UE to scale up its measured throughput to obtain an estimation of its current bandwidth⁶: *Suppose current cell-wide PRB utilization ratio is u , and the UE's current throughput is R , then at least a throughput of R/u can be supported without over-utilizing PRB.* Below we conduct a case-by-case analysis of this principle.

(i) *piStream client only (no legacy clients).* First, suppose the BS serves only one client who can thus enjoy all surplus PRBs. Given that the current PRB utilization u supports the client's current throughput R , the maximum achievable throughput when the client is granted all PRBs should equal:

$$B = R/u \quad (3.3)$$

Now, consider the more complicated cases when there are multiple piStream clients in the network, they will detect the same cell-wide PRB utilization u . Suppose client i is allocated an actual fraction u_i of the total PRBs, then it will scale u_i by $1/u$ accordingly. Even if they happen to scale up their rates at the same time, as shown in Figure 3.10(a), the resulting PRB utilization will not exceed 100% since $u_i/u + (u - u_i)/u = 1$. Therefore, all piStream clients can improve their rates without overshooting the bandwidth, as illustrated in Figure 3.10(a).

(ii) *piStream client(s) coexisting with conventional client(s).* In this case, Figure 3.9 has shown that the PRB utilization stays above the bandwidth utilization. Hence, a client can still scale up its throughput following Eq. (3.3) without overshooting its bandwidth. More specifically, suppose piStream client i is allocated a fraction u_i of the total PRBs. After rate scaling, the cell-wide PRB utilization becomes $\sum_i u_i/u + (u - \sum_i u_i) \leq \sum_i u_i/u + (u - \sum_i u_i)/u = 1$. A simple example is illustrated in Figure 3.10(b). Although in this case the total PRB utilization may be less than 100% after a single rate scaling, it can quickly ramp up to 100% after several successive rate scalings.

⁶This thesis focuses on the general cases where the LTE link bottlenecks the end-to-end network bandwidth [27]. We leave other corner cases to future work.

For the legacy clients in this scenario, firstly, if their traffic demands do not increase, they will not be affected by piStream at all because piStream clients only attempt to leverage the unused PRBs. Second, if the traffic demand of any legacy client increases after the PRB utilization reaches 100%, the LTE BS will invoke its proportional fairness scheduler to arbitrate the resource contention. Note that the same problem will happen even in a network with only legacy clients, and hence this is a general issue, not the limitation of our design.

(iii) *New client joining after full PRB utilization.* If a new client joins the network when the existing ones have exhausted the PRBs, the LTE BS's proportional fairness scheduler will take effect so that the new client gradually gains more PRBs. When the BS transfers PRBs to the new client, the cell-wide PRB utilization stays at 100%. Therefore, existing piStream clients will not scale up their rates any more. Furthermore, with a proportionally fair scheduler, the BS will allocate less resource to them, so the DASH senders serving existing piStream clients will slow down and the saved resources are gradually "transferred" to the new client.

Ultimately, when PRB is fully utilized, the BS's scheduler will be responsible for *ensuring fairness* among clients based on their link condition. As a client-side mechanism, piStream's rate scaling does not affect the BS's fair scheduler. However, it does help the clients to quickly reach the full PRB utilization state to use the network resource more effectively.

3.3.3 LRD-Based Video Adaptation

Given an estimation of current available bandwidth (the output of PIRS module), a piStream client runs a probabilistic LRD-based video adaptation (LVA) algorithm to select the rate for each next video segment, and balance the tradeoff between video quality and stalling risk.

Theoretical work [35] has shown that DASH performance can improve significantly if *future bandwidth* is known. However, the wireless channel fading and random traffic demand make bandwidth prediction very challenging in LTE.

Instead of attempting to predict the exact value of future bandwidth, the piStream

client takes advantage of the invariant burstiness, or long-range dependency (LRD) of LTE downlink traffic. Based on a model of its LRD profile, it estimates how long its current bandwidth-level is likely to remain consistent, and then chooses the per-segment video bitrate in a probabilistic way.

Modeling sojourn time of network bandwidth. The LRD property of Internet and local-area network traffic has been well established [36, 37]. The downlink resource sharing and large buffers in LTE intensifies such traffic patterns, which naturally causes the available bandwidth to exhibit LRD. To model LRD, we first define the *sojourn time* T_s as the period between two consecutive bandwidth jumping events, as illustrated in Figure 3.11. The bandwidth (estimated by the PIRS component) is considered to jump to another level when the highest video bitrate below the bandwidth (*i.e.*, the quantized bandwidth) changes.

The arrival of a sequence of bandwidth variations can be considered as a renewal process, with sojourn time equal to the inter-arrival time. As verified later in this section, the sojourn time distribution is relatively stable and follows a long tail or power law distribution. We use the widely adopted Pareto distribution to model such behavior:

$$\mathbb{P}\{T_s > t\} = \begin{cases} (\frac{\alpha}{t})^\beta, & t \geq \alpha \\ 1, & \text{otherwise} \end{cases} \quad (3.4)$$

where α and β are referred to as the *scale* and *shape* parameter for the Pareto model.

Based on the Pareto model, the piStream client estimates the probability $\mathbf{p}(t)$ that its bandwidth will sojourn at current level for one more video segment of duration Δt , conditioned on the observation that it has already been staying at this level for a duration t :

$$\mathbf{p}_s(t) = \mathbb{P}(T_s > t + \Delta t | T_s > t) = (\frac{t}{t + \Delta t})^\beta \quad (3.5)$$

We define $\mathbf{p}_s(t)$ as the *sojourn probability*. Eq. (3.5) implies that $\mathbf{p}_s(t)$ increases

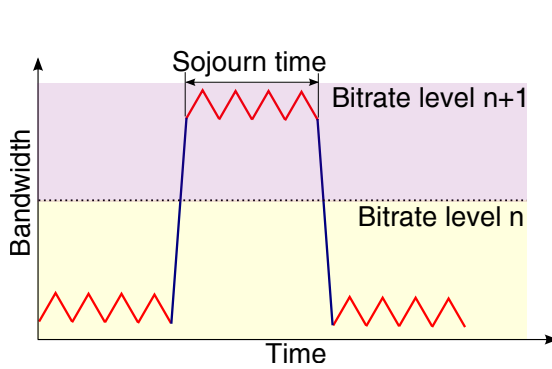


Figure 3.11: A simple example for the bandwidth sojourn time.

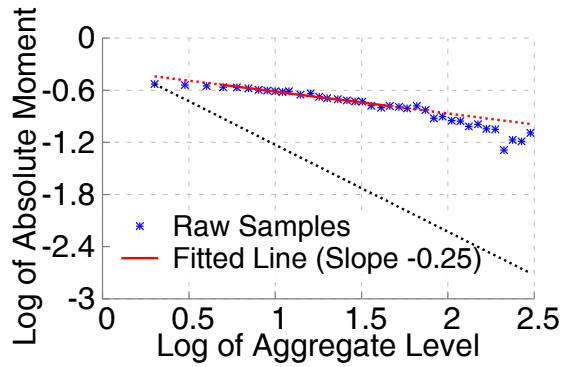


Figure 3.12: A Hurst parameter 0.75 indicates the long range dependency.

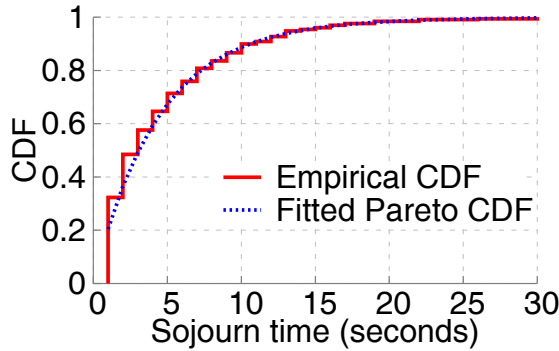


Figure 3.13: Sojourn time distribution fits a Pareto distribution.

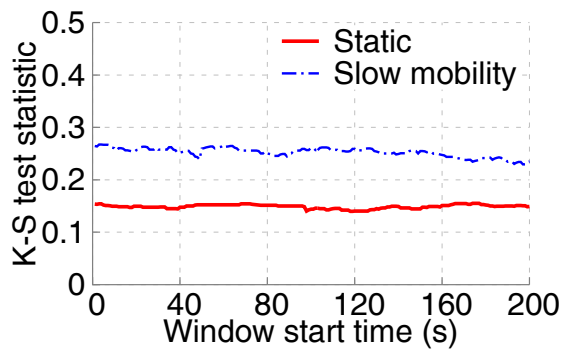


Figure 3.14: K-S test statistic. (difference between the empirical and the fitted CDF)

with t . In other words, the UE's bandwidth is more likely to sojourn at its current level if it has been staying there for a longer duration. Note that α disappears in this equation since typical video segment length $\Delta t > \alpha$, and thus we only use the first case in Eq. (3.4) to compute the conditional probability.

LRD model validation. We first use experiments to verify the LRD pattern of the LTE downlink bandwidth. We collect 5 time series of bandwidth values by running saturated Iperf sessions, each lasting 300 seconds. We use the absolute moment method [49] to estimate the Hurst parameter H for the time sequence of bandwidth. When $0.5 < H < 1$, the sequence has long range dependency, and

the degree of long range dependency increases as H gets closer to 1. Figure 3.12 gives the log-log plot of the absolute moment. The slope of the regression line is $k = -0.25$, and accordingly the Hurst parameter $H = 1 + k = 0.75$ [49], indicating strong long range dependency of the LTE bandwidth.

We further verify the goodness-of-fit for the Pareto model using the same bandwidth time series. To incorporate the impact of mobility, we collect an additional set of bandwidth trace in a car during slow driving with constant speed and direction. To process the time samples of bandwidth, we run a sliding window with size of 60s across all the samples and compute the Pareto fit of the samples in the window. The empirical CDF of bandwidth sojourn time and its Pareto fit in a particular time window under static environment are plotted in Figure 3.13. We see that more than 40% of the bandwidth level lasts longer than 5 seconds, which leaves enough room for the LVA design to take effect. Moreover, 90% of the bandwidth levels last no more than 10s. Hence, existing DASH design (*e.g.*, FESTIVE [33]) that uses a long smoothing window (typically 20s) may not adapt fast enough to follow this bandwidth variation.

Besides, in Figure 3.14, we report the Kolmogorov-Smirnov Test (K-S test) statistic $D = \sup_x |F_n(x) - F(x)|$ in each sliding time window, which uses the difference between the empirical CDF $F_n(x)$ and the fitted CDF $F(x)$ to characterize the goodness-of-fit. It is worth noting that a high mobility level may reduce the goodness-of-fit of our model. We will discuss the corresponding design to handle this issue in the end of this section.

From the results we can observe that: (i) The K-S test statistics are small, indicating small difference between empirical CDF and fitted CDF. (ii) The K-S test statistics are stable in time domain, which suggests the stability of the Pareto model parameters. (iii) The goodness-of-fit in high mobile environment is worse than that in static environment due to the bandwidth variations caused by channel fluctuation which are not captured by the Pareto model.

It should be noted that prior work [50] reported Poisson (memoryless) distribution of LTE downlink traffic. However, the experiments only examines short-term, packet-level inter-arrival time distribution, which does not conflict with the bursty

traffic pattern at larger time scale observed here⁷.

LRD-based adaptation design. The LVA algorithm proceeds as summarized in Algorithm 1 and detailed below.

(i) First, the client periodically (empirically set to 60s) applies a maximum-likelihood regression[51] to recent bandwidth samples to estimate the Pareto parameter β in Eq. (3.4).

(ii) Meanwhile, the *rate scaling module* continuously reports current bandwidth estimation to the adaptation logic. When the bandwidth (quantized to match the video bitrate levels) experiences a jump from L_{old} to L_{new} at time t_0 , the client starts to count the sojourn time T_s , the time that the bandwidth has been staying at the new level L_{new} . Based on Eq. (3.5), the client then computes the bandwidth sojourn probability $p_s(t)$: For the next video segment the network bandwidth will stay at the new level L_{new} with probability $p_s(t)$ or go to any other level with probability $1 - p_s(t)$.

(iii) Finally, the client performs stochastic video bitrate selection based on the sojourn probability $p_s(t)$. With lower sojourn probability, the new bandwidth is less likely to last for the next video segment, hence the client is reluctant to switch to the bitrate matching the new bandwidth. In contrast, higher $p_s(t)$ indicates a possibly more stable bitrate level thus the client is more inclined to switch.

To reduce the impact of a probabilistically wrong decision, LVA performs a sanity check based on the client's current buffer level. When the buffer level is low (we empirically set the threshold to 2000ms), even if a rare mistake can incur intolerable buffer underrun. Thus, the client acts more conservatively and reduces the video bit rate upon every bandwidth decrease it experiences.

Handling mobility. The bandwidth variations caused by channel fluctuations in mobility are unpredictable and not captured in our bandwidth sojourn time model. This leads to reduced goodness-of-fit in mobile environment and thus less accurate adaptation. To handle mobility, we use the accuracy of our model to guide the aggressiveness of the adaptation. More specifically, the client proactively evaluates the level of mobility based on the goodness-of-fit of bandwidth samples in

⁷Based on our private correspondence with the authors.

Algorithm 1 LRD-based video adaptation (LVA).

Input: B_n – Bandwidth estimation of the n -th segment
 L_{new} – Bitrate level after bandwidth variation
 L_{old} – Bitrate level before bandwidth variation
 t_0 – When last bandwidth variation happened
 $Q(\cdot)$ – Quantize bandwidth to video bitrate

Output: V_{n+1} – Video bitrate for the next video segment

- 1: /*Finish downloading the n -th segment at t */
- 2: Update Pareto parameter β using historical bandwidth
- 3: **if** $Q(B_n) == L_{new}$ **then** /*Check bandwidth variation*/
- 4: $T_s = t - t_0$; /*update the sojourn time T_s */
- 5: **else**
- 6: $T_s = 0$; $t_0 = t$; /*reset the sojourn time*/
- 7: /*update the bitrate levels*/
- 8: $L_{old} = L_{new}$; $L_{new} = Q(B_n)$;
- 9: Update bandwidth sojourn probability following Eq. (3.5)
- 10: **if** $\text{rand} < p_s$ **then** /*Stochastic video bitrate selection*/
- 11: $V_{n+1} = L_{new}$; /*switch to the new bitrate level*/
- 12: **else**
- 13: $V_{n+1} = L_{old}$; /*stay at the old bitrate level*/

current sliding window. A worse goodness-of-fit indicates higher mobility, which then forces the client to act more conservatively, *i.e.*, the client becomes more likely to reduce the video bitrate upon bandwidth decrease even when the sojourn time model suggests this decrease is temporary.

3.4 piStream Implementation

We prototype piStream on a cross-layer testbed comprised of a PHY-layer resource monitor and a PHY-aware DASH layer. Figure 3.15(a) and (b) illustrate the implemented system architecture and the prototype testbed, respectively. Since the Android phone lacks native support for many of the libraries essentially needed to implement piStream and also cannot be directly connected to the USRP, we implement most piStream components on the laptop as shown in Figure 3.15(b). Note that this does not affect the fidelity of our evaluation because the phone is

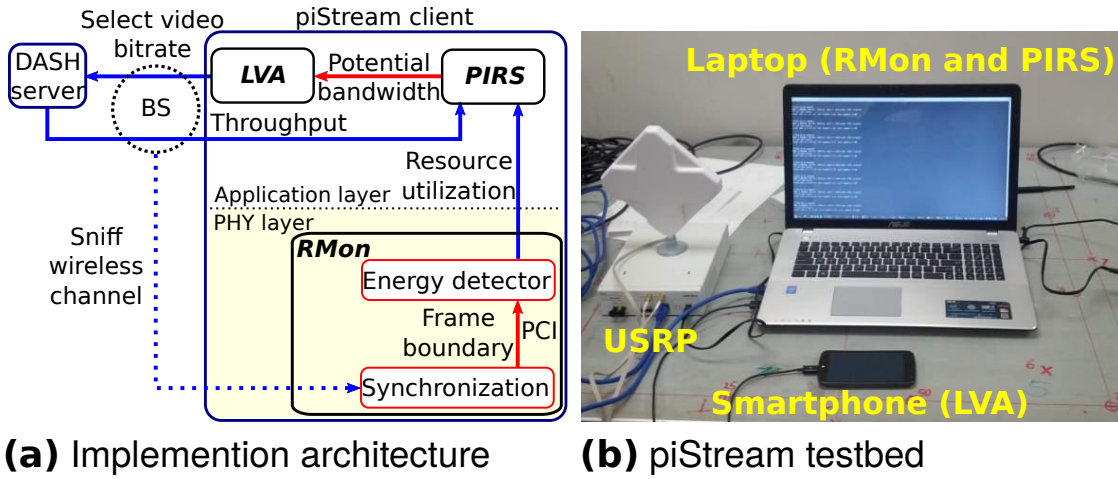


Figure 3.15: piStream implementation.

still the LTE client and all traffic goes through the LTE network.

Real-time LTE radio resource monitor (RMon). We implement piStream's *real-time* resource monitor over the USRP N210 software equipped with WBX daughter-board and an antenna covering the 690-960/1710-2620 MHz LTE band. The resource monitor is executed on USRP's PC host as a C module. It processes the LTE downlink signal samples captured by the USRP, and extracts the cell-wide PHY resource allocation information, following the frequency-domain energy-detection algorithm in §3.3.1. The resource (*i.e.*, PRB) allocation statistics are computed over a moving window of 200 ms, and passed on to the upper layers online. Since this 200ms time granularity for PRB utilization output is much smaller than typical video segment length in DASH (2000 ms in our experiments), our implementation can perform DASH adaptation in real time. We average the PRB utilization statistics in each segment duration before performing rate-scaling to its throughput.

Our resource monitor RMon essentially implements major PHY-layer primitives that exist in every UE, plus the customized energy detector in piStream. Specifically, we have implemented the following crucial sub-components following the LTE standard [5]:

(i) *Synchronization.* The *synchronization* block enables a UE to read the physical cell ID (PCI) of the current cell and identify the downlink frame boundaries.

(ii) *Signal energy detection.* The *energy-based data signal detector* in RMon realizes fast and robust monitoring of PRB allocation. The frame boundaries provided by the synchronization block are used to determine the starting points of data symbols, so that we can perform FFT to estimate the frequency-domain resource allocation. Meanwhile, as detailed in §3.3.1, the PCI is used to locate the cell-specific reference signals (CRS) and determine the energy detection threshold, which varies following the CRS energy for different PRBs to accommodate the frequency selectivity.

PHY-aware DASH client. We implement piStream’s adaptation logic by extending the DASH player from the GPAC[31] project. To interface the extended DASH layer with the RMon module, we tether an LTE smartphone (via USB) and a USRP radio (via Ethernet) to the same laptop PC host. Then we expose the PHY resource utilization statistics to the DASH player by setting up a named pipe between the resource monitor RMon and the DASH player. To leverage the PHY resource utilization statistics, we replace the default DASH adaptation logic in GPAC’s DASH player with our implementation of the PHY-informed rate scaling (PIRS) and LRD-based video adaptation (LVA) algorithms in piStream.

DASH server. We set up an MPEG-DASH server with high bandwidth (66 Mbps measured with Iperf over a wireline network) on Amazon EC2 using Apache 2. This follows typical scenarios where the LTE downlink, instead of the wireline backbone, acts as the bottleneck for video streaming. The server hosts multiple DASH video data sets with typical 2s video segment length.

Benchmark protocols for comparison. To benchmark piStream’s performance, we implement state-of-the-art DASH adaptation protocols including BBA[32], FESTIVE[33], and PANDA[38] in the GPAC’s DASH player[31] by replacing its default video rate adaptation algorithm which quantizes throughput of the latest segment as the bitrate selection. All implemented algorithms are summarized in Table 3.1.

(i) *BBA* denotes a broad class of buffer-based algorithms proposed in [32]. Among these variants, we implement BBA-2, a hybrid algorithm which uses throughput to control video bitrate in the startup phase. After reaching the steady state, the client then maps its current buffer level to the video bitrate selection using a linear function.

Algorithm name	<i>BBA</i>	<i>FESTIVE</i>	<i>PANDA</i>	<i>GPAC</i>	<i>piStream</i>
Bandwidth estimation	N/A	Harmonic mean throughput	N/A	Latest throughput	PIRS & LVA
Buffer knowledge	Yes	No	No	No	Yes
PHY information	No	No	No	No	Yes

Table 3.1: Implemented DASH algorithms.

(ii) *FESTIVE* algorithm selects video bitrate based on the harmonic mean of historical throughput measurements. As recommended in [33], our implementation adopts the averaging window over 20 segments and 30 seconds target buffer size. We also implement *FESTIVE*'s stateful bitrate selection, *i.e.*, we delay the switch to bitrate level k by k segments, but decrease the bitrate level immediately if necessary.

(iii) *PANDA* allows a client to keep increasing its video bitrate to probe the bandwidth until observing a throughput decrease, which is taken as a congestion indicator. Then it reduces video bitrate based on the difference between bitrate and throughput of the last video segment. Our implementation exactly follows the default *PANDA* parameters in [38], the parameter k controlling the rate reduction speed is set to 0.14, which means slow rate reduction after a congestion.

3.5 piStream Evaluation

In this section, we first validate each component in *piStream*, including the real-time monitor (RMon), PHY-aware bandwidth scaling (PIRS), and LRD-based video adaptation (LVA) (§3.5.1). Then we combine everything together and perform a thorough system-level test under various network conditions (§3.5.2). All experiments (except one with trace-driven emulation in §3.5.1) are conducted over real-world Verizon LTE network on channel 13 (746-756MHz) using Samsung Galaxy Nexus smartphones. They also use the same server and DASH data sets specified in §3.4.

The performance evaluation focuses on two major QoE metrics [52]: (i) The *average video bitrate* represents video quality. It is defined as the mean video bitrate over all streamed video segments. (ii) The *video stall rate* represents playback smoothness. It is defined as the percentage of video stalls, *i.e.*, the number of video segments

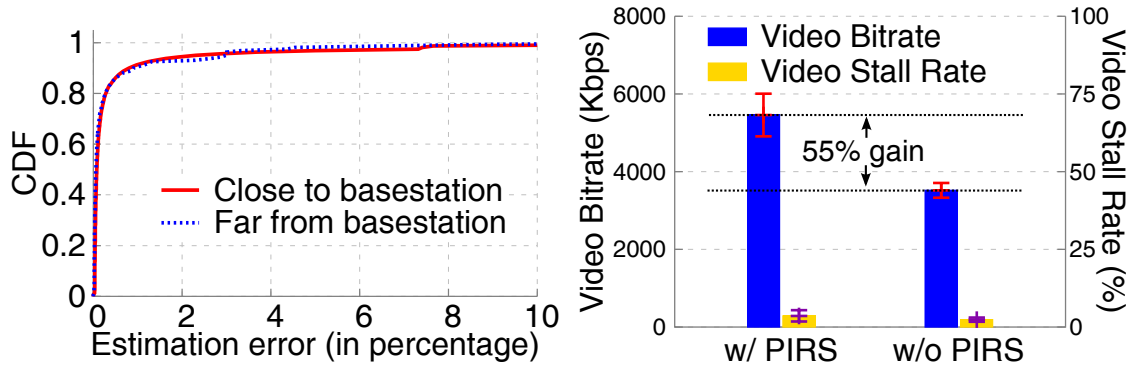


Figure 3.16: Our resource monitor RMon outputs accurate resource utilization. Figure 3.17: PIRS component alone improves the performance.

that stall the video divided by the total number of streamed video segments to obtain the video stall rate. We obtain these metrics by logging the bitrate of every video segment and the buffer level on the client testbed in real time.

3.5.1 Micro-benchmarks

Validating the Radio Resource Monitor (RMon)

We first examine how accurate piStream’s resource monitor RMon can estimate the PRB utilization statistics. We evaluate the PRB utilization (in percentage) for a single-user video streaming. Meanwhile, we record the bandwidth utilization (in percentage). As discussed in §3.4, LTE downlink channel is usually the bottleneck between server and client. Hence, if RMon is accurate, its PRB utilization output should equal bandwidth utilization for this single user. Estimation error is defined as the difference between PRB utilization and bandwidth utilization. The experiments are performance in static environment during idle hours so that the bandwidth utilization is *w.r.t.* the same bandwidth. We repeat the experiments under two link conditions, created by moving the client close by and far from the BS while ensuring connectivity. In each case, we compute RMon’s estimation error over 200 seconds and plot its CDF.

Figure 3.16 shows that RMon is highly accurate — it produces the same output

as the ground truth for more than 90% of the time. In case it errs, it differs by no more than 10%. In addition, its accuracy is only slightly affected by link condition. Therefore, its PRB statistics can be reliably used by piStream’s higher-layer modules. In our system tests (§3.5.2), we will demonstrate the synergistic performance under a variety of conditions including node mobility.

Validating the PHY-informed Rate Scaling (PIRS)

To verify the eventual impact of PHY-informed rate scaling, we evaluate how much performance improvement it can achieve in DASH video streaming. In this experiment, the RMon keeps sniffing the LTE downlink channel during DASH video streaming and reports the PRB utilization to the DASH client periodically. The client then scales up its bandwidth estimation based on the latest PRB utilization and directly sets the quantized bandwidth estimation as the bitrate for the next video segment. To exclude the influence of downlink bandwidth variation, we test a single static client during late night for 10 consecutive runs.

Figure 3.17 compares the performance of using PIRS or simply the throughput for video bitrate adaptation. We see that PIRS module alone improves the average video quality by 55% at the cost of only 1% increase in the number of video stalls. Therefore, PIRS can effectively overcome the slow ramp-up behavior typically seen in throughput-based bandwidth estimators (§3.2).

Validating the LRD-based Video Adaptation (LVA)

In this micro-benchmark, we compare LRD-based video adaptation with several widely adopted DASH adaptation algorithms: (i) Latest-throughput-base adaptation, which uses bandwidth experienced by the latest video segment as the expected future bandwidth. (ii) To handle the bandwidth variation, adaptation based on the smoothed historical bandwidth is also widely used. A straightforward solution is to compute the arithmetic mean. (iii) Harmonic mean based bandwidth smoothing has demonstrated superior performance [33] than the arithmetic mean since it is more robust to larger outliers.

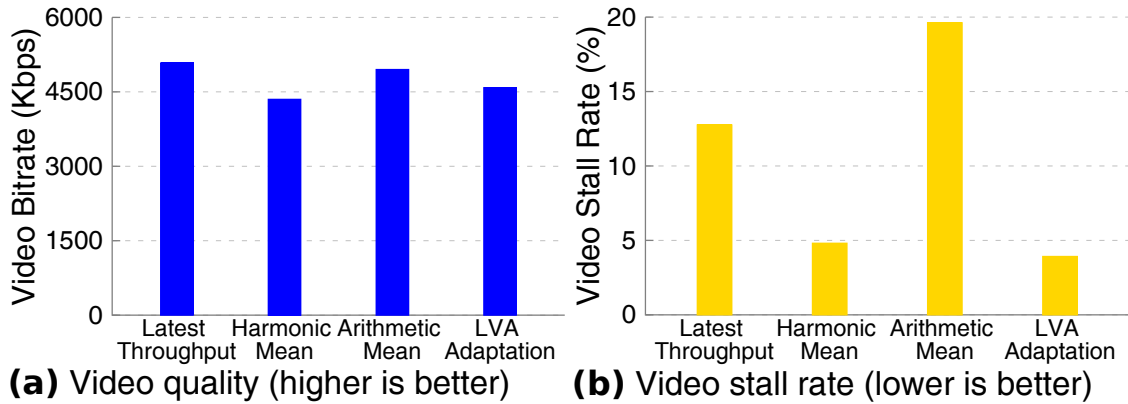


Figure 3.18: Trace-driven emulation with perfect knowledge of current & historical (but not future) bandwidth.

We use trace-driven emulation only for this micro-benchmark to isolate the effect of inaccurate bandwidth estimation in other algorithms. Instead of using the C++ implementation as in all other real-world experiments, we implement the algorithms again in MATLAB. The DASH video set used here is the same as that in all other real-world experiments. The transmission time is emulated within MATLAB based on the video segment size and the LTE downlink bandwidth in the collected traces. The traces contain a time series (at second level time granularity) of available LTE downlink bandwidth during busy hours for the target UE, collected by running a saturated Iperf TCP session. All DASH algorithms are run over the same trace. Arithmetic and harmonic mean based adaptation algorithms work on a granularity of 20 segments, as suggested in [33]. To investigate video stalls, we compute the buffer level at the client as a time series based on the bitrate selection for each video segment and the bandwidth at that time, where the buffer level may drop to 0 and cause a video stall when the selected bitrate exceeds the bandwidth. Note that among DASH schemes, only piStream can obtain an accurate estimation of the *current available* bandwidth and use it for video bitrate adaptation. Yet for a micro-benchmark comparison of adaptation algorithms, we assume the current bandwidth (not future bandwidth) is known and used by all algorithms.

(i) *Video streaming performance of adaptation algorithms.* We first compare the video streaming performance of these algorithms. Figure 3.18 plots the results of this

trace-driven emulation. Compared to the latest-throughput-based adaptation, LVA reduces the video stalling rate from 13% to 4%, at the cost of slight video quality loss (9%). Meanwhile, compared to the harmonic mean based adaptation, LVA improves video quality by 5% and the video stall rate by 1%.

(ii) *Effectiveness of adaptation algorithms.* To further investigate the reason behind the performance improvement, we look into the accuracy of adaptation by comparing the bitrate selection and the actual bandwidth of a video segment. Figure 3.19 shows that, 57% of the time, our LVA algorithm picks the video bitrate exactly matching the bandwidth, which is the highest among all tested algorithms. Moreover, we also see that the LVA algorithm has less overshoots and undershoots in the adaptation compared to other algorithms.

These results explain what Figure 3.18 shows. First, the latest-throughput-based adaptation has the highest video bitrate but lots of video stalls since it has more overshoots and less undershoots than other algorithms. LVA algorithm has the smallest video stall rate since it has the highest accuracy in adaptation (the peak at $x = 0$ in Figure 3.19). Both the harmonic and arithmetic mean based adaptation have low accuracy in adaptation, hence much worse performance.

3.5.2 System Level Test

We now compare piStream with state-of-the-art DASH adaptation algorithms (§3.4), including GPAC’s DASH player[31], BBA [32], FESTIVE [33], and PANDA [38].

piStream Client in Real-world LTE Network

We first conduct experiments with a static piStream client playing MPEG-DASH video over populated commercial LTE network. To ensure a practical test environment with sufficient link dynamics, *e.g.*, other clients joining or leaving the network, we perform the experiments in a library with about 100 students inside during the afternoon, where our resource monitor constantly reports around 30% background PRB utilization caused by other LTE clients.

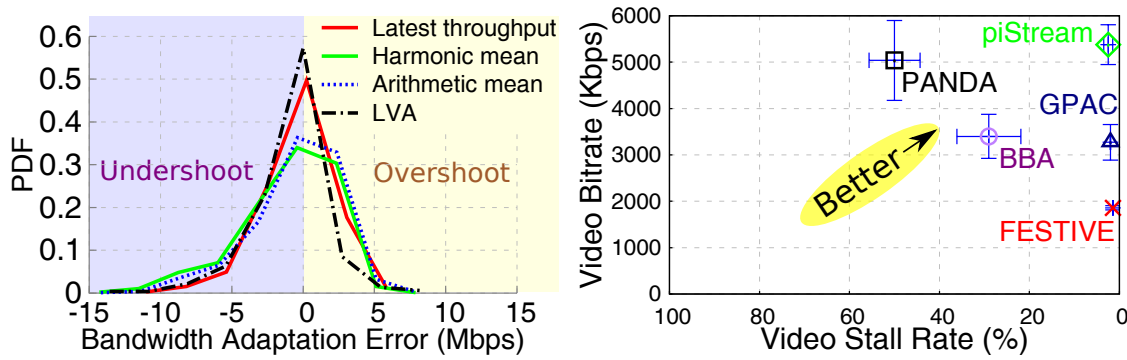


Figure 3.19: LVA has the lowest over- Figure 3.20: Average video bitrate & shoot prob.(18% lower than the runner-up)

Before the experiments, we first use Iperf to measure the end-to-end bandwidth using wireline network to ensure there are no other bottlenecks between the DASH server and the client. We also use OpenLTE[30] to scan the LTE downlink channel to ensure there is no inter-cell interference. Each of our test lasts around 200 seconds, and we repeat each experiment with 10 runs.

(i) From Figure 3.20, we see that piStream substantially boosts video quality compared with alternative approaches. In particular, it improves the average video rate by 64% over GPAC, 125% over FESTIVE, 61% over BBA, and comparable to PANDA. This performance boost in video quality is mainly attributed to the PHY-informed bandwidth scaling, which enables the piStream client to select higher video bitrate than other algorithms without exceeding the bandwidth.

Meanwhile, piStream maintains a low video stall rate of close to 0. This implies that the LRD-based video adaptation can minimize the risk of buffer underrun, despite the aggressive video rate selection upon bandwidth scaling. (ii) GPAC's DASH player experiences a lower video stall rate compared to the trace-driven simulation (*i.e.*, the latest-value-based adaptation in Figure. 3.18) where the current downlink bandwidth is assumed to be known. In this real experiment, the GPAC client needs to execute its historical averaging based bandwidth estimation, which tends to be conservative, resulting in less video stalling.

(iii) The FESTIVE algorithm relies on the harmonic mean of the historical

throughput to control the video bitrate. Despite its effectiveness in wireline networks [33], the harmonic mean takes a long time to converge. Under the high bandwidth variation of LTE, it results in severe bandwidth underutilization, despite a comparable stalling rate with piStream.

(iv) Surprisingly, the BBA algorithm exhibits an unacceptably high video stall rate (23%). It should ideally mitigate stalling by reducing the video rate whenever it sees a low buffer level. By examining its adaptation logs, we found that, given a high buffer level, BBA decides on a high video bitrate regardless of the current downlink throughput. Such uninformed aggressive choice tends to drain the buffer off before the high-rate video segment finishes downloading.

(v) The PANDA algorithm is built on a bandwidth probing mechanism and should ideally make more informed decisions. Yet the results show 50% of stalling rate despite its comparable video quality with piStream. We found that PANDA probes the bandwidth until observing a decrease in the downlink throughput, which is taken as an indicator for congestion. However, it may have already overshoot the bandwidth before experiencing such a throughput decrease. Also, its slow end-to-end probing can hardly keep track of the bandwidth in real-time.

In summary, piStream makes the best tradeoff between video quality and stalling rate compared with a wide range of video adaptation protocols that adopt rate-based, buffer-based and probing-based approaches. These adaptation algorithms fall short of agility and precision in the presence of high bandwidth variation in LTE. piStream's PHY-informed design effectively overcomes such limitations.

Impact of Mobility

We further evaluate piStream's performance by running it inside a moving vehicle in two outdoor scenarios: *slow-driving* (20 to 25 mph in an urban downtown area) and *fast-driving* (45 to 50 mph in a suburban area). The software-radio running piStream's RMon module is driven by a 120V power-inverter inside the vehicle. To ensure similar effect of mobility to all algorithms, we conduct the experiment by driving through the same route when testing different algorithms. Since our

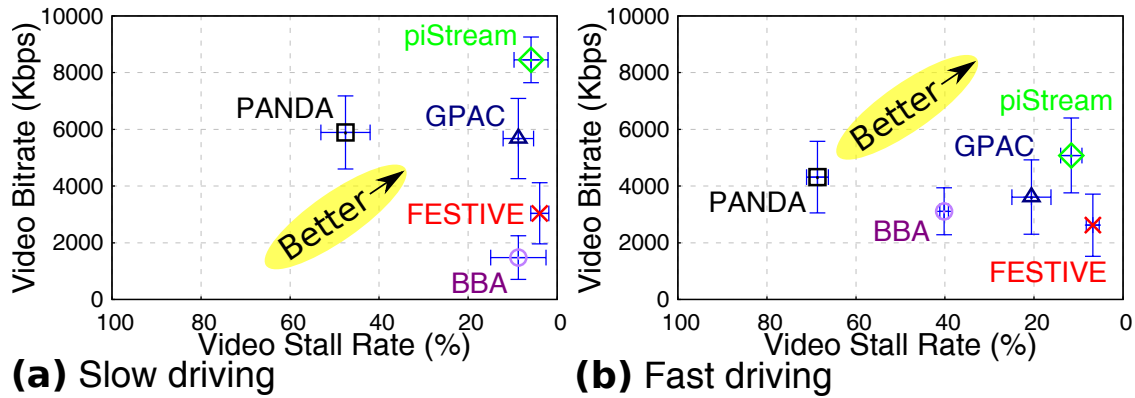


Figure 3.21: piStream outperforms other DASH algorithms in mobile environments. current spectrum monitor design cannot properly handle the hand-off between cells, we need to ensure that the car remains in current cell during the evaluation time. Therefore, we only stream for 100 seconds for each algorithm.

Figure 3.21(a) and (b) plot the experimental results. First, we see that the performance of all algorithms suffers under higher mobility, which is expected since mobility is known to worsen the channel condition and bring more bandwidth variations. Notably, the average video rate in the slow driving case is even higher than that in the static indoor environment (Figure. 3.20). This is because the higher signal strength outdoor can partly offset the negative effect of mobility.

piStream maintains the highest video quality among all tested algorithms and a low video stalling rate. This also implies that the RMon module in piStream is robust and reports accurate PRB utilization ratio even in the high mobility cases. In contrast, we found the control information decoding based method in LTEye[29] easily fails due to high bit errors induced by channel fading. It missed 40% of the downlink PRBs filled with higher energy, and it erroneously detected more than 1000 UEs in 20 seconds within a small cell. This poor performance is caused by the CRC failure, since LTE is designed to only let the UE to decode its own downlink allocation (§3.3.1).

The performance ranking of other tested algorithms also remains similar as in the static case. These algorithms can not keep track of the high bandwidth variation, and hence the video quality degradation becomes even more pronounced.

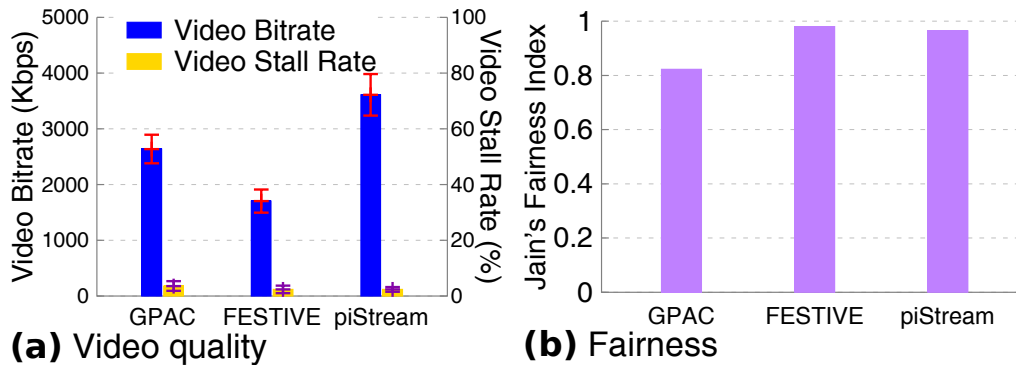


Figure 3.22: Performance with multiple DASH clients.

Multiple piStream Clients Coexisting

Will piStream's bandwidth scaling method cause unfairness when multiple clients coexist? We have provided a negative answer with basic intuitions in §3.3.2. Now we conduct experiments with four clients⁸ running piStream concurrently and evaluate the Jain's fairness index. We only compare piStream with GPAC and FESTIVE, since BBA and PANDA keeps an unacceptably high stalling rate.

Figure 3.22(a) shows that piStream maintains the highest video quality among all users, with comparable/lower stalling rate than other approaches. On average, a piStream client enjoys 42.3% and 108% higher video rate than GPAC and FESTIVE, respectively. More importantly, Figure 3.22(b) shows a Jain's fairness index of close to 1 among all users. This implies that each piStream client takes a fair share of the available resource when enforcing the bandwidth scaling. Note that GPAC tends to result in low fairness (index around 0.8), with its simple historical throughput based adaptation, whereas FESTIVE has trade video quality for fairness into its adaptation algorithm, *i.e.*, sacrificing many bitrate increase opportunities in its stateful bitrate selection, thus achieving good fairness.

⁸ piStream can support many more users than 4, but we believe fairness is meaningful only when all concurrent clients have good QoE (*e.g.*, with stalling rate of below 10%). Therefore we only serve 4 users to trade concurrent user number for QoE.

3.6 Summary

Video streaming is surging to dominate the cellular network traffic today. However, current adaptive video streaming protocols fall short of the expected QoE even under LTE's high bandwidth provisioning. To address the problem, we present piStream, a DASH-compatible adaptive video streaming framework that exposes LTE's PHY layer information to facilitate video rate adaptation. piStream's PHY-informed design enables a more accurate bandwidth estimation and agile video rate adaptation. Extensive experiments on a real-time prototype show that piStream outperforms state-of-the-art video streaming algorithms in both video quality and playback smoothness. Under typical video streaming environments, piStream achieves around $1.6\times$ video quality (bitrate) gain over the runner-up algorithm while maintaining a lower video stalling rate. We believe piStream's PHY-informed design can be applicable to a wider range of LTE applications than what we have explored.

4 ACCELERATING MOBILE WEB USING CELLULAR LINK INFORMATION

4.1 Introduction

Web access has become a main portal of content consumption on mobile devices, and user-perceived Web performance becomes a vital metric for network services. Over the past years, wireless service providers have strived to improve the cellular network performance to match the demanding Web access. However, the mobile Web loading performance does not match up to the infrastructure upgrade. Nationwide 4G vs. 3G speed tests in the US showed $10\times$ throughput improvement [53], but the typical Web latency only decreases by around half [8]. Given the typical mobile Web size of below 1.5 MB, and median 4G LTE throughput of 12 Mbps [8, 54, 53], one would expect the Web loading latency to be well below 1 second. In reality, however, most mobile webpages take 2 to 9 seconds to be loaded over LTE [8], which falls below both the true potential of high-speed LTE link and the user expectation, as 47% of consumers expect a Web page to load within 2 seconds or less [55].

The disappointing mobile Web performance roots in the poor interaction between HTTP—the application-layer protocol for the Web, and TCP—the underlying transport-layer protocol. It is well established that TCP performs best when there is a steady stream of data packets and returning ACKs to keep the network pipeline full. This is because TCP implicitly uses the data packets to probe the network, and it takes time to gradually ramp up the data load towards the network's available bandwidth. Since HTTP generates short, bursty flows, TCP lacks time to converge, thus severely underutilizing the network capacity. Such laggy responses are exacerbated in cellular networks with relatively long RTT and dynamically changing bandwidth. In addition, existing TCP protocols use packet delay or loss as congestion indicators to facilitate bandwidth probing. However, the unstable RTT makes the delay metric unreliable and vulnerable to spurious timeout [8]. The loss metric,

on the other hand, tends to overreact to wireless link losses, and is insensitive to real congestion because of the deep buffers at the LTE BS [20, 43, 56].

Given the mismatch between TCP's continuous packet flow requirement and HTTP's bursty traffic pattern, we propose to replace TCP's probing mechanism with a direct estimate of bandwidth from the LTE physical layer. This principle builds on several observations: (i) The end-to-end path for most cellular connections is bottlenecked by the last hop [26], particularly because content delivery networks (CDN) and Web caches shorten the wireline path. Hence it suffices to track the cellular link bandwidth, and gracefully detect and handle the corner cases when the wireline backbone becomes the bottleneck. (ii) Since an LTE BS itself already enforces fair scheduling [57], the transport-layer protocol should focus on fully utilizing the available physical bandwidth. (iii) LTE BS's available frequency/time resources are shared among clients. Thus, it is feasible for a client to estimate the resource utilization within the shared medium from its PHY layer signaling with the BS.

We realize these principles through a novel system design called Cellular Link-Aware Web loading (CLAW). CLAW accelerates mobile Web loading using a bandwidth exploration mechanism facilitated by cellular link statistics available on commodity smartphones. It replaces the conventional transport-layer bandwidth exploration with a PHY-layer informed scheme that can estimate available bandwidth almost within one RTT, without relying on the application traffic pattern. In this dissertation, we focus on integrating this idea in TCP, the default transport layer protocol to convey HTTP data, but the same bandwidth exploration principle can also be applied to UDP-based HTTP rate control like QUIC [58] from Google. The key challenge here lies in the limited PHY information available to clients. Ideally a client would need a global view of the BS's resource usage, so as to estimate the amount of resources and hence bandwidth available to it. But the PHY signaling channel is encrypted and each user equipment (UE) can only observe the resource allocated to itself [59, 29]. Although the BS could separate a dedicated control channel to broadcast the resource usage as public information, this requires non-trivial update to the LTE MAC/PHY stack and causes huge compatibility issue.

To overcome the constraints, we design a cellular load analyzer incorporating a quantitative model that allows an LTE phone to extrapolate cell-wide resource utilization, by leveraging public PHY statistics of the downlink channel. Prior work in LTE resource analysis needs fine-grained per-subcarrier channel information [59, 29], which can be obtained only through a software-radio monitor. Even if such information eventually becomes available, it entails non-trivial signaling overhead when streamed in real-time from the LTE modem and to user space. In contrast, our model enables accurate cell load analysis even with abstracted diagnostic information (*e.g.*, average signal power and modulation level), which is readily available on commodity LTE phones.

Given the estimation of available frequency/time on the PHY layer, CLAW further translates it into usable bandwidth, and feeds it back to facilitate rate control at the TCP sender (*i.e.*, Web server). Unlike prior LTE resource analyzers [59, 29, 16], CLAW can estimate both cell-wide and individual client’s resource usage, enabling more informed rate control under competing traffic. Moreover, a CLAW client harnesses its link-layer retransmission and queue overflow statistics to discriminate wireless loss from real congestion, preventing unnecessary sender rate fallback.

In addition, CLAW incorporates a simple bottleneck detector, which identifies the rare cases of wireline bottleneck, through a consistency check between the PHY resource utilization and transport-layer window evolution. The detection does not require any dedicated probing traffic and can complete within a few RTTs. When a wireline bottleneck occurs, CLAW falls back to an AIMD-style bandwidth exploration, which resembles legacy TCP but still harnesses certain PHY information such as packet losses and usable PHY resources.

We have implemented CLAW on an Android client that executes the idle resource estimation and bottleneck identification, and an HTTP server that replaces legacy TCP’s rate control with CLAW’s cellular-informed rate control. We benchmark CLAW’s performance against conventional TCP variants, along with two state-of-the-art transport protocols, CQIC [16] and Verus [21], which are designed for cellular networks. A thorough evaluation demonstrates that CLAW can quickly

track the available bandwidth resource, leading to 27% to 66% latency reduction across a wide range of websites and network conditions. We also observe that loss/delay based TCP protocols often lead to large performance variation, and CLAW reduces the worst-case latency (which is likely to be the bottleneck to user experience) by up to 73%.

To our knowledge, CLAW represents the first work to quantitatively model the relation between LTE’s resource utilization and the PHY-layer statistics on commodity cellphones. The model can be adapted by a broader range of wireless protocols, including not only web loading, but also video streaming [59], background traffic scheduling [60], *etc.* CLAW is also the first to achieve zero-overhead bottleneck detection and explicit loss differentiation over LTE. The CLAW design relies on PHY information, but entails no PHY-layer modification and is readily deployable on current LTE devices.

4.2 Background

Web loading is a sophisticated procedure involving both networking and computation. In general, it begins by downloading an HTML page which refers to the objects (*e.g.*, images) within the requested webpage, the browser then iteratively parses the HTML file while downloading the referred objects. In parallel with the downloading process, the client device progressively renders the retrieved objects on its display. Page loading time (PLT) is typically used as the performance metric, which accounts for both object downloading and processing time, and directly impacts end-user experience. Optimizations for both the downloading and computation are vital to enhancing the mobile Web performance. This thesis focuses on the networking part, we further discuss about this in §4.7.

HTTP has been the de facto vehicle protocol to deliver Web content, accounting for more than 82% of the traffic that ends at mobile devices [61]. Many of today’s Web services adopt HTTP 1.1, which use short-lived TCP connections to deliver objects, one connection per object. This inevitably incurs long latency as modern Web sites contain tens to hundred level objects.

Our CLAW design builds on prior measurement insights which revealed that mobile Webs’ performance bottleneck lies in the long RTT [62]. Since cellular networks’ long RTT roots in the legacy infrastructure [63], the key to accelerating Web download is to reduce the number of RTTs needed to retrieve the objects. The HTTP/2, inspired by Google’s SPDY [8] and ratified by the IETF in 2015[64], optimized the data transfer paradigms, allowing multiple outstanding object requests through one TCP connection. To date, HTTP/2 has been supported by all main-stream mobile browsers and content providers. However, recent measurement studies revealed disappointing HTTP/2 performance in cellular networks [8], due to *poor interaction between TCP, HTTP and the radio link layer*. In particular, TCP cannot track the high RTT variations and often treats an RTT surge as timeout, which in turn leads to spurious retransmissions and extended Web loading time.

The key to cutting the Web loading round-trips lies in the underlying transport layer, *e.g.*, HTTP/2[64] adopts the innovation to reuse a conventional TCP connection. However, conventional TCP protocols treat the network path as a black box, and attempt to infer network congestion from end-to-end metrics like delay and loss, which have to be generated through trial-and-error probing across many round-trips and hence lack responsiveness. In effect, most Web loading sessions are short and finish well before TCP converges to the network bandwidth [65]. Explicit congestion control protocols [66, 67] advocate more informed rate adaptation based on router’s feedback. But the deployment is rare to date, as they require new router hardware and sophisticated queue management. CLAW can be considered as providing a better-informed explicit congestion feedback that quantitatively characterizes how much more traffic load can be added before it congests the network. Yet it is much easier to deploy, as it harnesses the cellular bottleneck, and only requires the client to act as a single-point rate estimator.

4.3 Challenges for Mobile Web Over LTE

In this section, we present the root causes to the unsatisfactory mobile Web performance in LTE networks, and highlight the challenges in resolving these issues.

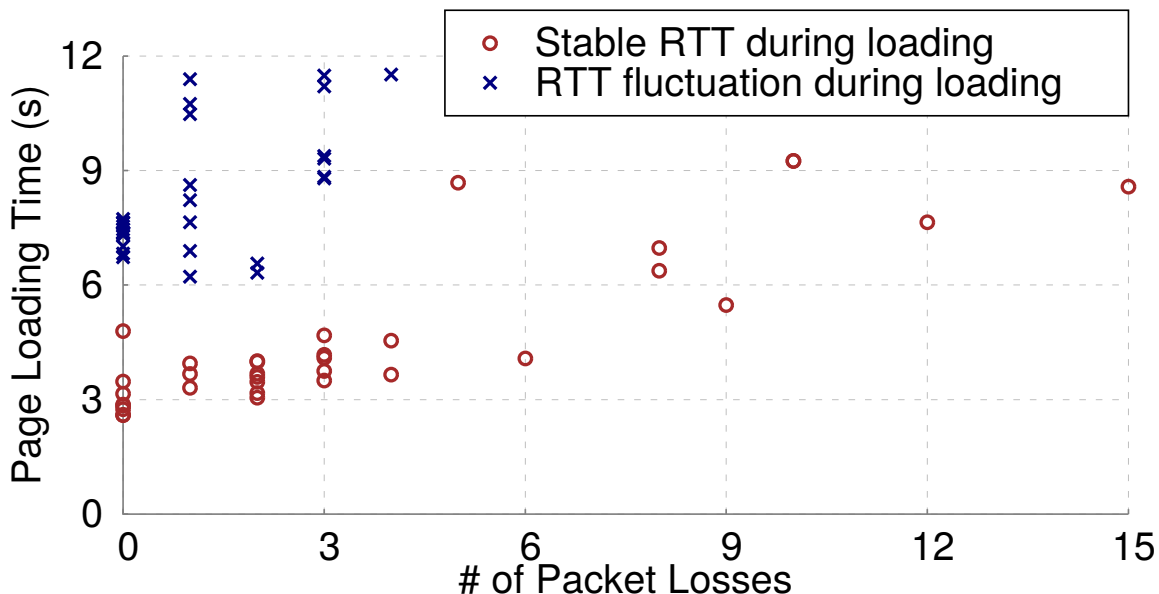


Figure 4.1: Page loading latency over LTE is affected by both RTT and packet losses.

The large and unstable RTTs mislead TCP adaptation. Although LTE adopts a more flattened network architecture and reduced the number of gateways compared with 3G, its 70ms average TCP handshake RTT remains much longer than wireline networks and even WiFi [53], and the RTT over LTE quickly inflates with the number of bytes in flight (it easily exceeds 100ms with only 100KB data in flight). In addition, LTE links often experience large RTT variation due to link-layer packet retransmission/reordering, channel quality change, and traffic dynamics of competing LTE clients. The RTT’s standard deviation is typically $3\times$ to $4\times$ of the mean for static LTE nodes, and even larger for mobile ones [63]. Such large RTT variations lead to the poor performance of conventional TCP during mobile Web loading (§4.6), which falsely treats a temporary RTT surge as packet timeout caused by network congestion.

Fig. 4.1 shows one case study, where we load the same static webpage 50 times on an LTE phone (detailed page hosting setup is in §4.6). The loading events scatter sparsely across the day to include various network conditions. The server uses the default TCP Cubic, and each page loading involves around 2700 packet transfers. From the results, we observe that packet losses are rare over LTE networks. However,

even a small number of lost packets can significantly slow down the Web loading, which is consistent with existing observations[68]. Furthermore, under few or no packet losses, sometimes the Web latency still surges due to TCP’s aforementioned sensitivity to RTT variations: a temporary RTT surge misleads the server to assume a network congestion/timeout and exit TCP slow start early, even when its current sending rate is still far below the optimum. To minimize such negative effects caused by cellular RTTs, *an efficient transport protocol should be able to quickly converge to the network bandwidth, with least number of round-trip probes*. Quick convergence also enables accurate adaptation to LTE channel dynamics, preventing bandwidth underutilization (due to underestimation) and self-inflicted congestion (due to bandwidth overestimation).

TCP overreacts to LTE link losses. Although LTE’s link-level retransmission improves reliability, packet losses are still unavoidable, and even a few losses across an entire Web loading session can substantially increase latency [69]. This is because TCP protocols typically treat packet loss as a congestion indicator, and multiplicatively reduce the sending rate even when the loss is caused by fluctuations of wireless link quality rather than a network congestion, resulting in poor bandwidth utilization and hence slower flows. As shown in Fig. 4.1, across the experiments, we observed very low packet loss rate, less than 1% even in the worst case. But we found even a few more packet losses lead to explosive growth of latency, which echoes the large-scale measurements in [69]. Moreover, due to unpredictable wireless losses, the Web latency becomes very unstable across runs. The worst-case latency can be extremely high and ruin the user experience, as the users are sensitive to performance degradations.

The short mobile Web flows hinders the sending rate from quick convergence to the network bandwidth. TCP’s bandwidth exploration is propelled by application traffic. Its congestion window size accumulates as more packets are acknowledged over previous round-trip transmissions. This property fares poorly with HTTP that creates sparsely distributed packet bursts — measurements studies showed that 90% of Web flows finish within TCP’s slow start phase [65], *i.e.*, well before it converges to network bandwidth.

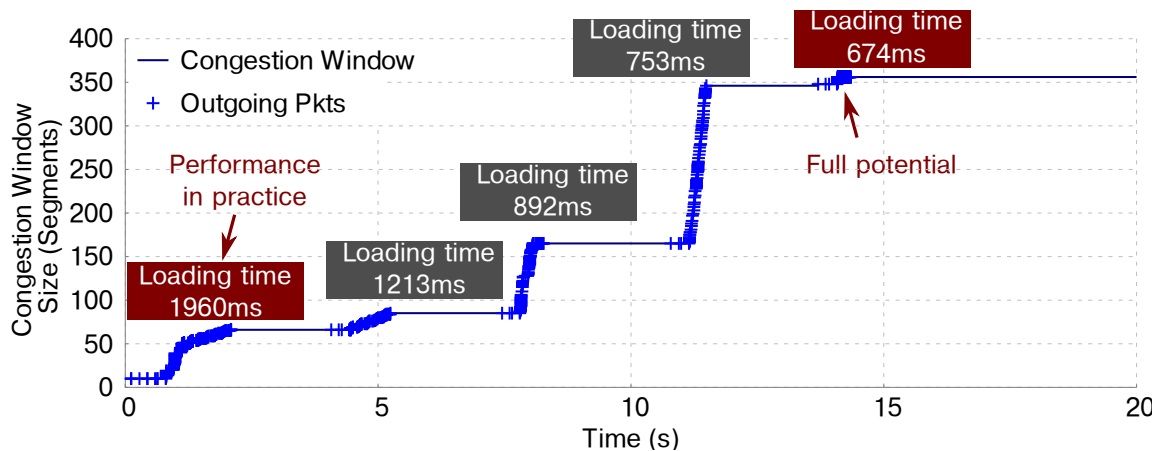


Figure 4.2: A snapshot of TCP sender cwnd for 5 back-to-back loading events visiting the same static HTTP/2 webpage (with browser cache disabled).

To showcase the problem, we load a static HTTP/2 webpage on an LTE phone for 5 times consecutively with small time interval. Fig. 4.2 plots the loading time and evolution of TCP congestion window (cwnd) size. In the 1st trial, due to the slow start mechanism, TCP aggressively increases the cwnd by one upon receiving each ACK (acknowledgement) packet. The cwnd ideally doubles per RTT if it were for bulk transfer, but in fact accumulates slowly due to lack of sufficient HTTP data to fill in the window. In the following trials, since HTTP/2 reuses the TCP connection, the cwnd gradually accumulates and the 5th trial's latency almost reduces to 1/3 of the 1st one. Obviously, TCP fails to fully explore the available bandwidth as mobile Web traffic lacks sufficient data to drive TCP's self-clocked rate adaptation. Thus, *an efficient rate control algorithm for mobile Web should be able to estimate the network bandwidth regardless of the source bitrate.*

4.4 CLAW Design

We propose Cellular-Link-Aware Web-loading (CLAW), which harnesses LTE's PHY-layer statistics, including signal energy, packet loss and modulation schemes, to accelerate the network bandwidth exploration for mobile Web traffic. CLAW aims to approach a design ideal that makes TCP converge to network bandwidth

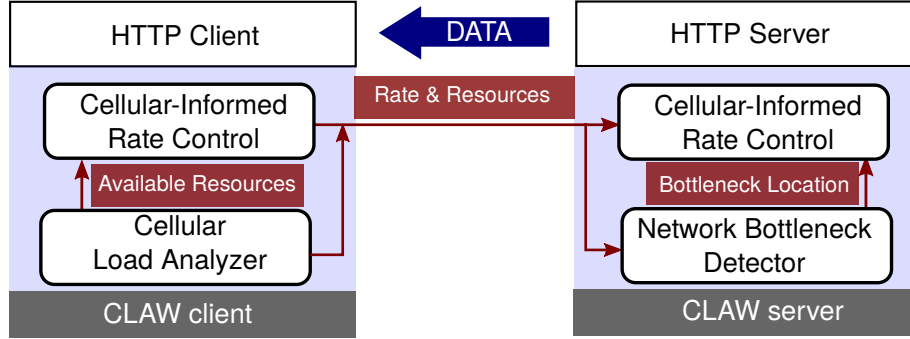


Figure 4.3: CLAW system architecture.

within one RTT. Below we outline CLAW's operations and components that work towards the design goal.

CLAW consists of three mechanisms running on the HTTP client (*i.e.*, LTE phone) and server (Fig. 4.3). The CLAW client runs a *cellular load analyzer* that models the cell-wide downlink traffic load on BS and estimates available resources, based on the real-time PHY statistics locally available from the phone's diagnostic interface. The client then performs *cellular-informed rate control* to map available resources into additional traffic that can be carried over the network path, and accordingly optimizes the congestion window size to be fed back to the CLAW server. Meanwhile, the client employs a *cellular bottleneck identifier* to detect and adapt to the rare cases with wireline rather than LTE link being the network bottleneck. In what follows, we introduce CLAW's design components in detail.

4.4.1 Cellular Load Analyzer

Instead of relying on the irresponsive network congestion indicators like packet delay/loss to probe the network capacity, CLAW can quantitatively assess the gap between current traffic rate and the network capacity based on the *cell load*, *i.e.*, the volume of allocated radio resources (time/frequency) on the downlink channel. It then translates the load estimation into a proper sending rate for the server that fully utilizes the network resource without causing congestion.

Ideally, the BS knows the cell load and can pass such information to all clients.

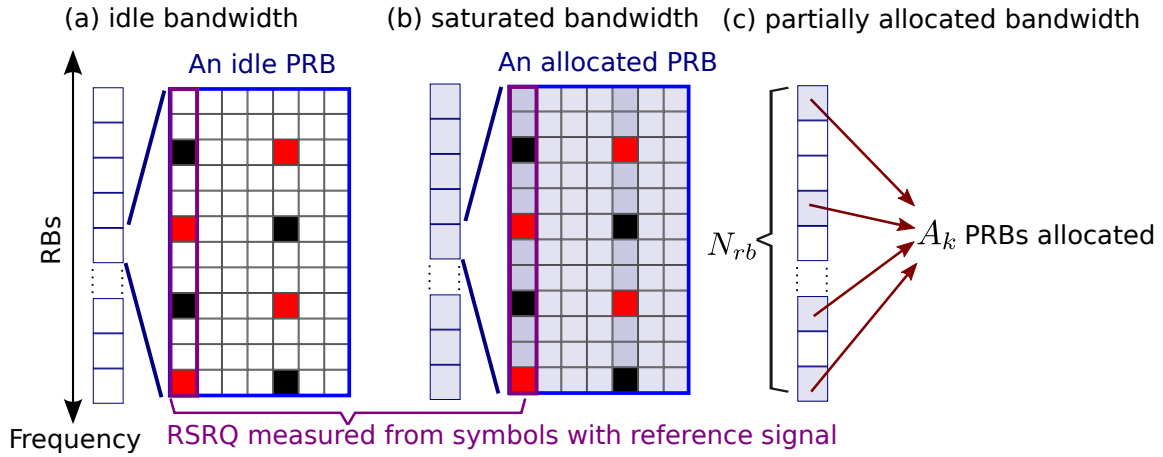


Figure 4.4: LTE reference signal under different resource allocation status (example shows a 2-antenna BS).

But such a new signaling channel requires modifications to the LTE MAC/PHY standard. Alternatively, cell load information may be obtained if a client can decode the BS's downlink control channel which specifies the resource allocation. But the BS logically isolates the control channel among different clients[29], so that the standard LTE clients are blind to each other's resource usage. More specifically, on commodity smartphones, although the diagnostic interface can expose the per-subframe resource allocation statistics to the user space, it only shows the resource allocated to the phone itself, while the cell-wide resource utilization information remains unavailable. Substantial hardware modifications or external sniffers[70, 59] are needed to enable an LTE client to decode the downlink control channel dedicated for other clients. The aim of CLAW's cellular load monitor is to make precise cell load estimation by harnessing the PHY metrics available on unmodified commodity phones.

Quantitative cell load estimation based on RSRQ. The RSRQ metric is originally provided to LTE clients to assist local decisions like handoff and bitrate adaptation. By digging deep into the LTE resource structure, CLAW repurposes RSRQ and quantitatively maps it to the cell load, *i.e.*, a BS' cell-wide downlink resource allocation. In what follows, we describe the analytical model for the mapping.

(i) When the downlink channel is idle, the client only receives reference signals, and thus the total received power over one PRB is $N_{\text{ref}}P$, where N_{ref} is the number of REs occupied by reference signals in one PRB, and P is the RSRP. Meanwhile, the RSSI is measured within the OFDM symbols carrying reference signals. Since the OFDM symbols span the entire bandwidth or N_{PRB} PRBs, the resulting RSSI is:

$$I_{\text{idle}} = N_{\text{PRB}}N_{\text{ref}}P \quad (4.1)$$

Per LTE specification, $N_{\text{ref}} = 2$ and 4 , for $N_{\text{ant}} = 1$ and 2 , respectively, where N_{ant} denotes the number of antennas on the BS. Combining Eq. (2.1) and (4.1), the corresponding RSRQ equals $1/N_{\text{ref}}$, *i.e.*, $1/2$ and $1/4$ (or -3 dB and -6 dB), respectively.

(ii) Now we analyze a saturated downlink channel with full resource utilization. The reference signals from different BS antennas do not overlap [5], so the received power of an RE carrying reference signal still equals P , whereas the power on an RE carrying data becomes $N_{\text{ant}}P$. As one PRB spans across $N_{\text{subc}}^{\text{PRB}} = 12$ REs, the power of one PRB equals $N_{\text{ref}}P + (N_{\text{subc}}^{\text{PRB}} - N_{\text{ref}})N_{\text{ant}}P$, and the RSSI over the fully occupied bandwidth is:

$$I_{\text{full}} = N_{\text{PRB}}(N_{\text{ref}} + (N_{\text{subc}}^{\text{PRB}} - N_{\text{ref}})N_{\text{ant}}) \cdot P \quad (4.2)$$

From Eq. (2.1) and (4.2), we infer that on a saturated LTE channel, $\text{RSRQ} = 1/12 \approx -11\text{dB}$ and $\text{RSRQ} = 1/20 \approx -13\text{dB}$ for BS with $N_{\text{ant}} = 1$ and 2 , respectively.

(iii) For a downlink channel with intermediate traffic load, consider a subframe k with A_k out of the total N_{PRB} PRBs allocated, the corresponding RSSI consists of the power over the A_k occupied PRBs and $(N_{\text{PRB}} - A_k)$ idle PRBs:

$$I_k = A_k I_{\text{full}} + (N_{\text{PRB}} - A_k) I_{\text{idle}} \quad (4.3)$$

Combining with Eq. (2.1), we have the RSRQ:

$$Q_k = \frac{N_{\text{PRB}} P}{A_k I_{\text{full}} + (N_{\text{PRB}} - A_k) I_{\text{idle}}} \quad (4.4)$$

Given I_{idle} from Eq. (4.1) and I_{full} from Eq. (4.2), we can derive the relation between Q_k , the RSRQ, and k , the number of allocated PRBs in this subframe:

$$Q_k = \frac{1}{A_k (N_{\text{subc}}^{\text{PRB}} - N_{\text{ref}}) N_{\text{ant}} / N_{\text{PRB}} + N_{\text{ref}}} \quad (4.5)$$

To validate this model, we record the number of allocated PRBs for each subframe with an RSRQ measurement when performing web loading over LTE. Our result combines records from multiple BSs in our area, all of which have 10MHz bandwidth ($N_{\text{PRB}} = 50$) and 2 antennas. Fig. 4.5 plots the average RSRQ value for each subframe-level PRB allocation number. We can observe that the measurement results match the theoretical computation well. As a result, the fraction of allocated PRBs in a time window, or cell load, can be reliably estimated based on the LTE-frame level RSRQ measurements following Eq. (4.5), which is the foundation of CLAW design.

Estimating available resources for a client. Following the model in Eq. (4.5), CLAW computes the instantaneous resource utilization, *i.e.*, fraction of allocated PRBs in a subframe, as:

$$r_k = A_k / N_{\text{PRB}} = \frac{Q_k^{-1} - N_{\text{ref}}}{(N_{\text{subc}}^{\text{PRB}} - N_{\text{ref}}) N_{\text{ant}}} \quad (4.6)$$

In practice, CLAW reads Q_k , N_{ant} from the phone's diagnostic interface, and all other parameters in Eq. (4.6) can be derived from the LTE standard, following our foregoing analysis.

To combat the burstiness in network traffic and the model variance shown in Fig. 4.5, CLAW accesses the current cell load L by averaging r_k across all K samples

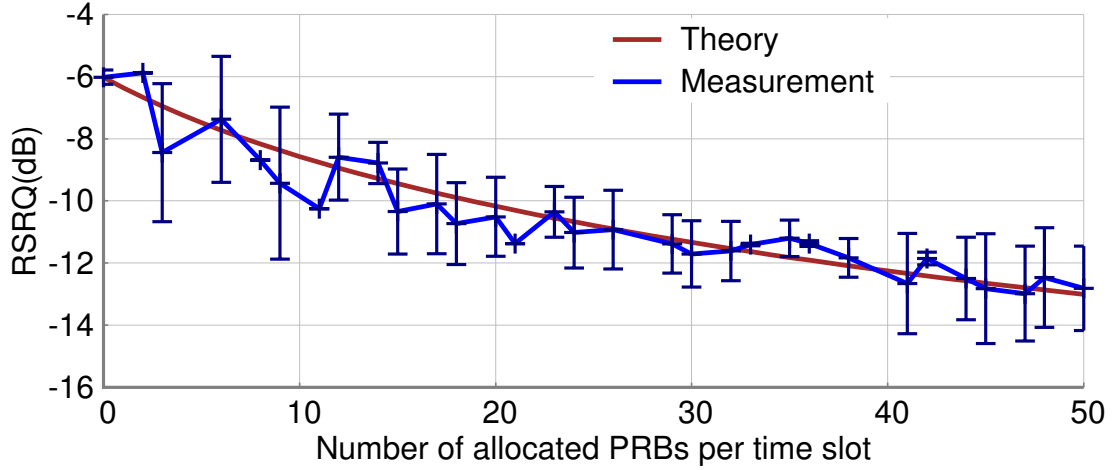


Figure 4.5: CLAW models LTE downlink resource allocation using a phone’s locally sensed RSRQ metric. This figure compares the model with measurement (error bars show std.).

of RSRQ collected during a sliding *time window* of length T :

$$L = \left(\sum_{k=1}^K r_k \right) / K \quad (4.7)$$

An interesting fact is that if we set the window size T as the round-trip time (RTT), the traffic capacity of the currently unused radio resources in this window is exactly the gap between current TCP congestion window and its target value — the bandwidth-delay product. Therefore, CLAW uses RTT as its time window size T .

Given the cell load L , a CLAW client can gauge the *number of PRBs allocated to all clients* within this time window T , as $N_a = N_t L$, where N_t denotes the total number of PRBs within T . N_t can be easily obtained following the LTE specification: Since each subframe lasts for 1ms, the number of subframes in a time window equals its length T in milliseconds. Also, one subframe consists of two time slots, each containing N_{PRB} PRBs, so the total number of PRBs within T equals $N_t = 2N_{\text{PRB}}T$.

Then the client can obtain N_I , the *number of idle PRBs* within T :

$$N_I = N_t - N_a = 2N_{\text{PRB}}T(1 - L) \quad (4.8)$$

Denote the *number of PRBs allocated to the client itself* as S_k , which can be read from the phone's diagnostic interface for every scheduled LTE subframe. At the end of a time window T , the client obtains N_s , the sum number of PRBs allocated to itself within T :

$$N_s = \sum_{k=1}^T S_k \quad (4.9)$$

Finally, CLAW estimates N_f , the number of *available* PRBs in this time window T , which consists of its currently consumed PRBs and additional idle PRBs it can leverage:

$$N_f = N_s + N_I \quad (4.10)$$

4.4.2 Cellular-Informed Rate Control

Given the estimation of currently available radio resources, CLAW performs cellular-informed rate control to shape the traffic bitrate to the optimum by adapting the transport-layer congestion window (cwnd) size. Algorithm 2 outlines our design and we provide more details in what follows.

Cellular-Informed Bandwidth Exploration

Quick bandwidth convergence facilitated by cellular link statistics. During bandwidth exploration stage, CLAW boosts the server's sending rate to network capacity based on available radio resources. After estimating the volume of available PRBs (N_f) in current RTT-sized time window T_i , the CLAW client translates N_f into the congestion window size matching the potential network capacity. Specifically, from the phone's diagnostic port, the CLAW client reads the Modulation and Coding

Algorithm 2 Cellular-Informed Rate Control.

Input: $\overline{\text{MCS}}$ – Average MCS in sliding time window
 N_a – Sum number of PRBs allocated to all clients
 N_t – Total amount of PRBs in the sliding time window
 N_f – The number of PRBs available to this CLAW client
 $\mathcal{P}$ – Bottleneck location (0 is cellular bottleneck)

Output: cwnd – to be used by server

```

1: /*CLAW Client*/
2: Slide forward RTT-sized time window by one subframe
3: Estimate BDP  $\hat{B}$  based on  $\overline{\text{MCS}}$  and  $N_f$  following Eq. (4.11)
4: The optimal congestion window  $W_{\text{claw}} = \hat{B}$ 
5: Feedback  $W_{\text{claw}}$  to the CLAW server
6: Repeat the steps above
7: /*CLAW Server*/
8: On receiving a new  $W_{\text{claw}}$ 
9: if ( $\mathcal{P} == 0$ ) then /*Cellular bottleneck*/
10:   if  $N_a/N_t < 1$  then
11:     /*Cellular-Informed BW Exploration*/
12:      $\text{cwnd} = W_{\text{claw}}$ 
13:   else /*On saturated channel, act as aggressive as Cubic*/
14:      $\text{cwnd} = \max(W_{\text{cubic}}, W_{\text{claw}})$ 
15: else /*Wireline bottleneck*/
16:    $\text{cwnd} = W_{\text{cubic}}$ 

```

Scheme (MCS) for every allocated LTE subframe. It then estimates the cellular link capacity $\hat{B}$ that can be supported by the N_f PRBs under average MCS in current time window:

$$\hat{B} = T_i F(N_f/T_i, \overline{\text{MCS}}) \quad (4.11)$$

where the mapping rule $F(\cdot)$ is specified in the LTE standard [71, Table 7.1.7.1 and 7.1.7.2], which maps the MCS and the number of PRBs per subframe (N_f/T_i) to the *Transport Block Size*, i.e., the number of bytes that can be carried over a subframe. This size $F(N_f/T_i, \overline{\text{MCS}})$ is further multiplied by T_i , the number of subframes in a time window, to obtain the potential capacity $\hat{B}$. Since CLAW uses RTT as the time window length (§4.4.1), $\hat{B}$ essentially equals the bandwidth-delay-product, i.e.,

the ideal congestion window size W_{claw} that best fits current network condition. W_{claw} is updated every time the sliding time window T moves forward, it is then fed back to the server, which sets W_{claw} as the congestion window (cwnd).

CLAW's cellular-informed bandwidth exploration design essentially addresses the conflict between the sparse Web traffic and the data-driven network capacity probing mechanism discussed in §4.3. Even with a small volume of traffic, CLAW can still instantaneously *converge to the available LTE bandwidth in one RTT*.

Hybrid bandwidth exploration mode. Note that a fully utilized LTE downlink channel does not necessarily imply the client has converged to its fair-share bandwidth. For instance, when the cellular downlink is saturated and all PHY resources are fully exploited, a new CLAW user that just joined the cell will keep estimating $W_{\text{claw}} = 0$, thus failing to explore the network bandwidth. To handle such situations, CLAW employs a *hybrid bandwidth exploration* mechanism, in which the sender maintains another window size W_{cubic} following the legacy TCP Cubic algorithm, and finally selects the larger one between W_{claw} and W_{cubic} as the actual cwnd. To avoid the false alarm of channel saturation caused by the traffic burstiness, we allow CLAW to switch to TCP Cubic only after it keeps detecting no idle resources for two consecutive RTTs, and then compensate it for the window size it could have increased in these two RTTs when it performs the mode switching. This hybrid bandwidth exploration design ensures that even over a saturated cellular link, the newly joined CLAW clients can still compete for the bandwidth as aggressively as the legacy TCP flows.

To demonstrate its effectiveness, we showcase a running instance of the hybrid bandwidth exploration in Fig. 4.6. We plot the server's cwnd and mark the sending time of each outgoing packet during a Web loading session over a saturated LTE channel. We see that the CLAW sending rate freezes in the beginning since there are no idle resource to explore. But after the 2-RTT inertia, mode switching is triggered: it first has a steep cwnd increase which follows the aforementioned window size compensation, and then ramps up with TCP Cubic's slow start operations.

Multi-user fairness. CLAW's cellular load awareness enables a client to detect the activities of the competing downlink clients in the same cell, which allows it

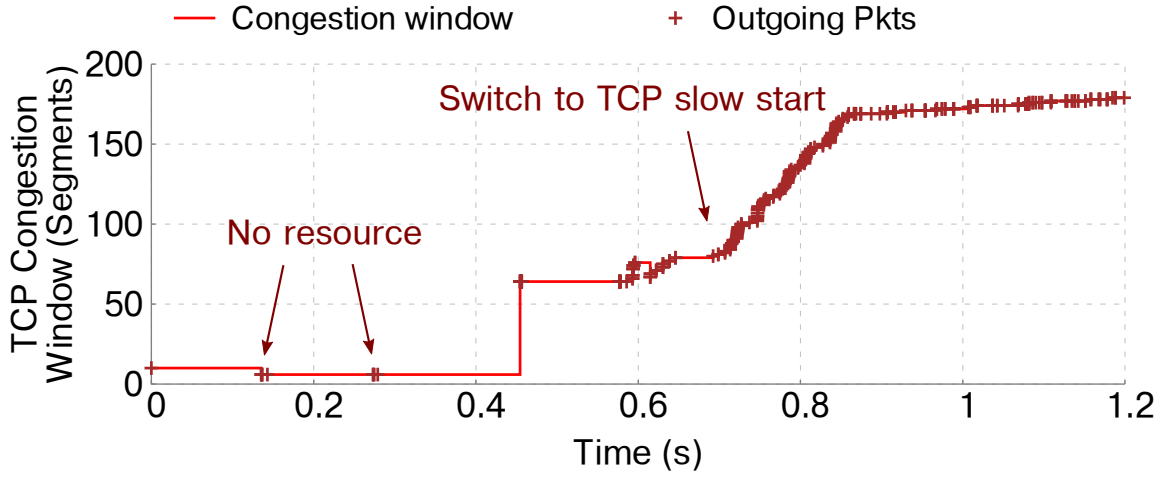


Figure 4.6: A running instance of the cellular-informed hybrid bandwidth exploration over an LTE channel saturated by existing users.

to explicitly achieve multi-user fairness. In contrast, conventional TCP is blind to competing clients and can slowly approach fair-share only for long-lived flows based on probing-and-trial. Specifically, CLAW design ensures fairness no matter when coexisting with conventional TCP flows or other CLAW flows. In what follows, we further explain how CLAW ensures the fairness on a case-by-case basis.

(i) When a CLAW client joins the LTE cell after legacy TCP clients, it only takes its current allocated resources plus the idle resources. It does not attempt to preempt the resources occupied by other clients, and thus it will not degrade the performance of other users. If the existing clients have already saturated the LTE downlink, CLAW's hybrid bandwidth exploration ensures that the CLAW client can fall back to TCP Cubic, thus guaranteeing its fairness when competing with the legacy TCP flows.

(ii) When a TCP Cubic client joins the LTE cell after existing CLAW clients, its experienced level of fairness is exactly the same as competing with conventional TCP Cubic clients, since the CLAW clients will fall back to TCP Cubic after the idle resources are exhausted.

(iii) When multiple CLAW clients join the LTE cell over time, the clients arriving later keep observing zero available resources. In such cases, the hybrid bandwidth exploration design makes all the CLAW clients fall back to TCP Cubic, so the

fairness level is the same as conventional networks with competing TCP Cubic flows.

(iv) *When a client leaves the LTE cell, the hybrid bandwidth exploration mode guarantees that the CLAW client can immediately consume the released resources.*

We emphasize that falling back to TCP Cubic ensures at least the same level of fairness compared to the legacy TCP Cubic, but CLAW can achieve even better fairness than TCP Cubic if we allow the existing clients to intentionally reduce their traffic rate when the new clients starting with low rates are detected (total resource utilization N_a increases but self resource utilization N_s remains unchanged). As a result, *CLAW achieves better fairness than conventional TCP since its resource awareness enables its rate adaptation to explicitly follow the BS's resource scheduling.*

Discriminating Link Loss from Congestion

Packet losses over cellular links, though infrequent, can severely disturb conventional TCP's congestion estimation, and lead to slow and unstable Web loading (§4.3). CLAW overcomes this fundamental limitation by discriminating the wireless loss and congestion loss using cellular link statistics available on commodity phones. LTE's RLC (Radio Link Control) layer handles the packet retransmission process on the downlink channel. The RLC layer statistics are available through the phone's diagnostic port, which include the number of packet losses, number of retransmitted packets and an overflow indicator for the RLC-layer packet queue. The CLAW client reports the RLC-layer loss events to the server, which then compare this information with the transport-layer packet losses. For instance, if a transport-layer loss happens along with RLC-layer losses caused by poor channel quality, the server notices it is not a congestion loss, and thus refrains from reducing the sending rate (congestion window). In contrast, the server will slow down immediately if the reported reason of packet loss is the buffer overflow at the LTE BS.

To facilitate the loss information feedback, CLAW marks the reserved bits in the header of the TCP ACK packets to represent different type of RLC loss events, so that the server can distinguish the wireless loss and congestion loss by reading

these marks in the duplicate ACKs triggered by losses. §4.5 introduces how we implement this ACK flagging in the Android OS.

4.4.3 Network Bottleneck Detector

CLAW’s cellular-informed rate control mechanism essentially boosts the traffic bit-rate to fit the client’s share of cellular network capacity. However, in some corner cases, the network bottleneck may appear at the wireline path rather than the cellular link, and sending the traffic at the cellular capacity will lead to congestion at the bottleneck. To achieve a robust design, CLAW needs to detect and adapt to the rare scenarios where wireline bottlenecks the network.

Online bottleneck detection. CLAW’s network bottleneck detector builds on a simple principle originating from its bandwidth exploration mechanism – If CLAW boosts the sending rate to harness currently unallocated resources, but the client’s resource usage does not increase correspondingly, then there should be a bottleneck along the wireline path between the server and the BS.

To demonstrate the feasibility of this principle, we record the LTE resource allocation when loading the same Website, across a wireline bottleneck and cellular bottleneck, respectively (detailed implementation in § 4.6.1). The results in Fig. 4.7 contrast the resource allocation patterns in the two cases, which show clear distinction.

Following this principle, the workflow of the network bottleneck detector design is outlined in Fig. 4.8. It runs in parallel with the cellular-informed rate control, at a time granularity of RTT level. Specifically, our design compares the volume of resources (N_f) available to a client, and its consequently allocated amount of resources (U) after one RTT. If $U < \delta N_f$, the detector declares a wireline bottleneck, otherwise a cellular bottleneck. The multiplicative factor $0 < \delta < 1$ controls the safe margin to accommodate for the RTT variations that can occasionally make U less than expected due to some delayed packets. We empirically set δ to 0.8. To further combat the network variations, the bottleneck is asserted only after consistent observations over a sliding window of 3 RTTs.

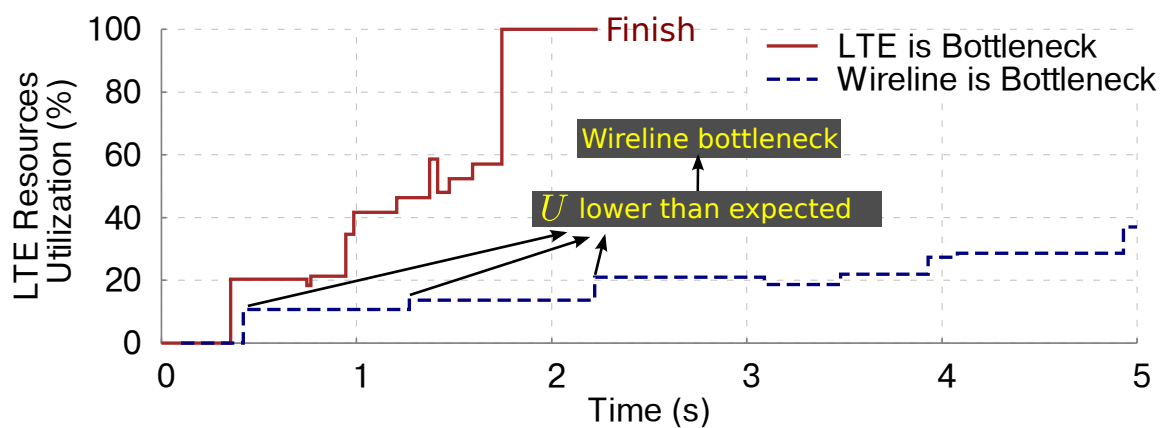


Figure 4.7: CLAW detects wireline bottleneck based on resource allocation pattern during bandwidth exploration.

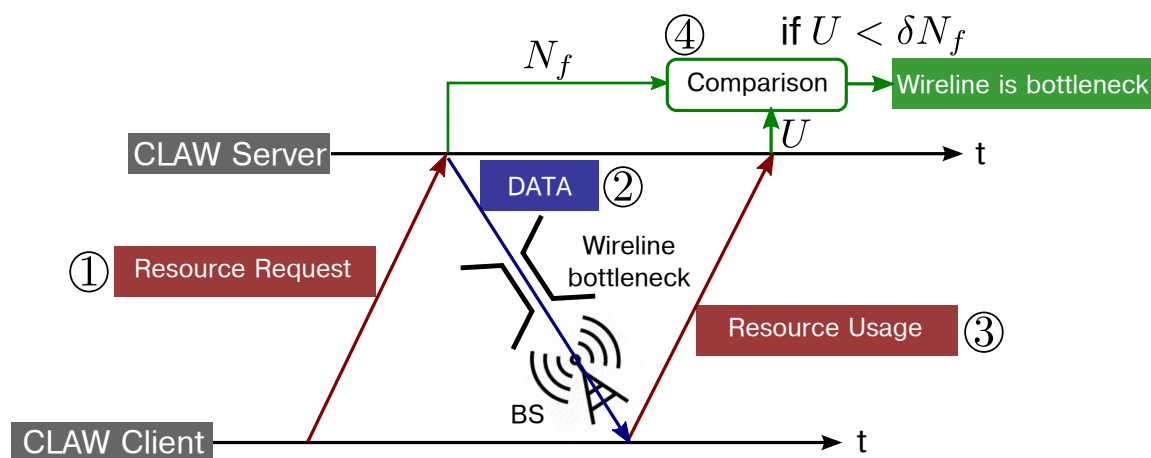


Figure 4.8: The network bottleneck detector in CLAW.

We emphasize that the bottleneck detector should be used as a sanity check. In most common cases, the wireless and wireline bottlenecks should have a non-trivial gap, even under channel variations, cross traffic disturbances, *etc.* So even a coarse check suffices. If these two happen alternately and inconsistently for a client, then they may be comparable and CLAW should degrade to the conventional TCP to avoid unnecessary oscillation.

Adapting to wireline bottleneck. After detecting a wireline bottleneck, CLAW falls back to TCP-style rate control. However, we emphasize that this fallback mode is still superior over conventional TCP, as it can still use cellular link statistics to discriminate wireless losses from congestion losses. Since the capability to discriminate between wireline and cellular bottlenecks is essentially a byproduct of the cellular-informed rate control and incurs zero overhead, CLAW client can run it in the background in parallel with web loading. Since it takes 3 RTTs to detect a wireline bottleneck, to avoid this latency, the client can also cache the decision based on server’s IP address and reuse it for later sessions.

4.5 CLAW Implementation

Client side. Our implementation reads the logs containing the cellular link statistics from the diagnostic interface available on most cellphone models with Qualcomm chipsets. In Android system, the logs are passed to user space through `/dev/diag`, following the implementation of MobileInsight[72]. From such logs, we decode the RSRQ, MCS, PHY resource (PRB) allocation, subframe index, and other statistics in real time with LTE subframe (1ms) granularity. The output logs of the Qualcomm chipsets rearrange the cellular link statistics into obscure formats that can be decoded only by Qualcomm’s QCAT, a proprietary software tool that parses the logs offline. To automate the real-time decoding, we reverse-engineer the log format by matching the raw HEX payload and QCAT’s decoding results. We timestamp each decoded cellular link statistics based on which subframe it comes from, *e.g.*, the 7th subframe within the frame with System Frame Number (SFN) 100 has timestamp 107 (the elapsed milliseconds since the SFN counting started). Since the SFN cycles

between 0 to 1023, we unwrap this value to avoid ambiguity. The decoded cellular link statistics are then fed to CLAW’s cellular load analyzer implemented in the client’s user space to facilitate the estimation of available resources following our model in § 4.4.1, and the estimation of the optimal congestion window size best fitting current network capacity following the algorithm in § 4.4.2.

In addition, the HTTP traffic pattern occasionally causes long idle durations, *e.g.*, when waiting for the response of HTTP GET, in which no downlink LTE traffic arrives. In such circumstances, the phone’s radio enters the idle state and measures the downlink channel infrequently to save energy (only 5 RSRQ measurements per second), and hence is largely blind to the background traffic. Fortunately, since there is no traffic at all, the CLAW flow does not need to update the TCP congestion window. Therefore, the CLAW algorithm is frozen in such idle durations and takes effect only when a TCP connection is activated. In contrast, LoadSense[60] attempts to estimate the background traffic load before the client cuts into the cellular channel, where the radio is still in idle state and listens to the channel infrequently for every 200ms. Hence LoadSense can only get a rough correlation between the RSRQ and cell load, and a binary cell load estimation of “idle” or “busy”. As a result, the on-the-fly operation of CLAW guarantees its more accurate cell load estimation over existing works.

For prototyping purpose, our current CLAW client implementation uses a UDP path to carry the optimal congestion window size for every 100 ms from the client to the server. Our future implementation will substitute this out-of-band feedback with an in-band signaling using the TCP Option field which can be more responsive. In addition, the client should also feedback the cellular-related packet loss events caused by BS buffer overflow or poor wireless channel (§4.4.2). To precisely synchronize the RLC-layer loss events reported by the diagnostic interface and the corresponding TCP packet loss, CLAW piggybacks the feedback in the TCP header like ECN-based TCP variants. To keep compatibility with current packet format, we use two of the 6 reserved bits in the TCP ACK’s packet header (`th->res1`), one as the flag for the wireless channel loss, the other for the BS buffer overflow. To implement such a feedback channel in the TCP packet header, we modified

`tcp_input.c` of the Android kernel to mark the wireless loss and buffer overflow notification bits, and `tcp_output.c` of the server kernel to extract these bits. We also add a customized `sysctl` variable to the Android kernel so that the CLAW application running in the user space can pass the identified loss & overflow events to the kernel.

Server side. We setup our test websites using an Apache HTTP/2 server [73] on an Amazon EC2 instance which also acts as the CLAW sender. At the server side, we tune the Linux TCP buffer size to guarantee the performance over high-speed networks like LTE. Besides, we set `net.ipv4.tcp_no_metrics_save` to 1 to ensure each Web loading test will not be affected by previous tests. To guarantee the performance of HTTP/2 which reuses TCP connection, `net.ipv4.tcp_slow_start_after_idle` is set to 0 as suggested in the HTTP/2 Wiki[74]. Upon receiving feedback containing the client’s estimated `cwnd` size, the server passes it to CLAW’s cellular informed rate control algorithm (§4.4.2), which runs in the user space and writes its `cwnd` size decision to the kernel via a customized `sysctl` interface. The new window takes effect by superseding the legacy TCP window just before the `cwnd` value is used for rate throttling within the packet transmission function `tcp_write_xmit()`. Such simple retrofitting allows flexible fallback to legacy TCP when CLAW identifies a wireline bottleneck (§4.4.3). For evaluation purpose, our modified server kernel also logs the packet losses, retransmissions, ACKs and behaviors of CLAW in real time. Since running CLAW implementation may change the TCP state parameters and HTTP/2 reuses TCP connection, to minimize the impact of the previous test to current Web loading, we reinitialize the TCP congest control state parameters by calling `tcp_init_congestion_control(sk)` in the TCP output engine at the end of each CLAW test. Overall, the majority of CLAW implementation lies in the client’s user-space, and it only involves 41 lines of kernel patch at the server side.

4.6 CLAW Evaluation

Experiment setup. We benchmark CLAW’s performance against both classic TCP variants and state-of-the-art congestion control schemes for cellular networks, in-

Algorithm Name	CLAW	CQIC	TCP Cubic	TCP Vegas	Verus
Bandwidth Exploration	LTE-resource-based	N/A	TCP slow start	TCP slow start	TCP slow start
Congestion Avoidance	LTE-resource-based	SINR-based	loss-based	delay-based	delay-based
Congestion indicator	Resource drain out & loss	N/A	loss & large delay	large delay only	loss & large delay
LTE link information	PRB allocation, MCS, TBS, RSRQ, RSRP	CQI	No	No	No

Table 4.1: CLAW and benchmark schemes.

cluding CQIC [16] and Verus [21] which showed superior performance over alternatives (*e.g.*, Sprout [20]). Following [16], the CQIC implementation takes the LTE TBS (§4.4.2) controlled by the BS’s MCS adaptation and resource allocation as the PHY capacity estimation. This essentially upgrades the original CQIC from HSPA+ to LTE [16]. We implement Verus in the server’s Linux kernel following the source code (UDP-based) [75]. The original Verus updates `cwnd` every 5s, longer than a typical Web session. We thus reduce the time granularity to 500ms. Verus’s bandwidth-delay profile is built during the slow start phase as suggested in [21]. For fair comparison, the tests with different schemes are conducted back-to-back to ensure similar channel conditions, and we employ the kernel’s `sysctl` interface to enable fast and flexible switch between the algorithms.

Webpages and PLT measurement. We download and host a list of the Alexa top sites on an Amazon EC2 server that runs the HTTP/2 and CLAW’s sender implementation. Following [54], the server hosts all Web objects of a website, including the dynamic objects generated by JavaScript, to avoid the clash between domain sharding and HTTP/2. More details about the sites we used can be found in §4.6.2. Our evaluation uses page loading time (PLT) as the primary performance metric, which directly impacts end-user experience. We use Chrome Developer Tools [76] which are widely adopted in Web loading related studies [77, 78, 79] to measure the PLT of a webpage over Android phones. The phones run the Chrome browser, with cache disabled to isolate individual Web loading tests.

Configurations for the phone. The smartphones used in our implementation are rooted so that the application layer can access and read from the diagnosis interface. We emphasize that no hardware modification is required, which is a big advantage over software-radio based cross-layer solutions like piStream[59]. Sometimes the phone may switch between LTE and HSPA when both networks

are available, especially in the mobility experiments. Since the CLAW design is customized for LTE, we select "LTE Only" as the preferred network type inside the diagnostic and general setting mode of Android system to prevent unexpected network switching.

4.6.1 Micro-benchmarks

Validating the cellular load analyzer. We first evaluate the accuracy of the cellular load analyzer and its real-time cell load tracking capability. We generate an `iperf` TCP traffic stream with bitrate increasing linearly by 0.5 Mbps per second, between the server and LTE phone. The experiment runs in late night isolate the interference from background traffic. Figure 4.9 plots the raw cell load (frame-level granularity) as scattered points, the smoothed cell load curve using a 200ms (approximately RTT sized) sliding window following Eq. (4.7), and the network layer traffic pattern measured at the phone using `tcpdump`. The results show that the raw cell load vary drastically over time, with a polarized resource allocation pattern – the BS either allocates most PRBs in a subframe or leave it empty, implying that it enforces frame aggregation. After smoothing, we can observe a high consistency between the PHY-layer cell load estimated following §4.4.1 and the corresponding network layer traffic intensity, which verifies the accuracy of the cellular load analyzer design. In addition, this result demonstrates the feasibility of tracking cell load in real time using the phone’s diagnostic interface.

Validating the cellular-informed rate control. To demonstrate the effectiveness of *Cellular-Informed Rate Control* in CLAW design, we load the same webpage using CLAW and TCP Cubic, the most widely deployed TCP variant. For fair comparison, we run the two experiments consecutively over the same idle LTE channel.

Fig. 4.10a plots the sender’s `cwnd` over time. We see that CLAW explores the network bandwidth much faster than TCP Cubic. Once the data transfer starts, it first quickly maps the PHY resource to a close-to-optimum `cwnd`¹, and then

¹The sender side `cwnd` surge happens at around $t=500\text{ms}$, which exceeds one RTT. This issue is caused by the overhead to read the diagnostic interface, and process & feedback the computed

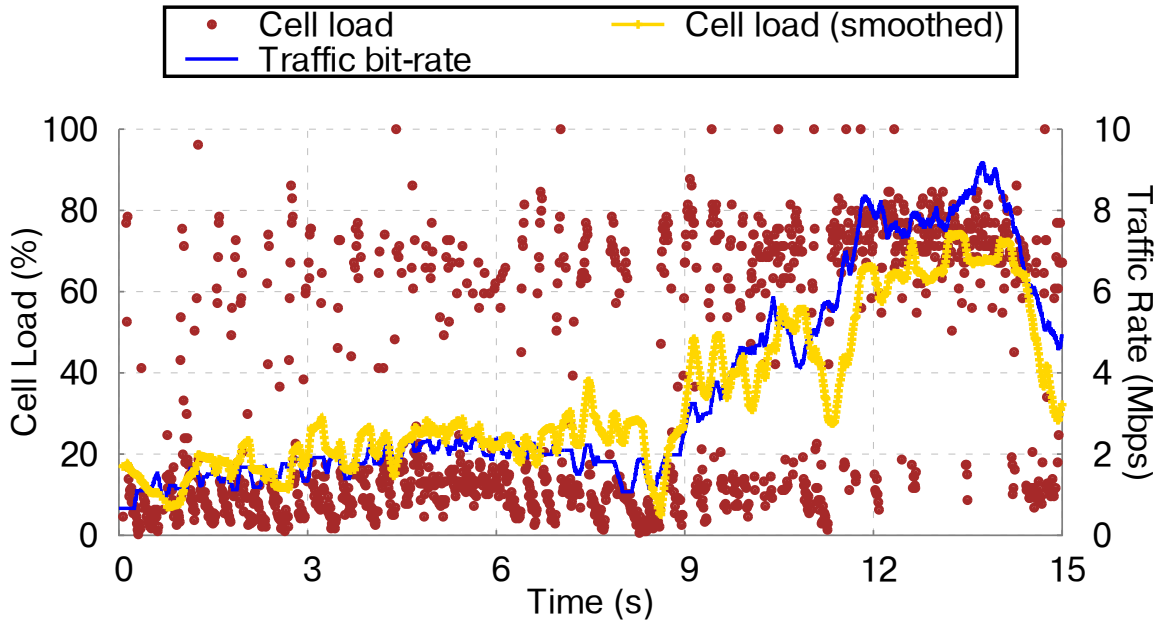


Figure 4.9: A microscopic demonstration of how CLAW’s cell load analyzer can reliably track the cell load.

keeps adapting the *cwnd* based on channel quality. Its downloading finishes early at 1.7s due to effective utilization of the bandwidth. In contrast, TCP Cubic can only gradually explore the bandwidth first via slow start, and then overshoots the bandwidth and jams the LTE downlink, because its congestion indicator, packet loss, is always masked by the large LTE buffer[80]. With queue building up at the BS, the RTT surges up and eventually exceeds RTO (Retransmission Timeout), which in turn causes multiplicative rate decrease. Such trial-based probing operations lead to long PLT (2.6s). Fig. 4.10a also plots one additional experiment that shows a case when Cubic’s slow start fails due to RTT variation (§ 4.3), which causes early termination of slow start and hence even longer PLT (3.2s).

Fig. 4.10b further compares the corresponding resource utilization reported from the client. We can observe that the BS’s PHY resource allocation follows the sender’s traffic intensity, and CLAW exploits LTE resources much more efficiently than TCP Cubic.

cwnd in the application layer. A more refined implementation can further reduce such overhead and improve the responsiveness.

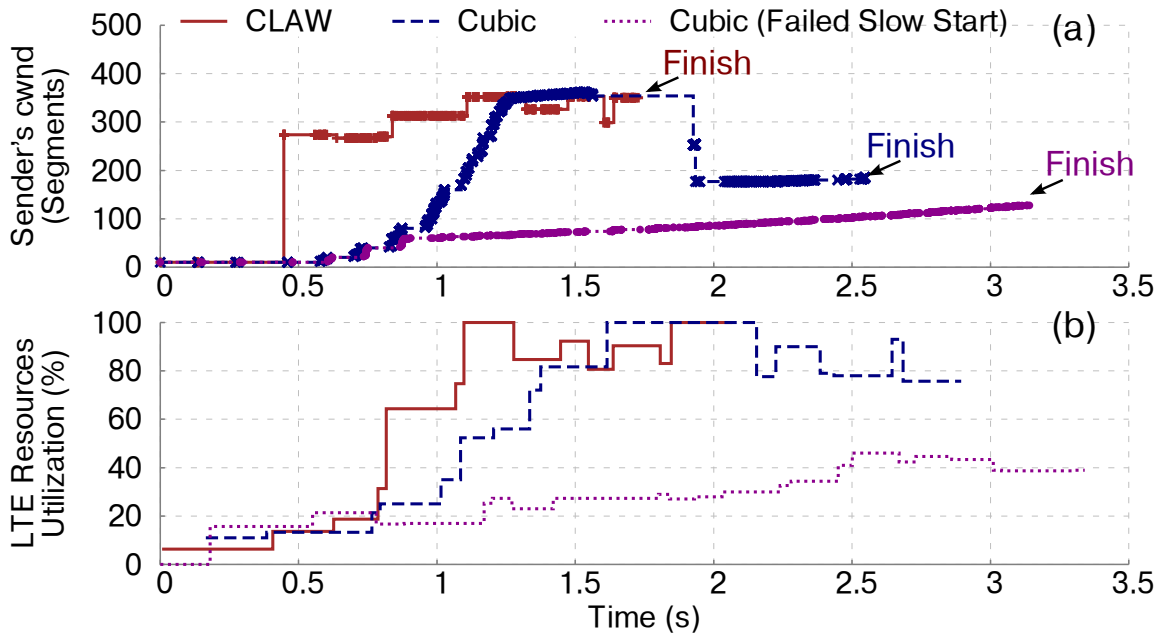


Figure 4.10: (a) CLAW server explores the network bandwidth faster than TCP Cubic (b) CLAW client is thus allocated more LTE PHY resources than TCP Cubic.

To inspect CLAW's bandwidth utilization in comparison with other Linux TCP variants, we create a bare-bones HTML page with a single 2MB image, and load the page 20 consecutive times using each algorithm. Fig. 4.11 plots the average PLT and its standard deviation across 20 trials. CLAW obviously outperforms all the TCP variants and, compared to the runner-up algorithm TCP Cubic, it reduces the average PLT by 3.44s (56%). Remarkably, it also significantly reduces the PLT variance, reducing the worst case PLT by 9.92s (73%) compared with Cubic.

Fig. 4.12 further shows a microscopic inspection for the *packet-level* RTT of CLAW, along with the loss-based (Cubic) delay-based (Vegas) TCP. Since Vegas throttles rate on high RTT, it shows 48ms (15%) smaller mean RTT than CLAW, whereas interestingly its PLT is $2.2\times$ longer! This is because Vegas often overreacts to delay spikes, resulting in extremely small queues and leaving the LTE BS barely anything to send. On the other hand, the BS's deep buffer makes Cubic insensitive to queue build up, misleading it to send at a high rate which causes self-inflicted congestion [20] and hence large RTT.

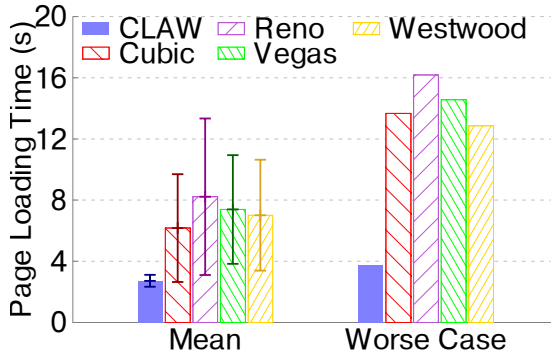


Figure 4.11: Web loading latency for a bare-bone Web page with a 2MB image.

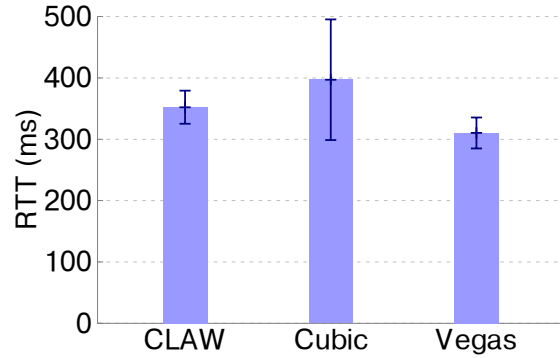


Figure 4.12: Packet latency (RTT) during loading Web page with a 2MB image

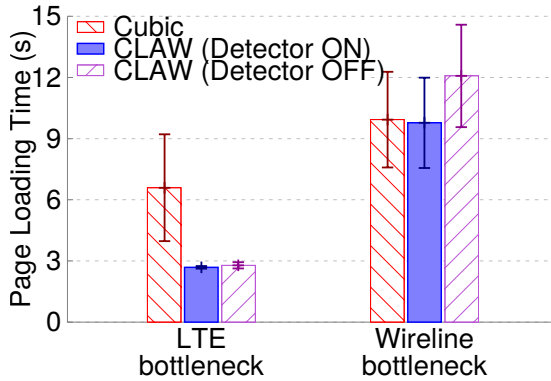


Figure 4.13: CLAW's bottleneck detector handles the rare wireline bottleneck.

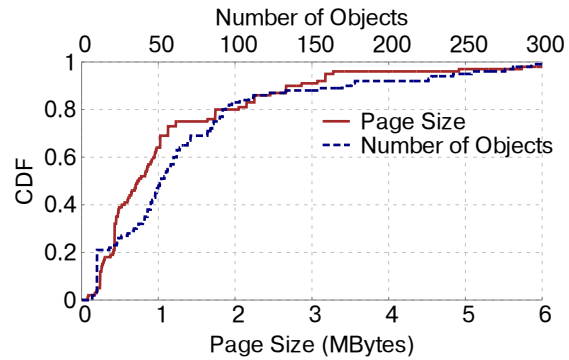


Figure 4.14: Profiling Alexa top 200 Web-site sizes.

Effectiveness of CLAW's bottleneck detector. Since wireline bottleneck rarely occurs, to evaluate the bottleneck detector, we manually throttle the Web server's network interface at 30 Kbps using the Dummynet kernel module [81]. We run the same bare-bones page downloading experiment as above. Fig. 4.13 shows that, without the bottleneck detector, CLAW's performance becomes comparable to Cubic, because attempts to leverage the idle resources on the LTE channel when its sending rate already exceeds the wireline bottleneck capacity, which causes high RTT, packet loss or retransmission timeout. In contrast, we can see 2.3s (20%) latency reduction after turning on CLAW's bottleneck detector and fallback mechanism, which is also slightly better than the TCP Cubic performance since CLAW's

fallback mode can still distinguish wireless losses. This experiment verifies that the bottleneck detector can effectively guarantee CLAW's performance in the rare case of wireline bottleneck.

4.6.2 System Level Test

In this section, we perform a system level test that compares CLAW's PLT against other transport protocols, across various Websites and network conditions. To control the experiment environment, unless otherwise stated, all tests load identical webpages on the same remote server. Each test loads the page for 20 times consecutively. The interval between adjacent tests is around 10s, long enough to ensure isolation and prevent HTTP/2 from reusing the previous cwnd.

CLAW with Different Websites

We conduct experiments with popular webpages selected from the top 200 Alexa list [82]. Fig. 4.14 plots the CDFs for both the page size and number of objects.

What kind of Webpages benefit the most from CLAW? We first evaluate CLAW's performance on 20 selected webpages that cover a wide spectrum of size and object number. To rule out the influence of other factors, we experiment on the idle LTE channel during late night at a static location, with other settings following the default configuration discussed at the beginning of this section. The scatter plot in Fig. 4.15 shows CLAW's PLT reduction *w.r.t.* TCP Cubic (in percentage) of each tested Website. Note that it is unlikely to have a page with small size but large number of objects, thus the upper-left corner of Fig. 4.15 remains blank.

We can observe that (i) CLAW reduces PLT by 27% to 48% compared with Cubic. (ii) CLAW's performance gain increases with larger page size or larger number of objects. Since CLAW accelerates page loading by faster downloading, it can save more time if the content downloading time accounts for more across the entire page loading process (§4.2). In addition, with more data packets, conventional TCP is more likely to experience at least one outage event, *e.g.*, overreacting to loss/RTT (§ 4.3), which causes extended PLT.

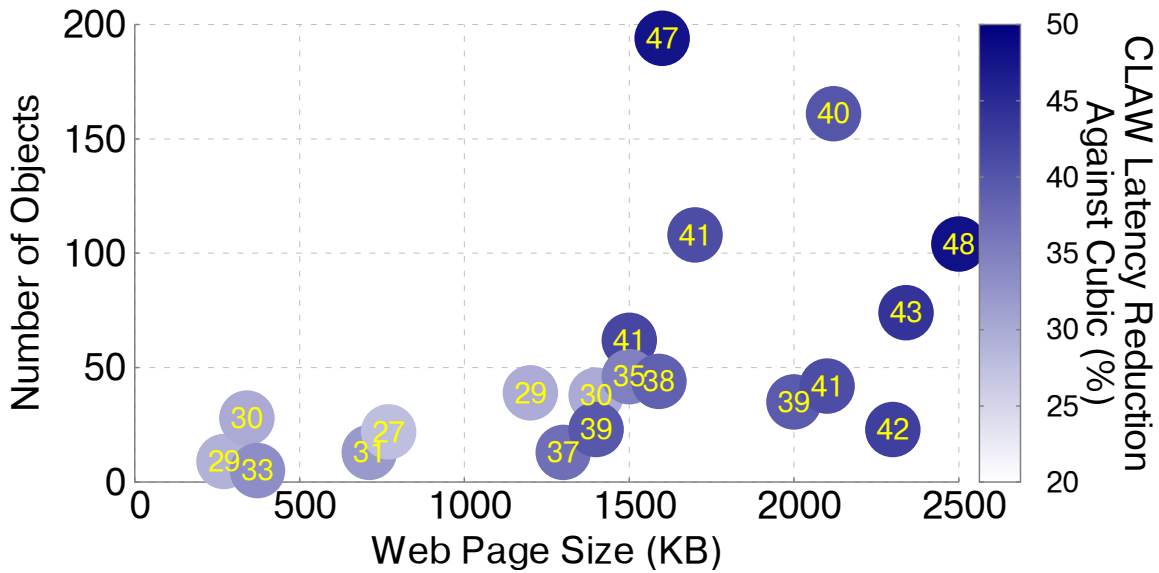


Figure 4.15: CLAW over different websites (number in circle is the percentage of latency reduction over Cubic).

PLT under different transport protocols. We further compare CLAW and other benchmark protocols over 5 randomly selected popular webpages. From the results in Fig. 4.16, we can observe that (i) CLAW achieves the minimum latency and CQIC is always the runner-up. CQIC has around 50% higher PLT than CLAW because CQIC restricts itself to the currently allocated PHY resource, whereas CLAW effectively estimates the BS's remaining resources that it can exploit. (ii) Among the two delay-based congestion controls, Verus's PLT is even worse than TCP Vegas, because it is designed for long-lived flows, and cannot reliably learn the bandwidth-delay profile within the short HTTP flow.

CLAW under various network conditions

In the following set of experiments, we evaluate the impact of various network conditions to the page loading performance of CLAW and benchmark protocols. To isolate the influence of webpage size and object number, we test the same typical LinkedIn profile webpage (that people may need to frequently check over mobile phones) across the following experiment sets.

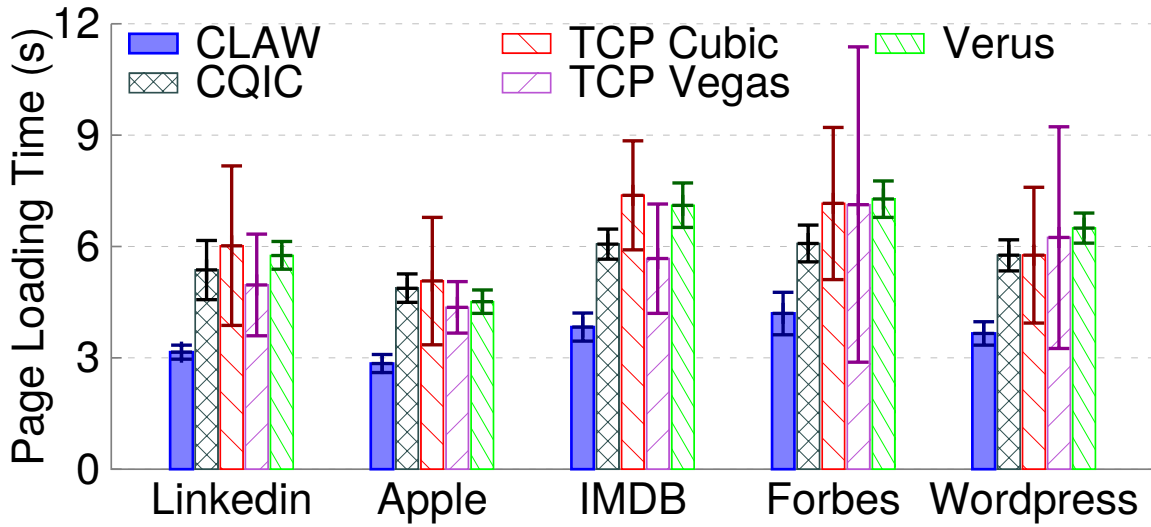


Figure 4.16: PLT under different transport protocols.

Under different LTE cell loads. To evaluate the impact of cell loads, we run the Web loading test at different time of the day – noon for a congested LTE cell, and late night for an idle cell. We used CLAW’s cellular load analyzer to inspect the cell load before the tests and found that the background cell load is around 70% during the noon and close to 0% during late night.

From the results in Fig. 4.17a, we make the following observations: (i) CLAW achieves the lowest PLT under different cell loads. For the idle channel, it reduces the latency by 5.06s (66%) against TCP Cubic and 1.63s (38%) against the performance runner-up, CQIC, while on the busy channel the reduction becomes 6.63s (59%) and 4.87s (51%), respectively. (ii) A CQIC client can only estimate the cellular link capacity supported by the fraction of resources allocated to itself [16], because it is unable to sense and harness the unallocated radio resources. This unfairly penalizes itself under a busy channel, leading to a sharp latency growth of 5.21s (55%) compared with the idle case. (iii) The delay-based TCP Vegas performs poorly over busy channel (mean PLT 8.04s), due to its well-known conservative rate control when coexisting with loss-based protocols [83]. Verus suffers from a similar problem, worsen by its inability to acquire a reliable bandwidth-delay profile.

Under different LTE channel qualities. We evaluate how the LTE channel

quality affects the PLT by running tests at places with high (12dB SINR) and low (5dB SINR) channel quality reported by the phone. The results in Fig. 4.17b show that the PLT increases under lower channel quality for all schemes since the throughput unavoidably degrades and translates into longer downloading time. Yet, CLAW always achieves the lowest latency: it reduces the latency by 2.2s (44%) against the widely-deployed TCP Cubic and 1.04s (27%) against the performance runner-up, CQIC. Similar to CLAW, CQIC can also adjust the downstream rate following the channel quality, but it is incapable of exploiting unallocated idle resources (§4.2) and hence its performance always falls behind CLAW. Besides, lower channel quality tends to increase packet loss, but CLAW's PLT only increases by 0.76s, smaller than all other schemes, due to its capability of explicit loss differentiation (§ 4.4.2).

Under different outdoor mobility scenarios. We further run the tests in a moving vehicle under three outdoor mobility scenarios: (i) *slow driving* with around 10 mph within a university campus, (ii) *city driving* with around 20 mph in an urban area and (iii) *fast driving* with around 50 mph in a suburban highway. We carefully choose the spots on the route to perform Web loading so that the cross-cell handoff does not happen during a web loading.

From the results (Fig. 4.17c), we can observe that CLAW again achieves the lowest PLT across all the mobility scenarios. This proves that CLAW's cellular congestion monitor and rate controller perform consistently well even under very high channel dynamics. It may be counter-intuitive that the PLT decreases with the mobility level. We found the reason lies in the channel quality: the slow driving mode crosses routes between tall buildings which severely blocks the LTE signal; city driving partially suffers from similar effect; the highway driving occurs in relatively empty space with plenty of line-of-sights between the client and BS, resulting in extremely high link quality – our congestion monitor shows the corresponding TBS size is almost $3\times$ higher than the slow driving case.

Over different cellular operators. As a cross-validation, we run CLAW on the same phone supporting two operators possibly with different resource allocation policies. Fig. 4.17d shows that, despite the different average PLT due to different channel conditions, CLAW shows similar fraction of PLT reduction across the

two operators, compared with alternative schemes. In particular, compared to TCP Cubic, CLAW reduces the page loading time by 1.6s (36%) for Operator A and 1.0s (35%) for Operator B. We also observe that Operator B provides better performance for all algorithms under the same channel quality, which owes to Operator B's BS scheduling algorithm that schedules the LTE client with shorter interval to propel the TCP self-clock. Regardless of the BSs used by different operators, CLAW achieves consistent performance gain over TCP Cubic since the BS's scheduling strategy equally impacts them. Due to lack of phone hardware, we are unable to test a wider range of operators. But we believe the experiment hints to the fact that the CLAW's design is not limited to a certain cellular operator.

Over different LTE hardware. To demonstrate that CLAW can always accelerate Web loading regardless of the peak LTE data rate and processing power supported by the hardware, we cross-validate CLAW over two mainstream phone models: Nexus 5 with category[84] 4 LTE and Nexus 5X with category 9 LTE chipset, supporting 150 Mbps [85] and 450 Mbps peak downlink rate [86], respectively. The newer Nexus 5X is also equipped with more powerful hardware, *e.g.*, CPU, that accelerates the network stack and OS services. From Fig. 4.17e, we observe that the latter can significant accelerate page loading, due to higher PHY bitrate and more powerful hardware. CLAW again shows consistent performance gain over both phone models. Interestingly, CLAW's latency reduction over TCP Cubic is around 35% for both Nexus 5 and Nexus 5X. In fact, although the low-end phones like Nexus 5 may lack proprocessing power hence suffer from higher computational time, they typically have LTE hardware with lower category number or data rate. Therefore, reducing the network latency is still crucial for mobile Web acceleration.

With HTTP/1.1 and HTTP/2. Although CLAW is designed with HTTP/2 in mind, we have conducted experiments to verify its backward compatibility with HTTP/1.1. The same server is used to test different HTTP versions to isolate the influence of server hardware and the network path to the server. We switch the server between different HTTP versions by enabling/disabling the `http2` module of the Apache server. The results in Fig. 4.17f show worse performance for all tested algorithms when degrading the server from HTTP/2 to HTTP/1.1. In addition,

compared to TCP Cubic, CLAW reduces the page loading time by 3.05s (46%) over HTTP/2 and a smaller 1.28s 18% over HTTP/1.1. This is because even though the network bandwidth explored by CLAW allows the Web contents to be downloaded all at once, without the multiplexing feature of HTTP/2, CLAW can only download the Web contents sequentially under HTTP/1.1, which raises the latency and reduces CLAW's performance gain.

TCP friendliness and multi-user cases

We conduct a microscopic examination of CLAW's TCP friendliness, by concurrently tapping the reload buttons of the same webpage on two phones of the same model, so as to create the worse-case traffic overlapping. We evaluate 3 configures with different combinations as shown in Fig. 4.19. By comparing Fig. 4.19a and Fig. 4.19b, we see that CLAW is as fair as TCP Cubic itself: when coexisting with CLAW, TCP Cubic achieves similar level of latency compared to when coexisting with another TCP Cubic flow. Note that in the CLAW-Cubic coexisting scenarios, CLAW will not sacrifice its own performance for Cubic, rendering similar performance to the CLAW-CLAW coexisting scenarios in Fig. 4.19c, which owes to CLAW's gentle aggression – it exploits the idle resources without touching the resource occupied by other users, until the idle resources are drained off (§4.4.2).

Due to lack of LTE phones, we were unable to test a large number of CLAW users concurrently. Instead, we develop a simulator to simulate a network with LTE bottleneck. The BS uses a simple round-robin scheduler, and the PHY parameters, RTT, loss rate, are configured to be consistent with our empirical measurements. We implement and run CLAW and Cubic separately. Each of our experiments lasts for 30s with up to 45 users, wherein each user starts a 750KB page loading at a random time. The result (Fig. 4.18) shows that, the CLAW users consistently outperform Cubic even under a heavily loaded network. As the network saturates, CLAW's benefit diminishes as it has less idle resource to explore and falls back to the hybrid bandwidth exploration most of the time. Consistent with our reasoning in § 4.4.2, CLAW shows a higher level of fairness (in Jain's index) than Cubic.

4.7 Discussion

Comparison with BS-based solutions. CLAW works with commodity phones and existing LTE infrastructure, which makes it readily deployable. In contrast, redesigning the BS to explicitly convey bandwidth information, as suggested in [87], requires fundamental modifications to the LTE infrastructure, phone hardware and MAC/PHY standard. Furthermore, such solutions require close cooperation between content providers and the cellular operators, whereas a software-based solution like CLAW is more feasible for the vast majority of Web services like online retailers.

CLAW's design is also transferable to future-generation cellular networks, as long as the clients can collect certain PHY statistics as CLAW does for 4G LTE networks. Furthermore, future cellular networks may deploy more TCP proxies and caching servers to optimize the Web latency. CLAW could be the perfect match for these servers as the cellular link is almost surely the bottleneck in such scenarios.

Computational latency vs. networking latency. Recent research [88, 89] reveals that the computational time also plays an important role in the Web loading latency. However, even under the extreme cases with a tiny RTT of 5ms, the content downloading time still accounts for more than 40% of the total latency [89], and starts to dominate under higher RTT of 150ms. We emphasize that the RTT quickly inflates during data transfer due to network queuing[53, 21, 56, 20], and can easily exceed 150ms with only 200KB data in flight[53], which further justifies CLAW's strategy of accelerating the bandwidth convergence with fewer round-trips.

Generations of smartphones have been embracing escalating CPU power to minimize the computational latency, which always outpaces the network upgrades. In addition, to catch up with the rapid growth of phone screen size/resolution, the Web content sizes will scale accordingly to offer better user experience, which further elevates the networking latency. We believe that CLAW's data transport protocol can better prepare the mobile networks for future demands. It can certainly be combined with page optimization [90] and better object fetching strategies [77] to further cut the PLT.

Remarkably, with shorter PLT, a CLAW user’s radio interface stays in the energy-hungry active state for shorter duration, which directly translates into energy saving. We also plan to explore this benefit in our future work.

4.8 Summary

We have presented the design, implementation and evaluation of CLAW, which accelerates mobile Web loading through a physical-layer informed transport protocol. CLAW extracts the LTE cell load, packet loss and bitrate information based on the PHY statistics available locally on a smartphone. By exposing such fine-grained information, CLAW unleashes the transport-layer from data-driven bandwidth probing, and enables judicious rate control under channel dynamics and losses. CLAW improves mobile Web performance remarkably, in terms of both average latency and stability. We believe its PHY-informed design can benefit a broader range of mobile protocols and applications.

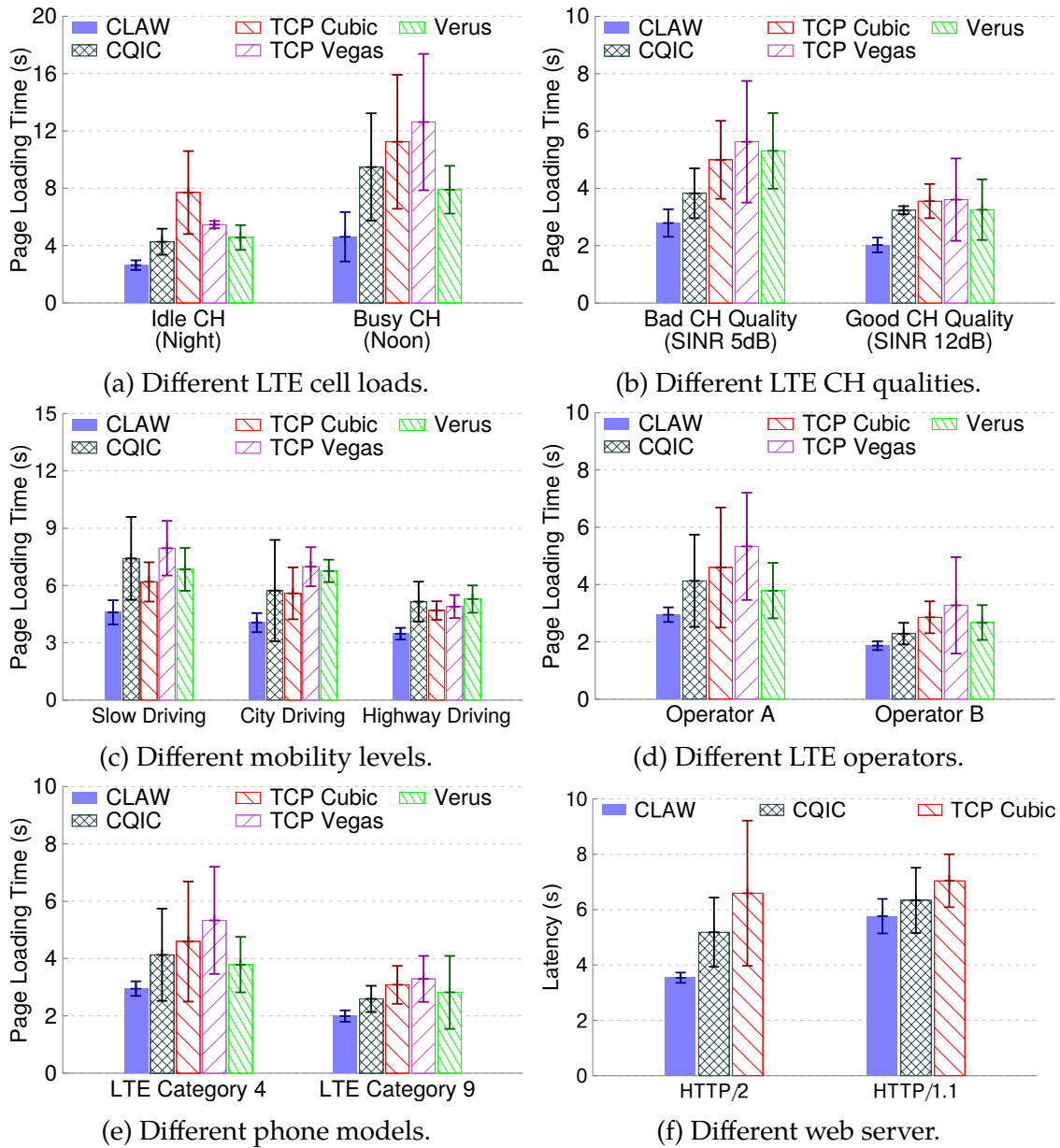


Figure 4.17: CLAW outperforms benchmark schemes under various scenarios.

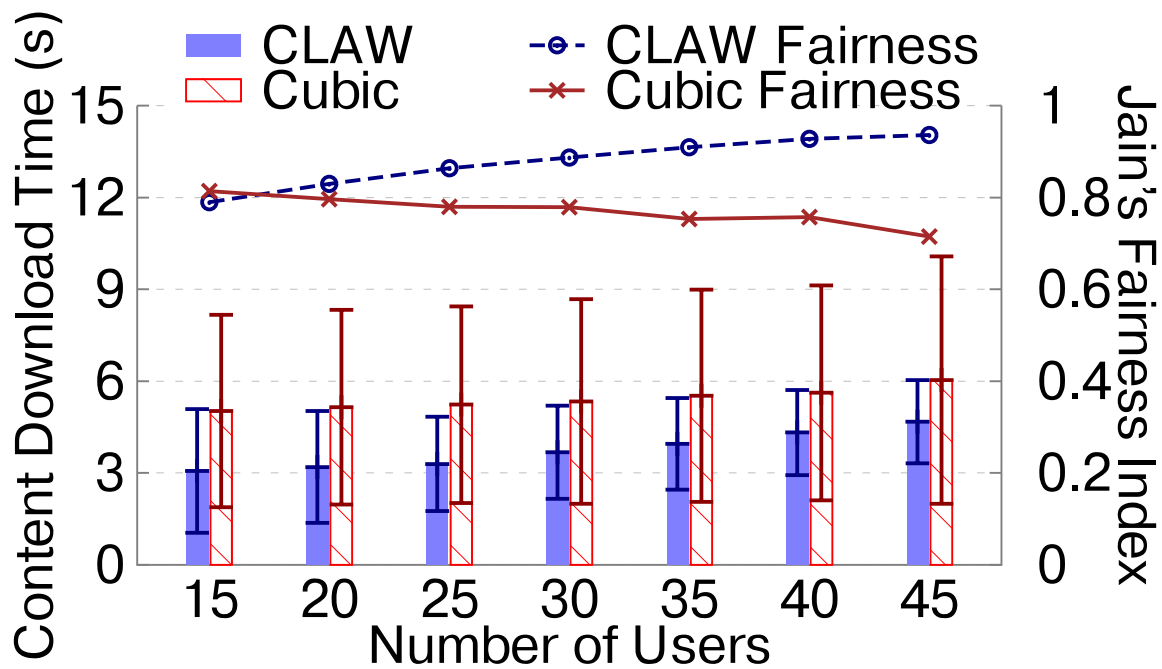


Figure 4.18: Scalability test.

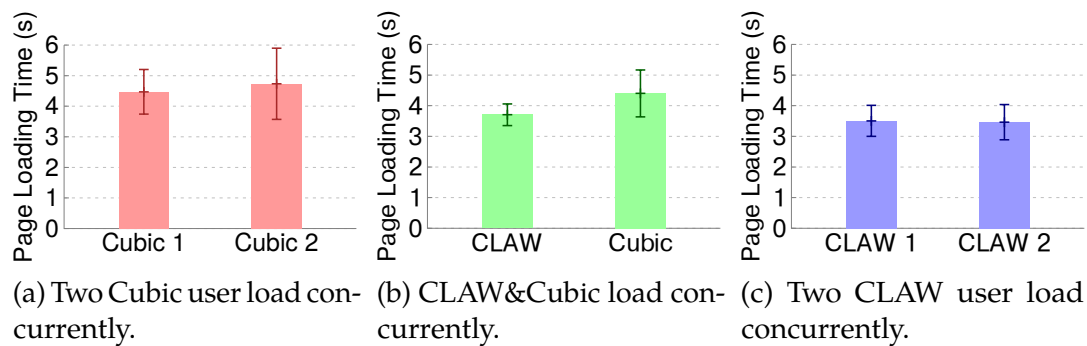


Figure 4.19: TCP coexistence and multi-user scenarios.

5 FREEZE-PRECAUTIONARY INTERACTIVE VIDEO CHAT OVER LTE

5.1 Introduction

The rapid proliferation of smartphones has triggered a tremendous growth of video telephony over cellular network infrastructures. The number of mobile video chat users will soon reach 2.1 billion, imposing a looming challenge to the limited radio resources in cellular networks. Unlike video streaming services that are mostly delivered over TCP/HTTP [59], interactive video communications use UDP/RTP as a substrate layer [91]. Despite decades of research at the transport layer, congestion control for delay-sensitive UDP flows remains under-explored [92]. Well-established congestion control principles, such as packet ordering and loss probing, mainly target network utility maximization, but often cause excessive delay for interactive video.

Recently, the IETF has been standardizing congestion control protocols specifically for real-time media [93]. These protocols aim for a “clean-slate” RTP/UDP based transport layer for interactive video. Existing proposals along this direction [94, 95] use either delay or loss as congestion indicator, and assimilate many congestion control principles that have endured the test of time. But the performance of such protocols is still unsatisfactory over LTE cellular networks [96]. The root cause lies in a conflict between the responsiveness requirement of video telephony and the elusive cellular bandwidth, which manifests in three aspects: (i) Delay/loss based end-to-end metrics need damage (*e.g.*, queue fill-up) to be done before reaction, or need multiple RTTs to get materialized, which fares poorly with the delay-sensitive video chat. (ii) LTE BS uses larger buffers (up to a few seconds for certain operators [20]), which absorb the link bandwidth variations due to channel dynamics, but cause excessive delay before queue overflow or packet loss occurs. (iii) LTE BS maintains separate queues for different users [43, 27], which helps QoS isolation but also renders delay-based protocols unable to react quickly to new competing

traffic.

In this chapter, we propose pyChat, a cross-layer video adaptation framework that resolves the above challenges and enables responsive video chat over LTE. At the core of pyChat is a clean-slate RTP-based congestion control framework, built on the IETF/Web3C WebRTC APIs [97], but augmented by LTE PHY-layer information. Instead of treating the end-to-end path as a black box, a pyChat user extrapolates an explicit congestion indicator by directly from its LTE channel, which is the network bottleneck in the common cases [26, 27]. The LTE radio resource comprises a limited number of frequency-time blocks called physical resource block (PRB). A sudden rise of PHY resource utilization may be caused by (i) increasing traffic demand and resource contention among users, (ii) decreasing channel quality of certain users which then consume more PHY resources to sustain current bitrate. Either can end up with temporary queue buildup and hence frozen video frames. Since the PHY resource utilization ratio essentially reflects the PHY-layer *cell load*, we use these two terms interchangeably. Our empirical measurements (§5.3) reveal that more than 90% of video freeze events are preceded by a surge in cell load. Such correlation exists for both the LTE downlink and uplink. More importantly, the gap between these two events spans from a few hundred milliseconds to a few seconds. Therefore, upon detecting a cell load surge, a pyChat sender has abundant time to proactively reduce its rate before freezing occurs.

The actual realization of this principle entails non-trivial challenges and design choices. Specifically, we found most freezing events occur when the cell load is well below 100%, and when the video bitrate is well below the long-term average bandwidth. We refer to such phenomenon as *micro-congestion*. Micro-congestion is mainly attributed to short-term queue buildup, which cannot be captured by conventional congestion control mechanisms that attempt to estimate the saturation bandwidth. To prevent video freeze when micro-congestion happens, we can throttle the video bitrate on every cell load surge. However, not every cell load surge will end up with a video freeze and the aforementioned strategy can be over-conservative. To answer the question of *whether to throttle the rate to prevent video freeze*, a pyChat user predicts the video freeze probability based on the LTE

cell load using a stochastic model it built online (§5.4.1). Given the estimated video freeze probability, to answer the question of *when and how much to throttle the rate to prevent video freeze*, pyChat employs freeze-precautionary rate control (FPRC) (§5.4.2), a joint RTP rate control and video adaptation protocol to optimize the video Quality of Experience (QoE). Since the video freeze probability can be predicted within a finite-horizon, FPRC looks ahead of time and uses a finite-horizon dynamic programming algorithm to choose a rate that balances the tradeoff between higher video bitrate (quality) and risk of video freeze.

We have implemented pyChat as a real-time congestion control and video adaptation framework, and integrated it with the WebRTC API [98], which serves as the back-end of many browser-to-browser video chat services [99]. To obtain the LTE cell load in real time, we implement an LTE channel status monitor based on software radio, which evaluates the LTE uplink/downlink resource utilization with frame-level granularity. This monitor serves as a PHY proxy that is tethered to an LTE smartphone, and passes the congestion metrics to its browser, which hosts the WebRTC video codec and pyChat adaptation. We expect in the near term, a firmware update can allow commodity smartphones to directly extrapolate the frame-level PHY congestion metrics from their channel state information (CSI), just as in current WiFi transceivers [47]. Besides, readily deployable (but less accurate) cell load monitoring based on RSRQ[60] from commodity smartphones is discussed in §5.7.

Our run-time experiments demonstrate that pyChat can effectively predict and prevent video freezing using the PHY-aware congestion metrics. Its miss rate is around 10% to 20%, depending on the burstiness of the BS's traffic. This level of prediction accuracy suffices to prevent majority of the long-latency/freezing events in existing reactive congestion control protocols. We benchmark pyChat's video chat performance against the standard WebRTC framework (used in Google Hangout, and running the Google Congestion Control protocol), and other state-of-the-art congestion control and video adaptation schemes, including Sprout, Skype, FaceTime and piStream [20, 100, 101, 59]. We conduct the experiments under various network environments, including busy LTE channel with dynamic

competing traffic, and mobile LTE channel with rapidly varying channel quality. pyChat demonstrates superior performance over benchmark schemes in all these scenarios. It only has 0.5% to 1.1% of video freezing ratio in idle/busy LTE cells, in comparison to 3% to 35% in alternative schemes. In particular, compared with WebRTC using Google Congestion Control (GCC) backend, pyChat can double the video quality while reducing freezing rate from 9% to close to 0%. Our experiments also demonstrate that pyChat, similar to other video telephony applications, does not affect the performance of competing TCP flows due to their prioritization on minimizing latency.

Although prior work has touched upon cross-layer bandwidth estimation and video adaptation [16, 59], pyChat represents the first to address the congestion control problems unique to latency-sensitive video telephony in LTE cellular networks. The specific contributions of pyChat can be summarized as follows:

(i) We propose a physical-layer aware congestion metric that can predict delay spikes in real-time video traffic, which differs from previous protocols that either estimate saturation bandwidth [16, 59] or react passively to long latency [94].

(ii) We design practical mechanisms to reliably obtain the congestion metric, taking into account cellular channel dynamics. We further propose a simple online adaptation mechanism, which models the relation between the PHY-aware congestion metric and video performance, and uses a look-ahead algorithm to optimize the video telephony QoE.

(iii) We implement pyChat and integrate it with the WebRTC as a general congestion control stack for browser-based video telephony. Our experiments verifies that pyChat substantially reduces the video delay compared with state-of-the-art protocols and proprietary video chat applications.

pyChat runs at the LTE users, can directly support WebRTC-compatible applications and does not require any modification to the LTE BS. It respects the end-to-end congestion control principle [102]. It needs no middlebox deployment at cellular carriers, or additional signaling channel between the BS and video chat clients.

5.2 A Primer on The WebRTC Standard

Open-access, browser-based interactive media applications are gaining traction recently owing to promotion from main Internet standardization bodies – the IETF and W3C – which ratified the WebRTC. WebRTC comprises a set of substrate protocols layered on RTP/UDP, to enable a Web application to communicate real-time media between the browsers on different devices. WebRTC opens up a wide range of real-time interactive applications, and has been adopted by the browser version of mainstream video telephony services, including Google Hangout, Skype, Slack, and Facebook Messenger [99, 103]. Historically, a primary impedance to video-specific transport protocol is legacy compatibility. But WebRTC encapsulates RTP transport layer and video adaptation into a single browser stack, making it an ideal vehicle for cross-layer video telephony protocols.

Currently, WebRTC uses the Google Congestion Control (GCC) as default transport protocol. GCC runs rate-based congestion control at both the sender and receiver. At receiver side, GCC adopts the basic “probe-overuse-reprobe” work flow to explore the bandwidth, similarly to conventional TCP protocols. When the network bandwidth is under-utilized, GCC client gradually raises its bandwidth estimation and feeds back this estimation so that the sender increases its rate accordingly. By probing up in this way, the receiver’s bandwidth estimation will eventually reach its available bandwidth. The overuse detector is triggered then, which is defined as the difference between inter-arrival time (ΔT_a) and inter-departure time (ΔT_d) of a pair of consecutive packets. If $\Delta T_a > \Delta T_d$, the later packet must have suffered from extra queuing delay, thus an increasing value of $\Delta T_a - \Delta T_d$ implies queue building up and bandwidth overuse. If this overuse indicator exceeds a predefined threshold γ , the GCC receiver reduces its bandwidth estimation multiplicatively to avoid congestion.

At sender side, GCC updates its sending bitrate following the receiver-side bandwidth estimation feedback. Meanwhile, it also employs the packet loss as a congestion indicator: under high packet loss rate, the sender reduces its sending bitrate which is, however, lower-bounded a theoretical estimation based on TCP

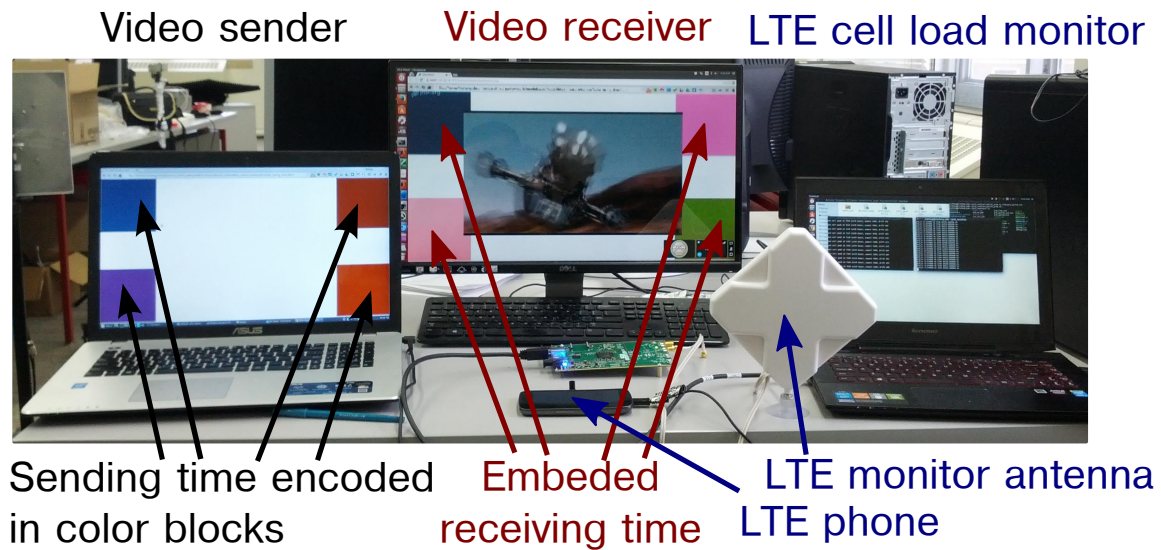


Figure 5.1: Customized testbed for evaluating real-time congestion control protocols for interactive video.

Friendly Rate Control (TFRC) [104] to ensure fairness when coexisting with TCP flows. Furthermore, the sender passes its sending bitrate to the video encoder as the *target video bitrate*. WebRTC supports VP8, VP9 or H.264 codecs, which can adapt the video quality based on the target bitrate. In this way, WebRTC synchronizes the GCC-based transport-layer rate adaptation and video rate adaptation, and uses the former to bound the latter.

5.3 Motivation

In this section, we conduct experiments to reveal the fundamental limitations of conventional congestion indicators when applied to interactive video in LTE networks. We use the IETF/W3C WebRTC framework as a baseline, which represents the state-of-the-art in congestion control for real-time interactive media.

5.3.1 Measurement Setup

Figure 5.1 illustrates the measurement testbed that we build to profile and diagnose congestion control protocols for real-time interactive video. At a high level, our testbed can be considered as an open-source version of Google Hangout (browser-integrated) [105] – Except the video display and session management which uses JitSi Meet [106], it employs the same WebRTC framework integrated with Chromium, with GCC based transport layer. We set up the video chat between two users: one is a PC connected to our campus wireline network, while the other is a Verizon 4G LTE smartphone (Galaxy Nexus). Video chat performance is measured at both sides, reflecting the congestion adaptation when LTE uplink and downlink acts as the bottleneck respectively.

To profile video delay, we developed a customized delay measurement tool, which measures end-to-end delay for every video frame displayed on the screen (§5.5 describes the design detail). This delay metric reflects the user-perceived video performance. To measure the video bitrate, we directly extract the incoming/outgoing video bitrate from the Chromium browser by modifying its source code and sending the values via shared memory after turning off the sandbox that blocks the usage of shared memory. We develop a simple C program to read this shared memory in real time. To ensure repeatability and controllability, we use virtual webcam (ManyCam [107] for testing FaceTime under iOS and WebcamStudio [108] for all other tests under Linux) to feed a movie scene into the video chat application. The video scene has 1080P resolution, and WebRTC’s video encoder adapts the video bitrate to create the variable bitrate (VBR) video stream based on network condition. The average video output rate is around 3 Mbps.

5.3.2 Limitations of Delay/Loss-based Congestion Control

Delay-based congestion indicator

We first evaluate the responsiveness of delay-based congestion indicator in WebRTC/GCC. Before the experiments, we first run a short iperf session to obtain a

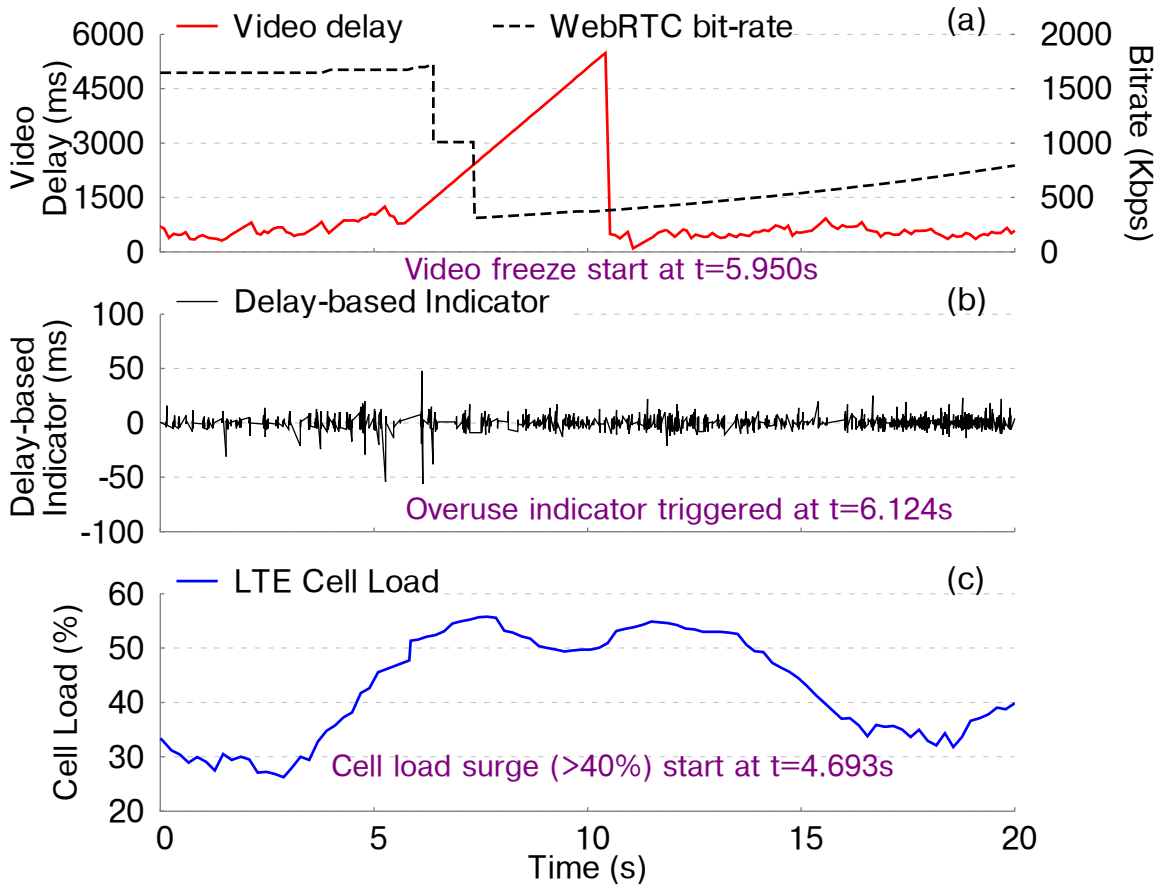


Figure 5.2: Snapshot of video chat statistics over an LTE downlink (Long-term measurements are in §5.4.1).

rough estimation of the available bandwidth, ensuring it suffices to support the 3 Mbps source video bitrate.

As mentioned in §5.2, GCC adopts a *delay-based overuse indicator*, which triggers multiplicative rate decrease when its value exceeds a predefined threshold. In order to investigate the responsiveness of this overuse indicator, we record the real-time video chat statistics including the per-video-frame delay, the value of delay-based overuse indicator and the PHY-layer LTE cell load. Aligning these metrics in time domain allows us to diagnose the underlying mechanism of rate control over LTE networks¹.

¹It is worth noting that the motivational results in this section are only snapshots to showcase the

Figure 5.2 plots the time series of statistics at the LTE client side measured during a WebRTC video session when LTE downlink bottlenecks the network path. We first observe that the video freezes at $t = 5.950\text{s}$ from Figure 5.2(a), *i.e.*, the video stalls at a particular frame whose delay keeps accumulating over time. Note that GCC sharply throttles the video bitrate after the freezing occurs. However, the video frame delay still escalates to nearly 6000ms until next frame arrives. What triggers GCC's rate throttling? From Figure 5.2(b), we see that GCC's delay-based congestion indicator peaks at $t = 6.124\text{s}$, which is slightly later than the video freeze. This leaves no room for the sender to slow down in a precautionary way, other than react passively.

Figure 5.2(c) shows the time series of LTE cell load in the same experiment, measured using a real-time LTE cell load monitor that we developed (§5.5). The LTE downlink cell load varies between 26% and 55% most of the time. Interestingly, it never exceeds 60% even when the video freezes. This implies that a cell load peak can cause bursty queue buildup at the BS and result in a video freeze, even when the BS has enough PHY resource to support higher bandwidth. We refer to such phenomenon as *micro-congestion*, to distinguish it from conventional non-real-time congestion control systems that focus on bandwidth overuse detection. Furthermore, by contrasting Figure 5.2(a) and Figure 5.2(c), we see that the onset of the cell load surge (4.693s) comes well before the video freeze (5.950s), implying that the former can be a *precautionary indicator of micro-congestion*.

We then repeat the experiments for the case when the LTE phone acts as video sender. Although our LTE cell load monitor can only obtain the uplink resource utilization of our user itself, it still approximates the cell-wide congestion status (§5.5). Figure 5.3 plots the video delay and uplink cell load trace containing a few video freezing events. Consistent with the downlink behavior, every video freeze is preceded by a cell load peak, corresponding to a micro-congestion.

In summary, *conventional delay based congestion indicator materializes as an aftereffect of video freezing, and hence it cannot react promptly when video chat meets the highly*

problems in the rate control operation. More conclusive results based on long-term measurements can be found in §5.4.1.

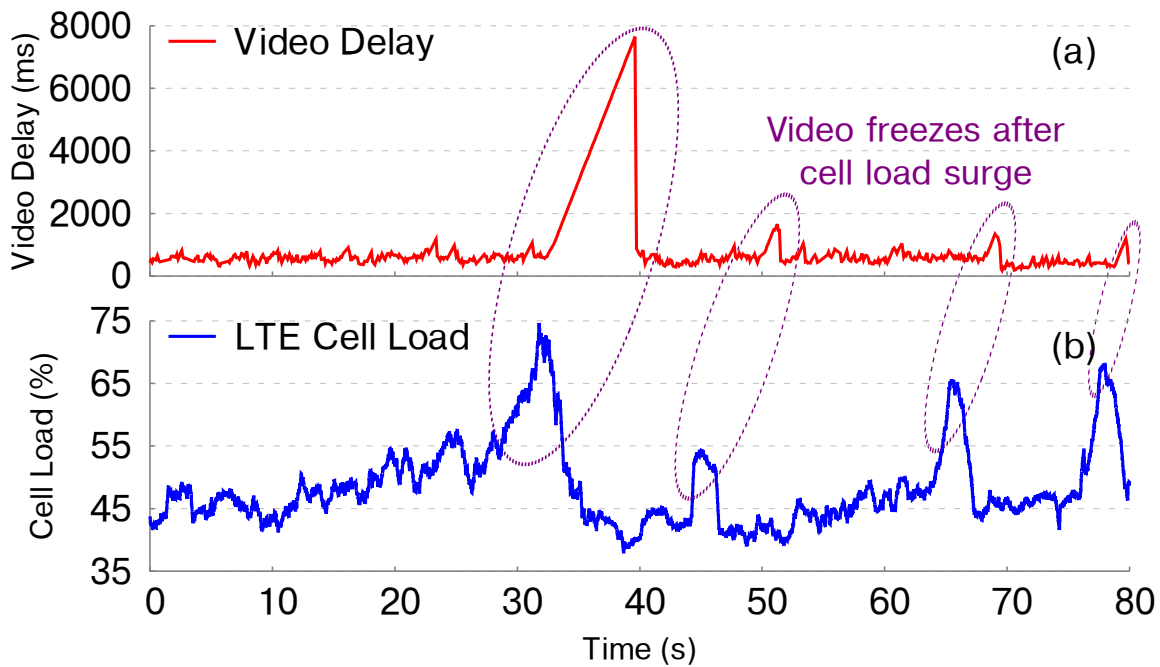


Figure 5.3: When sender streams live video over LTE uplink, every video freeze comes *after* a cell load surge, which is also observed in the downlink scenario (Figure 5.2).

dynamic LTE channel. The causing relation between micro-congestion and video freezing motivates us to use cell load pattern as a precautionary congestion indicator, which is the key principle in pyChat.

Loss-based Congestion Indicator.

Loss-based congestion indicator is known to perform poorly in cellular networks, especially for delay-sensitive applications, due to the notorious bufferbloat problem [80]. LTE BS is over-buffered [56, 27], with up to a few seconds of buffer size [20] to absorb the wireless channel variation. As a side effect, the large buffer prevents packet loss when the sending bitrate overshoots the cellular link bandwidth, thus loss-based congestion control algorithms are neutralized as they fail to detect the bandwidth overuse. In our experiments, by adding debug logs in the Chromium source code, we found that GCC's loss-based congestion control is never triggered under various LTE network conditions, This is reasonable as GCC's loss-based

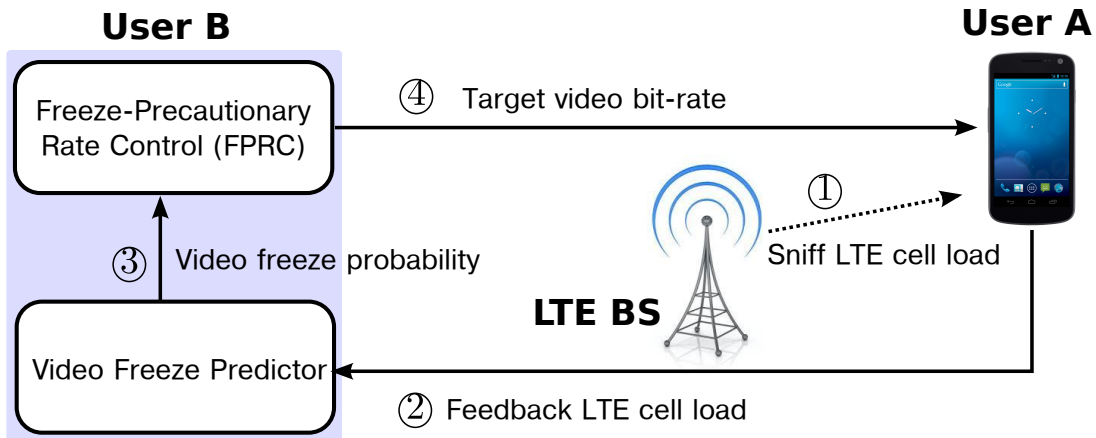


Figure 5.4: pyChat workflow.

congestion control is particularly design for the networks with small buffers [94] rather than over-buffered LTE networks.

5.4 pyChat Design

pyChat is a cross-layer framework that harnesses LTE's PHY-layer cell load information to prevent video freezes while maximizing the quality of interactive video chat. Compatible with the WebRTC standard, pyChat executes both the congestion control and video adaptation at the end hosts. So it can be deployed without modifying operators' infrastructures, nor does it require the BS to discriminate video traffic. We elaborate on such advantages further in §5.7.

In a peer-to-peer video chat between two pyChat users, each user acts as both video sender and receiver. Figure 5.4 outlines the workflow: ① A pyChat user (LTE client) continuously monitors the LTE cell load or the PRB utilization in its current cell ². ② It then feeds back the cell load information to the other peer (video sender) via WebTC's RTCP channel for every 200ms. ③ The video sender's *freeze predictor* predicts the video freezing probability at its receiver using a stochastic

²Recent work showed that LTE cell load can be obtained at the LTE client by passively sniffing the channel instead of modifying the BS [59, 29, 60]. We enhance the approach in [59] to make it work in real time (detailed in §5.5 and 5.7).

model, which is continuously refined based on the cell load information, along with latest video delay statistics and the video traffic it received/sent. ④ The freezing probability is then employed in the sender's *freeze-precautionary rate control* (FPRC) algorithm, which adapts video target bitrate. pyChat's video adaptation mechanism considers not only video quality and freezing risk, but also video stability. It employs WebRTC's built-in video encoder to generate the output video whose bitrate matches the target bitrate, but in a smooth-out manner to prevent abrupt resolution changes that hurt user QoE [33].

In the rest of this section, we describe the rationale and detailed design of pyChat's modules.

5.4.1 PHY-Assisted Video Freeze Predictor

pyChat's freeze predictor builds on our observation of the micro-congestion effect (§5.3.2). Before delving into the predictor design, we conduct an anatomy of the micro-congestion, revealing its generality and the causes behind.

Diagnosis of Micro-Congestion

The measurements in §5.3 reveal that every video freeze comes after a LTE cell load surge, referred to as *micro-congestion*. To leverage this finding for responsive video adaptation, we first diagnose micro-congestion and video freezes by answering several important questions.

Why does cell load surge before video freeze? For interactive video, increased traffic and reduced channel quality can be two immediate causes of video freeze. With increased traffic from either competing users or the video receiver itself, the total traffic going through the LTE BS can exceed its capacity. The same happens when the reduced channel quality leads to a capacity drop. In either case, the video freezes as the LTE-provisioned bitrate falls below the video bitrate. On the other hand, LTE cell load, or PRB utilization, is a natural indicator of the instantaneous network condition — A surge in LTE cell load directly implies the two immediate causes. With increased traffic, the BS needs to allocate more PHY resources to

sustain more traffic. Similarly, with reduced channel quality, each PHY resource block carries fewer bits, and thus for the same traffic bitrate, the user consumes more PRBs.

To validate the above reasoning, we conduct real-world experiments over commercial LTE networks. First, to demonstrate that increased traffic causes higher PHY resource utilization, we generate downlink TCP traffic to a LTE smartphone using `iperf`, and compare the PHY resource utilization of different traffic bitrates. For each traffic bitrate, we stream traffic for 10 minutes to collect enough LTE PHY frames (each is 10ms long), and then compute the CDF of the per-frame PRB utilization, distributed across all collected LTE frames. To ensure no other users in the LTE cell disturb our results, our experiments are conducted in late night. We use the phone's LTE RSRQ information to identify an idle channel (RSRQ around -6dB denotes an idle channel [60]). The receiving phone is static in all experiments. We repeat our tests for 3Mbps, 5Mbps and 10Mbps TCP traffic.

As shown in Figure 5.5, under higher traffic bitrate, a higher PRB utilization manifests: First, we see a higher percentage of heavily used LTE frames, *e.g.*, the peaks at around 60% frame utilization. Second, we see less unused LTE frames, *e.g.*, there is no unused (0% utilization) LTE frame when we use 10Mbps traffic to saturate the LTE downlink. We further examine the relation between LTE channel quality and cell load. To create adverse channel condition, we intentionally wrap the phone with metal foil, which causes about 6dB drop in channel quality (reported from the phone's LTE RSRP metric). Figure 5.6 shows that, to sustain the same 5Mbps traffic bitrate, a user consumes more PHY resources when its channel quality degrades.

In sum, a surge in the LTE cell load immediately reflects either increased traffic or reduced wireless channel quality, both of which are the root causes of video freezes. That is why cell load surge happens before every video freeze and can be used to predict video freezes.

Is the time interval between cell load surge and video freeze long enough for video adaptation? To use LTE cell load for video adaptation, the video receiver needs to feed back its cell load value to the sender. The feedback delay can be large

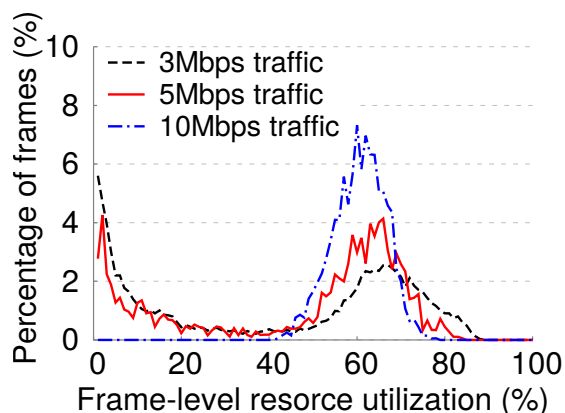


Figure 5.5: PHY resource utilization increases with traffic demand.

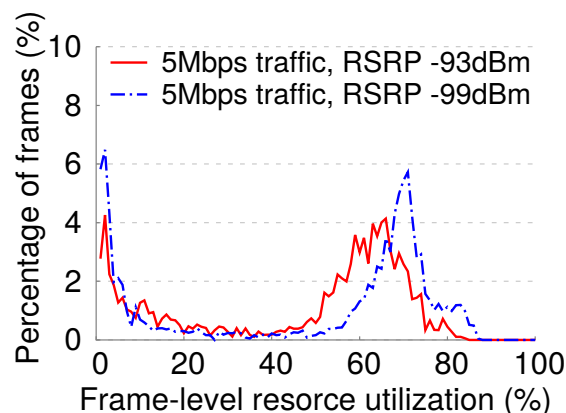


Figure 5.6: PHY resource utilization increases under lower channel quality.

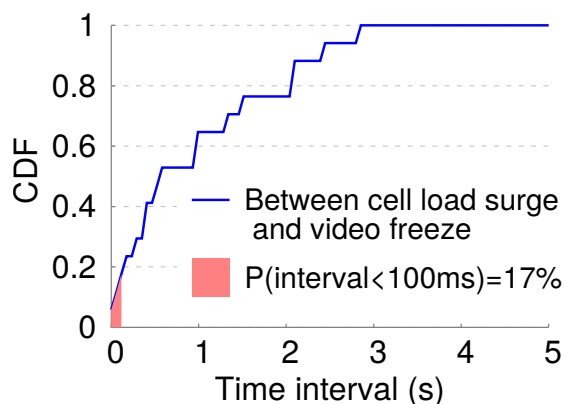


Figure 5.7: Distribution of the time interval between cell load surge and its following video freeze.

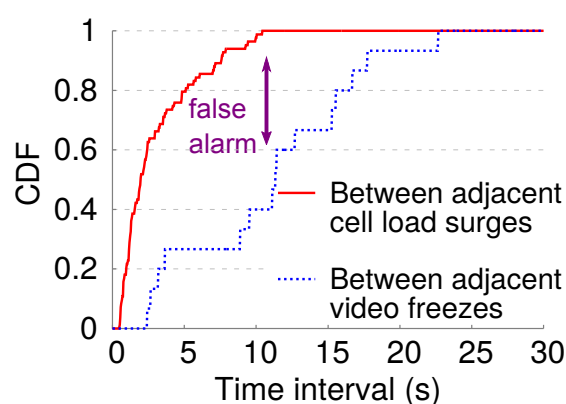


Figure 5.8: Distribution of the time interval between consecutive cell load surges, and that between video freezes.

in LTE networks, up to 100ms in certain cases [27]. Will the video freeze before the sender gets the feedback? Our long-term WebRTC measurements showed that it is the rare case. Following similar setup as in §5.3.1, the experiments in Figure 5.7 demonstrate that in 83% of the cases, the video freezes 100ms later than cell load surge. In other words, even after considering feedback delay in the control loop, cell load still offers an early prediction of video freeze. In contrast, conventional congestion indicators like packet loss or delay materialize only after the video performance drops, and hence they fall short of responsiveness.

Will cell load increase always cause a video freeze? Since video freeze comes after a cell load surge, to prevent video freeze, can we simply reduce the video bitrate on detecting every cell load surge? Unfortunately, not every cell load surge can result in video freeze. In other words, the cell load surge is a necessary but insufficient condition of video freeze. For example, when a new user joins the LTE cell but the BS still has surplus PHY resources, the video will not freeze even though the cell load may increase temporarily. To investigate how often such scenarios happen in practice, we compare the frequency of occurrence of cell load peaks ($> 40\%$ PHY resource utilization) and that of video freezes (frame delay $> 2s$). Figure 5.8 shows that the time interval between any two consecutive cell load peaks is below 10 seconds, but around 60% of the video freezes happen more than 10 seconds later than the preceding freeze event. In other words, many cell load surges do not end up with a video freeze. As a result, simply reducing the video bitrate on every cell load surge can be over-conservative as it incurs many false alarms.

Modeling the Relation between Video Freezes and Cell Load

Given that mapping every surge in cell load to a video freeze is highly suboptimal, what kind of cell load surge will cause video freeze? To answer this question, we employ an online model to estimate the video freeze probability upon a cell load surge.

To build the model, one key observation is that the LTE network bandwidth evolution depends on current wireless channel quality, competing LTE users' traffic,

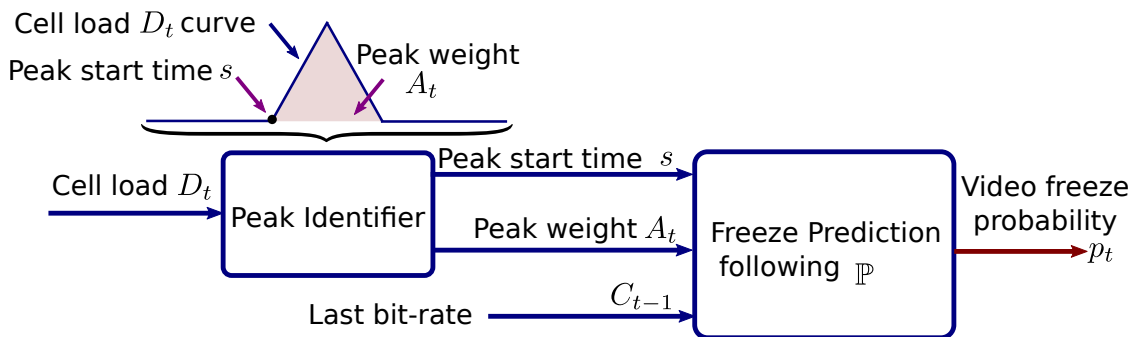


Figure 5.9: pyChat's PHY assisted video freeze predictor.

and the LTE BS's scheduling policy. Our previous measurements has shown that cell load already reflects both the traffic intensity going through the BS (Figure 5.5) and the wireless channel quality variation (Figure 5.6). Therefore, for a particular BS, the cell load peaks that can cause video freeze should have persistent patterns. We can capture such patterns by allowing each pyChat user to build a probabilistic prediction model, and refine the model as it roams across BSs. The model characterizes the joint distribution over cell load surges, the user's own video traffic demand and video freezes. With this model, pyChat can estimate the video freeze probability and adapt accordingly by monitoring the cell load and its own traffic demand in real-time.

(i) Modeling the joint distribution. Figure 5.9 illustrates pyChat's PHY-aware freeze prediction mechanism. To build the prediction model, the user slices time into epochs, each lasting 200ms (typical LTE RTT [43] and adaptation granularity of current WebRTC implementation in Chromium). We define a *video freeze event* at time t as $Z_t \in \{0, 1\}$, where 1 means video freeze and 0 otherwise.

To characterize the cell load surge, we define the *peak weight* A_t , which is the area under the time-domain cell load curve from the peak start time s to current time t . It essentially approximates how much accumulated traffic has gone through the BS within this period, and reflects how busy the BS is. While the peak start time s is the onset time when the cell load peak closest to current time t started to grow, more specifically, when the cell load in an epoch is more than doubled compared with the previous average, and the cell load value itself is greater than the value that

caused the latest freeze (at least above 40% according to our measurement). Besides, the user's video traffic demand at t can be represented by the bitrate request C_{t-1} that it made at the previous epoch $t - 1$. Note that in WebRTC, such requests are sent from the video receiver to the sender, and are accessible from both transport and application layers.

With the above set of variable, we can define the joint distribution as $\mathbb{P}(Z_t, A_t, C_{t-1})$. Given this distribution, by measuring $A_t = A^*$, and $C_{t-1} = C_{t-1}^*$ in real time, the user can estimate the probability of delayed frame deliver at current time t as a conditional probability: $p_t = P(Z_t = 1 \mid A_t = A^*, C_{t-1} = C_{t-1}^*)$.

(ii) How to build the joint distribution online? To estimate the video freeze probability online, pyChat needs to build the joint distribution $\mathbb{P}$ and refine it over time. A straightforward solution is to use legacy rate control like GCC while building $\mathbb{P}$, and then switch to pyChat algorithm after $\mathbb{P}$ materializes. But such a multi-variable distribution takes a long time to converge. In pyChat, we leverage the positive correlation between video delay and cell load to accelerate the online refinement the joint distribution $\mathbb{P}$ while running pyChat. Specifically, although unaware of the precise relation between video freeze and cell load peak, suppose $A_{t+1}^* \geq A_t^*$ and/or $C_t \geq C_t^*$ (i.e., when comparing epoch $t + 1$ with t , the BS has higher traffic intensity and/or the video demand is higher), we are sure that the video freeze probability $p_{t+1} \geq p_t$. Also note that the freeze prediction model can calibrate itself. When it has low precision, the inaccurate video adaptation causes more video freezes, and such video freezes in turn serve more samples to refine $\mathbb{P}$. As a result, we can start with a rough model with low accuracy, which uses predefined coarse mapping from cell load to video freeze following their positive correlation. And then we gradually refine it at runtime to achieve increasing accuracy. In our pyChat implementation, we simply bootstrap the model using previous p_t values obtained from offline experiments.

(iii) Online estimation of the video freeze probability. Given the joint distribution $\mathbb{P}$, the user predicts the probability of a video freeze whenever it detects a cell load peak. Specifically, it measures A_t and C_{t-1} , then computes the video freeze

probability as:

$$\begin{aligned}
 p_t &= P(Z_t = 1 \mid A_t = A_t^*, C_{t-1} = C_{t-1}^*) \\
 &= \frac{\mathbb{P}(Z_t = 1, A_t = A_t^*, C_{t-1} = C_{t-1}^*)}{\mathbb{P}(A_t = A_t^*, C_{t-1} = C_{t-1}^*)}
 \end{aligned} \tag{5.1}$$

5.4.2 Freeze-Precautionary Rate Control (FPRC)

To fully exploit the video freeze probability estimated by the video freeze predictor, we propose Freeze-Precautionary Rate Control (FPRC), a joint RTP congestion control and video adaptation protocol customized for highly dynamic LTE networks. The core idea of FPRC's optimization model is to balance the video quality and video freeze. Generally, when a cell load surge is detected, reducing the RTP and video bitrate immediately can always reduce the risk of video freeze. But this comes at the loss of video quality. As shown in §5.4.1, not all cell load increases end up with video freeze. For example, when the video freeze probability is relatively low on a cell load surge, we may still hold current video bitrate for a longer time to get higher video quality.

In the rest of this section, we detail the FPRC design. As outlined in Algorithm 3, FPRC has two modes: When LTE cell load remains low, FPRC uses the *bandwidth exploration mode* (§5.4.2) to explore the “available” bandwidth and improve the video bitrate (quality). On detecting a cell load peak, FPRC switches to the *freeze prevention mode* (§5.4.2), which employs dynamic programming to determine when to reduce the video bitrate and reduce it by how much.

When and how much to decrease the rate?

Finite-horizon video freeze prediction. After detecting a cell load peak starting at time s , the video freeze predictor can predict the video freeze probability p_t for a future time epoch $t > s$ till a finite horizon $T + s$. Given real-time cell load measurements, A_t in Eq. (5.1) has deterministic value at time epoch t . Then the video freeze probability p_t becomes a function of C_{t-1} , the rate selection made at time epoch $t - 1$. In other words, $p_t = p_t(C_{t-1})$. We can make such prediction

Algorithm 3 Freeze-Precautionary Rate Control (FPRC).

Input: t : Current time epoch
 s : Starting time of the cell load peak
 $p_t(\cdot)$: Video freeze probability, function of rate level
 D_t : Current cell load
 D^* : Cell load threshold
 C_{t-1} : Previous bitrate selection
 C_{t-1}^+ : Next bitrate level above previous bitrate

Output: C_t : Traffic bitrate

```

1: if Cell load peak detected at time  $s$  then
2:   /* Freeze prevention mode */
3:   Obtain  $p_t(\cdot)$  from the freeze predictor at time  $t$ 
4:    $i = s + T$ 
5:   /* backward induction from  $s + T$  to  $t$  */
6:   while  $i \geq t$  do
7:     Compute the optimal bitrate  $C_i$  following (5.2)
8:      $i = i - 1$ 
9:   Set current bitrate to the optimal  $C_t$ 
10: else /* Bandwidth exploration mode */
11:   if  $D_t < D^*$  then
12:      $\alpha = 1 + D^* - D_t$ 
13:      $C_t = \min(C_{t-1}^+, \alpha \cdot C_{t-1})$ 
14:   else
15:     Switch to freeze prevention mode

```

for any time epoch t within a finite horizon $T + s$. Beyond $T + s$, the video freeze probability is simply 0 since the network totally recovers from the micro-congestion if it does not end up with a video freeze. Our FPRC design uses $T = 3000\text{ms}$ which is chosen empirically based on the measurements in Figure 5.7.

Finite-horizon dynamic programming solution. The above observation inspires us to formulate an optimization problem based on finite-horizon dynamic programming. Although the WebRTC video encoder takes continuous video bitrate value as input, to reduce the computational complexity, we quantize the continuous video bitrate range into K bitrate levels. In pyChat implementation, we empirically set $K = 100$ to divide the $0 \sim 3\text{Mbps}$ bitrate range with 30Kbps resolution as we have up to 3Mbps bitrate for the highest video quality. Let C_s denote the video

bitrate used when the cell load peak starts, the bitrate values for selection should not exceed C_s as it is risky to increase the bitrate when the video is likely to freeze. Therefore after detecting the cell load peak, the pyChat user can only choose from the quantized rate levels in the range $[0, C_s]$. The target of our online rate control algorithm is to determine the optimal bitrate level C_t at current time epoch t that maximizes a utility function U_t , which is defined as the sum utility expectation accumulated from t till the time horizon $T + s$ at which the video freeze probability drops to 0.

Suppose we are at time epoch $t > s$ and the rate control algorithm selects rate C_{t-1} for the previous time epoch. If the video does not freeze at t , we define the accumulated utility U_t as the sum of current rate selection C_t and the future utility U_{t+1} that accumulated from time epoch $t + 1$ till the time horizon $T + s$. If the video freezes, we define the corresponding utility loss as L . On this video freeze, FPRC sharply throttles the traffic bitrate and the algorithm starts over in the bandwidth exploration mode and waits for the next cell load peak, so there is no U_{t+1} if the video freezes at t . Mathematically, we have:

$$U_t = (1 - p_t(C_{t-1})) \cdot (U_{t+1} + C_t) - p_t(C_{t-1}) \cdot L \quad (5.2)$$

With $p_t(\cdot)$ obtained from the video freeze predictor and predefined freeze penalty L , the dynamic programming problem defined by Equation (5.2) can be solved by backward induction. More specifically, at the time horizon $T + s$, we have $p_t = 0$, thus the optimal choice at $t = T + s$ is simply not to reduce the bitrate on cell load surge, *i.e.*, to hold the bitrate C_s . Given this boundary condition, we can solve the dynamic programming problem by backward induction following line 5 ~ 9 in Algorithm 3.

Overhead analysis The complexity of the dynamic programming algorithm for FPRC is $O(K \cdot T)$. Based on our measurements (Figure 5.7), an empirical value of $T = 3000\text{ms}$ is used and each time epoch or rate adaptation interval is 200ms in current WebRTC-based implementation. Hence the FPRC algorithm incurs low computational overhead and can safely run in real time.

When and how much to increase the rate?

When LTE cell load remains low, the video freeze probability is nearly 0. In this safe region, the rate control algorithm can explore the “available” bandwidth to improve the video bitrate (quality). FPRC harnesses the PHY-layer cell load information to explore the network bandwidth in a responsive way. Note that the cell load also implies the amount of remaining PHY resources in this cell. The user can increase the sending rate quickly when there are sufficient resources remaining, while it should keep or reduce its current sending rate when the resources almost drain out. In FPRC design, we first observe the BS’s typical cell load under saturated traffic (*e.g.*, around 60% in Figure 5.5) to determine the cell load threshold D^* . During a runtime video chat session, if current cell load is D_t , the user applies a multiplicative increase factor α to its sending bitrate as:

$$\alpha = 1 + D^* - D_t \quad (5.3)$$

When D_t is well below D^* , the user increases the sending bitrate quickly, as it is sure that this will not overuse the bandwidth or freeze the video. As D_t approaches D^* , the sending bitrate plateaus as less remaining resource indicates higher risk of video freeze. It is worth noting that to guarantee the smoothness of video bitrate adaptation, for every time epoch t , the pyChat user is only allowed to increase video bitrate to the adjacent higher video bitrate level below $\alpha \cdot C_{t-1}$. Finally, if D_t exceeds D^* (which is possible as competing traffic emerges), FPRC will stop bandwidth exploration and switch to the aforementioned freeze prevention mode. In contrast, existing congestion control protocols like GCC uses a hard-coded constant α as they are unaware of the PHY bandwidth bound — they can only estimate the bandwidth by saturating the path and freezing the video.

Smoothing Video Bitrate Adaptation

Existing work [33] has shown that the smoothness of video bitrate adaptation also plays an important role for QoE. In other words, we should avoid sharp and frequent

changes in the video bitrate. pyChat considers the smoothness of video adaptation in both the freeze prevention mode and the bandwidth exploration mode. First, in freeze prevention mode, its utility function implicitly priorities the smoothness: when we have to reduce the video bitrate, to maximize the accumulated video bitrate with video freezing penalty (our utility function), the dynamic programming algorithm must reduce the video bitrate as late as possible and as less as possible, as long as the video does not freeze. Second, in bandwidth exploration mode, pyChat is only allowed to adapt between adjacent bitrate levels to avoid sharp video bitrate changes.

5.5 pyChat Implementation

Real-time LTE PHY-Layer cell load monitor. Since current smartphones do not expose the PHY resource utilization statistics to the user space, we implement pyChat’s LTE PHY-layer cell load monitor using the USRP software radio³, whose signal processing is done on a PC host tethered to our LTE smartphone. Our cell load monitor estimates the resource utilization within every LTE frame (10 milliseconds), which has orders of magnitude finer granularity requirement than the real-time LTE monitor implementations in existing works. For example, piStream [59] monitors the resource utilization of every segment during DASH video streaming. It only needs a time granularity of several seconds, equivalent to the DASH video segment duration. In contrast, pyChat’s freeze predictor requires the fine-grained resource utilization statistics in order to build the probabilistic prediction model, and to be responsive to potential video freezes given the shallow buffer of the video chat application.

To compute the frequency usage via FFT analysis, the monitor must precisely synchronize with the LTE BS, in order to obtain the starting sample index of each data symbol. Our resource monitor leverages LTE BS’s known preambles (referred to as PSS and SSS), broadcast every 0.5ms time slot, to achieve synchronization. To search for a PSS on a 10MHz LTE channel, we need to correlate the preambles

³Alternative solution that does not require extra client-side hardware is discussed in §5.7.

with all 76800 samples in a 0.5ms time slot. Our implementation has applied heavy multi-thread parallelization and core dedication on the PC host to prevent such heavy computational load from hurting the real-time video performance.

Integrating pyChat in Chromium browser. We prototype pyChat’s congestion control and video adaptation mechanisms based on the WebRTC module in the open-source Chromium browser. Our implementation keeps the WebRTC and browser API unchanged, so it is compatible with any browser-based JavaScript video chat applications, such as JitSi Meet [106], sipML5[109], and SIP.js [110]. pyChat’s transport/application layers run in parallel with the PHY resource monitor. The latter acts as an independent process, whose output is passed to the WebRTC module every 10 ms via shared memory. To implement pyChat inside Chromium, we modify the GCC implementation in WebRTC source code. More specifically, we add the interface to read the cell load passed through shared memory, the video freeze predictor, and the FPRC rate control logic whose output replaces GCC’s output. Along with pyChat, we have also implemented two state-of-the-art video rate control algorithms Sprout [20] and piStream[59] inside Chromium. Since the WebRTC client already has the variable for the incoming bitrate, to implement Sprout, we record the historical incoming bitrate values, use them to fit a Poisson distribution, and set current bitrate to the 95% lowerbound of this Poisson distribution, following the description in [20]. For piStream, as we can already read the cell load values in real time, by dividing the incoming bitrate by the cell load value, we immediately get the piStream’s bitrate selection.

Real-time end-to-end video delay measurement. Our experimental evaluation involves a comparison with commercial video telephony applications which do not allow for video frame interception and timestamp measurement. To circumvent such issues, we develop a non-invasive video delay measurement tool. The core idea is to watermark the sending time of a video frame in the video frame itself. The receiver extracts the sending time from the received video frame and subtracts it from local time to compute the screen-to-screen delay. The sender and receiver are synchronized using NTP which has millisecond level precision. To make the timestamp display robust to adverse LTE channels and video frame corruption, we

encode the sending timestamp (with millisecond resolution) using colored blocks. Each block representing one decimal bit, and 10 colors with uniform separation in the RGB code space are used.

To embed the color blocks into the video frame, we use html5 to create a webpage containing the color blocks, which change their pattern every millisecond. This webpage is rendered on the screen and mixed along with the video we want to send through the virtual webcam. We put the color blocks at the four corners of the screen to ensure that it can adapt to different screen size, and that it is not masked by the video display (cf. Figure 5.1). Ultimately, with this delay measurement tool, we can get a time sequence of delays for the video frames, and compute average video delay, identify video freeze and evaluate delay distribution for any video chat application without access to its source code.

5.6 pyChat Evaluation

In this section, we first use micro-benchmark experiments (§5.6.1) to validate pyChat’s design components, including the cell load monitor, freeze predictor and the FPRC algorithm. Then we conduct extensive system level tests on our real-time pyChat prototype and benchmark it against state-of-the-art video rate control algorithms and commercial video chat applications under various network conditions (§5.6.2). We conduct experiments over the Verizon LTE network on channel 13 (downlink 746-756MHz, uplink 777-787MHz) using Samsung Galaxy Nexus SCH-i515 smartphones. Similar to the setup in §5.3, to ensure the video bitrate is controllable and repeatable, we use virtual webcam to inject the same 720P movie clipped by FFmpeg to the video call applications for all experiments.

5.6.1 Micro-benchmarks

Verifying the cell load monitor. We first examine the accuracy and real-time monitoring capability of pyChat’s PHY-layer cell load monitor. To this end, we tweak the iperf source code so that it can generate a TCP traffic stream whose bitrate

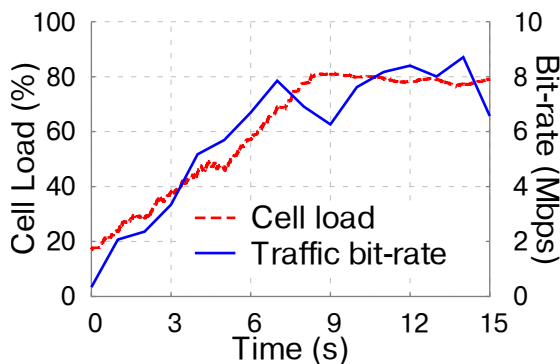


Figure 5.10: PHY cell load follows the

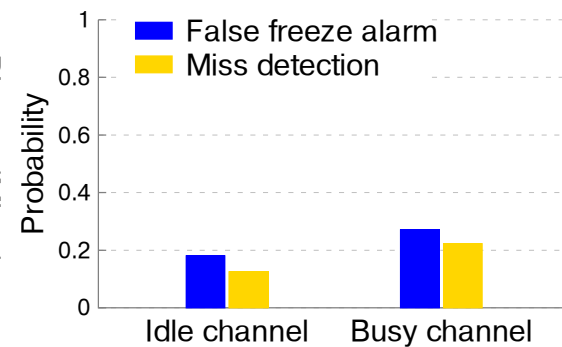


Figure 5.11: Accuracy of the video freeze predictor.

increases linearly over time. We run this iperf session between an Amazon EC2 server and LTE phone, while monitoring the cell-wide PHY resource utilization. The experiment was conducted in late night to avoid interfering traffic. We repeat this experiment for both LTE uplink and downlink, and then use a 200ms sliding window to average the resource utilization (200ms is also the typical RTT and adaptation granularity of FPRC). Figure 5.10 plots the resource utilization or cell load output from a downlink experiment, which shows a highly consistent trend with the increase of traffic bitrate. A similar observation is made for the uplink but omitted due to space constraint. This result again proves that the PHY cell load is an accurate indicator of the network congestion condition. In addition, it validates the feasibility of real-time cell load monitoring, and the effectiveness of our monitor implementation.

Verifying the PHY assisted freeze predictor. In this experiment, we train pyChat’s video freeze predictor offline, and then evaluate its accuracy in video chat sessions. A fully online version of pyChat will be evaluated in our system-level test (§5.6.2). We conduct the tests during both late night (idle channel) and noon (busy channel) on a campus LTE network. In each test, we first run a WebRTC video chat session for 10 minutes while monitoring the cell load. With the collected video delay, received bitrate, and cell load statistics, we derive the prediction model following §5.4.1. At run-time, the predictor estimates the video freeze probability on every cell load surge. We evaluate the prediction accuracy using the false alarm

and miss detection (prediction) of video freeze. We use 90% probability as the false alarm threshold: if the predictor outputs a probability higher than 90% but the video does not freeze within 3 seconds, then it is considered as a false alarm. Similarly, we define a 10% threshold for the miss detection. Figure 5.11 shows the freeze prediction accuracy of pyChat. We see that the predictor achieves 18% false alarm and 10% miss detection rate on the idle channel, versus 23% and 20% on the busy channel. The latter is slightly higher because of the elusive competing traffic during busy hours incurs more unpredictable network condition changes, and hence compromise the predictor's accuracy. Moreover, the miss detection rate is lower than false alarm rate, which is a desired trait because pyChat needs to err on the conservative side, preventing a possible freeze rather than catching a risky rate-increase opportunity. Overall, although the error rate ranges from 10% to 23% depending on channel conditions, this is already significantly better than the state-of-the-art WebRTC congestion control framework, which has failed to catch all the freezing events in this experiment.

Verifying the FPRC algorithm. We then verify the effectiveness of the FPRC algorithm, which adapts the transport layer and video rate to optimize QoE. Figure 5.12 plots the sending bitrate and cell load of a running WebRTC session controlled by FPRC. We can observe that FPRC reduces the video bitrate immediately on detecting cell load peak ② and ③ in Figure 5.12, which demonstrates its adaptation responsiveness. Furthermore, when the video bitrate is low, even we have a cell load peak (peak ① in Figure 5.12), the video freeze probability is still low and there is no need to throttle the rate, we see that pyChat behaves exactly in this way. In addition, we see that FPRC can effectively exploit the network bandwidth by quickly increasing bitrate (around $t=30s$ in Figure 5.12) when both the video bitrate and the cell load are low. In contrast, traditional algorithms like GCC can only slowly (to prevent sudden freeze) explore the available bandwidth, which often costs tens of seconds like in Figure 5.2(a).

We then evaluate FPRC's performance of video quality over a longer period, which is measured by the received video bitrate at the WebRTC client. We compare FPRC with vanilla WebRTC which uses GCC for rate control. All tests are performed

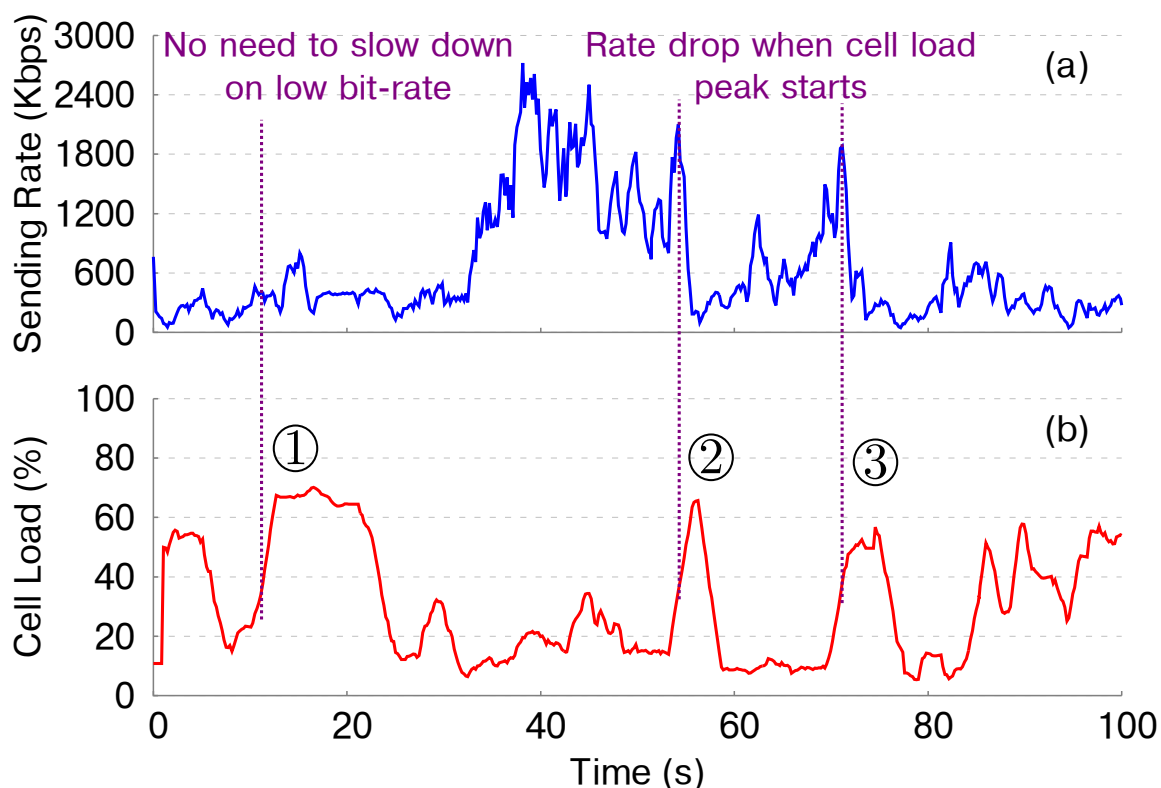


Figure 5.12: FPRC adapts the traffic bitrate following the cell load to prevent video freezes.

in an office environment during normal working hours. The LTE phone acts both a video chat sender and receiver, and we measure the performance of both upstream and downstream. We repeat each test run by 20 times, each lasting 5 minutes. Figure 5.13 plots the average bitrate and *std.* as error bar. We observe that FPRC almost achieves $1.92\times$ higher bitrate over GCC, which translates into a almost doubled video quality. This verifies that FPRC can respond promptly to the availability of PHY resource, and improve bandwidth utilization. On the other hand, Figure 5.14 plots the distribution of video delay across all received video frames. We see that the 90-percentile latency of FPRC is similar to that of GCC. However, FPRC's worst-case latency, which determines the amount of video freezes, is significantly lower than GCC. In particular, only 2% frames in FPRC experience a video delay longer than 1s but up to 1.3s, whereas 10% frames in GCC experience a delay longer

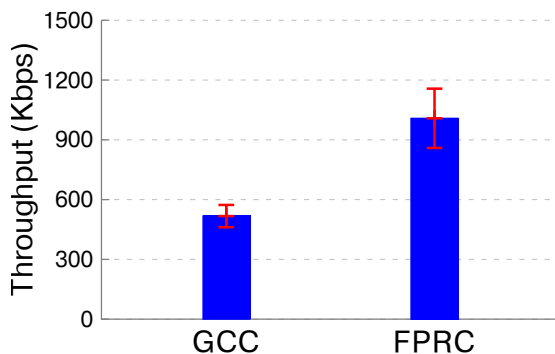


Figure 5.13: FPRC increases bitrate when unlikely to freeze thus achieves higher video quality than GCC.

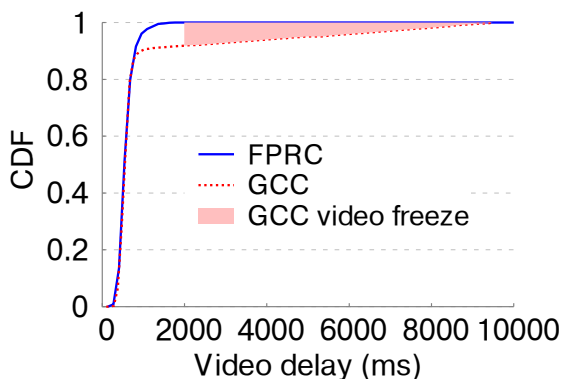


Figure 5.14: FPRC reduces bitrate when video tends to freeze thus achieves lower video delay than GCC.

than 1s and up to 9.5s. If we empirically define freezing as a frame arrival with $> 2s$ latency, then GCC experiences a freezing rate of 8% which degrades QoE substantially, while FPRC almost has no freezing events across all the tests.

5.6.2 System Level Test

We now compare the performance of our pyChat prototype with state-of-the-art congestion control and video rate adaptation algorithms including Sprout [20] and piStream [59], and commercial video telephony applications including Skype, Google Hangout and FaceTime. To avoid repetition, we mainly focus on the case when the LTE phone is a downstream video chat receiver. In our implementation, the pyChat, Sprout and piStream are all prototyped on top of the WebRTC's transport layer (§5.5). In addition, they all use the Jitsi Meet [106] as video front-end and session management. We set up our own video session signaling and RTP relay server on Amazon EC2 to eliminate unexpected bottleneck at the relaying server. Before each experiment, we also run iperf bandwidth measurement to guarantee no unknown bottleneck exists between both ends of the video call. To implement piStream (originally developed for DASH[28] video streaming where the client can pre-buffer the video content to be played in future), we apply its video rate adaptation algorithm to each way of the video chat stream, and allow it to access

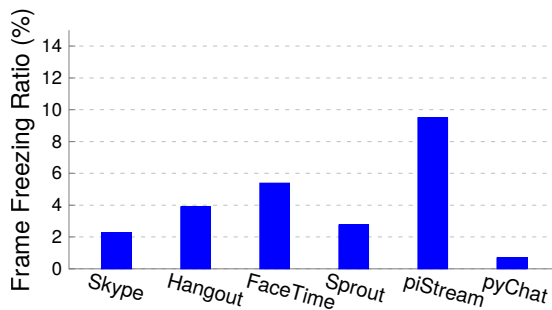


Figure 5.15: Video freezing ratio in idle channel.

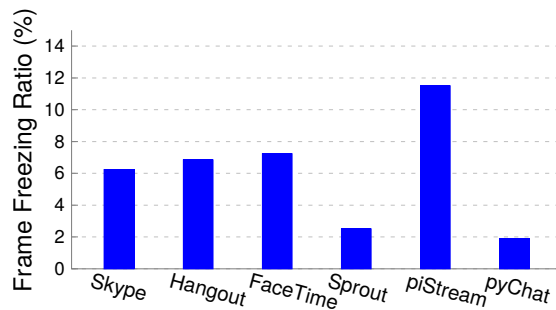


Figure 5.16: Video freezing ratio in busy channel.

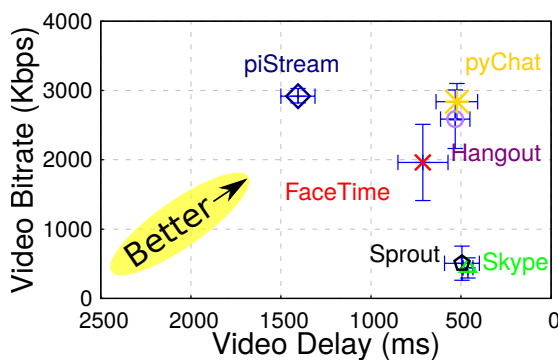


Figure 5.17: Video quality & delay in idle channel.

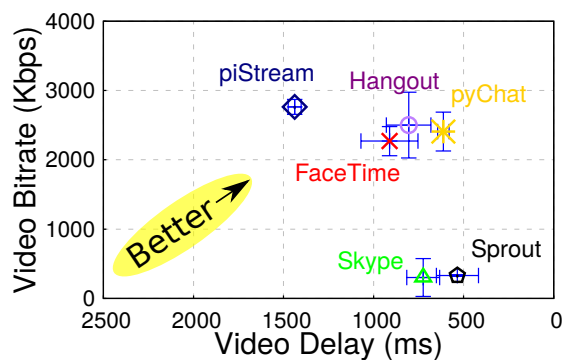


Figure 5.18: Video quality & delay in busy channel.

the same cell load information as pyChat.

Under varying background traffic loads. We first run the video chat applications at late night (idle channel) and noon (busy channel), respectively. For a sanity check, we used our cell load monitor to measure the background cell load, which is close to 0 in the late night, and mostly above 30% in the noon. In our tests, each video call session lasts around 3 minutes, and for each scheme we repeat the test by 10 rounds. During the experiments, the LTE phones remain static on the table to isolate the channel variation caused by mobility.

Figure 5.15 and 5.17 show the performance under an idle channel, while Figure 5.16 and 5.18 show the busy hours. From the results, we make the following observations. (i) pyChat achieves the second highest video quality, only slightly lower than piStream, among all tested schemes. It improves the average video

bitrate by $5\times$ to $6\times$ over Skype, $1.1\times$ over Google Hangout, $1.2\times$ to $1.5\times$ over Facetime and $5\times$ to $6\times$ over Sprout. Such benefit mainly comes from pyChat's FPRC, which exploits the bandwidth promptly when it knows the risk of video freeze is low. More importantly, pyChat has a low freezing ratio of 0.5% to 1.1%, which is multiple folds lower compared with all other solutions, and it also has close-to-lowest average latency. This advantage roots in pyChat's awareness of PHY cell load surges, which cannot be handled by any high-layer congestion metrics.

(ii) Sprout has no bandwidth exploration mechanism as it adapts the rate only based on historical throughput, rather than the unexplored "available" bandwidth. If the video starts with a low bitrate, which is the common strategy of most video call applications, Sprout will barely have the chance to increase its bitrate. Even though we are already using the most aggressive Sprout variant which essentially use a value higher than 95% of the historical incoming rate as the sending rate [20], the resulting bitrate is still too low. In fact, Sprout claims to work only in a saturated channel where bandwidth exploitation is unnecessary, and its performance was verified in a simulated environment with a bandwidth saturator to ensure such operating condition is met [20].

(iii) piStream performs very aggressive bandwidth exploitation, which keeps scaling up the video bitrate to achieve close-to-100% expected cell load. Such a strategy grants it high video quality, and works well for one-way video streaming where the client maintains a playback buffer of several seconds. However, when applied to interactive video, piStream suffers severe video freezes since full resource exploitation is not the correct strategy for interactive video. As shown in §5.3, interactive video freezes due to micro-congestion even before the BS is saturated. For conventional video streaming, this phenomenon is masked by the large receive buffer. However, when we migrate the aggressive video streaming algorithm like piStream to interactive video with very small buffer due to the strict delay constraints, the video freezes frequently since there is no longer a large buffer to absorb the high video delay variations caused by aggressive bandwidth exploitation. We further discussion the difference between interactive video and one-way video streaming in §5.7.

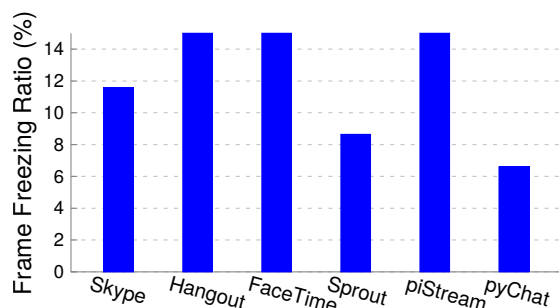


Figure 5.19: Video freezing ratio under driving mobility.

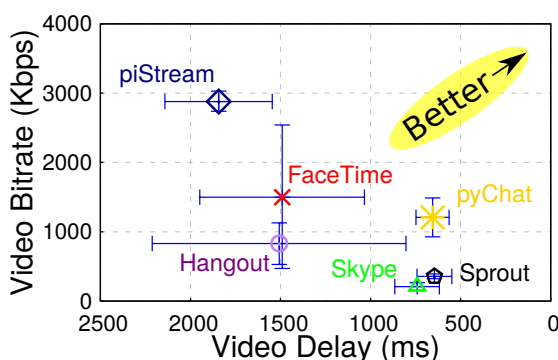


Figure 5.20: Video quality&delay under car driving.

(iv) Unlike other commercial video call applications, Skype behaves in a very conservative manner. It gives up exploring the network bandwidth and keep a low video bitrate. Such strategy helps it to achieve lower video delay (Figure 5.18, green point) and video freeze ratio (Figure 5.16) than both FaceTime and Google Hangout. However, this comes at severe video quality sacrifice.

Under varying channel quality caused by mobility. We further evaluate pyChat on a vehicle moving at 40 mph in a suburban area. Each test video session lasts for 2 minutes and is repeated along the same driving route. The results in Figure 5.19 and 5.20 demonstrate that all tested schemes experience higher freezing ratio compared with the static scenario. Skype and Sprout are less affected by mobility, as they always keep a low video bitrate. But even such conservative schemes cannot prevent video freezes caused by channel variations. On the other hand, more aggressive schemes (Hangout, FaceTime and piStream) suffer from severe freezing (up to 35%) and latency variation. In contrast, pyChat's 6% freeze ratio is the lowest, against 9% to 35% in other schemes since any channel quality drop during mobility will cause a cell load surge (§5.4.1), which then triggers the responsive rate throttling of pyChat to avoid video freeze. Clearly, responsiveness to channel condition change is the key to LTE video telephony under mobility scenarios, and pyChat's awareness of PHY cell load enables it to make the best tradeoff between video quality and freezing risk.

Coexisting with TCP. Given the fact that HTTP/TCP traffic accounts for a large

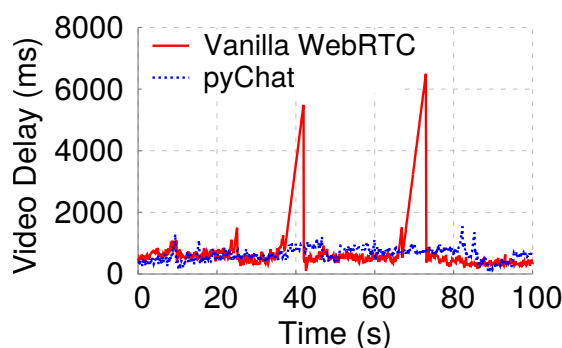


Figure 5.21: Vanilla WebRTC freezes when coexisting with TCP flow. pyChat does not.

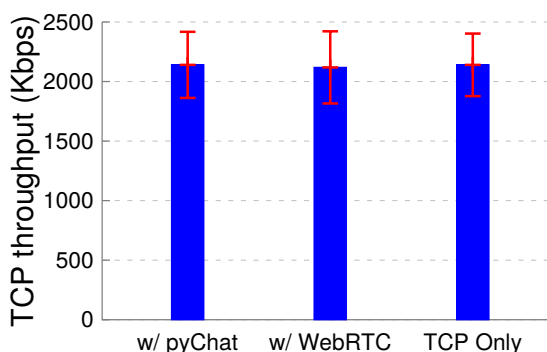


Figure 5.22: Both pyChat and Vanilla WebRTC do not affect the TCP throughput.

fraction of the wireless Internet traffic, it is nature to ask two fairness-related questions: First, will pyChat be unfairly affected by competing TCP traffic? Second, will pyChat video jam the competing TCP traffic? In this experiment, we demonstrate that neither happens due to pyChat's prioritizing of delay performance. Our tests run a iperf session with 2Mbps TCP traffic to a second user in parallel with the video chat user. Figure 5.21 shows that while coexisting with TCP flow, vanilla WebRTC experiences severe video freezes, whereas pyChat achieves a stable video delay and no freezing. From Figure 5.22, we further observe that pyChat does not affect the TCP performance at all. In other words, pyChat prevents video freezes by throttling its own traffic rate rather than suppressing coexisting TCP flows.

5.7 Discussion

Infrastructure support vs. end-host solution. Existing work [111] proposed the BS-centric video adaptation over LTE, which assumes the BS perform centralized rate control in its scheduler. However, such a solution requires the LTE BS to be able to discriminate application-layer traffic, which is non-trivial. It also requires the BS to intercept the control plane of every end-to-end interactive video session, which may harm the flexibility and scalability of interactive video applications. Furthermore, the BS-centric solution requires heavy modifications to the LTE MAC

scheduler, which has been built into hardware and entails high retrofitting cost. In contrast, pyChat is a host-centric solution that performs end-to-end rate control. It remains compatible with current LTE infrastructures and hence is readily deployable. Nonetheless, its adaptation principles will remain applicable if the cellular BS can directly provide congestion signaling in future.

Note that although our prototype implementation still employs a software radio as a smartphone proxy to monitor the LTE cell load, recent work [60] has shown that cell load estimation can be done using COTS phones. Specifically, RSRQ, an LTE PHY layer metric readily accessible from the user space, can be leveraged for coarse-grained cell load estimation. We plan to integrate this RSRQ-based cell load estimation into pyChat in future work.

Video chat vs video streaming. Existing work on LTE video streaming, such as piStream [59], also leveraged the LTE PHY-layer cell load information to maximize bandwidth utilization. However, such approaches no longer work for interactive video. As shown in §5.3, interactive video freezes even before the bandwidth saturation due to the micro-congestion phenomenon. This is because compared to conventional video streaming in which the receiver can pre-buffer tens of seconds of video, the interactive video is generated in real-time, and delivered with a few hundred milliseconds of tight delay constraint. Our experiments in §5.6.2 have demonstrated that piStream-like bandwidth exploitation results in severe freezing effects for interactive video.

5.8 Summary

Existing congestion control protocols can only react passively to delay/loss events, which fall short of responsiveness and harm video telephony's QoE, especially in highly dynamic LTE networks. In this chapter, we have presented measurement evidences that demonstrate the feasibility and effectiveness of using physical channel information to predict video telephony performance over LTE networks. Our joint congestion control and real-time video adaptation framework, pyChat, harnesses such a PHY aware predictor as a new congestion metric, to optimize the balance

between video quality and freezing rate. Given the recent standardization activities in mobile edge computing architectures that advocate low-layer feedback [87], we believe pyChat holds potential as a practical mechanism to meet the growing challenges and demands in wireless video telephony.

6 360° PANORAMIC VIDEO TELEPHONY OVER LTE CELLULAR NETWORKS

6.1 Introduction

Panoramic videos, also known as 360° videos, is gaining traction in the consumer electronics and mobile application ecosystem. Many video content providers [112, 113] recently started to stream 360° videos, which are usually captured by omnidirectional cameras, and can be viewed by virtual reality head-mounted-displays (VR HMDs), such as Oculus Rift, Google Daydream, HTC VIVE, and Samsung GearVR. Besides fetching 360° videos stored on a server, the emerging mobile 360° stereoscopic cameras [114, 115, 116, 117], combined with the mobile HMDs, can potentially enable a new wave of *interactive 360° video* services that capture & stream 360° videos in real-time. Some examples include elevating current 2D video chat into 360° video chat, panoramic live broadcasting for outdoor events and adventures, and flying a drone remotely as if sitting inside a virtual cockpit (Fig. 6.1). However, existing works mainly investigate on-demand 360° video streaming from content providers, while the area of interactive 360° video telephony remains mostly unexplored.

In this chapter, we present POI360, the first PORTable Interactive 360° video telephony system running over existing LTE cellular infrastructure. Two major challenges emerge in realizing POI360: (i) *Portability*. To perform ubiquitous real-time streaming that exceeds the Wi-Fi coverage, POI360 relies on the mobile broadband for uploading/downloading the live 360° video stream. Unfortunately, current cellular network has not kept pace with the boom in mobile devices and applications. The close viewing distance of near-eye display for VR HMD entails high resolution to maintain a reasonable quality-of-experience. Furthermore, a 360° video contains views in every direction, which translates into a bitrate of tens of Mbps and drastically overloads the LTE uplink (median bandwidth around 2.2 Mbps [118]). (ii) *Interactivity*. POI360 needs to confront an intrinsic conflict between the strict

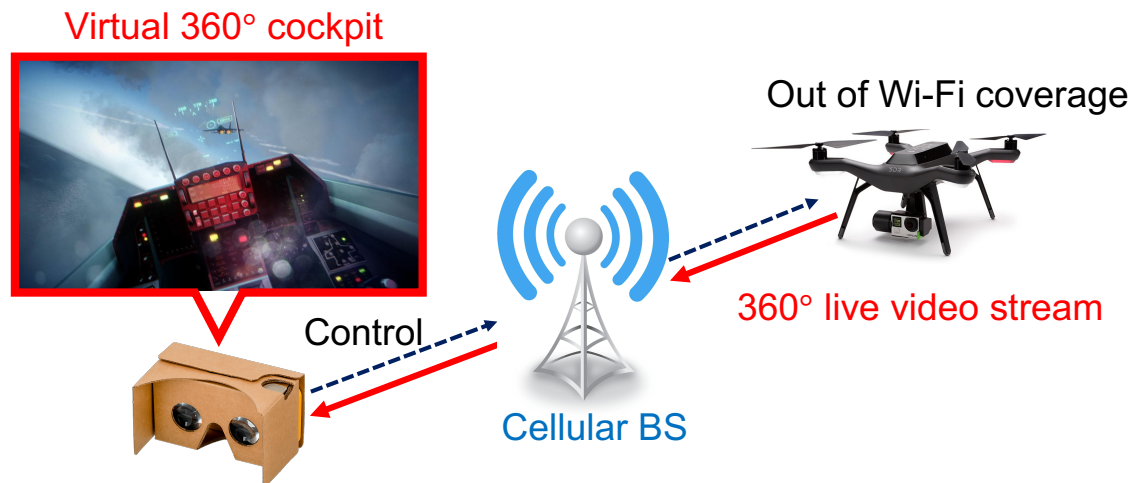


Figure 6.1: Example application scenario for portable interactive 360° video telephony over cellular networks.

latency requirement of interactive video and the highly dynamic cellular network condition. The situation is even more challenging when the heavy 360° video traffic pushes the cellular link to its limit, as a fluctuating cellular link bandwidth can easily fall below the high traffic load. POI360's rate control algorithm must detect this and reduce the traffic rate immediately, otherwise the video will freeze after a congestion.

To meet these challenges, POI360 design achieves both efficient compression to curtail the overwhelming bitrate of 360° video stream, and responsive rate control to facilitate smooth real-time video telephony. (i) Although existing works for 360° video compression [119, 120, 121, 122] can effectively reduce the traffic load by prioritizing the encoding quality of the region-of-interest (ROI) in a 360° frame, the highly dynamic cellular network latency may easily fail such ROI-based compression, as the sender's ROI knowledge may lag behind the remote user's actual ROI, causing mismatches between the sender-allocated encoding quality and user-perceived display quality, and thus severe quality fluctuations (§6.3). POI360 answers this challenge with an adaptive spatial compression algorithm – Instead of sticking to static compression ratios for ROI and non-ROI regions, it dynamically tunes the encoding quality distribution across the 360° panorama based on the ROI

update responsiveness, thus guarantees a stable user-perceived quality during user ROI change. (ii) To boost the responsiveness of rate control, since cellular uplink is the typical network bottleneck for interactive video (§6.3), POI360 monitors the cellular device's uplink firmware buffer occupancy to instantaneously detect any uplink congestion. Upon congestion, POI360 degrades its uploading bitrate to the windowed-sum of cellular uplink transport block size (TBS), which is the current bandwidth guaranteed to be achieved.

We have implemented a POI360 prototype over commercial LTE cellular networks and smartphones. Targeting VR HMD using smartphone as screen, POI360 runs in mobile browsers, and our implementation integrates WebRTC [98] which serves as the back-end of most interactive video services [99], and WebGL, which converts the 360° WebRTC video stream to the HMD-compatible stereoscopic view in the browser. We implement POI360's cellular-link-informed rate control in the WebRTC module of the Chromium browser. The required cellular physical-layer statistics, including the uplink firmware buffer occupancy and TBS, are read from the diagnostic interface on commodity phones. Therefore, POI360 requires no hardware modification and is readily deployable. Our run-time experiments demonstrate that POI360's adaptive spatial compression algorithm significantly outperforms existing benchmark algorithms, and POI360's rate control is much more responsive than conventional control mechanisms that rely on end-to-end delay/loss metrics. We have also conducted field tests under various locations and mobility levels, which confirm POI360's effectiveness in enabling real-time interactive 360° video services over LTE cellular networks.

The contributions of POI360 are summarized as follows:

(i) To the best of our knowledge, POI360 is the first portable interactive 360° video telephony system customized for existing cellular network infrastructure.

(ii) POI360 features adaptive spatial compression for 360° video telephony, which dynamically tunes the compression strategy to combat the system lag of user ROI update, in contrast to existing works that stick to static compression ratios for ROI/non-ROI regions.

(iii) POI360's awareness of cellular link statistics enables quick adaptation to

cellular uplink congestion and thus smooth 360° video telephony, while existing works[59, 16] focus on the throughput of on-demand downlink streaming.

6.2 Background of 360° Video

360° or panoramic videos are typically shot using a fisheye camera [123] or an array of cameras whose field-of-view (FoV) together cover the 3D space. The captured video frames are then stitched and projected onto a sphere, which is further mapped to a planar format in order to be compressed using standard video encoders. The world map is a classical example of sphere-to-plane conversion, following an equirectangular projection. But this straightforward conversion incurs redundant pixels especially near the poles of the sphere. To further compress the video data, alternative conversion schemes instead project the sphere into a cube [124] or pyramid [125], which can be further unwrapped onto a cube plane.

Nonetheless, the bitrate of a 360° video (*e.g.*, tens of Mbps for 4K video [119]) typically exceeds the link throughput of average users on a cellular or wired access network [53]. To alleviate the bandwidth crunch, the concept of region-of-interest (ROI) streaming has been explored extensively [121, 120, 126, 127, 128]. Human eyes have limited FoV, and the vision resolution is high at the central visual field (fovea) but drops off roughly quadratically with distance from the center. Accordingly, the resolution of tiles near the visual periphery or in the region with low viewing probability can be compressed more aggressively. As shown in Fig. 6.2, by estimating the viewer's foveation using head-orientation sensors inside a VR HMD, the video server can selectively stream video tiles—those within the ROI will be delivered with high resolution while the remaining ones will have minimal quality. Such ROI streaming mechanisms can bring the bandwidth cost to a level that can be supported even in cellular networks [122]. Besides orientation sensors, alternative approaches can further narrow down the ROI, *e.g.*, using an eye-tracking camera to track the viewer's focus [128], using semantic analysis of the video frame content or crowd-sourced viewing statistics [129, 119, 122]. Although this work focuses on the head-orientation-based ROI compression, our compression algorithm can be

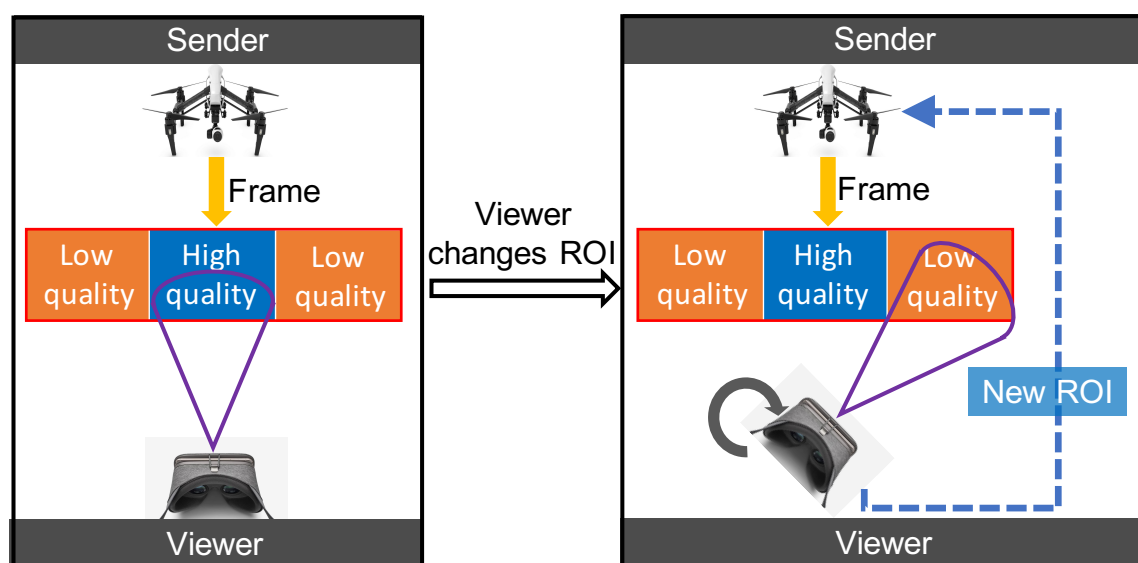


Figure 6.3: Problem for ROI-based compression: viewer ROI change causes user-perceived quality fluctuation.

work latency of ROI feedback, the panoramic video sender and viewer may have inconsistent knowledge of the viewer's current ROI, thus there can be a temporary mismatch between the sender-allocated quality and user-perceived quality for the ROI region whenever the user view changes, causing quality fluctuations and user sickness. In §6.6, we further verify this phenomenon through real-time experiments (Fig. 6.13b). We note that motion-based ROI prediction [130, 131] cannot fully solve this problem as its prediction horizon is still shorter than the end-to-end latency of mobile interactive video sessions (§6.7).

Solution space I. To mitigate the quality fluctuation, we need to address the trade-off between the compression aggressiveness and the stability of ROI quality. An aggressive compression mode can focus the quality on a small ROI while minimizing the non-ROI bitrate. But it is sensitive to the ROI changes as the user's FoV can easily enter the non-ROI region. So it applies only to networks with small feedback latency. On the other hand, a conservative compression mode that distributes the quality more evenly across spatial regions is suitable for high-latency networks in order to keep stable video quality in ROI region even under

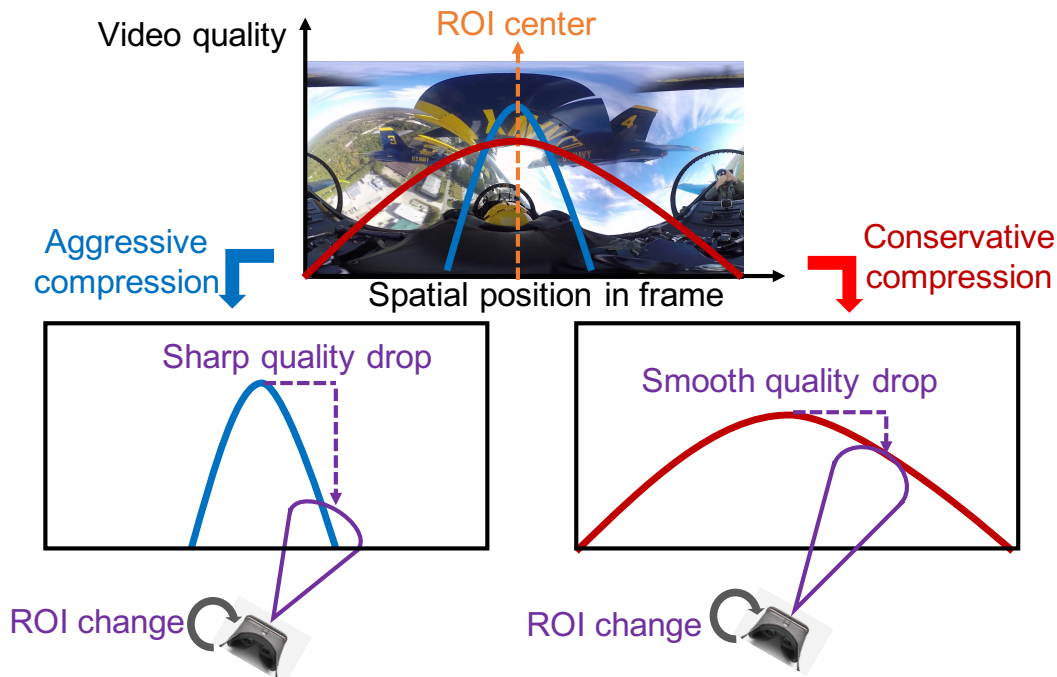


Figure 6.4: Trade-off between compression ratio and user-perceived quality stability during ROI update.

laggy ROI feedback, but has to sacrifice the quality itself to fit the entire frame within limited network bandwidth. To make the situation worse, cellular networks have highly dynamic latency [53], causing unstable responsiveness of the end-to-end ROI update.

To resolve these issues, instead of sticking to a fixed compression mode, we can dynamically switch across different modes based on the responsiveness of ROI update, as shown in Fig. 6.4. This essentially follows the cross-layer design principle – latency of the network layer ROI feedback can be used to control the application layer traffic load compression. In this way, we can aggressively reduce the traffic load when current network condition allows the sender to responsively update its ROI knowledge, and conservatively compress the stream to maintain a stable video quality when the end-to-end ROI update becomes sluggish.

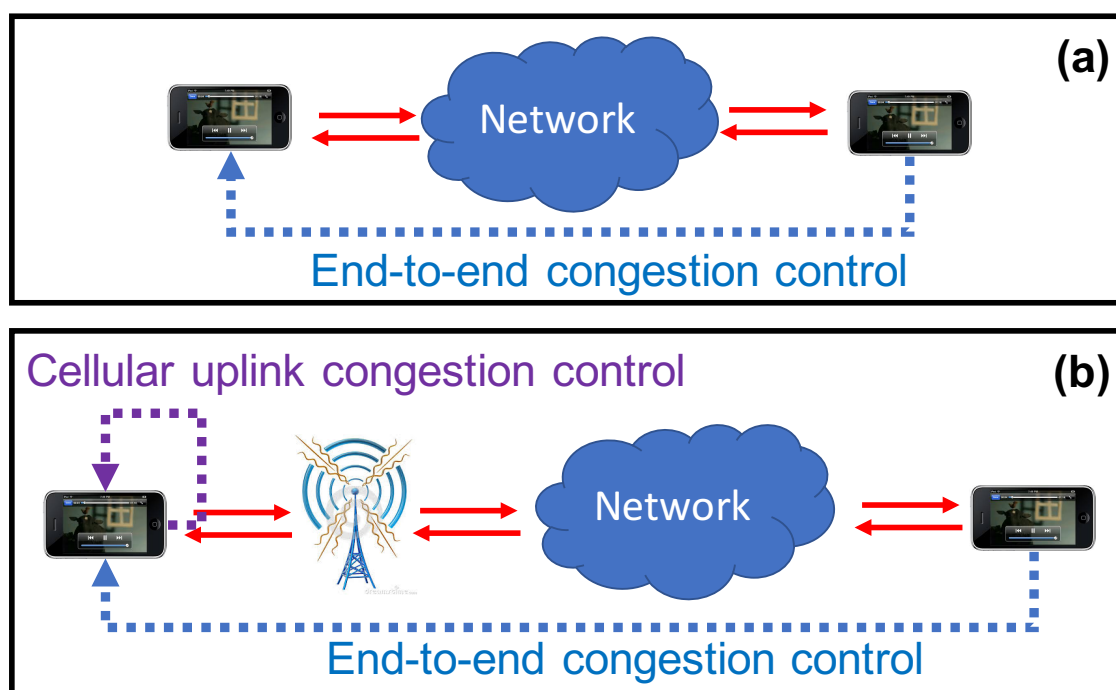


Figure 6.5: (a) Conventional end-to-end rate control. (b) Cellular uplink congestion control for interactive VR.

6.3.2 Video Rate Control Responsiveness

Challenge II. The rate control algorithms must responsively adapt the panoramic video encoding bitrate based on network conditions, to avoid congestion and video freeze. However, existing video bitrate adaptation algorithms for interactive videos commonly adopt end-to-end network condition estimation (§6.2), as illustrated in Fig. 6.5. Such strategies fail to meet the responsiveness requirement of interactive panoramic video, especially over highly dynamic cellular networks (experimental evidences will be shown in §6.6).

Solution space II. For panoramic video telephony, cellular uplink is typically the network bottleneck, but needs to sustain the same traffic load as the downlink or wireline segments. Therefore, the sender can actually bypass the sluggish end-to-end metrics, and directly execute congestion detection and bandwidth estimation for the cellular uplink alone.

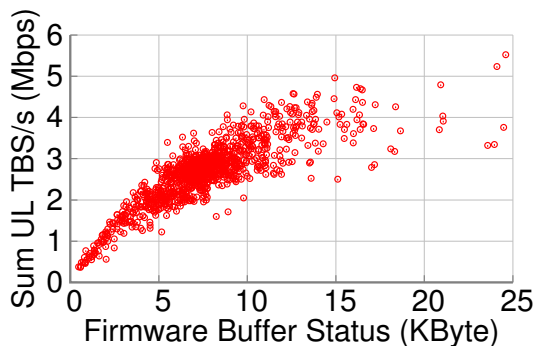


Figure 6.6: The relation between uplink firmware buffer occupancy and TBS.

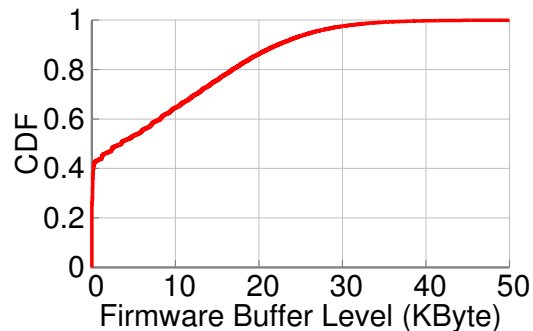


Figure 6.7: Uplink firmware buffer level under WebRTC default rate control.

6.3.3 Uplink Bandwidth Underutilization.

Challenge III. Due to the use of proportional fair scheduling in LTE networks [132], the user device’s uplink firmware buffer has an unique feature – its service rate depends on its own buffer occupancy. Fig. 6.6 showcases this effect, where we measure the buffer size and uplink throughput R_{phy} (indicated by the TBS) on an LTE smartphone. With small buffer size, the R_{phy} increases almost linearly with the buffer size, and saturates when reaching the uplink bandwidth. Unaware of this effect, generic video rate control frames, like WebRTC, simply set the RTP sending rate R_{rtt} to match the video bitrate R_v . However, a temporary uplink bandwidth surge can drain the buffer quickly, and the RTP sender may leave the buffer empty as it needs to follow the R_v cap, even if there exists pending traffic at the video application layer. As an evidence, Fig. 6.7 plots the CDF of uplink buffer level when streaming a 4K panoramic video through an LTE phone (more details about the setup are in §6.5). We see that the uplink buffer becomes empty for 40% of the time, even though the traffic always exceeds the available bandwidth.

Solution space III. The knowledge of the uplink firmware buffer level opens a new opportunity to address this challenge. In particular, the sender can push the firmware buffer level to a “sweet” region which is far from congestion but still high enough to harness the bandwidth provisioned by the BS’s uplink scheduling. As the video encoding bitrate R_v should not be changed too fast to avoid quality fluctuations that may degrade user experience, we keep updating the RTP sending

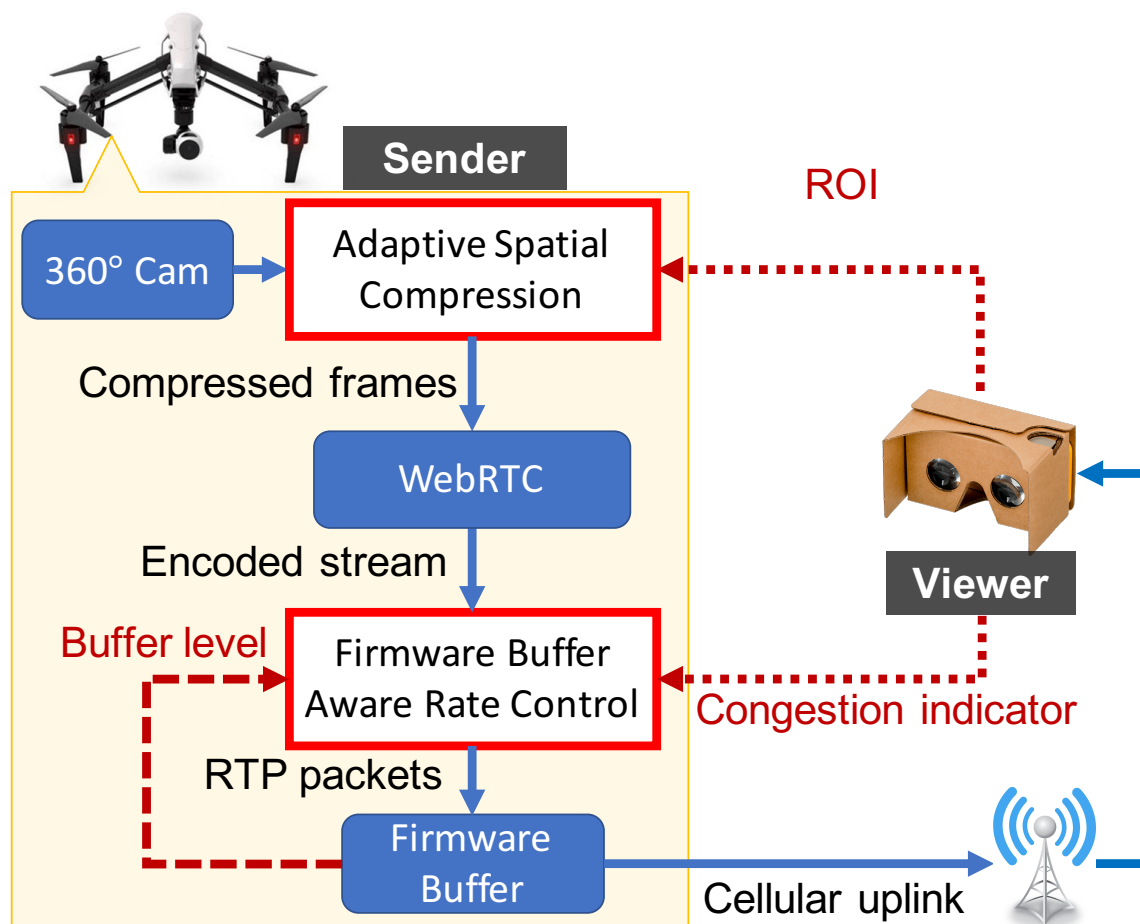


Figure 6.8: POI360 system architecture.

rate R_{rtp} based on the instantaneous buffer level to keep it in the “sweet” region.

6.4 POI360 Design

In this section, we present our system design to exploit the solution space discussed in §6.3. POI360 is built over the WebRTC[98] framework. As outlined in Fig. 6.8, the POI360 sender first applies *adaptive spatial compression* to perform ROI-based spatial compression to the input 360° video stream while adapting the compression mode based on current network condition, the ROI-compressed stream is then encoded

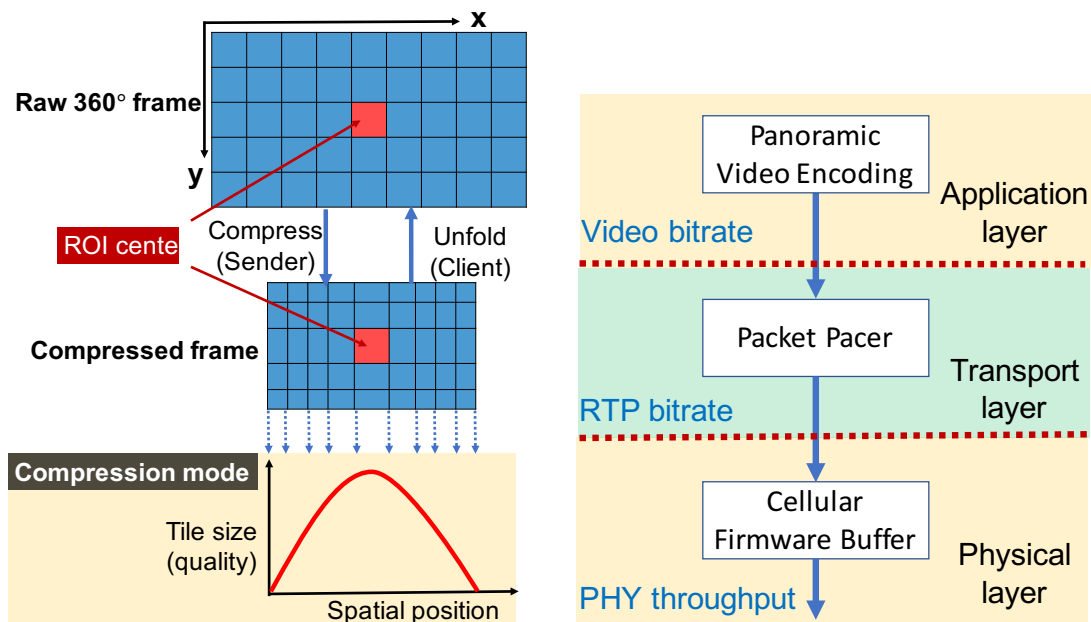


Figure 6.9: POI360 video compression Figure 6.10: POI360 rate control model.

by the WebRTC framework’s encoder (by default the VP8 encoder) which performs temporal compression to the frames. The sender then uses *firmware buffer aware rate control* to efficiently deliver the encoded stream to the remote viewer wearing a VR HMD. Packet losses and rebuffering during streaming are handled by WebRTC’s builtin mechanisms[133]. In what follows, we introduce POI360’s system model (§6.4.1), and then describe its two major design modules (§6.4.2 and §6.4.3) in more detail.

6.4.1 System Model

360° frame compression. As illustrated in Fig. 6.9, POI360 compresses 360° video stream at the video frame level. Each raw 360° video frame is first spatially segmented into small tiles, which are then compressed separately based on their relative position *w.r.t.* the user’s current ROI centre. Finally, all tiles of a video frame are stitched together and sent through the cellular uplink. Based on Fig. 6.9, we

define several compression parameters to be used across this chapter. Let i denote a tile's position in the x-axis while j denote its position in the y-axis in a 360° video frame. The center of the user's current ROI is a tile whose position is denoted as $r = (i^*, j^*)$. We define *compression level* l_{ij} as the ratio of tile size before and after compression, and *compression matrix* L is the matrix of compression level l_{ij} for all tiles. We further define *compression mode* $\mathcal{F}$ as the mapping that determines the compression level l_{ij} of each tile based on the distance between its position (i, j) and the current ROI center $r = (i^*, j^*)$:

$$l_{ij} = \mathcal{F}(i - i^*, j - j^*) \quad (6.1)$$

The basic principle behind POI360's compression mechanism is to use higher compression level for the tiles further away from the ROI center r , and the center itself should always have the lowest compression level $l_{\min}$. Under a certain compression mode, when the ROI center shifts with the user's head position, the compression matrix will be updated following Eq. (6.1), which is essentially a cyclic shift based on the shift of ROI center.

360° video stream rate control. As illustrated in Fig. 6.10, POI360 uses a cross-layer rate control model at the 360° video sender, which contains two important buffers: the video buffer in the application layer, and the cellular firmware buffer in the LTE physical layer. Below are the crucial parameters:

360° video encoding bitrate (R_v) determines the bitrate of the compressed video source, *i.e.*, traffic injection rate (video quality) into the video buffer. When the network is congested, the sender should throttle R_v .

RTP sending rate (R_{rtp}) in the transport layer parameter controlling the outgoing traffic of the video buffer and the input traffic into the cellular firmware buffer.

PHY-layer throughput (R_{phy}) determines the outgoing traffic of the cellular firmware buffer. R_{phy} is controlled by the BS's uplink scheduling policy. However, as will be clarified soon, POI360 can indirectly affect R_{phy} by controlling R_{rtp} .

From the diagnostic interface on commodity smartphones, we can extract the LTE physical-layer statistics, *i.e.*, the uplink firmware buffer level (B) and uplink

transport block size or TBS (P_{tbs}), for every LTE uplink subframe [72].

6.4.2 Adaptive Spatial Compression

POI360's adaptive compression module aim to tame the 360° video quality fluctuations caused by ROI changes. As discussed in §6.4.1, for ROI-based 360° video stream compression, the sender needs to keep updating the compression matrix $L(t)$ based on $r_s(t) = (i_s^*, j_s^*)$, its current knowledge of the client-side ROI. Due to the end-to-end feedback latency d_f , the sender's knowledge of the viewer's ROI position $r_s(t)$ lags behind the actual value $r_c(t)$, in other words, $r_s(t) = r_c(t - d_f)$. As a result, when the viewer shifts her ROI to $r_s(t)$ at time t , the sender will keep using the previous ROI $r_c(t - d_f)$ to compress the 360° video stream until $t + d_f$. In this duration, the user view has shifted to the new region, while the sender has not updated the compression level for this region yet. So the viewer observes a quality drop in ROI since time t , and then experiences a quality surge at time $t + d_f + d_v$ (where d_v denotes the one-way video frame delay), as it takes some time for the sender to be aware of an ROI change and deliver the video frames compressed with updated ROI to the viewer. Such quality fluctuations severely degrade the user experience[33].

To address this issue, POI360's adaptive spatial compression allows the sender to gauge the responsiveness of the end-to-end ROI update, and then adopt the spatial compression mode $\mathcal{F}$ that best fits current responsiveness level. Under swift ROI update, the sender can aggressively compress the non-ROI regions. On the other hand, under sluggish ROI update, the sender should create a smooth quality transition from ROI to the non-ROI regions, as the compressed content may be temporarily displayed in the new ROI. The key challenge here is to *precisely estimate the end-to-end ROI update responsiveness*. As shown in Fig. 6.11, this responsiveness is affected by 3 factors: (i) *The ROI feedback delay* d_f determines how long it takes for the sender to be aware of an ROI change. (ii) *The one-way video frame delay* d_v reflects how long it takes for the 360° video stream compressed with the updated ROI to arrive at the client. (iii) *The ROI stability* shows how often the user changes

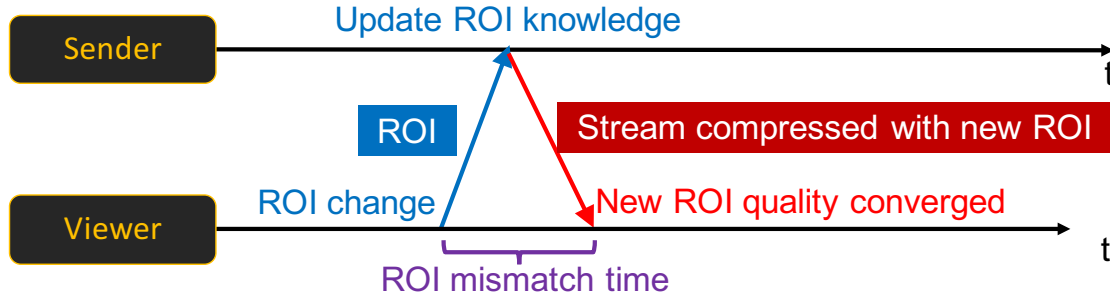


Figure 6.11: ROI mismatch time.

the ROI and by how much.

POI360 employs a single metric, the *ROI mismatch time* (M) to capture all these factors. As shown in Fig. 6.11, M is defined as the time interval during which the 360° video sender and client have inconsistent knowledge of the user's ROI. When the network causes higher ROI feedback latency, M will be higher. Similarly, when the network causes higher video frame delay, it takes longer for the frames with updated ROI to reach the client, leading to higher M as well. Finally, when the user switches the ROI consecutively, inconsistency becomes more severe, again leading to higher M .

POI360 measures M by monitoring the ROI video quality at the client side: Ideally, the ROI region should always enjoy the highest quality. In other words, the ROI compression level $l(i_c^*, j_c^*)$ should equal $l_{\min}$. However, when the user quickly switches to a new ROI, her head may temporarily point to a low-quality region. Following Eq. (6.2), POI360 measures M at the time scale of every 360° video frame.

$$M = \begin{cases} \max(t - t_0, d_v), & \text{if } l(i_c^*, j_c^*) \neq l_{\min} \\ d_v, & \text{otherwise} \end{cases} \quad (6.2)$$

where the client starts counting the time on detecting the ROI change at time t_0 , and computes $M = \max(t - t_0, d_v)$ for the frame coming at time $t > t_0$ until the quality in the new ROI converges to the highest level. For the frames in which current ROI already has the lowest compression level, *e.g.*, when the user keeps

the ROI, we have $M = d_v$, as the update latency for potential ROI change must be higher than current frame delay d_v . The client then uses a sliding time window to average across the frame-level measurements of M , and periodically feeds back the average M to the sender, with the feedback interval the same as the video frame interval. After receiving M , to achieve a balance between traffic load reduction and the stability of user-perceived ROI quality, the sender then switches between K pre-defined compression modes $\mathcal{F}_1, \dots, \mathcal{F}_K$ listed in the order of decreasing compression aggressiveness (smoother quality drop for regions further away from ROI). Our POI360 implementation uses $K = 8$ and empirically determines the index of the compression mode $\mathcal{F}_{i_m}$ to select as $i_m = \max(8, \lceil M/200\text{ms} \rceil)$. After the adaptive spatial compression, the ROI-compressed video stream is then fed to WebRTC's codec (by default VP8), which further performs the temporal compression.

6.4.3 Firmware Buffer Aware Congestion Control

POI360's Firmware Buffer Aware Congestion Control (FBCC) aims to address the challenge of irresponsive rate adaptation, following the solution principle in §6.3. In what follow, we describe the details and rationales behind our design choices.

Panoramic video encoding bitrate control

The core principle behind FBCC's video bitrate control is to let the sender promptly detect the overuse of cellular uplink bandwidth based on its local cellular firmware buffer occupancy, instead of waiting for the remote peer's end-to-end congestion measurement feedback. Upon detecting bandwidth overuse, the panoramic video sender immediately reduces its encoding bitrate to fit the instantaneous uplink bandwidth, which is again computed based on the PHY-layer statistics.

How to detect cellular uplink congestion? Conventional congestion control algorithms, like the GCC used in WebRTC, treat the network as a blackbox and rely on the end-to-end network delay/loss metrics to detect congestion. As a result, if a congestion happens, it takes at least one RTT for the sender of conventional rate

control to detect it. In effect, due to the large buffers at the LTE BS [50], those metrics become extremely irresponsive, taking up to seconds to react to a congestion.

In contrast, FBCC is customized for interactive panoramic video services running over cellular networks, exploiting the fact that the last-hop cellular path typically bottlenecks the network path. FBCC is essentially a network-assisted rate control algorithm, which leverages the uplink firmware buffer occupancy to enable quick detection of potential congestion. Specifically, let $J \in \{0, 1\}$ be the congestion indicator with $J = 1$ showing a congestion. $B(t)$ denotes the firmware buffer occupancy at time t , and Δt denotes the report interval of firmware buffer occupancy from the phone's chipset, FBCC estimates a bandwidth overuse and potential congestion based on the following conditions:

$$J = \begin{cases} 1, & \text{if } B(t - (n - 1)\Delta t) > B(t - n\Delta t) \\ & \text{for } n = 1, \dots, K \text{ and } B(t) > \Gamma(t) \\ 0, & \text{otherwise} \end{cases} \quad (6.3)$$

Essentially, the FBCC estimates a congestion on detecting both of the following events: (i) A continuous increases of firmware buffer occupancy $B(t)$ for K consecutive reports from the chipset. To guarantee the responsiveness, POI360 uses a small $K = 10$. (ii) Current firmware buffer occupancy $B(t)$ exceeds a threshold $\Gamma(t)$, which is set to the long-term average buffer level and keeps being updated online.

How to throttle the rate on detecting a cellular uplink congestion? After detecting the uplink bandwidth overuse, the rate control algorithm should reduce the panoramic video encoding bitrate to avoid potential congestion. While classical congestion-control solutions throttle the rate following empirical curves, POI360 can precisely cut the encoding bitrate to fit the instantaneous cellular uplink bandwidth. In particular, the sum of the TBS (P_{tbs}^w) for all W LTE subframes (with 1ms length) in a time window divided by the length of the time window W is the *current* PHY-layer LTE uplink throughput:

$$R_{phy} = \frac{\sum_{w=1}^W P_{tbs}^w}{W} \quad (6.4)$$

When an uplink bandwidth overuse is detected ($J = 1$), the throughput R_{phy} on this saturated uplink channel is exactly the available uplink bandwidth R_{bw} :

$$R_{bw} \begin{cases} = R_{phy}, & \text{if } J = 1 \\ > R_{phy}, & \text{otherwise} \end{cases} \quad (6.5)$$

As a result, after detecting a uplink bandwidth overuse, FBCC directly sets the encoding bitrate to T_u based on Eq. (6.4). Note that POI360 resorts to T_u only under congestion, because T_u represents the current rate that the client can *exploit*, but cannot indicate the potential rate it can *explore*.

Handling congestion happening elsewhere. In certain rare cases, congestion can happen elsewhere along the end-to-end path, *e.g.*, due to the unpredictable heavy cross traffic. POI360 detects and handles such cases by degrading to the legacy WebRTC [98] end-to-end congestion control, Google Congestion Control (GCC). More specifically, let R_{gcc} denote the bitrate selected by the legacy GCC algorithm, and t^* denote the moment when a cellular uplink congestion is detected, POI360 combines both the cellular uplink congestion control and legacy end-to-end congestion control for encoding bitrate adaptation as follows:

$$R_v(t) = \begin{cases} \frac{\sum_{w=1}^W P_{tbs}^w}{W}, & \text{if } t^* \leq t \leq t^* + 2RTT \\ R_{gcc}(t), & \text{otherwise} \end{cases} \quad (6.6)$$

As shown in the above equation, (i) When the cellular uplink becomes the bottleneck ($J = 1$ detected at $t = t^*$), POI360 immediately reduces the encoding bitrate R_v to the cellular uplink bandwidth R_{bw} computed by Eq. (6.5). In this way, POI360 is more responsive than GCC as its congestion control does not wait for the reduced R_{gcc} feedback from the remote peer. Furthermore, our solution is also more accurate as it directly converges to the cellular uplink bandwidth, instead of sharply reducing the rate and probing the bandwidth again. (ii) When $J = 0$, the congestion happens elsewhere, or there is no congestion at all. By setting the encoding bitrate R_v to R_{gcc} , the selected bitrate from the WebRTC client, POI360 reuses the legacy WebRTC

congestion control logic to handle network congestion elsewhere. Similarly, if there is no congestion, cellular uplink congestion control will not be triggered, so we still have $R_v = R_{gcc}$

Note that after a bandwidth overuse occurs at the cellular uplink ($t^* < t \leq t^* + RTT$), legacy GCC at the client side will also detect a bandwidth overuse and then notifies the sender to throttle the rate by sharply reducing the R_{gcc} to feedback, which arrives the sender one RTT after our algorithm reduces the rate to R_{bw} . In other words, the different responsiveness of sender-based and client-based algorithms may causes two consecutive rate reductions on a single bandwidth overuse event. To avoid this problem, as in Eq. (6.6), our design keeps forcing the panoramic video encoding bitrate to the PHY-layer uplink bandwidth R_{bw} within a duration of 2 RTTs (in cases the RTT itself changes) after an uplink bandwidth overuse is detected.

We also emphasize that FBCC reduces the 360° video encoding bitrate, rather than the transport-layer bitrate, when bandwidth overuse occurs. As shown in Fig. 6.10, if we throttle the transport-layer bitrate on an uplink bandwidth overuse, it simply changes the queuing location from the firmware packet buffer to the application-layer packet buffer, but the total queue length remains unchanged. The only ways to clear the queues are to *throttle the source traffic rate* and *increase the physical-layer service rate*. The aforementioned encoding rate control essentially throttles the source traffic rate. In what follows, we introduce how FBCC boosts the physical-layer service rate by properly adapting the transport-layer bitrate.

Cellular-Link-Informed RTP Rate Control

As discussed in §6.3.3, conventional rate control may unnecessarily leave the cellular firmware buffer empty due to the poor interaction between the video bitrate control and transport rate control. FBCC addresses this issue by pushing the local firmware buffer level to the “sweet spot” that maximizes bandwidth utilization without causing congestion. Since it cannot directly change the firmware buffer level, FBCC indirectly controls the buffer level by controlling the buffer’s input – the RTP sending

bitrate. The algorithm works with a time granularity of D_p , which is the feedback interval of the buffer level from the chipset (40ms in our test device).

At the beginning of each time epoch with duration D_p , the RTP rate controller monitors the occupancy of firmware buffer $B(t)$. When it drops below the target buffer level B_{phy}^* corresponding to the optimal cellular uplink rate, it updates its current RTP sending rate $R_{rtp}(t)$ following:

$$R_{rtp}(t) = R_{rtp}(t - D_p) + (B_{phy}^* - B(t))/D_p \quad (6.7)$$

The physical meaning of Eq. (6.7) is to increase the RTP bitrate so that the buffer level can reach the target level when the next time epoch starts. The optimal firmware buffer level B^* depends on the hardware of the phone, which can be learnt from previous transmissions. Since it is a intrinsic parameter to the low layer, we only need a one-time calibration of B_{phy}^* using arbitrary data transmission.

It is worth noting that POI360's RTP rate control does not require the cellular uplink to be the network bottleneck. Regardless of the bottleneck location, it is always better for the sender to deliver the uplink traffic load to the BS as fast as is allowed currently. We also note that this design does not affect the fairness to other cellular users, because the LTE BS itself executes fair scheduling.

6.5 POI360 Implementation

Browser-based adaptive spatial compression. We implement POI360's spatial compression algorithm using HTML5. Specifically, every raw 360° video frame is divided to small tiles by writing each tile region to a separate HTML5 canvas. The size of each tile is compressed based on current compression mode $\mathcal{F}$. The compressed tiles are then stitched together in one HTML5 canvas, in which the sender also embeds its current compression mode and its knowledge of the viewer's ROI. Finally, WebRTC captures and encodes (the default encoder is VP8) the outgoing video stream from this canvas. After receiving the compressed video stream, the client first infers the sender's compression matrix using the ROI and compression

mode information embedded inside each received video frame and then unfolds this compressed frame accordingly. Then the client uses WebGL to render the unfolded video frame as stereoscopic view for left and right eyes, which can be viewed with stereoscope VR headsets like Google cardboard.

Since the 360° video stream is captured from real world, rather than generated based on a real-time 3D engine, and the ROI-compressed video frame dimension become comparable to that of conventional video, the computational overhead and latency of 360° video encoding and rendering for POI360 are comparable to WebRTC-based conventional video telephony like Google Hangout.

Besides rendering the stream, the client also keeps updating current ROI based on the user input like the motion sensor reading on the phone, and periodically feeds back current user ROI to the sender via WebRTC data channel. The feedback interval is set to the video frame interval. Meanwhile, the client also keeps monitoring the responsiveness (M discussed in §6.4.2) of the end-to-end ROI update, which is also fed back via the WebRTC data channel. Based on M , the sender then adapts its compression mode following §6.4.2.

Firmware Buffer Aware Congestion Control. POI360's cellular-link-informed rate control obtains the physical-layer statistics from the diagnostic interface, which is available on most commodity phones. So it requires no hardware modifications and is readily deployable. Following the implementation of MobileInsight[72], the POI360 prototype uses our customized real-time log decoder to read the phone's diagnostic interface and obtains the LTE uplink TBS and the uplink firmware buffer level for every 40ms. These cellular physical layer statics are then written to shared memory which is also accessed by POI360's FBCC algorithm. We integrate FBCC into the WebRTC module in the open-source Chromium browser by modifying the GCC implementation in WebRTC source code.

360° video telephony performance measurement. (i) *End-to-end video frame delay measurement.* The core idea is to embed the sending timestamp of a video frame inside the frame itself, the receiver then extracts the sending time from the received frame and subtracts it from local time to compute the delay. The sender and receiver are synchronized using NTP. Each decimal bit of the sending timestamp

(with millisecond resolution) is encoded using a colored square block, with the number from 0 to 9 mapping to 10 colors with uniform separation in the RGB code space. These blocks are then appended to the side of the outgoing video frame. At the client side, the RGB values of pixels within each colored block region of the received frame are averaged and then mapped back to the decimal number.

(ii) *360° video quality measurement*. Video quality measurement requires comparison between the original and the received video frame. However, for 360° video, the users only care about the quality within ROI, as other regions are not rendered. Therefore our measurement system matches sender-side and client-side frames in both time and space domain to facilitate 360° video quality measurement. In particular, the client only dumps the ROI of its received frame, and the corresponding ROI position (i_c^*, j_c^*) . The sender, on the other hand, may have stale ROI knowledge due to the network latency, so it has to dump the entire frame. Finally, in the ROI comparison, we crop the ROI region from the original frame based on the client's corresponding ROI position (i_c^*, j_c^*) .

6.6 POI360 Evaluation

In this section, we first use micro-benchmark experiments (§6.6.1) to validate POI360's design components, including the adaptive spatial compression (§6.6.1) and the FBCC (§6.6.1) rate control. Then we conduct extensive system level tests for our real-time POI360 prototype under various network conditions (§6.6.2). Our experiments run on a commercial LTE network using LG Nexus 5 D820 smartphones. To obtain representative user-controlled ROI rotation during a 360° video telephony session, the ROI is controlled by 5 individual users. Since the users' ROI updates behavior can have large correlation with the video content, we use different 360° video for each of the individual user to avoid overfitting. To guarantee repeatable 360° video traffic load for experiments, we use virtual webcam[134] to live stream the same 360° video with 4K resolution when repeating the experiments for a user, and POI360 splits every frame to 12×8 tiles for compression. During each experiment, we run the 360° video telephony session for 5 minutes and repeat for

10 times.

6.6.1 Micro-benchmarks

Adaptive Spatial Compression

We first verify POI360's dynamic spatial compression. To highlight the effectiveness of POI360's design choices customized for cellular networks, the same set of experiments are also repeated over a wireline network on a university campus. Specifically, we run a POI360 session with two laptops, and the Internet access of both laptops are provided by either wireline connection or LTE. To isolate the impact of network protocols, we use GCC, WebRTC's default rate control (§6.2), as the transport layer.

We compare POI360 with two benchmark panoramic video compression schemes including Conduit[135] and Pyramid encoding[136]. Conduit crops the ROI region from the panorama video frame and only streams the cropped parts. To avoid displaying blank frame, we still send the non-ROI regions for Conduit but with the lowest possible quality. Pyramid encoding centers the panorama video frame at the ROI, and then allocates the center region the highest quality and aggressively compresses the contents in the 4 corners based on their distance to the center. Based on our system mode (§6.4.1), Conduit can be considered as a very aggressive compression mode with a sharp quality distribution curve, whereas Pyramid encoding adopts a conservative compression scheme with smooth quality distribution, so it can better handle the ROI shifting at the cost of higher traffic load. However, both algorithms are incapable of dynamically adapting the compression modes based on the network conditions. Besides, motion-prediction based compression[122] does not work in our application scenario (§6.7), thus is not compared here.

User-perceived video quality in ROI. We first evaluate the user-perceived ROI quality based on the aforementioned setup. First, Fig. 6.12a and 6.12b plot the average PSNR within the ROI region over both cellular and wireline networks, with error bars showing the standard deviation of PSNR value. We can observe that POI360's compression algorithm achieves the highest PSNR within the ROI

MOS Video Quality	PSNR Range (dB)
Excellent	> 37
Good	$31 \sim 37$
Fair	$25 \sim 31$
Poor	$20 \sim 25$
Bad	< 20

Table 6.1: Mapping PSNR to Mean Opinion Score (MOS).

region. In wireline networks, all the algorithms perform reasonably well. However, Conduit and Pyramid suffer from significant quality loss when running over cellular networks, with 11 to 13 dB lower PSNR than POI360. Such disparate performance comes from POI360's capability to adapt the compression modes based on the network latency that affects the end-to-end ROI update responsiveness. POI360 can effectively switch to the proper spatial compression mode that best fits the current network condition. In contrast, when a viewer changes ROI in Conduit, the remote peer cannot update its knowledge of the viewer's ROI in time. So the viewer can only see compressed regions with low quality. Pyramid shows slightly higher PSNR than Conduit, as it has a smoother spatial quality distribution, and hence less quality drop during the ROI shift.

To understand the viewer's subjective experience, in Fig. 6.12c and 6.12d, we further plot the PDF of the Mean Opinion Score (MOS) computed based on the frame-level PSNR, following the well-known relation between the MOS and PSNR[137] summarized in Table 6.1. We see that POI360 achieves the best MOS over both cellular and wireline networks. In particular, over cellular networks, 52% of its frames have good quality while 4% have excellent quality. Conduit does not provide any frame with good or excellent quality, and Pyramid only has 7% frames with good quality. Even in wireline networks, POI360 has more than half of the frames

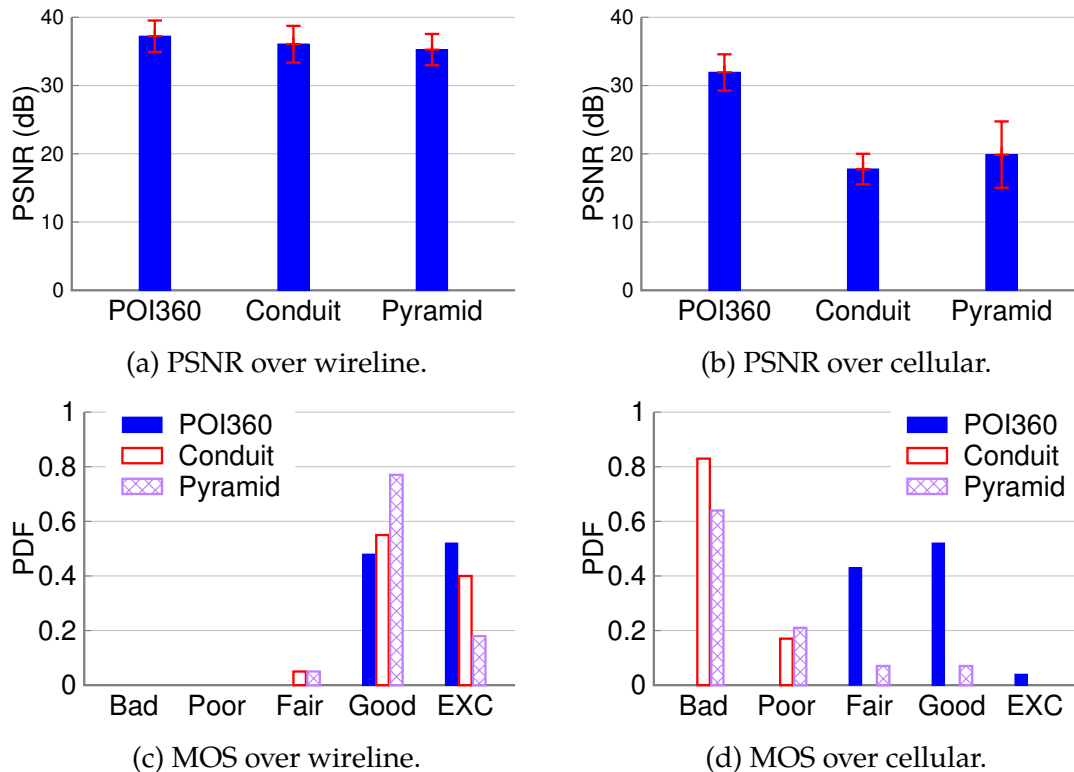


Figure 6.12: Comparing video quality under POI360 compression and benchmark algorithms.

with excellent quality and the rest of the frames with good quality, substantially outperforming both Conduit and Pyramid.

We emphasize that POI360's performance gain mainly comes from its principle of *adaptive* compression. The compression itself is taking effect just like other schemes. For example, given a 360° video with 12.65Mbps raw bitrate, the received stream bitrate at the POI360 client is only around 3Mbps, reduced by 76% (more in §6.6.1 and Fig. 6.17a).

Video quality stability in ROI. Besides video quality itself, the short-term stability of video quality is also vital for user experience[33]. We characterize this *stability* by the standard deviation of the ROI compression level in a 2 second sliding window. The CDF plot in Fig. 6.13a shows that all the algorithms have small quality variation in wireline networks with relatively stable network condition and

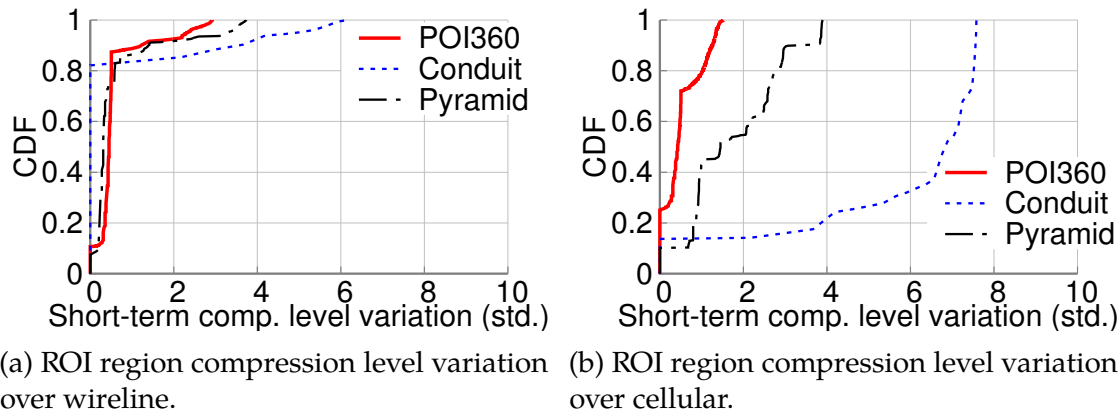


Figure 6.13: Short-term ROI compression level variation (std. value of ROI compression level in a 2s window).

small RTTs. In contrast, in cellular networks (Fig. 6.13b), Conduit and Pyramid show poor stability, with $5\times$ and $14\times$ higher *std.* than POI360. This is because the large and unstable RTT of cellular networks makes the end-to-end ROI update laggy, causing the fluctuating compression level in the actual ROI. The problem is especially severe for Conduit, as it only has 2 compression levels, thus ROI shifting triggers unacceptable video quality oscillation between the high/low levels. Using more compression levels, Pyramid smooths out the transition, thus achieving better stability than Conduit. However, its rigid spatial compression mode still cannot fit all conditions in the highly dynamic cellular networks. In contrast, POI360 achieves the best stability as it can always adapt to the compression mode best fitting current network condition.

360° video frame delay. We now analyze the frame-level delay performance collected from the same set of experiments. Fig. 6.14a plots the CDF of frame delay over the wireline network. We can observe that POI360's compression algorithm achieves the lowest video delay. Furthermore, in cellular networks (Fig. 6.14b), POI360 achieves a median delay of 460ms, 15% less than Conduit which aggressive compression, because its adaptive design can switch to more aggressive compression modes than Conduit under bad network condition, or prioritize the video quality when the network allows. In other words, POI360's better ROI quality and stability

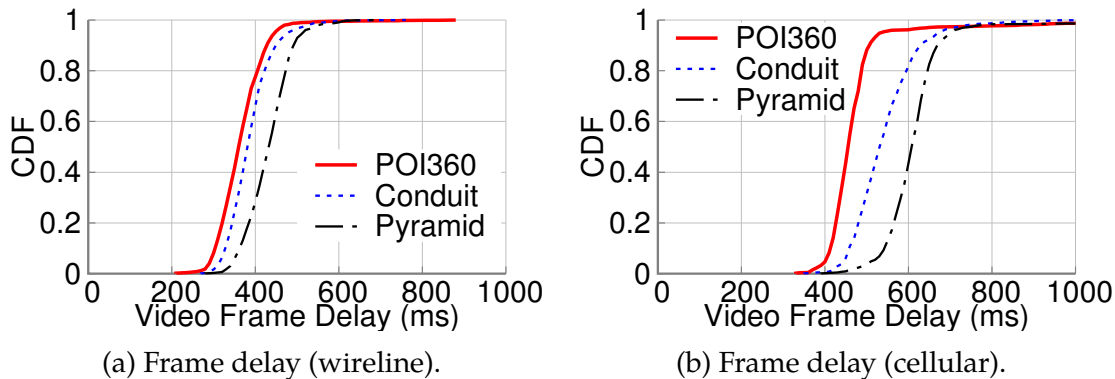


Figure 6.14: 360° video frame delay under POI360 compression and benchmark algorithms.

come without sacrificing the crucial delay performance. This delay is also comparable to the conventional video telephony delay over cellular networks measured in existing works[7]. We emphasize that end-to-end frame delay is not frame interval, as shown in Fig. 6.2, *e.g.*, a video stream can be delayed by 460ms while keeping a 36FPS frame rate. Pyramid encoding has higher video delay than Conduit, as it compresses the non-ROI regions less aggressively. In contrast, POI360 strikes a balance between video quality and delay: When the end-to-end ROI update is responsive, it can aggressively crop and stream only the high-quality ROI like Conduit; When the ROI update becomes sluggish, it can distribute the video quality more evenly across the frame like Pyramid encoding.

Video frame freezing ratio. Based on the video frame delay, we then compute the frame freezing ratio, the most crucial user experience metric. We define *freezing ratio* (F_R) as the percentage of video frames that experience higher than 600ms delay. Fig. 6.15a shows that all algorithms work properly in wireline networks, with F_R below 2%, and POI360 has the lowest F_R of 0.6%. However, in cellular networks (Fig. 6.15b), both Conduit and Pyramid encoding fail with an unacceptable F_R of 8% to 17%. In contrast, POI360's adaptive compression tames the F_R below 3% under the network dynamics.

In sum, *POI360's adaptive compression substantially outperforms the benchmark algorithms in terms of video quality, stability and delay, especially in cellular networks with*

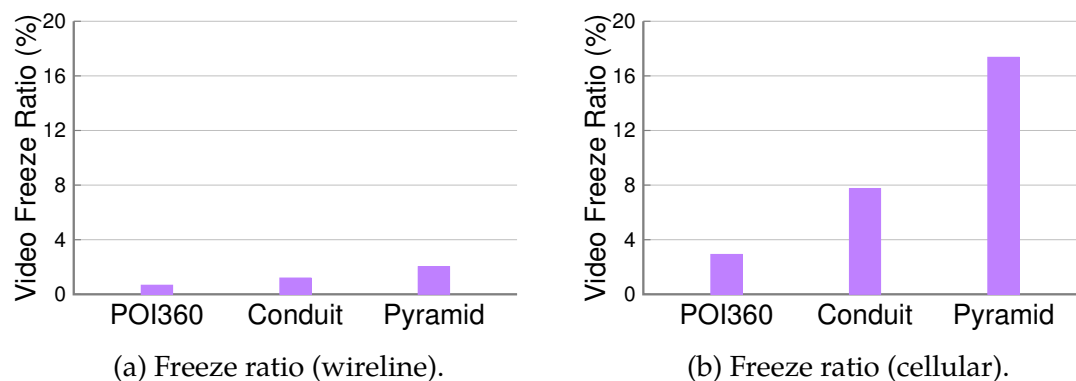


Figure 6.15: 360° video freeze ratio under POI360 compression and benchmark algorithms.

long RTT and highly dynamic network conditions. However, there is still a performance gap between cellular and wireline, especially in terms of freezing ratio. In what follows, we show how POI360's cross-layer rate control helps bridge this gap.

Firmware Buffer Aware Congestion Control

Recall that POI360's FBCC module aims to improve the cellular uplink bandwidth utilization by pushing the firmware buffer occupancy to a proper level. To verify its effectiveness, we repeat the panoramic video telephony sessions with each lasting for 200 seconds for both POI360 over FBCC and POI360 over GCC—the default WebRTC rate control used in the foregoing experiments. Both configurations share the same application-layer dynamic panoramic video compression module in §6.6.1. The scatter plot in Fig. 6.16 shows the firmware buffer level and the per-second uplink TBS (throughput) collected across the panoramic video telephony sessions. We split this figure into 3 regions: (i) *the low usage region* with uplink throughput below 2 Mbps; (ii) *the high usage region* where the uplink throughput exceeds 2Mbps and keeps increasing with the buffer level; (iii) *the overuse region* where the uplink throughput no longer increases with the firmware buffer level as the uplink is saturated. It is clear that FBCC maintains the buffer level around the “sweet spot” in the high usage region that is far from congestion but high enough to secure sufficient bandwidth utilization. In contrast, GCC's buffer occupancy stays at the

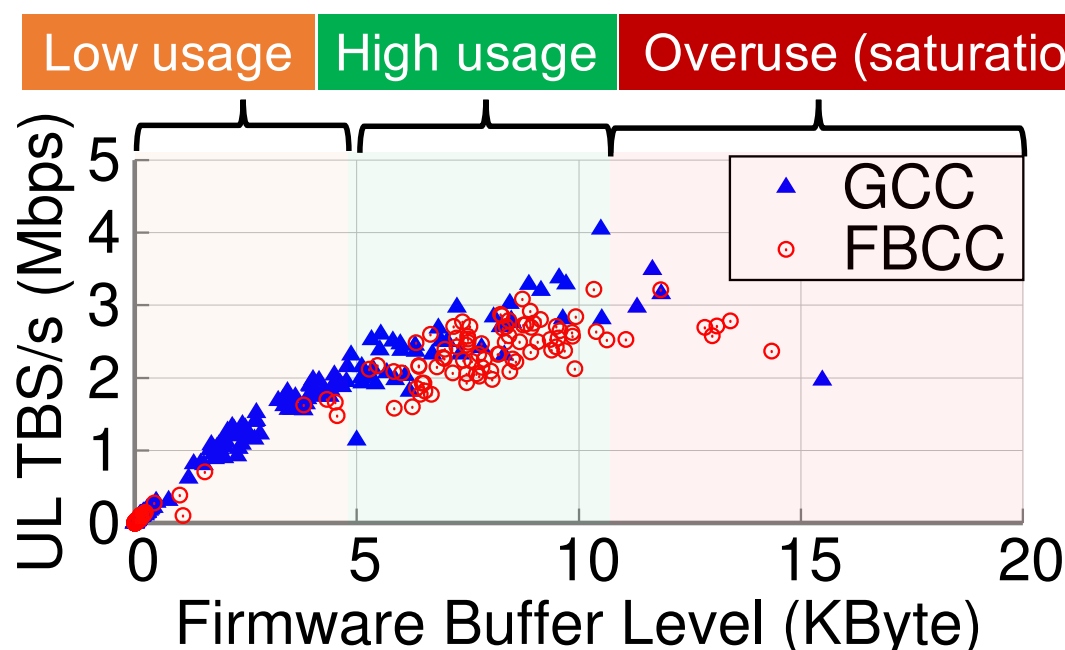


Figure 6.16: FBCC is allocated higher uplink TBS than GCC as it maintains a proper firmware buffer level.

low usage region for a substantial fraction of the samples, indicating that it severely underutilizes the network bandwidth.

We proceed to evaluate the impact of the transport layer rate control on the performance of panoramic video telephony. Following the same experimental setup discussed at the beginning of this microbenchmark, we compare POI360 running over FBCC against POI360 running over legacy GCC. From Fig. 6.17a, we can observe that the throughput of both configurations are almost identical at around 3Mbps, as they share the same compression algorithm which compresses the panoramic video stream bitrate to such a level below the current cellular uplink bandwidth.

However, the throughput variation differs a lot. Specifically, GCC's throughput has a 1.2Mbps standard deviation, 57% higher than FBCC. This unstable throughput is mainly caused by its inefficient adaptation logic, which explores the network bandwidth by gradually increasing the sending rate, and reacts to the bandwidth overuse by sharply throttling the traffic bitrate. In contrast, FBCC converges to the

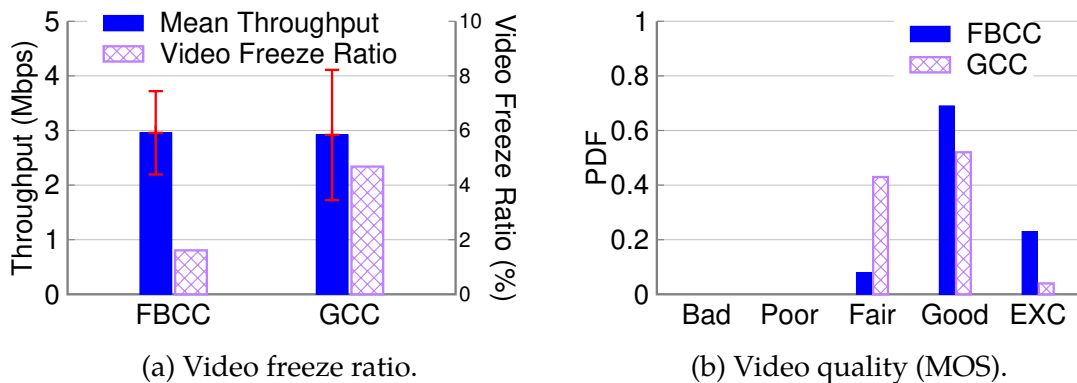


Figure 6.17: Comparing POI360 performance with FBCC or GCC (WebRTC’s default rate control).

network bandwidth quickly as it can directly estimate the cellular uplink bandwidth based on the firmware buffer level and TBS following Eq. (6.5) when the buffer status indicates a congestion, and then responsively throttles the rate to the estimated bandwidth. As a result, the extra cross-layer information helps FBCC achieve a low F_R of around 1.6% over cellular networks, which is significantly lower than the F_R of GCC (4.7%), and is even close to the wireline case (Fig. 6.15b). The stability of throughput also translates into high video quality. Fig. 6.17b shows the MOS quality computed based on the PSNR collected from the same experiment. We see that FBCC has much higher percentage of video frames with good (69%) and excellent (23%) quality than GCC. In contrast, GCC has more than 40% of the frames with only fair quality, as its throughput frequently vibrates to low levels.

6.6.2 System Level Test

We now integrate POI360’s design components and conduct a system level evaluation under various network conditions.

Different background traffic load. Since POI360 runs on cellular networks with the uplink being the typical bottleneck, we verify its resilience under different uplink traffic load within the same cell. We conduct the experiments in two campus environments: (i) *light traffic load environment* during early morning when most users are off campus; (ii) *heavy traffic load environment* in the noon just after class,

when most roam around campus with their cell phones. All experiments are performed at the same static location to rule out the influence of signal strength and mobility.

Fig. 6.18a plots the video freezing ratio F_R and PSNR. We observe that the overall F_R is only around 1% under light traffic load. Even during the busiest hour, the freezing rate is only around 4%, and PSNR drops by 2 dB. Under light traffic load, the MOS metric shows a relatively larger fraction of frames with excellent quality. But even with heavy competing traffic, majority of the frames are in either excellent or good condition, and none is poor or bad condition. Therefore, our results demonstrate that POI360 is robust to the background traffic load. Such robustness owes to its awareness of cellular link statistics like the uplink firmware buffer level, and its ability to effectively maintain an adequate uplink rate by manipulating the buffer level.

Different LTE channel quality. Since POI360 aims to realize the ubiquitous capturing and streaming of 360° video, it is important to understand the impact of the location-dependent cellular link quality on its performance. To create different channel qualities, we place our prototype at different locations and evaluate the performance under different levels of received signal strength (RSS): (i) *weak signal* inside a concrete parking garage, where the RSS is -115dB as reported by the phone's firmware. (ii) *moderate signal* at an outdoor parking lot partially shadowed by a tall building (-82dB RSS). (iii) *Strong signal* in an open parking lot without tall buildings nearby (-73dB RSS). The experiments are conducted during weekends when the cellular channel is mostly idle.

Fig. 6.18c shows the resulting PSNR and video freeze performance. We first see that the F_R is not affected much by the RSS. Even under weak channel, the F_R is below 3%. However, this comes at the cost of lower video quality (Fig. 6.18d). Under a weak channel, none of the frames show excellent quality, whereas the strong channel has 31% of the frames with excellent quality. The different trends for video freeze and video quality can be explained by the stability of RSS. Although the channel can be weak which forces POI360 to use low video quality, as long as the RSS does not fluctuate, POI360's rate control can always converge to the

network bandwidth and stably stream the video without freezing.

Different mobility level. One potential application of POI360 is to enable 3D experience sharing through drones or inside moving vehicles. Different mobility levels in turn affects the LTE network condition. To evaluate the impact, we place our experimental platform inside a moving vehicle with 3 different speed configurations: (i) 15 mph, a typical slow driving speed within the residential area; (ii) 30 mph, driving along a road in an urban area; (iii) 50 mph, driving along a highway. We repeat each test 10 times along the same route. During the experiments, the phone is under seamless LTE coverage.

Fig. 6.18e shows the performance in terms of F_R . We see that slow driving has almost no impact on F_R compared with the static experiments. But F_R increases to 7% under city driving speed and 9% on the highway. Fig. 6.18f further plots the MOS video quality computed based on the PSNR, we can observe that although the video quality decreases under higher mobility level, even under the speed of 50 mph, all frames still have either excellent (20%) or good (80%) quality, which mainly owes to the less building blockage and thus higher RSS (typically -60dB) around the highway test route.

6.7 Discussion

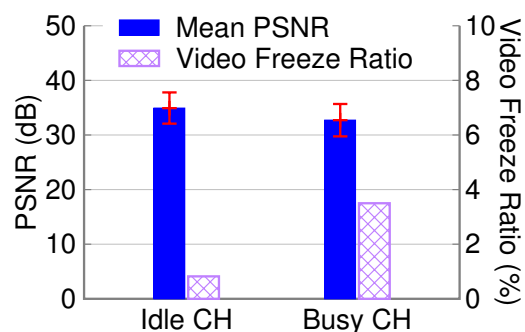
Why motion-based ROI prediction cannot solve the problem? (i) For rendering 360° frames locally, motion-based ROI prediction [130, 131] helps smooth the display quality. However, such prediction only works reliably at a short time scale. In typical use, the average head angular velocity is 60 deg/sec while acceleration can go up to 500 deg/sec^2 as reported by Oculus VR[130], *i.e.*, we cannot predict where the head will stop at if the horizon is more than 120 ms later, which is below the typical end-to-end frame latency of cellular video. (ii) For on-demand 360° video multicast, recent work[138] suggests using the motion patterns learnt from existing users to predict the ROI of current user, as users perform similar motion patterns given the same video content. However, 360° video telephony is different from on-demand

streaming, as the content is generated in real time. Therefore there are no such existing contents to learn from.

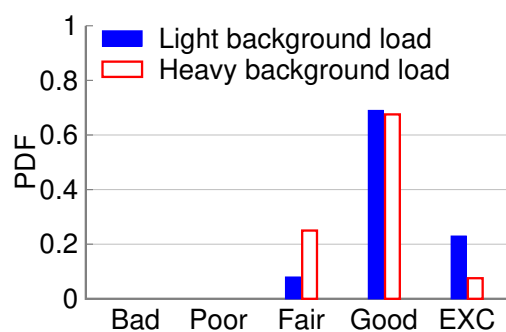
Improving the ROI update responsiveness. Although POI360 smooths out the ROI quality transition by adaptive compression, it still takes the cellular RTT before the quality of new ROI converges. The key to reduce this convergence time is shortening the end-to-end network path. In current 4G LTE network, traffic still goes to the Internet even both ends of the video telephony connect to the same BS. In future works, mobile edge computing can be used to enable the relaying at the edge BS, thus significantly shortens the path and accelerate the quality convergence of POI360.

6.8 Summary

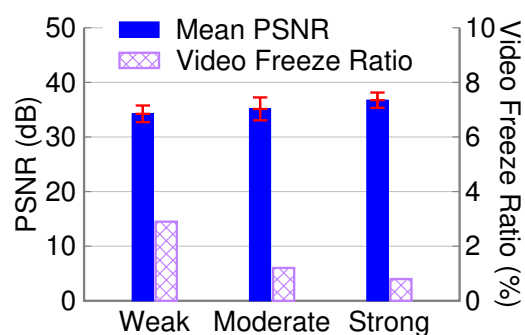
Although on-demand 360° video streaming for recorded contents has been studied extensively, real-time interactive 360° video telephony remains mostly unexplored. In this dissertation, we investigate the feasibility of realizing interactive 360° video telephony services over LTE cellular networks, and present our readily-deployable solution POI360, which employs adaptive compression to balance the traffic load reduction and user-perceived video quality, and cellular-link-informed rate control design to boost the responsiveness to potential congestion. Finally, a real-time prototype of POI360 is implemented, which validates our design in various network conditions. We believe this work can inspire new applications that bring the conventional interactive video services to the 360° world.



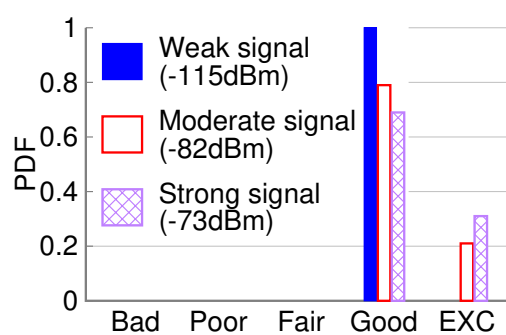
(a) Freeze ratio & PSNR with different background load.



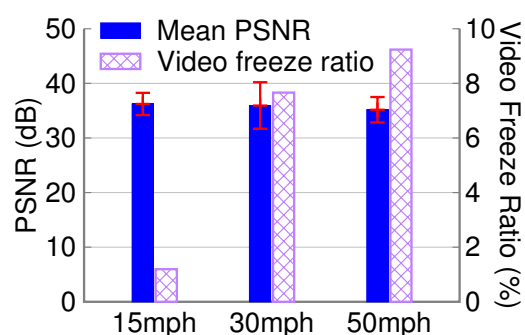
(b) Video quality (MOS) with different background load.



(c) Freeze ratio & PSNR with different signal strength.



(d) Video quality (MOS) with different signal strength.



(e) Freeze ratio & PSNR at different driving speed.



(f) Video quality (MOS) at different driving speed.

Figure 6.18: System-level POI360 evaluation.

7 RELATED WORK

This chapter provides a brief survey on the prior research efforts related to this dissertation from various perspective. First, we survey the numerous related works on transport-layer rate control for wireless networks. Then we introduce the existing measurement works that profile the LTE network performance and investigate the reasons of the observed LTE performance issues. Next, we elaborate prior solutions for on-demand video streaming over cellular networks. We then investigate existing works on improving the mobile Web performance. After that, we discuss about both the commercial solutions and research efforts for interactive video chat. We close this chapter by discussing researches in the hot area of 360° panoramic video streaming.

7.1 Transport Layer Rate Control over Wireless

7.1.1 Split-Connection Protocols

Dating back to 1990s, radio links are highly unreliable in the premature stage of wireless networks, and split-connection protocols [9, 139, 140] were proposed to improve the data transport over the lossy wireless link. The basic idea is to break the end-to-end connection into two separated parts at the wireless BS, thus shield the remote sender from wireless losses. More specifically, the intermediate sender at the BS can handle all retransmissions for the packet losses on the wireless link, while such losses are masked from the original TCP sender to prevent the unnecessary congestion control triggered by packet losses. However, existing measurement works [10] have revealed the poor performance of split-connection protocols in practice. Such a design principle can also severely slow down the handoff procedure when the user switches between different BSs. Furthermore, in modern wireless networks like LTE, the large BS buffer, efficient coding and ARQ-based retransmission in the RLC layer greatly boost the reliability of the radio link. Now wireless related losses are rare and bufferbloat caused by the lack

of packet loss becomes the new problem. In other words, the challenge that the split-connection protocols attempt to solve has mostly depreciated in 4G cellular networks.

7.1.2 Link-Layer Solutions

Instead of splitting the end-to-end transport-layer connection, the link-layer solutions [10, 11, 12] handle the wireless losses based on the retransmissions in the link layer. The basic idea of such works has been inherited in the 4G LTE cellular network, which incorporates the Radio Link Control (RLC) layer to handle the local retransmissions over the lossy wireless channel [13]. However, even with the help of the RLC layer, mobile services still suffer from poor performance, as the root cause of the performance issues is not the wireless link related loss.

7.1.3 Cross-Layer Solutions

A vast body of research has touched upon cross-layer protocols for wireless networks [15, 141, 142], but most of them fail to pinpoint the actual problems that cause the unsatisfactory mobile service performance. For instance, more than 30 years ago, explicit loss notification or ELN [14] was proposed to enable the sender to distinguish between congestion related loss and wireless channel related loss based on the PHY-layer information. However, in modern cellular networks, the large buffer and efficient retransmission shield the packet losses over wireless links from upper layers. Therefore it is no longer the fundamental limitation that bottlenecks the performance of modern mobile services.

Only recently, the practical cellular-specific problems are revealed through large-scale measurements over real-world cellular networks [27, 43, 53, 88]. More specifically, the highly dynamic cellular link bandwidth and delay, instead of wireless link losses, are the fundamental challenges for the data-hungry and delay-sensitive mobile services like video streaming, interactive video and VR. Recent work CQIC [16] represents the first practical cross-layer rate control solution over cellular networks, which builds a transport protocol atop UDP and adapts data transport

rate by estimating link capacity in HSPA+ cellular networks based on PHY-layer channel quality information (CQI). However, though bearing the cross-layer design principle, CQIC is only “wireless-channel-informed” rather than “physical-layer-informed”. In other words, it is only aware of the physical-layer wireless channel quality CQI. But for LTE physical layer where resources are shared among UEs, CQI alone cannot characterize the available network bandwidth – the resource allocation information is equally important. Without such knowledge, CQIC suffers from a fundamental limitation that it cannot identify whether the resources unallocated to itself are idle or occupied by other users. Thus, a CQIC client is unable to perform bandwidth exploration – it essentially estimates the bitrate supported by the fraction of resources it is *currently allocated*, but remains blind to the idle resources that it can further *explore*, which causes an over-conservative bandwidth estimation.

Loadsense [60] suggests using physical-layer metric RSRQ to estimate the cellular channel as idle or busy, and sneak traffic into the idle period to improve energy efficiency. However, it simply employs an empirical Support Vector Machine (SVM) to classify the “idle” and “busy” channel states, treating the underlying relation between cell load and RSRQ as a black box. In contrast, by digging deeply into the LTE physical-layer resource structure, our work proposes the first analytical model to characterize the quantitative relation between RSRQ and the load on the cellular link, thus can perform much more precise adaptation over LTE networks.

Cross-layer rate control typically assumes the last-mile cellular link bottlenecks the end-to-end connection. However, in some corner cases, the bottleneck may still appear in the wireline path. State-of-the-art bottleneck identification schemes like Qprobe [143] require massive dedicated probing packets, which makes them impractical for short-flow applications as the probing traffic itself overshadows data transmissions. In contrast, our bottleneck detector design presented in §4.4.3 is facilitated by the awareness of cellular network load. It bears zero overhead, can run in parallel with Web data transfer and discriminate between wireline and cellular bottlenecks in real time.

7.1.4 Network Bandwidth Estimation

Thanks to the advanced physical-layer radio communication technologies like MIMO and OFDM, 4G LTE cellular network achieves a high link bandwidth. But meanwhile the complicated cellular network architecture also brings an unavoidable high latency. As a result, the LTE network features a high bandwidth-delay-product (BDP) and thus can be categorized as Long Fat Network (LFN). Unfortunately, it is well-known that conventional TCP does not perform well over LFN [139], as the high latency slows down the convergence of TCP bandwidth exploration to the high network bandwidth. Meanwhile, the cellular link bandwidth is highly dynamic due to the physical channel variations. But conventional TCP protocol like TCP Reno or Vegas uses the aftereffects of congestion, like packet loss or delay, as the indicator of congestion, thus their rate control always falls behind the bandwidth variation. As a result, estimating the end-to-end network bandwidth is crucial for mobile service optimization.

The research on network bandwidth estimation starts even before the wide spread of wireless networks, and one of the standard approaches is to use packet pairs to probe the bottleneck link bandwidth [19]. Since wireless link is typically the bottleneck of an end-to-end connection, this approach can also be used for bandwidth estimation for wireless users. Besides, network bandwidth estimation has also been adopted in some of the existing TCP variants – TCP Westwood [18] estimates the end-to-end network bandwidth by monitoring the rate of returning ACKs, thus does not rely on the packet losses for network bandwidth exploration. However, over LTE cellular networks, existing bandwidth estimation mechanisms fail due to the unawareness of cellular physical layer rate control. More specifically, LTE BS allocates physical-layer spectrum resources (and hence link bitrate) to the mobile users based on current end-to-end traffic load of the connections, which is adapted by the end-to-end rate control protocols like TCP running in the upper layer. However, the bandwidth estimation of these end-to-end protocols in turn depends on the cellular resource allocation. For example, when the traffic load to a certain user is low, the BS will not allocate all idle resources to that user, and then

the end-to-end protocol will estimate a low network bandwidth. In other words, the end-to-end traffic load is bounded by the network bandwidth supported by the *currently allocated* physical-layer resources, rather than *all available* resources including the idle resources. In this way, the nested end-to-end rate control & physical-layer rate control essentially form a vicious cycle – low end-to-end traffic rate and low physical-layer rate can enforce each other, thus cause severe network bandwidth underutilization.

In recent years, the new challenges in 4G LTE cellular networks like buffer bloat [56] triggered a revisit of the transport protocol with end-to-end network bandwidth estimation. In particular, some of the works build empirical bandwidth estimation models to facilitate transport-layer rate control. Sprout [20] uses a stochastic model estimated from throughput probing to forecast network bandwidth. Verus [21] estimates the bandwidth following a delay-to-bandwidth mapping learned from a training phase. PCC [144] directly tests a wide range of sending rates and then picks the rate with the best measured performance.

Although these model-driven protocols showed high throughput for bulk data transfer, they are unsuitable for the mobile services discussed in this dissertation due to the following reasons:

(i) model-based solutions require a long training phase to build the empirical model, which conflicts with the mobile services of short time span (Web loading) or requiring good responsiveness (interactive video chat); It may also easily drain out the typical cellular user's limited data quota.

(ii) Due to the highly dynamic LTE environment, an empirical model will soon become non-reusable. For similar reasons, we cannot use pre-defined models. Frequently rerun the training phase to update the model can severely degrade the performance of mobile services;

(iii) Building the empirical performance model requires the sender to inject saturated traffic into the network path, thus most of these works assume saturated traffic flows. Such an assumption cannot hold for mobile Web loading with typically page size smaller than 2 to 3 MBytes[8], and may directly jam the video for interactive video chat.

7.1.5 TCP Friendliness

Traffic flows controlled by the TCP protocol, especially TCP Reno and Cubic, dominate the Internet. As a result, when introducing a novel rate control design, TCP friendliness would be a major concern. For instance, TCP Vegas [83] achieves much better performance than TCP Reno when running alone, but it never gets popular over the Internet, as its delay-based design tends to conservatively throttle its own rate when coexisting with the wide-spread TCP Reno and Cubic flows. In this way, the TCP Vegas flows always suffer from poor performance as it tends to yield to existing Reno and Cubic flows. TFRC [104] adopts a throughput equation to compute a sending rate which is reasonably fair when competing with TCP Reno flows as a function of the measured packet loss rate and RTT. The throughput equation is carefully designed to match the TCP Reno throughput, while the model-based adaptation guarantees a smoother sending rate change compared to conventional TCP that sharply throttles the sending rate on packet loss events.

In this dissertation, our target is not the throughput fairness. Since conventional TCP severely underutilizes the available cellular link bandwidth, if our design only achieves the same level of throughput as TCP, the bandwidth underutilization problem is not solved at all. Therefore, we have another reasonable definition for the fairness – when adding traffic flows running our protocols in the network, it should not affect the performance of existing TCP flows more than adding the same number of conventional TCP flows with the same traffic demands, but our user can still enjoy a higher throughput than conventional TCP by harnessing the idle physical resources that conventional TCP fails to utilize.

7.2 LTE Measurement Profiling

The study in [53] represents a first measurement of commercial LTE deployment, revealing architecture-level properties enabling lower-latency and higher throughput than 3G networks, as well as low-layer configurations (*e.g.*, large in-network queues) that cause high-layer anomaly. For instance, large in-network queues amplify the

impact of packet re-ordering and may trigger unnecessary timeout when link-layer retransmission occurs. Huang *et al.* [88] examined how radio state machine interacts with application traffic in LTE networks, focusing on the end-host energy wastage due to uninformed active radio state maintenance. More recent studies examined the impact of handover [145], traffic burstiness and fair scheduling policies [27] on LTE/HSPA transport-layer. All such measurement/simulation hints to the importance of cross-layer interaction in LTE networks.

7.3 Adaptive Video Streaming

Video streaming services have kept challenging mobile Internet infrastructure and protocol design. Conventional video streaming relied on propriety services, often on top of UDP [146, 147]. More recent on-demand video streaming services are converging towards the HTTP based dynamic adaptive streaming (DASH) [28], which allows repurposing commodity Web infrastructure (*e.g.*, CDN servers) for video streaming. This trend has proliferated many variants of DASH (*e.g.*, Microsoft Smooth Streaming, Apple HTTP Living Streaming). In present days, video content providers like YouTube and Netflix have widely deployed DASH on their servers

DASH-based streaming algorithms. The core of DASH is to accurately estimate the available network bandwidth, and then choose a video bitrate below it. Existing DASH algorithms strive for a balance between high video quality (bitrate) and the risk of video freezes. The key challenge lies in estimating the highly dynamic available network bandwidth over cellular links, which can only be roughly approximated in existing DASH-based works using either historical TCP throughput [33, 38], or video buffer dynamics [32]. In contrast, this dissertation has demonstrated that piStream overcomes the limitations of these approaches via cross-layer bandwidth estimation.

RAN-assisted streaming algorithms. In cellular networks, the Radio Access Network (RAN) performs RRM (Radio Resource Management), all PHY layer details are available at the BS. This fact inspires RAN-assisted video streaming solutions [148, 149], in which the video traffic scheduling algorithm runs on the BS

and controls the end-to-end traffic by sending non-LTE-standard control messages. Unfortunately, such RAN-assisted solutions require heavy modifications on the LTE BS or even the core network [149]. In contrast, this dissertation focuses on more practical and readily deployable solutions that requires no hardware modification to the cellular infrastructure.

7.4 Improving the Mobile Web Performance

Since HTTP is the dominating protocol to deliver Web contents, enhancing HTTP is a natural way to accelerate mobile Web. Google proposed SPDY [8], which optimizes the Web data transfer by allowing the client to fetch multiple outstanding objects simultaneously through a single TCP connection. SPDY then evolves into HTTP/2[64], which is considered as the next generation of HTTP protocol. However, SPDY or HTTP/2 still perform poorly over cellular networks [8] due to its poor coordination with the underlying TCP protocol and cellular physical layer. In other words, only optimizing HTTP itself does not solve the problem. Parallel to the protocol evolution, Web caching has been widely deployed through CDNs at the network edge. Browsers and intermediate DNS servers employ caching/prefetching to reduce DNS lookup latency, with up to 80% hit rate [150]. In addition, LTE cellular networks have widely adopted middle-boxes that take over the TCP establishment and termination to reduce TCP's handshake latency [43]. All these optimizations amortize the HTTP initialization latency and signifies the importance of reducing actual downloading time. Polaris [77] accelerates Web loading by using fine-grained dependency graphs to enable aggressive object fetching, which significantly reduces the web loading latency especially on large sites with complicated dependencies. However, due to the high RTT over LTE, typical mobile sites are optimized to reduce round-trip interactions by minimizing the content dependencies.

7.5 Interactive Video Chat over Cellular Networks.

Interactive video chat like Skype has existed for years, yet new challenges are emerging when it migrates to a mobile service over cellular networks.

Early video telephony services extensively explored DiffServ infrastructures that prioritize video traffic [151]. But due to the formidable cost, it is infeasible to guarantee the network capacity can outpace the unbounded growth of video traffic. Therefore, today's wireless video telephony commonly operates as a best-effort service.

Agile adaptation to network congestion status is the key theme of best-effort mobile Internet. Proprietary video telephony solutions commonly build their own rate/congestion adaptation protocols based on UDP [7]. Whereas the protocol details are unavailable, measurement studies reveal severe performance limitations, such as sluggish response to bandwidth change [152] and frequent freezing [7] especially over cellular networks. Cisco proposes a Network Assisted Dynamic Adaptation (NADA) algorithm [146], which regulates the sending rate based on explicit congestion notifications (ECN). Similar network-assisted mechanisms for cellular BS have been proposed by Intel and Skype [153], but require ECN support on intermediate routers.

Jain *et al.* [87] proposed a real-time congestion control protocol tailored for LTE, which requires an LTE BS to estimate the achievable throughput of each user, and feed such information to the video senders to guide its congestion adaptation. This requires the LTE BS, or a middle-box server close-by, to estimate the achievable throughput of each UE, and return the information directly to the server to guide its transport/application adaptation. Although controlled tests show potential, it remains unclear how to accurately estimate the achievable throughput, which (because of the BS's scheduler) is a function of channel condition and other competing users' traffic demand. In [132], Xu *et al.* enhance mobile interactive video using a machine learning model to predict cellular network performance based on historical application-layer statistics like throughput/latency. However, since application-layer itself may be trapped on a state with low bandwidth utilization

as discussed in §3.2, application-layer performance can no longer indicate the achievable throughput.

Recently, the IETF chartered the RMCAT working group [93] to standardize congestion control for real-time media. Google Congestion Control (GCC) [94] has been a leading proposal in RMCAT, and acts as the media transportation framework for mainstream browsers like Chrome and Firefox. However, GCC does not react well to wireless channel fluctuation, nor to the buffer buildup [96]. In addition, GCC takes a few tens of seconds to adapt to the initial conditions of the network, as it does not have a specific mechanism to solve the initialization problem [95].

Note that the on-demand video streaming systems discussed previously explore bandwidth aggressively to maximize video quality, and leverage several seconds' large buffer to absorb inaccurate bandwidth estimation. In contrast, with orders of magnitude tighter delay constraints, interactive video chat systems need to prioritize responsiveness, thus deserving a distinct design.

7.6 360° Panoramic Video Streaming

The past decade of research in 360° or panoramic video delivery has focused extensively on the encoding aspect [154], *i.e.*, mapping the original sphere video into planar format, and further splitting an entire frame into tiles [119, 124, 125, 155, 156]. Image compression based on ROI coding has been adopted by JPEG2000[157], well before the 360 degree videos become popular. In recent years, the concept of delivering a fraction of the tiles based on the viewer's ROI has also been investigated in related works [121] that focus on selecting the tiles intelligently to reduce the bandwidth cost, while ensuring high resolution for ROI tiles and maintaining a low probability of missing tiles, under certain bandwidth and latency constraint. To date, 360° video streaming mainly deliver pre-captured videos on demand, which can tolerate second-level buffering latency [158]. Through trace-driven simulation, Qian *et al.* [122] recently verified the feasibility of delivering tiled 360° videos through cellular networks for on-demand video streaming. Relatively less

research has examined the system-level issues in delivering 360° videos across highly dynamic wireless networks.

Driven by low-cost portable cameras, a new class of 360° video telephony applications are emerging, which can potentially support real-time immersive experience sharing. Through an omni-directional camera mounted on a drone [116, 117], headset [114], or handheld mobile device [115], real-time panoramic scenes can be captured and teleported to remote users, who can experience first-person 360° view when they wear the VR HMD. The Holoportation system from Microsoft [159] represents the state-of-the-art in streaming 360° videos for tele-presence. However, current version of Holoportation still requires 30-50 Mbps bandwidth, and only works within WiFi range rather than cellular coverage, which significantly limits its portability. Meanwhile, multiple practical problems remain open, *i.e.*, network protocol support for real-time streaming, and the critical trade-off between traffic load reduction and the stability of user-perceived video quality.

Ultimately, interactive 360° videos impose a higher bandwidth requirement on the mobile broadband than conventional video, thus call for a more agile adaptation to the highly dynamic network conditions. Proprietary video chat systems commonly build their own rate/congestion adaptation protocols based on UDP [7]. Whereas the protocol details are unavailable, measurement studies reveal severe performance limitations, such as sluggish response to bandwidth change [152] and frequent freezing [7] especially over cellular networks. Recently, the IETF chartered the RMCAT working group [93] to standardize congestion control for real-time media. Google Congestion Control (GCC) [94] has been a leading proposal in RMCAT, and acts as the media transportation framework for mainstream browsers like Chrome and Firefox. Alternative proposals [146, 153] regulate sending rate based on explicit congestion notification (ECN), but require ECN support on intermediate routers. In [132], Xu *et al.* proposed to use a machine learning model to predict cellular network performance based on historical end-to-end throughput/latency, which, however, still needs at least one RTT to detect a network change as an end-to-end solution.

8 CONCLUSION

8.1 Concluding Remarks

Despite the high speed cellular link, present-day mobile services still underperform over 4G LTE networks. The root cause lies in the poor cooperation between mobile service applications, network protocols, and cellular physical layer. In this dissertation, we propose the design principle of PHY-informed mobile service optimization to overcome the limitations of mainstream mobile services over LTE networks. Following this design principle, we first propose piStream, an adaptive video streaming protocol that leverages the physical-layer information in DASH video streaming adaptation algorithm to fully exploit the available cellular link bandwidth. We then design and implement CLAW, a cross-layer framework that accelerates mobile Web loading using cellular link information. Compared to piStream using software radio to sniff the cellular channel, CLAW adopts a novel analytic model to estimate the resource utilization status on the cellular channel, thus requires no hardware modification to commodity phones. Further, we propose pyChat, a freeze-precautionary rate control algorithm for interactive video chat, which can forecast video freeze in the near future using cellular link resource utilization as the indicator, and throttle the video bitrate correspondingly to prevent potential video freezes. Finally, we investigate the feasibility of 360° video telephony over existing LTE cellular infrastructure, and propose POI360, which employs a novel spatial frame compression algorithm that overcomes the limitation of conventional 360° video compression algorithm over high RTT cellular networks. Meanwhile it leverages the LTE uplink firmware buffer level extracted from physical layer to adapt traffic load and thus addresses the challenge of cellular uplink congestion.

8.2 Discussion & Future Works

Although this dissertation focuses on optimizing the mobile service performance over the present-day 4G LTE network, our design principles of PHY-informed mobile service optimization can also be potentially applied over future mobile networks, *e.g.*, 5G cellular networks. Specifically, the flexible network architecture and novel mobile services of 5G networks brings both opportunities and challenges for mobile service optimization. In what follows, we discuss about the limitation of our current works and how to further enhance the mobile service performance under the context of future 5G networks.

8.2.1 BS-Centric Solutions

This dissertation focuses on the end-to-end solutions without modification to the cellular BS, as the 4G cellular infrastructure has been widely deployed and frequently updating every existing BS is difficult. In other words, end users rather than the cellular operators are the target of our solutions. For instance, the online shop owners can equip the server hosting their Web site with our CLAW implementation, meanwhile distribute via GooglePlay to the users the shopping APP that reads the phone's diagnostic interface for bandwidth estimation. All the modifications happen at the ends of the connection, thus the online shop owners does not need to contact the cellular operator to reserve network and spectrum resources for the traffic visiting their Web sites. In this way, the distributed cross-layer design principle provides great flexibility.

However, the cellular networks are naturally centralized, with the BS scheduling all the resources and transmissions, thus a centralized service optimization design would have a global view of all the clients and provide better performance than the distributed solutions. Furthermore, to facilitate the delay-sensitive mobile services like interactive video, the distributed solutions can only use more efficient end-to-end rate control to avoid congestions and packet queuing for latency reduction, meanwhile adopt dedicated protocol design to tolerate the high latency. But the fundamental cause of the high latency over cellular network is the complicated

network architecture, which cannot be fully addressed without redesigning the BS. Over current 4G LTE networks, although the resource scheduling in the MAC layer can be modified via software updates, the physical layer operations and network architecture cannot be changed. But this situation may change in the era of 5G networks. In recent years, the concept of software-defined cellular network is gaining tremendous attention as an important feature of 5G networks. With more flexible network architecture, it will be easier to modify the physical layer operations inside the BS and expose the PHY-layer information to upper layer mechanisms like resource allocation. Since all the traffic scheduling and resource allocation tasks happen at the BS, physical layer of the BS has a global view of the cell, thus running the cross-layer algorithms on the BS can potentially provide better performance than end-to-end solutions. Meanwhile, such BS-centric design can also facilitate the cross-layer cooperation across multiple cells. For instance, using multi-path TCP to receive application data from multiple BS on different frequency bands. In this way, future BS can get involved in the physical layer informed mobile service optimization, which brings more optimization options and opportunities.

8.2.2 Physical-Layer Informed Mobile Edge Computing

Mobile edge computing (MEC [160]) has been a research hot spot in recently years. With more video content servers, web caching servers, proxy servers and dedicated application servers deployed in the cellular BS, the majority of mobile traffic will be simply between the mobile devices and these edge servers. Due to the much shorter end-to-end path, cellular link become the only network bottleneck and the design principle of PHY-informed mobiles server optimization can always be applied.

For instance, typical smartphones fail to satisfy the computational power demand of VR gaming due to the limited form factor and power supply, while the strict latency requirement of VR prohibits the cloud computing on remote servers. MEC provides the opportunity to address this challenge, as it enables computational offloading to the nearby servers inside the BS. However, the rendered VR contents become huge traffic load over the cellular link, thus dynamically adjusting

the fraction of rendering tasks to offload based on the instantaneous cellular link condition is vital for such mobile VR services, which is part of our future researches

REFERENCES

- [1] The Guardian. 2016. Mobile Web Browsing Overtakes Desktop for The First Time.
- [2] Cisco Systems, Inc. 2012. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012–2017.
- [3] Skype Team. 2016. Over 1 Billion Skype Mobile Downloads “ Thank You!
- [4] OpenSignal. 2016. State of Mobile Networks: USA.
- [5] 2014. LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation. *3GPP TS 36.211 version 12.3.0 Release 12*.
- [6] Ctrix. 2014. Mobile Analytics Report.
- [7] Yu, Chenguang, Yang Xu, Bo Liu, and Yong Liu. 2014. “Can you SEE me now?”: A Measurement Study of Mobile Video Calls. In *Ieee infocom*.
- [8] Erman, Jeffrey, Vijay Gopalakrishnan, Rittwik Jana, and K. K. Ramakrishnan. 2013. Towards a SPDY’ier Mobile Web? In *Acm conext*.
- [9] Bakre, Ajay, and BR Badrinath. 1995. I-TCP: Indirect TCP for Mobile Hosts. In *Proceedings of the 15th international conference on distributed computing systems (icdcs)*. IEEE.
- [10] Balakrishnan, Hari, Venkata N Padmanabhan, Srinivasan Seshan, and Randy H Katz. 1996. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *ACM SIGCOMM Computer Communication Review*.
- [11] Lin, Shu, and Daniel J Costello. 2004. *Error control coding*, vol. 2. Prentice Hall Englewood Cliffs.

- [12] DeSimone, Antonio, Mooi Choo Chuah, and On-Ching Yue. 1993. Throughput Performance of Transport-Layer Protocols over Wireless LANs. In *Ieee global telecommunications conference (globecom)*. IEEE.
- [13] Dahlman, Erik, Stefan Parkvall, and Johan Skold. 2013. *4G: LTE/LTE-Advanced for Mobile Broadband*. Academic Press.
- [14] Ding, Wenqing, and Abbas Jamalipour. 2001. A New Explicit Loss Notification with Acknowledgment for Wireless TCP. In *Personal, indoor and mobile radio communications, 2001 12th ieee international symposium on*. IEEE.
- [15] Shakkottai, Sanjay, Theodore S Rappaport, and Peter C Karlsson. 2003. Cross-Layer Design for Wireless Networks. *IEEE Communications magazine*.
- [16] Lu, Feng, Hao Du, Ankur Jain, Geoffrey M. Voelker, Alex C. Snoeren, and Andreas Terzis. 2015. CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks. In *Proc. of acm hotmobile*.
- [17] Jiang, Haiqing, Zeyu Liu, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Understanding Bufferbloat in Cellular Networks. In *Acm sigcomm workshop on cellular networks*.
- [18] Mascolo, Saverio, Claudio Casetti, Mario Gerla, Medy Y Sanadidi, and Ren Wang. 2001. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proceedings of the 7th annual international conference on mobile computing and networking*. ACM.
- [19] Hu, Ningning, and Peter Steenkiste. 2003. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE journal on Selected Areas in Communications*.
- [20] Winstein, Keith, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay Over Cellular Networks. In *Nsdi*.

- [21] Zaki, Yasir, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Acm sigcomm*.
- [22] Grossman, Dan. 2002. New Terminology and Clarifications for DiffServ. Tech. Rep.
- [23] Bernet, Yoram, Peter Ford, Raj Yavatkar, Fred Baker, Lixia Zhang, Michael Speer, Robert Braden, Bruce Davie, John Wroclawski, and Eyal Felstaine. 2000. A Framework for Integrated Services Operation over DiffServ Networks. Tech. Rep.
- [24] Babiarez, Jozef, Kwok Chan, and Fred Baker. 2006. Configuration Guidelines for DiffServ Service Classes. Tech. Rep.
- [25] GSM Association. IMS Profile for Conversational Video Service.
- [26] Zhang, Ying, A. Arvidsson, M. Siekkinen, and G. Urvoy-Keller. 2014. Understanding HTTP flow rates in cellular networks. In *Ifip networking conference*.
- [27] Xu, Yin, Zixiao Wang, WaiKay Leong, and Ben Leong. 2014. An End-to-End Measurement Study of Modern Cellular Data Networks. In *Proc. of passive and active measurement conference*.
- [28] Sodagar, Iraj. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. In *IEEE multimedia*.
- [29] Kumar, Swarun, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. 2014. LTE Radio Analytics Made Easy and Accessible. In *Proc. of acm sigcomm*.
- [30] Ben Wojtowicz. OpenLTE.
- [31] ENST. GPAC.
- [32] Huang, Te-Yuan, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proc. of acm sigcomm*.

- [33] Jiang, Junchen, Vyas Sekar, and Hui Zhang. 2012. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proc. of acm conext*.
- [34] Yin, Xiaoqi, Vyas Sekar, and Bruno Sinopoli. 2014. Toward a Principled Framework to Design Dynamic Adaptive Streaming Algorithms Over HTTP. In *Proc. of acm hotnets*.
- [35] Zou, Xuan Kelvin, Jeffrey Eрман, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K. Sinha. 2015. Can Accurate Predictions Improve Video Streaming in Cellular Networks? In *Proc. of acm hotmobile*.
- [36] Leland, W.E., M.S. Taqqu, W. Willinger, and D.V. Wilson. 1994. On the Self-Similar Nature of Ethernet Traffic. *IEEE/ACM Transactions on Networking* 2(1).
- [37] Crovella, Mark E., and Azer Bestavros. 1996. Self-similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Proc. of acm sigmetrics*. SIGMETRICS '96.
- [38] Li, Zhi, Xiaoqing Zhu, J. Gahm, Rong Pan, Hao Hu, A.C. Begen, and D. Oran. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications* 32(4).
- [39] Microsoft. Microsoft Smooth Streaming. <http://www.iis.net/downloads/microsoft/smooth-streaming>.
- [40] Apple. HTTP Live Streaming.
- [41] Adobe. HTTP Dynamic Streaming. <http://www.adobe.com/products/hds-dynamic-streaming.html>.
- [42] Hoque, M.A., M. Siekkinen, J.K. Nurminen, and M. Aalto. 2013. Dissecting Mobile Video Services: An Energy Consumption Perspective. In *Proc. of ieee wowmom*.

- [43] Becker, N., A. Rizk, and M. Fidler. 2014. A Measurement Study on the Application-Level Performance of LTE. In *Ifip networking conference*.
- [44] Gouta, Ali, Dohy Hong, Anne-Marie Kermarrec, and Yannick Lelouedec. 2013. HTTP Adaptive Streaming in Mobile Networks: Characteristics and Caching Opportunities. In *Modeling, analysis & simulation of computer and telecommunication systems (mascots)*.
- [45] Lawrence Berkeley National Laboratory. iperf3.
- [46] Qualcomm. QXDM.
- [47] Halperin, Daniel, Wenjun Hu, Anmol Sheth, and David Wetherall. 2011. Tool release: Gathering 802.11n traces with channel state information. *ACM SIGCOMM CCR* 41(1):53.
- [48] Sen, Souvik, Bozidar Radunovic, Jeongkeun Lee, and Kyu-Han Kim. 2013. CSpy: Finding the Best Quality Channel Without Probing. In *Proc. of acm mobicom*.
- [49] Taqqu, Murad S, Vadim Teverovsky, and Walter Willinger. 1995. Estimators for Long-Range Dependence: an Empirical Study. *Fractals*.
- [50] Winstein, Keith, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Usenix nsdi*.
- [51] Kotz, S., and S. Nadarajah. 2000. *Extreme Value Distributions: Theory and Applications*. London: Imperial College Press.
- [52] Oyman, O., and S. Singh. 2012. Quality of experience for HTTP adaptive streaming services. *IEEE Communications Magazine* 50(4).
- [53] Huang, Junxian, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An In-depth Study of

- LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proc. of acm sigcomm*.
- [54] Wang, Xiao Sophia, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2014. How Speedy is SPDY? In *Usenix nsdi*.
 - [55] Kissmetrics Blog. How Loading Time Affects Your Bottom Line.
 - [56] Jiang, Haiqing, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Tackling Bufferbloat in 3G/4G Networks. In *Proc. of acm internet measurement conference (imc)*.
 - [57] Pedersen, K. I., T. E. Kolding, F. Frederiksen, I. Z. Kovacs, D. Laselva, and P. E. Mogensen. 2009. An Overview of Downlink Radio Resource Management for UTRAN Long-term Evolution. *IEEE Communications Magazine*.
 - [58] Google. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. <https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02>.
 - [59] Xie, Xiufeng, Xinyu Zhang, Swarun Kumar, and Li Erran Li. 2015. piStream: Physical Layer Informed Adaptive Video Streaming Over LTE. In *Acm mobicom*.
 - [60] Chakraborty, Abhijnan, Vishnu Navda, Venkata N. Padmanabhan, and Ramachandran Ramjee. 2013. Coordinating Cellular Background Transfers Using Loadsense. In *Acm mobicom*.
 - [61] Erman, Jeffrey, Alexandre Gerber, Mohammad Hajiaghayi, Dan Pei, Subhabrata Sen, and Oliver Spatscheck. 2011. To Cache or Not to Cache: The 3G Case. *IEEE Internet Computing* 15(2).
 - [62] Wang, Zhen, Felix Xiao Zhu Lin, Lin Zhong, and Mansoor Chishtie. 2011. Why are web browsers slow on smartphones? In *Proc. of the workshop on mobile computing systems and applications (hotmobile)*.

- [63] Larson, Natalie, Džiugas Baltrunas, Amund Kvalbein, Amogh Dhamdhere, kc claffy, and Ahmed Elmokashfi. 2015. Investigating Excessive Delays in Mobile Broadband Networks. In *Proc. of the workshop on all things cellular (allthingscellular)*.
- [64] IETF. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). <https://tools.ietf.org/html/rfc7540/>.
- [65] Sun, Peng, Minlan Yu, Michael J. Freedman, and Jennifer Rexford. 2011. Identifying Performance Bottlenecks in CDNs Through TCP-level Monitoring. In *Proc. acm sigcomm workshop on measurements up the stack (w-must)*.
- [66] Katabi, Dina, Mark Handley, and Charlie Rohrs. 2002. Congestion Control for High Bandwidth-Delay Product Networks. *ACM SIGCOMM Computer Communication Review*.
- [67] Caesar, Matthew, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. 2005. Design and Implementation of a Routing Control Platform. In *Usenix nsdi*.
- [68] Flach, Tobias, Nandita Dukkupati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. 2013. Reducing Web Latency: The Virtue of Gentle Aggression. In *Acm sigcomm*.
- [69] ———. 2013. Reducing Web Latency: The Virtue of Gentle Aggression. In *Acm sigcomm*.
- [70] Bui, Nicola, and Joerg Widmer. 2016. OWL: A Reliable Online Watcher for LTE Control Channel Measurements. In *Proceedings of the 5th workshop on all things cellular: Operations, applications and challenges*.
- [71] 3GPP. 2015. LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures. *TS 36.213 version 12.4.0 Release 12*.

- [72] Li, Yuanjie, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. MobileInsight: Extracting and Analyzing Cellular Network Information on Smartphones. In *Acm mobicom*.
- [73] The Apache HTTP Server Project.
- [74] IETF HTTP Working Group. HTTP/2 Wiki.
- [75] Yasir Zaki. Verus Source Code on GitHub.
- [76] Google. Remote Debugging Android Devices.
- [77] Netravali, Ravi, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *Usenix nsdi*.
- [78] Zhu, Yuhao, Matthew Halpern, and Vijay Janapa Reddi. 2015. Event-Based Scheduling for Energy-Efficient qos (eqos) in Mobile Web Applications. In *2015 ieee 21st international symposium on high performance computer architecture (hpca)*. IEEE.
- [79] Li, Xianfeng, and Zhiqiang Bao. 2014. Performance Characterization of Web Applications with HTML5 Enhancements. In *Dependable, autonomic and secure computing (dasc), 2014 ieee 12th international conference on*. IEEE.
- [80] Gettys, Jim, and Kathleen Nichols. 2012. Bufferbloat: Dark Buffers in the Internet. *Communications of the ACM* 55(1).
- [81] Carbone, Marta, and Luigi Rizzo. 2010. Dummynet revisited. In *Acm sigcomm computer communication review*.
- [82] Alexa.
- [83] Hengartner, Urs, Jürg Bolliger, and Thomas Gross. 2000. TCP Vegas Revisited. In *Infocom 2000*.
- [84] 3GPP. LTE UE-Category.

- [85] Qualcomm. . Snapdragon 800 Processor.
- [86] ———. . Snapdragon 808 Processor.
- [87] Jain, A., A. Terzis, N. Sprecher, S. Arunachalam, K. Smith, and G. Klas. 2015. Requirements and reference architecture for Mobile Throughput Guidance Exposure.
- [88] Huang, Junxian, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2012. A close examination of performance and power characteristics of 4g lte networks. In *Proc. of acm mobisys*.
- [89] Nejati, Javad, and Aruna Balasubramanian. 2016. An In-Depth Study of Mobile Browser Performance. In *Proceedings of the 25th international conference on world wide web*. International World Wide Web Conferences Steering Committee.
- [90] Google. 2016. Accelerated Mobile Pages (AMP).
- [91] Xu, Yang, Chenguang Yu, Jingjiang Li, and Yong Liu. 2014. Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype. *IEEE/ACM Transactions on Networking* 22(3).
- [92] Jana, Shraboni, Amit Pande, An (Jack) Chan, and Prasant Mohapatra. 2013. Mobile Video Chat: Issues and Challenges. *IEEE Communications Magazine* 51(6).
- [93] IETF. 2015. RTP Media Congestion Avoidance Techniques (RMCAT). <https://datatracker.ietf.org/wg/rmcat/documents/>.
- [94] Google. A Google Congestion Control Algorithm for Real-Time Communication. <https://tools.ietf.org/html/draft-alvestrand-rmcat-congestion-03>.
- [95] Johansson, Ingemar. 2014. Self-Clocked Rate Adaptation for Conversational Video in LTE. In *Acm sigcomm workshop on capacity sharing*.

- [96] De Cicco, Luca, Gaetano Carlucci, and Saverio Mascolo. 2013. Experimental investigation of the google congestion control for real-time flows. In *Acm sigcomm workshop on future human-centric multimedia networking*.
- [97] Fund, F., Cong Wang, Yong Liu, T. Korakis, M. Zink, and S. S. Panwar. 2013. Performance of DASH and WebRTC Video Services for Mobile Users. In *Packet video workshop (pv)*.
- [98] IETF/W3C. 2015. WebRTC.
- [99] Tan, John. 2015. WebRTC Mobile: Facebook and Slack Are In.
- [100] Microsoft Inc. 2015. Skype.
- [101] Apple Inc. 2015. FaceTime.
- [102] Saltzer, J. H., D. P. Reed, and D. D. Clark. 1984. End-to-end Arguments in System Design. *ACM Trans. Comput. Syst.* 2(4).
- [103] skype communications sarl. 2014. skype for web (beta).
- [104] Handley, M., S. Floyd, J. Padhye, and J. Widmer. 2003. TCP Friendly Rate Control (TFRC): Protocol Specification.
- [105] Hart, Chad. 2015. How does Hangouts use WebRTC?
- [106] Jitsi Development Team. 2015. Jitsi Meet: an OpenSource WebRTC JavaScript Application.
- [107] ManyCam Development Team. ManyCam.
- [108] WebcamStudio Development Team. WebcamStudio.
- [109] Doubango Telecom. sipML5.
- [110] SIP.js Development Team. SIP.js.

- [111] Chen, Jiasi, Rajesh Mahindra, Mohammad Amir Khojastepour, Sampath Rangarajan, and Mung Chiang. 2013. A Scheduling Framework for Adaptive Video Delivery over Cellular Networks. In *Acm mobicom*.
- [112] YouTube. 2016. Uploading 360-degree Videos.
- [113] Facebook. 2016. Facebook 360 Video.
- [114] Moon, Mariella. 2016. 360fly Puts 4K Video Cams on Helmets.
- [115] Ryder, Adam. 2016. VR Action Camera Field Test: Ricoh Theta S and 360Fly 4K.
- [116] Popper, Ben. 2016. Drones and Virtual Reality Headsets are a Delicious Combination.
- [117] Lavars, Nick. 2016. Exo360 Drone Shoots VR Content on the Fly.
- [118] Guo, Yihua, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding On-device Bufferbloat for Cellular Upload. In *Proceedings of the 2016 acm on internet measurement conference*.
- [119] Gaddam, V. R., M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen. 2016. Tiling in Interactive Panoramic Video: Approaches and Evaluation. *IEEE Transactions on Multimedia* 18(9).
- [120] Makar, M., A. Mavlankar, P. Agrawal, and B. Girod. 2010. Real-Time Video Streaming With Interactive Region-of-Interest. In *Ieee international conference on image processing (icip)*.
- [121] Rondao Alface, Patrice, Jean-François Macq, and Nico Verzijs. 2012. Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach. *Bell Labs Technical Journal* 16(4).
- [122] Qian, Feng, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2016. Optimizing 360 Video Delivery over Cellular Networks. In *Proceedings of the workshop on all things cellular*.

- [123] Richoh. 2016. Theta S 360 Mobile Camera.
- [124] Facebook. 2016. Cubemap Transform Open Source Code.
- [125] ———. 2016. Next-generation video encoding techniques for 360 video and VR.
- [126] Sun, Xinding, J. Foote, D. Kimber, and B. S. Manjunath. 2005. Region of Interest Extraction and Virtual Camera Control Based on Panoramic Video Capturing. *IEEE Transactions on Multimedia* 7(5).
- [127] Cuervo, Eduardo, and David Chu. 2016. Poster: Mobile virtual reality for head-mounted displays with interactive streaming video and likelihood-based foveation. In *Proceedings of acm mobisys*.
- [128] Ryoo, Jihoon, Kiwon Yun, Dimitris Samaras, Samir R. Das, and Gregory Zelinsky. 2016. Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices. In *Proceedings of the international conference on multimedia systems (mmsys)*.
- [129] Pang, Derek, Sherif Halawa, Ngai-Man Cheung, and Bernd Girod. 2011. Mobile Interactive Region-of-interest Video Streaming with Crowd-driven Prefetching. In *Proceedings of international acm workshop on interactive multimedia on mobile and portable devices*.
- [130] LaValle, Steven M, Anna Yershova, Max Katsev, and Michael Antonov. 2014. Head Tracking for The Oculus Rift. In *Robotics and automation (icra), 2014 ieee international conference on*.
- [131] Azuma, Ronald Tadao. 1995. Predictive Tracking for Augmented Reality. In *Ph.d. dissertation, university of north carolina at chapel hill*.
- [132] Xu, Qiang, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. 2013. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *Proc. of acm mobisys*.

- [133] Holmer, Stefan, Mikhal Shemer, and Marco Paniconi. 2013. Handling Packet Loss in WebRTC. In *Image processing (icip), 2013 20th ieee international conference on*. IEEE.
- [134] Umlaeute. 2017. v4l2loopback - A Kernel Module to Create V4L2 Loopback Devices.
- [135] Aakash Patel and Gregory Rose. 2015. Conduit: Efficient Video Compression for Live VR Streaming.
- [136] Evgeny Kuzyakov and David Pio. 2016. Next-generation video encoding techniques for 360 video and VR.
- [137] Sen, Sayandeep, Syed Gilani, Shreesha Srinath, Stephen Schmitt, and Suman Banerjee. 2010. Design and implementation of an "approximate" communication system for wireless media applications. In *Sigcomm*.
- [138] Bao, Yanan, Tianxiao Zhang, Amit Pande, Huasen Wu, and Xin Liu. 2017. Motion-Prediction-Based Multicast for 360-Degree Video Transmissions. In *Ieee secon*.
- [139] Balakrishnan, Hari, Srinivasan Seshan, Elan Amir, and Randy H Katz. 1995. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of the 1st annual international conference on mobile computing and networking*. ACM.
- [140] Tian, Ye, Kai Xu, and Nirwan Ansari. 2005. TCP in Wireless Environments: Problems and Solutions. *IEEE Communications Magazine*.
- [141] Srivastava, V., and M. Motani. 2005. Cross-layer Design: A Survey and the Road Ahead. *IEEE Communication Magazine* 43(12).
- [142] Fu, B., Y. Xiao, H. J. Deng, and H. Zeng. 2014. A Survey of Cross-Layer Designs in Wireless Networks. *IEEE Communications Surveys Tutorials* 16(1).

- [143] Baranasuriya, Nimantha, Vishnu Navda, Venkata N Padmanabhan, and Seth Gilbert. 2015. QProbe: Locating the Bottleneck in Cellular Communication. In *Conext*.
- [144] Dong, Mo, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Nsdi*.
- [145] Nguyen, Binh, Arijit Banerjee, Vijay Gopalakrishnan, Sneha Kasera, Seungjoon Lee, Aman Shaikh, and Jacobus Van der Merwe. 2014. Towards Understanding TCP Performance on LTE/EPC Mobile Networks. In *Proc. of the workshop on all things cellular (allthingscellular)*.
- [146] Zhu, Xiaoqing, and Rong Pan. 2013. NADA: A Unified Congestion Control Scheme for Low-Latency Interactive Video. In *International packet video workshop (pv)*.
- [147] H. Lundin and S. Holmer and H. Alvestrand. 2013. A Google Congestion Control Algorithm for Real-Time Communication. Internet-Draft (Informational).
- [148] Internet Engineering Task Force. Mobile Throughput Guidance Signaling Protocol.
- [149] Chen, Jiasi, Rajesh Mahindra, Mohammad Amir Khojastepour, Sampath Rangarajan, and Mung Chiang. 2013. A Scheduling Framework for Adaptive Video Delivery over Cellular Networks. In *Proc. of acm mobicom*.
- [150] Sundaresan, Srikanth, Nick Feamster, Renata Teixeira, and Nazanin Magharei. 2013. Measuring and Mitigating Web Performance Bottlenecks in Broadband Access Networks. In *Proc. of internet measurement conference*.
- [151] Clark, David D., Scott Shenker, and Lixia Zhang. 1992. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. In *Acm sigcomm*.

- [152] Cicco, Luca De, Saverio Mascolo, and Vittorio Palmisano. 2011. Skype Video Congestion Control: An Experimental Investigation. *Computer Networks* 55(3).
- [153] Zhu, Jing, R. Vannithamby, C. Rodbro, Mingyu Chen, and S. Vang Andersen. 2012. Improving QoE for Skype video call in Mobile Broadband Network. In *Ieee global communications conference (globecom)*.
- [154] Chakareski, J. 2013. Adaptive Multiview Video Streaming: Challenges and Opportunities. *IEEE Communications Magazine* 51(5).
- [155] Quang Minh Khiem, Ngo, Guntur Ravindra, Axel Carlier, and Wei Tsang Ooi. 2010. Supporting Zoomable Video Streams with Dynamic Region-of-interest Cropping. In *Proceedings of the acm sigmm conference on multimedia systems (mmsys)*.
- [156] Ngo, Khiem Quang Minh, Ravindra Guntur, and Wei Tsang Ooi. 2011. Adaptive Encoding of Zoomable Video Streams Based on User Access Pattern. In *Proceedings of the annual acm conference on multimedia systems (mmsys)*.
- [157] Taubman, David, and Michael Marcellin. 2012. *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*. Springer Science & Business Media.
- [158] Khiem, Ngo Quang Minh, Guntur Ravindra, and Wei Tsang Ooi. 2011. Towards Understanding User Tolerance to Network Latency in Zoomable Video Streaming. In *Proceedings of the acm international conference on multimedia*.
- [159] Orts-Escolano, Sergio, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. 2016. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of acm symposium on user interface software and technology (uist)*.

- [160] ETSI. 2014. Mobile Edge Computing Introductory Technical White Paper. Tech. Rep.