Master's Thesis

An Extremely Low-latency Congestion Control for Mobile Cellular Networks

Shinik Park

Department of Computer Science and Engineering

Graduate School of UNIST

2019

An Extremely Low-latency Congestion Control for Mobile Cellular Networks

Shinik Park

Department of Computer Science and Engineering

Graduate School of UNIST

An Extremely Low-latency Congestion Control for Mobile Cellular Networks

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Shinik Park

06/25/2019
Approved by

_____

Advisor
Kyunghan Lee

An Extremely Low-latency Congestion Control for Mobile Cellular Networks

Shinik Park

This certifies that the thesis/dissertation of Shinik Park is approved.

06/25/2019

signature

_____

Kyunghan Lee

signature

_____

Changhee Joo

signature

_____

Hyoil Kim

Abstract

Since the diagnosis of severe bufferbloat in mobile cellular networks, a number of low-latency congestion control algorithms have been proposed. However, due to the need for continuous bandwidth probing in dynamic cellular channels, existing mechanisms are designed to cyclically overload the network. As a result, it is inevitable that their latency deviates from the smallest possible level (i.e., minimum RTT). To tackle this problem, we propose a new low-latency congestion control, ExLL, which can adapt to dynamic cellular channels without overloading the network. To do so, we develop two novel techniques that run on the cellular receiver: 1) cellular bandwidth inference from the downlink packet reception pattern and 2) minimum RTT calibration from the inference on the uplink scheduling interval. Furthermore, we incorporate the control framework of FAST into ExLL's cellular specific inference techniques. Hence, ExLL can precisely control its congestion window to not overload the network unnecessarily. Our implementation of ExLL on Android smartphones demonstrates that ExLL reduces latency much closer to the minimum RTT compared to other low-latency congestion control algorithms in both static and dynamic channels of LTE networks.

Table of Contents

1. Introduction 1

1.1 Exploiting Cellular Network Characteristics for Protocol Design 1

1.2 Gap from Ideal Latency: Existing Latency Optimized Protocols 2

1.3 ExLL Contribution 3

2. Observations 4

2.1 Measurement Setup 4

2.2 Cellular Network Characteristics 4

3. ExLL's Network Inference 7

3.1 Cellular Downlink 7

3.2 Cellular Uplink 10

4. ExLL Design 13

4.1 Control Algorithm 13

4.2 State Transition 14

4.3 MTE Calculation 15

4.4 mRE Calculation 17

4.5 Exit from Observation to Control 18

4.6 Recovery from Loss or Timeout 18

5. Evaluation 20

5.1 Receiver- vs. Sender-side ExLL 20

5.2 Performance in Static Channel 20

5.3 Performance in Mobile Channel 23

5.4 Performance with Multiple Flows 23

5.5 Non-Cellular Bottleneck Adaptation 26

6. Application Performance 29

7. Related Work 31

8. Conclusion 33

List of Figures

1 Mean and 95-th percentile RTT against the average throughput of various congestion control algorithms compared with that of sending a static congestion window around BDP over a real LTE network..…… 2

2 (a) RTT, throughput, and RSSI from three test runs in a mobile scenario, (b) RTT statistics (with average and 100% confidence interval) from channels with different RSSI values..…. 5

3 Throughput and RTT when (a) two cellular receivers in the same eNB download data with Cubic each, (b) one receiver downloads with Cubic and another downloads with Cubic cropped by BDP..…….. 6

4 Snapshot of received packets over time when downloading data with Cubic at the UE and the detailed packet receptions mapped onto allocated subframes (colored) in radio frames for the UE..……... 8

5 Sample calculation of $F(f_i)$ and $C(f_i)$ from packet receptions in a given radio frame f_i.. 9

6 Comparison of measured throughput with Cubic and bandwidth estimations from $F(\cdot)$ and $C(\cdot)$ over time…… 10

7 Concept of SR periodicity for cellular uplink scheduling in comparison with downlink scheduling……... 11

8 Snapshot of received packets and Acks over time and the distribution of RTT values observed at UE……... 11

9 State transition diagrams of the receiver-side ExLL and the sender-side ExLL…… 16

10 MTE shows the fastest response in detecting bandwidth changes that are reflected in the measured throughput later. MTE_S is slightly slower than MTE but is faster than the measured throughput…….. 16

11 A sample run of $apRTT$ and $mpRTT$ measured at a cellular receiver during the observation mode. $\hat{T}^{\mathrm{SR}}$ estimates 10 ms SR periodicity for the connected eNB…… 17

12 Congestion control behaviors and performance of receiver-side and sender-side ExLL compared in real LTE networks while the tested cellular receiver is (a) stationary or (c) mobile. (b) and (d) summarize the comparison of throughput and RTT between two implementations of ExLL……... 22

13 (a) A comparison between ExLL and BBR in a stationary LTE channel, (b) RTT and throughput performance comparison between ExLL and other protocols. ExLL outperforms other low-latency protocols and operates very closely to the ideal performance characterized by sending $1.0 \times$ BDP of the network……... 23

14 (a) A comparison between ExLL and BBR in the same mobile channel generated by in-lab LTE testbed, (b) RTT and throughput performance comparison between ExLL and other protocols in a

mobile channel. ExLL shows nearly 20 ms less RTT compared to BBR while getting throughput as much as Cubic…… 24

15 ExLL (a) persistently maintains lower latency while giving throughput fairness to second and third flows compared to BBR (b)……... 25

16 (a) ExLL takes control or hands it over to Cubic adaptively under cellular or non-cellular bottleneck for coexistence with Cubic. (b) ExLL runs as Cubic and achieves fairness when competing with Cubic flows under non-cellular bottleneck. (c) An ExLL flow experiencing non-cellular bottleneck coexists with two ExLL flows with no non-cellular bottleneck in a cellular link……. 28

17 (a) Average PLT and (b) Speed Index with 95% confidence interval measured from three popular web sites with or without application updates. ExLL substantially improves PLT and Speed Index especially when application updates coexist……. 30

18 Video channel switching time. ExLL switches to a new video much faster than TCP Cubic when background traffic exists……... 30

List of Tables

1 Server and Android device specification……..…… 4

2 SR periodicity estimation in a real LTE network and from our in-lab LTE testbed……... 18

3 Jain's Fairness Index from Multi-flow Scenarios.…….. 26

1. Introduction

Recent observations in the Internet revealed that TCP sessions often suffer from exceptionally long packet delay even in a small bandwidth delay product (BDP) network. Gettys et al. termed this phenomenon as bufferbloat [13] and diagnosed it as an over-buffering problem at the bottleneck link, mostly caused by TCP's loss-based congestion control mechanism that keeps pushing packets to the network until the bottleneck queue becomes full. Follow-up measurement studies [14, 18, 21] confirmed that the bufferbloat problem is often severe in cellular networks. They diagnosed that Cubic [15], the default TCP congestion control algorithm in Linux (hence, the default in most Android devices) and in Windows [7], is aggressive enough to quickly fill up cellular network buffers, resulting in up to several hundred milliseconds of additional packet latency.

1.1 Exploiting Cellular Network Characteristics for Protocol Design

In order to tackle such a latency problem in cellular networks, there have been various research proposals on designing a low-latency congestion control algorithm [10, 21, 23, 24, 33, 36]. These approaches, however, have not fully leveraged cellular network specific characteristics.

For example, for the downlink scheduling the base station (BS) schedules downlink packets towards multiple user equipments (UEs) at 1 ms granularity (a.k.a. transmission time interval, TTI), based on both the signal strength reported by each UE and the current traffic load [9]. This indicates that we can instantly infer the cellular link bandwidth by observing packet reception patterns in a very short time window instead of explicitly probing the bandwidth or measuring throughput for a relatively long period of time (e.g., a few RTTs). For the uplink scheduling, the BS needs to grant uplink transmission eligibility for each UE, which happens at a regular interval known as SR (scheduling request) periodicity [3]. We find that ignoring such a scheduling pattern often underestimates the minimum RTT and leads congestion control algorithms to run in unrealistic operating points.

Furthermore, the minimum RTT value is not generally affected by the channel condition between UE and BS due to adaptive MCS (modulation and coding scheme) selection used in LTE systems, which approximately eliminates MAC-layer retransmissions, but also not affected by other UEs connected to the same BS since per-UE queue is isolated.

The understanding of the aforementioned factors has important implications on the design of an extremely low-latency congestion control and its best performance in throughput and delay. In particular, to cope with highly dynamic channel conditions in cellular networks, it is vital to obtain both achievable maximum throughput and minimum RTT as quickly and accurately as possible, so that the congestion control algorithm can always exploit up-to-date BDP for low-latency control.

1.2 Gap from Ideal Latency: Existing Latency Optimized Protocols

In order to check the latency performance of existing low latency congestion control algorithms including Cubic, we perform our own measurement with an Android smartphone over a static LTE channel as shown in Figure 1. We compare the performance (mean and 95-th percentile) of existing latency optimized congestion control protocols along with a simple protocol that sends at a constant rate ($\beta \times BDP$). The minimum RTT and maximum throughput achievable for the tested channel are about 47 ms and 90 Mbps, respectively. While Cubic suffers from long packet latency of 230 ms, BBR [10], PropRate [23] and Verus [36] achieve 104 ms, 71 ms, and 76 ms, respectively, with similar or much lower throughput[1]. Low-latency algorithms show significant RTT suppression compared to Cubic, but their latency performance is still far from the ideal one characterized by sending the BDP variants (β = 0.9,1.0,1.1), which is to achieve about 68 ms at 90 Mbps throughput. It is intriguing that BBR and PropRate that are designed to track and utilize the BDP of the network are performing not as good as sending the BDP. However, considering the overhead from cycling several modes of operation to probe bandwidth and RTT, the existence of the performance gap is not surprising.

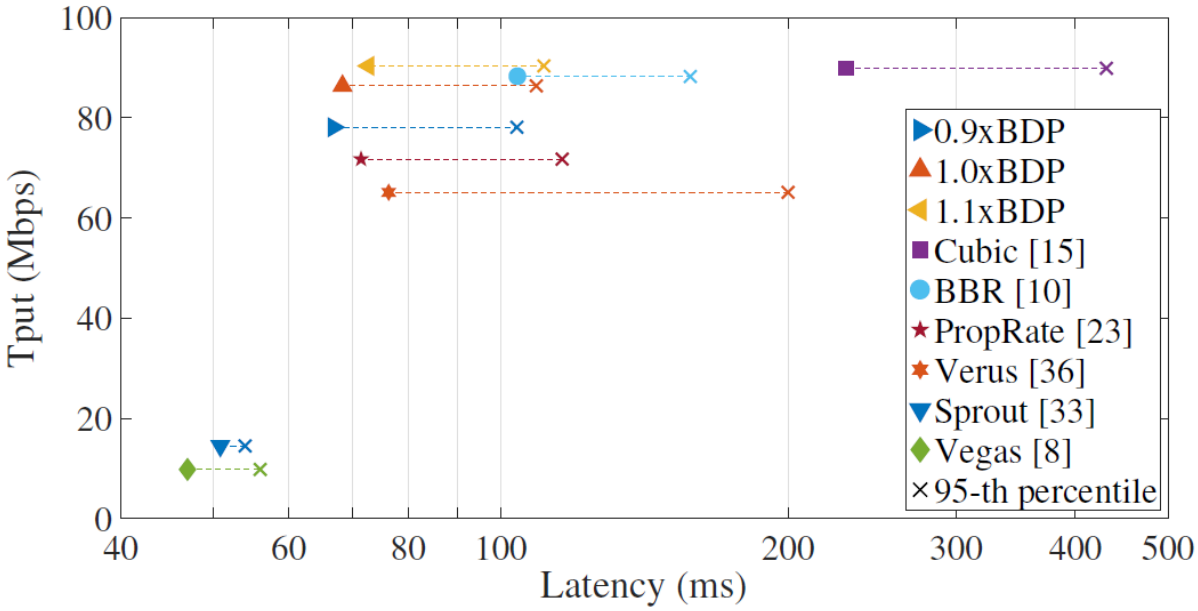


Figure 1: Mean and 95-th percentile RTT against the average throughput of various congestion control algorithms compared with that of sending a static congestion window around BDP over a real LTE network.

[1] We test protocols based on the source code provided by the authors. Tuning them to operate more aggressively to achieve maximum throughput is possible, which may, however, lead to significant increase in latency. Thus, we do not modify or tune them.

1.3 ExLL Contribution

To bridge the gap, we propose a new low-latency congestion control for mobile cellular networks, namely ExLL (Extremely Low Latency) that reduces latency as close to the minimum RTT while retaining the same level of throughput that Cubic achieves. To obtain such performance, instead of probing the network, ExLL estimates the bandwidth of cellular channels by analyzing the packet reception pattern at an LTE subframe granularity in the downlink and also estimates the minimum RTT more realistically by incorporating SR periodicity in the uplink. As these estimations can be done reliably at each UE, ExLL takes the receiver-side design as its first choice. With the bandwidth and latency estimations, ExLL adopts the control feedback in FAST [32] to compute its receive window (RWND). This receiver-side ExLL design is immediately deployable to cellular UEs without compromising servers as it makes the congestion control protocol running on the servers set its congestion window (CWND) based on the RWND from ExLL receiver. Furthermore, ExLL can be implemented as sender-side as well. We later demonstrate that both implementations have a minor performance gap in practice.

Our comprehensive experiments carried out over commercial LTE networks confirm that ExLL can always achieve shorter RTT which is much closer to the minimum RTT while retaining similar throughput of Cubic. More specifically, in a stationary scenario where an Android smartphone is stably connected to an LTE network with 50 ms of its minimum RTT and 75 Mbps of its maximum throughput, ExLL attains on average 66 ms RTT while maintaining throughput of 72 Mbps; BBR and Cubic attain about 110 ms and 261 ms RTT with about 70 Mbps and 75 Mbps. PropRate and Verus show lower throughput about 59 Mbps and 39 Mbps and attain around 61 ms and 92 ms RTT. In a mobile scenario where a smartphone user moves between good (-95 dBm) and bad channels (-125 dBm), ExLL retains around 61 ms and 45 Mbps while BBR and Cubic stay around 78 ms and 395 ms with about 40 Mbps and 46 Mbps. PropRate and Verus shows 53 ms and 63 ms but is with only 23 Mbps and 34 Mbps.

In summary, our contributions are three-fold.

- We develop novel techniques that can estimate the cellular link bandwidth and realistic minimum RTT without explicit probing, which can be easily extended to next-generation cellular technologies such as 5G.
- We incorporate the control logic of FAST into ExLL to minimize the latency even in dynamic cellular channel conditions.
- We implement ExLL in both receiver- and sender-side versions that give wider deployment opportunities. The receiver-side ExLL can provide an immediate solution for untouched commodity servers while the sender-side ExLL can provide a fundamental solution for 5G URLLC (Ultra Reliability and Low Latency Communication) services [28].

2. Observations

In this section, we present several observations from both commercial LTE networks and our controlled LTE tested. In Section 3, we introduce ExLL's cellular specific inference techniques to achieve low latency without sacrificing throughput.

2.1 Measurement Setup

We conduct measurements in commercial LTE networks and in our in-Lab LTE testbed. Software settings represented by kernel and Android OS versions and hardware specifications such as LTE chipsets equipped in cellular devices are summarized in Table 1. Our in-lab LTE testbed consists of indoor small cell base stations (i.e., eNBs) and EPC (Evolved Packet Core) software implementing MME (Mobility Management Entity) and S/P-GW (Serving/Packet Gateway) [1]. We put Android phones inside a shield box, so that the phones can communicate with the eNB via a pair of antennas inside the box. The signal attenuator installed between the shield box and the eNB can emulate a variety of RF conditions. For EPC, we use the open source NextEPC [26] that implements features up to 3GPP LTE Release 14.

Table 1. Server and Android device specification

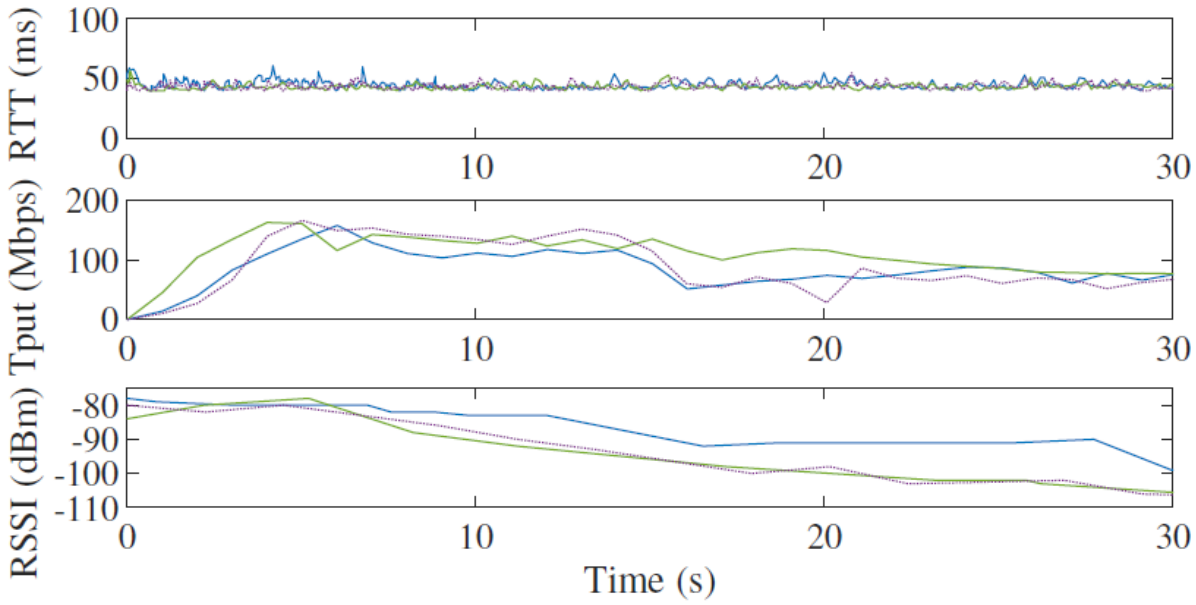
Real LTE	Processor	LTE Modem	Kernel	OS
Server	i7-6700K	-	Linux 4.13	Ubuntu 16.04
Client	MSM8992	X10 LTE	Linux 3.10	Android 8.1.0

in-lab LTE	Processor	LTE Modem	Kernel	OS
Server	i5-7200U	-	Linux 4.13	Ubuntu 16.04
Client	MSM8992	X10 LTE	Linux 3.10	Android 8.1.0

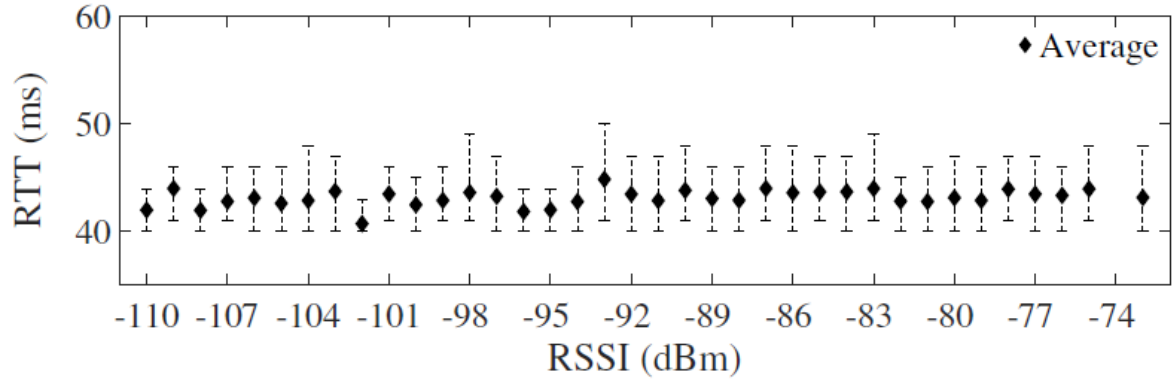
2.2 Cellular Network Characteristics

Max throughput and min RTT in the cellular network: We first test throughput and RTT in a commercial LTE network by physically moving two cellular devices from a certain location where the received signal strength indication (RSSI) is about -75 dBm to another location with -105 dBm. We measure the throughput of one device by downloading a large file from our in-lab server running Cubic. At the same time, we also measure the instantaneous RTTs by repeating ping tests in another device. Note that both devices are connected to the same LTE eNB. Figure 2 (a) shows RTT from ping tests, throughput from downloading, and RSSI over time from the three test runs. We can see that RTT

stays nearly at the same level even when the RSSI varies widely, but achievable throughput fluctuates accordingly.



(a) RTT, throughput, and RSSI from three test runs in a mobile scenario



(b) RTT statistics (with average and 100% confidence interval) from channels with different RSSI values

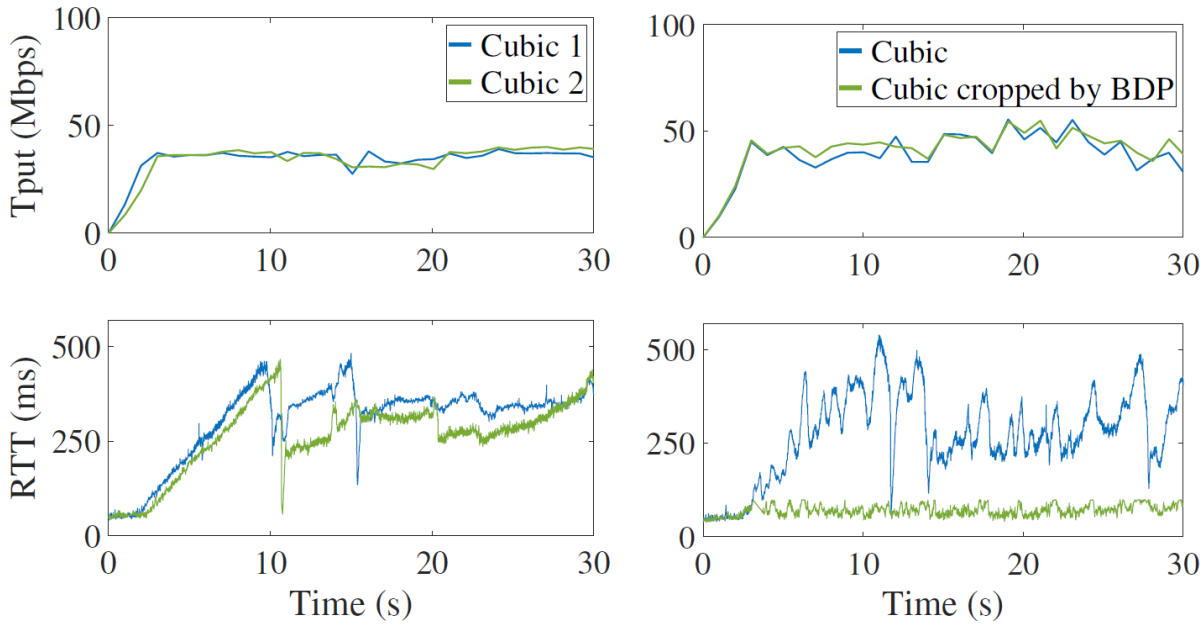
Figure 2: (a) RTT and throughput measurements in a commercial LTE network while moving cellular devices from a location with strong signal strength to a location with weak signal strength, (b) RTT statistics measured by ping tests over a commercial LTE network with different RSSI values.

Min RTT over different RSSI values: The minimum RTT observed from a network has been one of the most important component in delay-based congestion controls, as it indicates the RTT with nearly zero queueing delay. In the cellular network, however, it is unclear if the physical channel condition largely affects this. To demonstrate the impact of RSSI on the minimum RTT, we average 100 ping measurements for different levels of RSSI. As shown in Figure 2 (b), the minimum RTT (the lower

end of a confidence interval) is not much affected by RSSI. We conjecture that this is because of MCS adaption to a different channel quality for reliable packet transmissions in the MAC layer, which limits additional delay from retransmissions. Another interesting observation is that RTT variation is also very stable over different RSSI values. We find the rationale behind this stable RTT variation from the implementation of SR periodicity of LTE networks, which we investigate the details in Section 3.2.

Per-UE queueing[2] in LTE networks: To see if the LTE network employs a per-UE queue, we run two Cubic flows on two cellphones (one flow on each cellphone). Then, we replace one Cubic flow with a Cubic flow whose CWND is cropped by BDP. Figure 3 confirms that each UE has its own queue, not affecting the other competing UE's RTT while achieving the same throughput even by capping the CWND.

As discussed in [33], the per-UE queue in cellular networks is an important factor in designing a low-latency congestion control algorithm because the delay in the queue is only affected by its own control, not by other congestion control algorithms running on other cellular devices in the same BS. This motivates us to focus more on a receiver-side design that can turn Cubic flows from any servers to the receiver into low-latency flows.



(a) Both receivers download data with Cubic

(b) One receives data with Cubic while another does with Cubic cropped by BDP

Figure 3: Throughput and RTT when (a) two cellular receivers in the same eNB download data with Cubic each, (b) one receiver downloads with Cubic and another downloads with Cubic cropped by BDP.

[2] Each UE is given a default bearer that serves all best-effort traffic. For simplicity, we denote the queue of the default bearer as per-UE queue here.

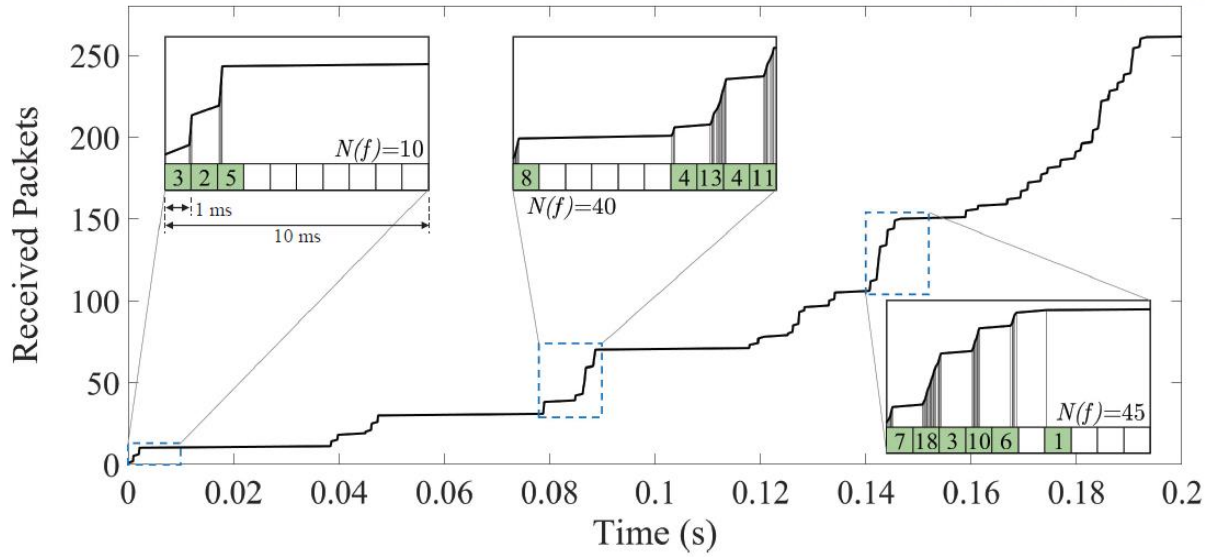
3. ExLL's Network Inference

ExLL leverages downlink and uplink scheduling mechanisms of cellular system to inform the design of an extremely low latency protocol without having throughput degradation.

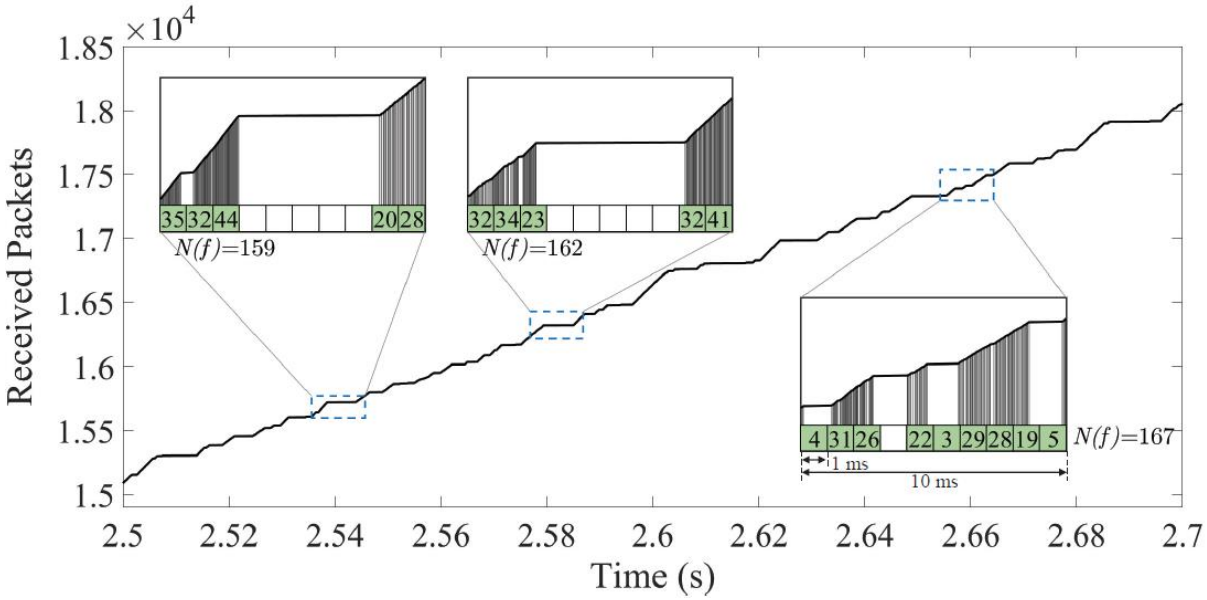
3.1 Cellular Downlink

In mobile cellular networks that experience frequent bandwidth fluctuations, an efficient probing becomes more important, as explored by recently proposed low-latency congestion control algorithms such as Verus, BBR, and PropRate. Although how to probe optimally is still an open question, we can postulate that probing in a way that injects packets excessively to the network even for short duration such that the network becomes temporarily overbuffered is far from optimum in terms of latency. To overcome this challenge, we can borrow recent proposals that enables the estimation of available cellular network bandwidth either from 1) extracting PHY-layer parameters [24, 34, 35] or 2) running a machine learning [17]. However, they also bring new practical issues such as extracting PHY-level information is only supported by specific chipset vendors (e.g., Qualcomm) with additional software tools (e.g., QXDM) that need rooting a device. A machine learning based approach [17] eliminates the need for such complications. However, learning parameters takes time, and thus momentary changes in cellular networks such as traffic load and channel condition are hard to be tracked in real time. To this end, ExLL takes a relatively simpler yet effective approach. ExLL observes packet receiving patterns in a cellular device and finds clues to estimate the available bandwidth from the scheduling behaviors of cellular networks without explicitly probing the network.

Downlink scheduling pattern: LTE systems are designed to schedule downlink packets by the unit of subframe whose duration is 1 ms [1]. Also, ten subframes are grouped to form a radio frame of length 10 ms. Each subframe consists of two slots of 0.5 ms duration and one slot contains multiple resource blocks (RBs) with 180 kHz bandwidth each [2]. The RB is the smallest resource unit allocatable to an LTE UE and the number of RBs in a slot is determined by the total bandwidth. For example, 10 MHz which is a typical bandwidth of commercial LTE networks has 50 RBs. According to [22], the physical-layer specification from 3GPP defines that an LTE network with 10 MHz bandwidth, 256 QAM for modulation, and 2x2 MIMO antennas can reach up to 100.8 Mbps. Therefore, the network can deliver 12,600 bytes during one subframe, thus each RB carries about 252 bytes. Provided that most commercial networks set their MTU sizes between 1,428 and 1,500 bytes [27], such an LTE network can carry at most 8.4 to 8.8 packets per subframe. When carrier aggregation in the LTE-Advanced networks [1, 30] is activated, multiple frequency bands (e.g., 2 or 3 bands) can add up and the total bandwidth increases to 20, 30, 40, or 50 MHz. Then, the data rate and the number of packets downloadable in a subframe increase accordingly.



(a) Downlink packet reception patterns in the beginning of a new flow



(b) Downlink packet reception patterns after a few seconds

Figure 4: Snapshot of received packets over time when downloading data with Cubic at the UE and the detailed packet receptions mapped onto allocated subframes (colored) in radio frames for the UE.

In Figure 4, we count the number of downlink packets received by a cellphone from a server running Cubic in the 45 ms RTT network. The cellular bandwidth is 50 MHz from carrier aggregation. As depicted, at every 1 ms, a group of packets are received while the group size varies from 2 to 44 by subframes. This is well aligned with the aforementioned description of LTE downlink. Furthermore, we present the packet counts by the unit of subframe (1 ms) and radio frame (10 ms) in the same figure. In particular, the colored subframes indicate *allocated subframes* during which packet receptions are made

by allocated RBs, and the numbers on the colored subframes denote the number of received packets during each subframe while $N(f)$ denotes the total number of packet receptions during one radio frame (10 ms). Figure 4 (a) showing the initial stage of the download implies that the number of received packets in a radio frame increases rapidly as CWND increases. Thus, in the initial stage, the downlink behavior temporarily depends on CWND growth. However, in a few seconds as Figure 4 (b) shows, the patterns of allocated subframes change, but the total number of packet receptions per radio frame becomes stable. This is because subframe allocations are averaged out in a radio frame, and the averaged packet reception is governed by the chosen MCS and the scheduling quota for each receiver. We define metrics for ExLL's bandwidth estimation below.

Cellular downlink bandwidth estimation: : We let f_i and $F(\cdot)$ denote the i-th radio frame for a given UE and a bandwidth estimation function that converts f_i into a value in Mbps, respectively. The operation of $F(f_i)$ is as simple as counting the received bytes during one radio frame divided by 10 ms. In a special case where f_i does not include any allocated subframe, such f_i is ignored. We also define a microscopic bandwidth estimation function, $C(\cdot)$, which focuses more on average packet reception intervals within a subframe to estimate the maximum channel bandwidth as follows:

$$C(f_i) = \frac{\sum_{j \in S(f_i)} b_{ij} / \Delta t_{ij}}{|S(f_i)|}, \quad (1)$$

where b_{ij}, Δt_{ij}, and $S(f_i)$ denote the amount of received bytes within j-th subframe (s_{ij}) of i-th radio frame, the time gap between the first packet and the last packet reception within s_{ij} and the index set of allocated subframes in f_i, respectively. By definition, $C(\cdot)$ captures the total channel bandwidth before it is split to multiple users. Therefore, in case when a BS is occupied by a single UE, $C(\cdot)$ is close to $F(\cdot)$. But in case with the BS serving multiple UEs, $C(\cdot)$ is much larger than $F(\cdot)$. Figure 5 illustrates how $F(\cdot)$ and $C(\cdot)$ are computed for a sample radio frame.

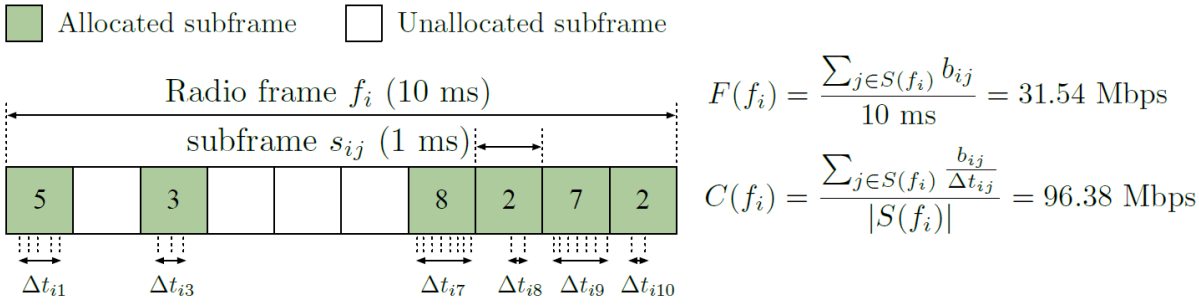
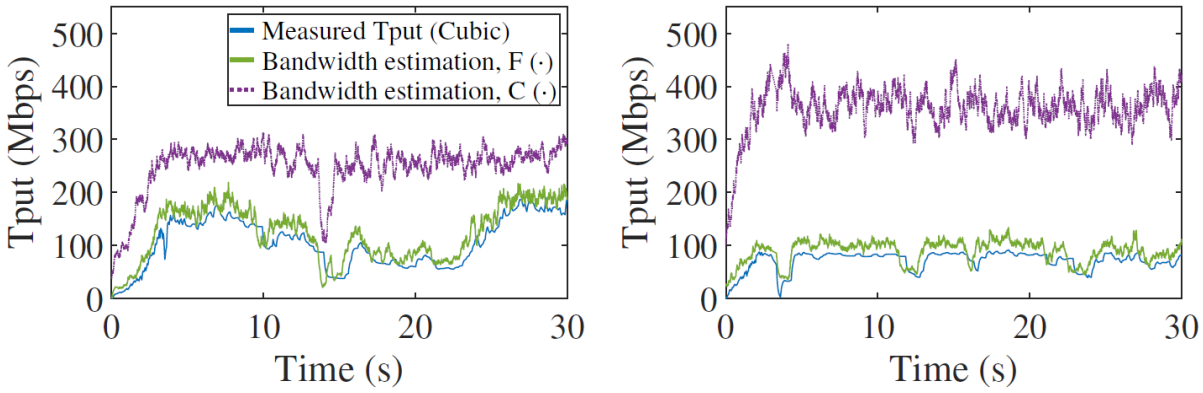


Figure 5: Sample calculation of $F(f_i)$ and $C(f_i)$ from packet receptions in a given radio frame f_i.

***F* and *C* over dynamic channels:** Figure 6 (a) and (b) present $F(\cdot)$, $C(\cdot)$, and the measured throughput on the UE in the carrier-aggregated channels of 30 MHz and 40 MHz, respectively, when

the UE downloads data for 30 seconds from a server running Cubic. Two interesting observations are found from these figures: 1) $F(\cdot)$ very precisely tracks the achievable network bandwidth before it measures the actual throughput, 2) $C(\cdot)$ estimates the channel bandwidth of 300 Mbps or 400 Mbps from 30 MHz or 40 MHz channel very precisely and quickly. When the MCS is degraded due to a poor channel condition, $F(\cdot)$ and $C(\cdot)$ instantly detect it as shown in the figures. $C(\cdot)$ estimates the best case performance for a single UE, but even when eNB serves only one UE, the achievable throughput can be lower than $C(\cdot)$ due to QoS settings of eNB such as UE-AMBR (aggregate maximum bitrate)[3,4] [3]. We find both metrics $F(\cdot)$ and $C(\cdot)$ are useful for different purposes. In Section 4.3 and 4.5, we detail the usage of them.



(a) Measurements from a real LTE network with 30 MHz channel

(b) Measurements from a real LTE network with 40 MHz channel

Figure 6: Comparison of measured throughput with Cubic and bandwidth estimations from $F(\cdot)$ and $C(\cdot)$ over time.

3.2 Cellular Uplink

LTE uplink scheduling is different from that of downlink. The biggest difference is that before obtaining an uplink scheduling grant from the BS, a UE needs to send its scheduling request by following the SR periodicity as depicted in Figure 7. Commercial LTE eNBs typically use SR periodicity chosen either from 5, 10, 20, 40, or 80 ms [29, 37]. While ExLL implements the SR periodicity inference algorithm in 4.4. we experimentally show such uplink scheduling patterns below.

Uplink scheduling patterns: Figure 8 (a) shows the receiving packet counts in a UE downloading data from a server and Figure 8 (b) shows the receiving Ack counts in the server. The time lines are adjusted

[3] This is defined to limit total throughput for each UE.

[4] In general, QoS settings in eNB are invisible to UEs. However, as specified in the LTE attach procedure of UE [4], some QoS settings can be shared from eNB to UE through ESM (EPS session management) messages. The purpose of this sharing is to let the UE give optional intelligence to applications that need better traffic control. Provided such QoS settings, fine tuning of $C(\cdot)$ can be further made.

to start from zero by the moment of receiving the first packet. As shown in Figure 8 (a) and (b), the granularity of Ack reception in the server is about 10 ms whereas the granularity of packet reception in the UE is about 1 ms. This implies that SR periodicity in the connected eNB for uplink is set as 10 ms.

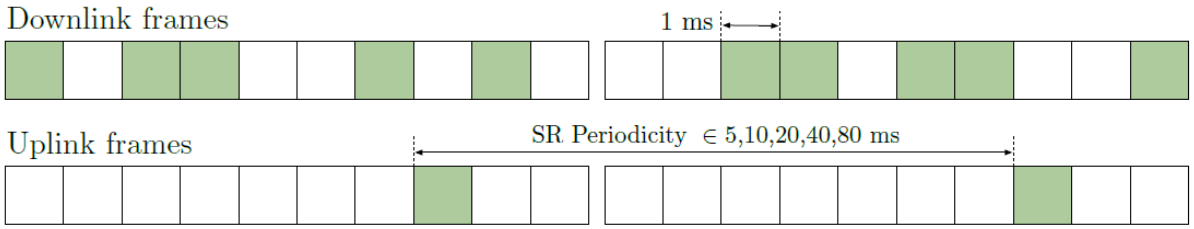
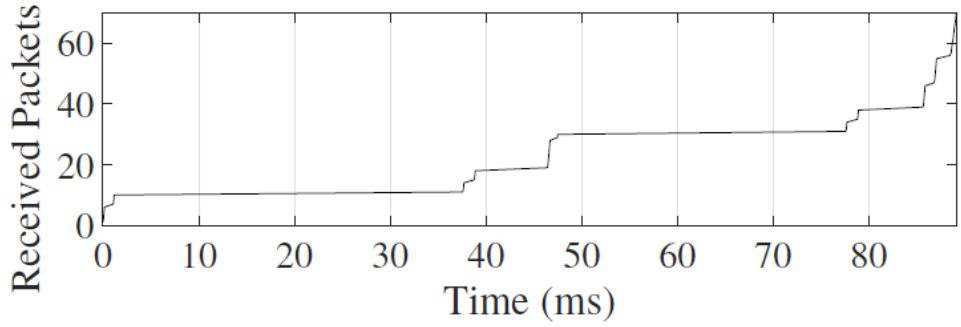
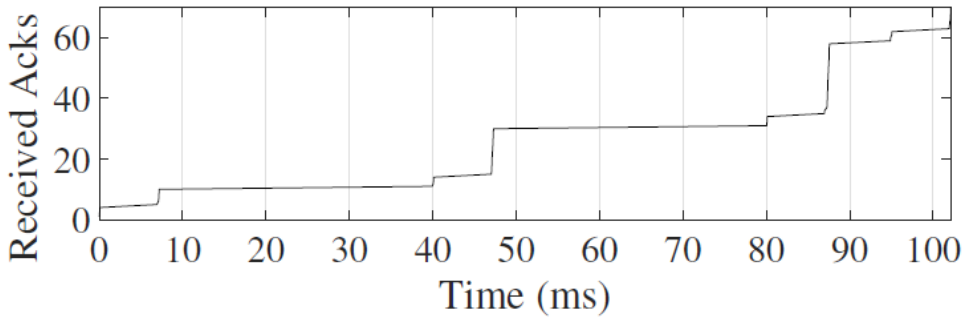


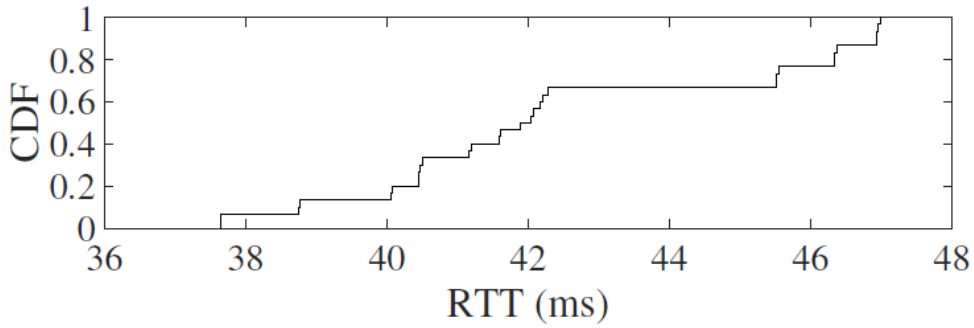
Figure 7: Concept of SR periodicity for cellular uplink scheduling in comparison with downlink scheduling.



(a) The number of received packets at UE



(b) The number of received Acks at sender



(c) CDF of RTTs collected at UE

Figure 8: Snapshot of received packets and Acks over time and the distribution of RTT values observed at UE.

RTT variation and min RTT: Figure 8 (c) shows the CDF (Cumulative Density Function) of per-packet RTT measured in the UE by pinging the server. We find that the RTTs from the ping test vary from 37 ms to 47 ms whose average is about 42ms. The gap between the maximum and minimum is about 10 ms that matches with SR periodicity. An important lesson is drawn here. If a low-latency congestion control simply takes the observed minimum RTT as its measure or target for controlling CWND, the control becomes overly conservative and loses throughput. To avoid such a problem, we develop a realistic estimation technique for the minimum RTT that takes SR periodicity into consideration[5], which will be detailed in Section 4.4.

[5] Note that a major cellular chipset vendor, Qualcomm, is reflecting this aspect in assessing their latency performance in LTE networks [25].

4. ExLL Design

ExLL aims at controlling its sending rate (i.e., CWND) so as to minimize latency while achieving throughput comparable with that of Cubic. Realizing this goal requires us to address two important challenges: 1) given a dynamic cellular channel whose achievable throughput and RTT vary, how do we track them precisely without explicitly probing the network?; 2) once achievable throughput and RTT become known, how do we use them to tightly control the CWND so as not to deviate from the desired operating point? In this section, we answer these questions and propose ExLL.

4.1 Control Algorithm

The well-known dilemma that every low-latency congestion control has is that the queuing in the bottleneck link should be minimized for latency, but it should be always non-empty for throughput. This dilemma becomes more challenging in dynamic networks. When the bottleneck bandwidth increases or decreases, the sending rate should quickly change accordingly, otherwise throughput loss or RTT increase occurs. We tackle this dilemma by revisiting the following control equation of FAST:

$$w_{i+1} = (1-\gamma)w_i + \gamma\left(\frac{mRE_i}{R_i}w_i + \alpha\right), \quad (2)$$

where $\gamma \in (0,1], \alpha > 0$ and w_i, R_i, and mRE_i denote CWND at time slot i, RTT measured at i, and the minimum RTT estimate at timeslot i, respectively.

The equation of FAST reduces CWND as the measured RTT deviates from the minimum RTT estimate (mRE) while persistently pushing CWND to grow by a constant incremental factor, α. This control equation lets a flow (say flow j) converge to the equilibrium data rate $x^{(j)} = \alpha^{(j)}/q^{(j)}$, where $q^{(j)}$ denotes the round-trip queueing delay of the flow (i.e., summation of all queueing delays on its routing path), which is dominated by the queueing delay at its bottleneck link. This equilibrium rate is known as the unique maximizer of a network utility maximization problem: $\max_{x\geq 0} \sum \alpha^{(j)} log x^{(j)}$ [32]. FAST probes and tracks the network bandwidth as fast as Cubic, provides weighted proportional fairness that does not penalize flows with large propagation delays, and suppresses queueing in the bottleneck link compared to Cubic. Nonetheless, it is hard to classify FAST as a low-latency congestion control because of α, the tuning parameter. In FAST, α plays many roles. It determines the amount of queueing in the bottleneck of a flow, which accumulates when having multiple flows, the agility of bandwidth adaptation, and the robustness in maintaining high throughput. Small α may give restrained queueing that is desirable for a low-latency congestion control, but it slows down the speed of adaptation. More seriously, small α loses its guarantee to achieve maximum throughput in a network in which RTT can fluctuate heavily. In such a network,

CWND can sometimes be overly reduced from RTT fluctuation leading to failures in always keeping the queueing non-empty.

ExLL is designed to provide a solution to the problems related to α by using our inference techniques in the UE, while retaining all the merits of FAST. Our solution is simple and does not complicate the control logic of FAST. It replaces α with $\alpha(1 - T_i/\,MTE_i)$, where T_i and MTE_i denote the measured throughput at time i and the maximum throughput estimate at time i, respectively. ExLL can obtain MTE_i at the UE using cellular downlink characteristics. Thus, the control equation of ExLL is given as follows:

$$w_{i+1} = (1-\gamma)w_i + \gamma\left(\frac{mRE_i}{R_i}w_i + \alpha(1 - \frac{T_i}{MTE_i})\right), \qquad (3)$$

Unlike FAST, w_i updates in ExLL is basically done by the receiver (UE), so w_i for the receiver-side ExLL means RWND; in the sender-side ExLL, the same equation updates CWND.

If MTE_i can be obtained precisely, the revised equation has a critical benefit over FAST. Even if α is chosen arbitrarily large for agile and robust bandwidth probing, ExLL does not to over-buffer the bottleneck link. This is because the increment given to congestion window from α diminishes to zero as the actual throughput T_i approaches the maximum throughput[6]. The equilibrium data rate of ExLL flow j is given as $x^{(j)} = \alpha^{(j)}/(q^{(j)} + \alpha^{(j)}/MTE_i^{(j)})$, where $MTE_i^{(j)}$ denotes MTE measured by flow j at time i. Similar to FAST, ExLL provides fairness to flows without any penalty in large propagation delays.

4.2 State Transition

ExLL can be implemented either at the receiver or at the sender. The receiver-side ExLL implementation has a significant advantage over the sender-side one as it can work with any server running Cubic as in Figure 9 (a). In order for ExLL receiver to take control by its RWND, the CWND of Cubic at the server should grow sufficiently so that min(*cwnd, rwnd*) is governed by RWND. Therefore, until Cubic increases its CWND by slow start beyond the cellular link bandwidth estimated by ExLL, ExLL stays in *observation* mode. As soon as the CWND grows sufficiently, ExLL receiver exits to control mode and starts to report RWND, computed from Eq. 3, back to the server. If Cubic experiences a packet loss and reduces its CWND below RWND, a recovery logic of ExLL receiver, which will be explained in Section 4.6, detects such an event by checking the difference between its RWND and CWND measured in the receiver. Upon detection, the recovery logic temporarily stores that RWND value and stops updating RWND until CWND of Cubic increases again to exceed that

[6] We use α = 200, γ = 0.5 by default. They work reliably in all experiments.

RWND. When Cubic resets by a timeout, the recovery logic also detects it and lets ExLL restart from observation mode.

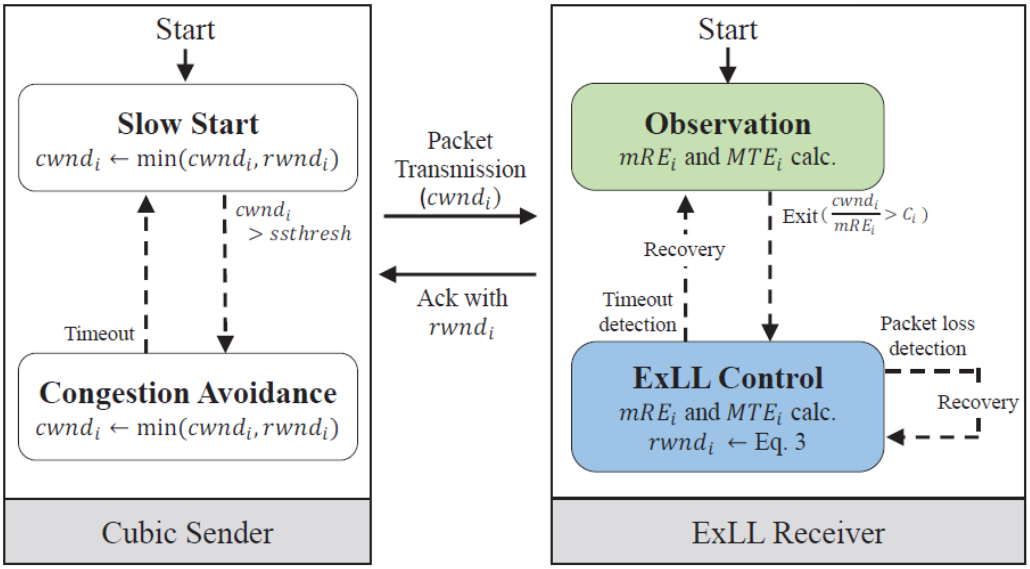
On the other hand, sender-side ExLL can have two design choices. We can let it work by itself similarly to FAST or let it be a plug-in module of Cubic. The former may operate efficiently in the network where all the flows rely on ExLL, but the latter would be preferable if there are cases for ExLL to coexist with Cubic flows. Here, we present the latter option that may give better deployment opportunity. As a plug-in, sender-side ExLL runs Cubic in the background as shown in Figure 9 (b). When a session is initiated, ExLL relies on Cubic to increase the CWND, $cwnd^C$, quickly by slow start. If $cwnd^C$ is determined to grow enough to achieve estimated cellular bandwidth from checking $cwnd^C/mRE_S > MTE_S$, the CWND of ExLL $cwnd^E$ starts to be computed, where mRE_S and MTE_S denote mRE and MTE obtained at the sender, respectively. From then on, ExLL overrides Cubic when $cwnd^E$ is smaller than $cwnd^C$ and vice versa. We explain the detailed computations of mRE and MTE below.

4.3 MTE Calculation

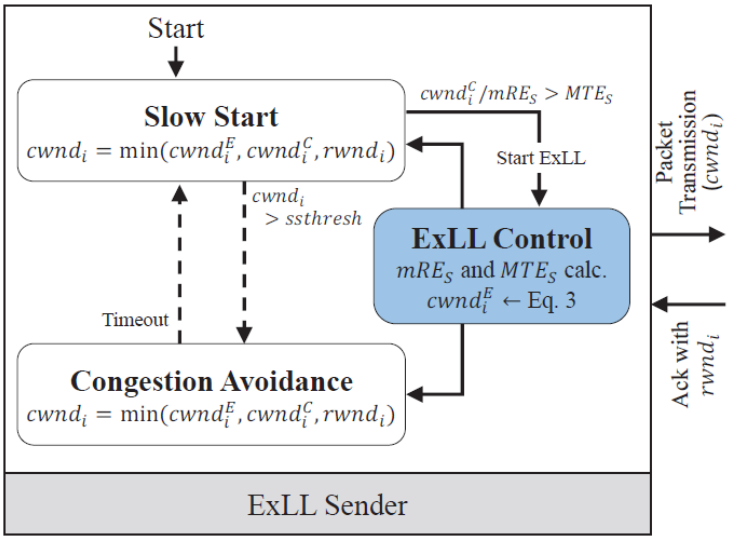
As UEs are scheduled by the BS, changes in the bandwidth of cellular link can be observed more precisely by the UEs than the servers. For the receiver-side ExLL implementation, we obtain MTE from the moving average of $F(\cdot)$, the estimated cellular bandwidth by observing the packet reception during the duration of one radio frame. Because $F(\cdot)$ is calculated at every radio frame except for the radio frames that have no allocated subframes, in most cases the moving average is updated at every 10 ms. In the cellular network with RTT of a few tens of milliseconds, ExLL receiver refreshes MTE several times during one RTT and uses the up-to-date MTE for the RWND computation. As analyzed in Section 3, $F(\cdot)$ is capable of figuring out cellular bandwidth changes much faster than the throughput measurement done by the sender.

For the sender-side ExLL implementation, we estimate the UE's receiving rate by calculating MTE_S from Acks received at the sender as $MTE_S = cwnd/\Delta t$, where Δt denote the time difference between the first Ack arrival and the last Ack arrival for the group of packets sent as CWND. MTE_S estimates how the packets sent as CWND are received in the receiver.

To see the difference in throughput estimation between the receiver-side and sender-side ExLL implementations, Figure 10 compares MTE, MTE_S, and the measured throughput between the server and the UE. MTE estimated in the receiver-side ExLL shows the fastest response in detecting bandwidth changes that are reflected in the measured throughput later. MTE_S, on the other hand, detects the changes faster than the measured throughput, but it is a bit slower than MTE while the difference is marginal.



(a) State transition diagram of Receiver-side ExLL



(b) State transition diagram of Sender-side ExLL

Figure 9: State transition diagrams of the receiver-side ExLL and the sender-side ExLL.

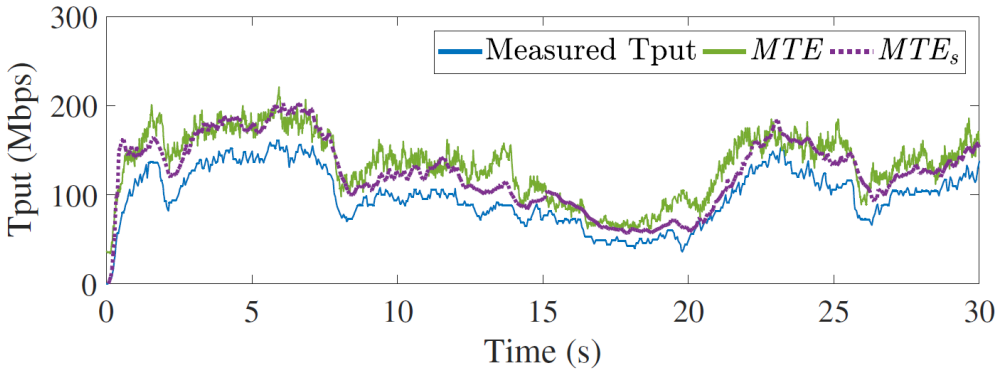


Figure 10: MTE shows the fastest response in detecting bandwidth changes that are reflected in the measured throughput later. MTE_S is slightly slower than MTE but is faster than the measured throughput.

4.4 mRE Calculation

Our finding in Section 3 confirms that setting the minimum RTT, by taking the minimum value among the observed RTT values can mislead the protocol control due to SR periodicity. For the receiver-side ExLL implementation, the minimum and the average per-packet RTT are tracked during observation mode, which are denoted by $mpRTT$ and $apRTT$, respectively. When the ExLL receiver switches to control mode, it first sets mRE as $mRE = mpRTT + D(2 \times (apRTT - mpRTT))$, where $D(\cdot)$ is the function that finds the most matching SR periodicity value either among 5, 10, 20, 40, and 80 ms from the observation of $\hat{T}^{SR} = 2 \times (apRTT - mpRTT)$.

ExLL receiver measures per-packet RTT by the time interval between the reception of a packet whose sequence number is n and the reception of a packet that is brought by the Ack for the packet n (typically from two consecutive packets sent from the sender).

We show a sample run of $mpRTT$ and $apRTT$ during the observation mode of the ExLL receiver in Figure 11. As the figure confirms, for the eNB that use SR periodicity of 10 ms, our estimation $\hat{T}^{SR}$ suggests a very close value to 10 ms, and thus $D(\hat{T}^{SR})$ becomes 10 ms. Table 2 shows the results of multiple SR periodicity estimations done over a commercial LTE network of 10 ms SR periodicity and over our in-lab LTE testbed with SR periodicity settings of 20 and 40 ms. In all cases, ExLL calculates $\hat{T}^{SR}$ that correctly converts to the ground truth SR periodicity. We also test the reliability of SR periodicity estimation in a multi-flow scenario in which an ExLL receiver initiates three download sessions, each lasting 30 seconds, sequentially with the interval of 10 seconds (as depicted in Figure 15). Even with such co-existing flows, we find that all three flows correctly estimate 10 ms as their SR periodicity.

In the sender-side ExLL implementation, the same logic runs with per-packet RTT measurement at the sender.

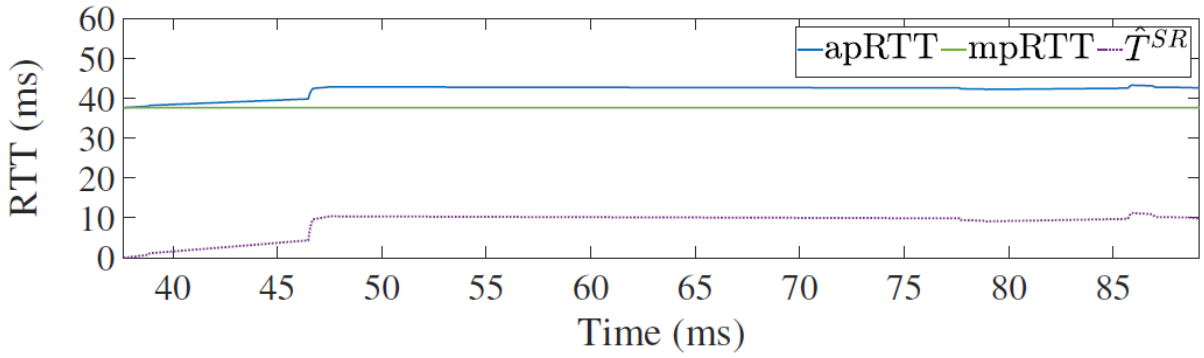


Figure 11: A sample run of $apRTT$ and $mpRTT$ measured at a cellular receiver during the observation mode. $\hat{T}^{\mathrm{SR}}$ estimates 10 ms SR periodicity for the connected eNB.

Table 2: SR periodicity estimation in a real LTE network and from our in-lab LTE testbed

Real LTE Network			In-lab LTE Testbed					
SR periodicity: 10 ms			SR periodicity: 20 ms			SR periodicity: 40 ms		
$mpRTT$	$apRTT$	$\hat{T}^{SR}$	$mpRTT$	$apRTT$	$\hat{T}^{SR}$	$mpRTT$	$apRTT$	$\hat{T}^{SR}$
37.65	42.55	9.8	24.98	34.18	18.4	23.74	40.69	33.9
38.61	42.58	7.9	23.78	35.29	23.0	22.82	40.47	35.3
38.59	43.04	8.9	25.17	37.96	25.6	24.01	41.15	34.3
36.85	41.90	10.1	24.85	33.16	16.6	22.99	41.60	37.2
39.16	42.89	7.5	24.52	36.73	24.4	23.67	41.25	35.2

4.5 Exit from Observation to Control

In order to exit from observation mode, ExLL receiver needs confirmation that the current CWND at the server exceeds the required size to fully exploit the cellular link bandwidth. If the CWND fails to grow that much, it means that the downlink flow has its bottleneck in a non-cellular link. In such a case, ExLL receiver stays in observation mode and lets Cubic running at the server control the flow to be compatible with other competing Cubic flows. It is known by BBR and Copa [6] that when competing with Cubic flows in the shared FIFO queue of a non-cellular bottleneck, a low-latency congestion control flow mostly loses its throughput. Thus, compatibility with Cubic is an important merit of ExLL.

For the cellular bottleneck case, the exact condition to exit from observation mode is as follows: $cwnd_i/mRE_i >$UE-AMBR. If CWND measured at receiver divided by its minimum RTT estimate, mRE_i, is larger than the maximum allowed cellular bandwidth for the receiver, RWND can safely govern CWND with no potential throughput loss. Unfortunately, UE-AMBR is hardly known at the receiver since it is one of operator-configured parameters. Thus, we conservatively use C_i, the estimated channel bandwidth which is the moving average of $C(\cdot)$ at the moment as the substitute of UE-AMBR. Using C_i guarantees no throughput loss.

4.6 Recovery from Loss or Timeout

The role of recovery mechanism is two-fold: 1) when the CWND of Cubic at the sender becomes smaller than the RWND provided by ExLL receiver, which happens mostly due to packet losses, recovery lets

ExLL receiver stop computing and updating RWND until the CWND exceeds the RWND again, 2) when Cubic at the sender initiates a new slow start due to timeout, recovery lets ExLL restart for fresh measurements. Below we describe in more detail.

In most LTE networks where packet losses are nearly perfectly concealed from transport layer thanks to mild MCS selection [9, 18], the measured CWND in ExLL receiver from counting received packets during one RTT matches with its computed RWND as long as RWND is governing the CWND of Cubic at the sender. However, when a packet loss or a timeout occurs, Cubic's CWND at the sender temporarily shrinks, so ExLL receiver can observe the measured CWND is lower than its RWND. Upon this observation, recovery focuses on the size of the measured CWND – 1) if it is equal to the initial CWND of Cubic, recovery determines that a timeout happened and restarts ExLL from its observation mode; 2) otherwise, recovery determines that packet losses occur. It reserves RWND as RWND_{ρ} and stops updating RWND until the measured CWND recovers. During this waiting period, Cubic's control temporarily governs ExLL. As soon as the measured CWND exceeds RWND_{ρ}, RWND computation as in ExLL control equation restarts and ExLL takes the control back. By doing so, we remove potential confusion of ExLL in controlling its RWND. We omit the recovery logic for sender-side ExLL due to similarity.

5. Evaluation

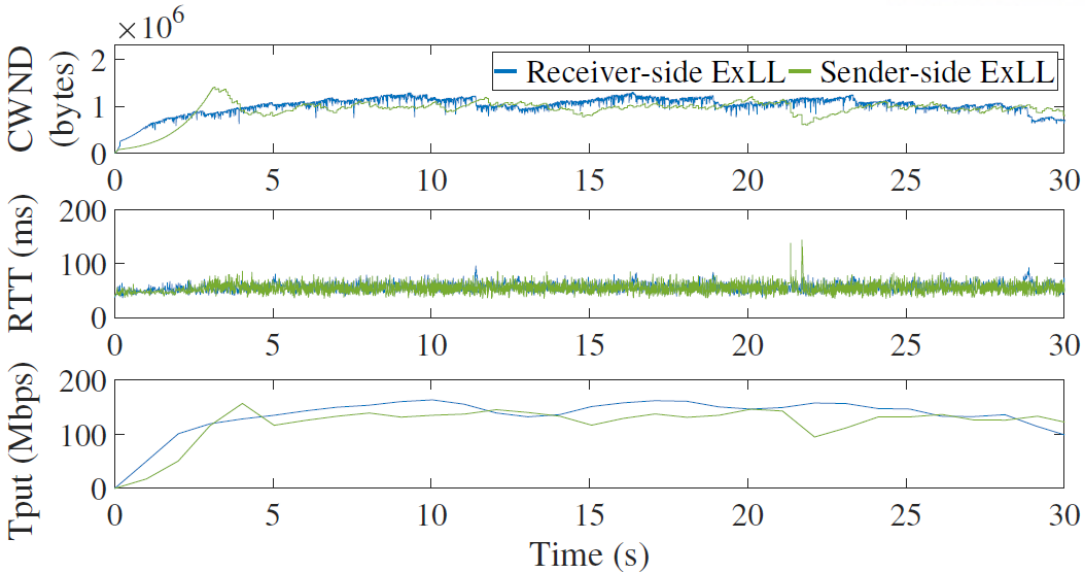
In this section, we first provide a comparison between receiver-side and sender-side ExLL. Then, using receiver-side ExLL, we extensively evaluate ExLL in comparison with other protocols in stationary and mobile LTE networks. We then further examine the performance ExLL in multi-flow and non-cellular bottleneck scenarios. We implement receiver-side ExLL on Android smartphones (Nexus 5X) by patching the kernel. The number of lines added to or modified in the kernel 3.10 of Android 8.1.0 is 327 in total. The modifications are made in two files: tcp_ipv4.c, and tcp_input.c. Sender-side ExLL is implemented by modifying tcp_input.c in Linux kernel 4.13. The number of lines modified is 114. We use TCP Probe installed in the server to monitor and throughput, CWND, and RTT for both of sender-side and receiver-side ExLL.

5.1 Receiver- vs. Sender-side ExLL

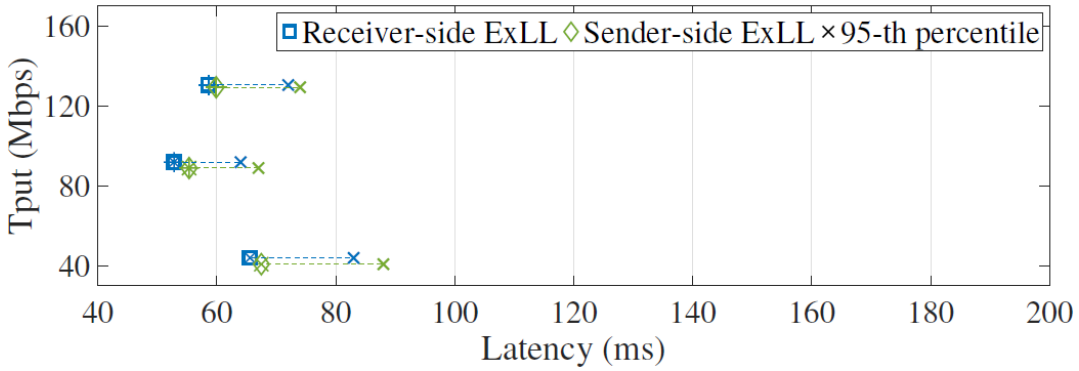
We first compare the behaviors of receiver-side ExLL and sender-side ExLL in Figure 12 (a) from a real LTE network whose RSSI is stable at -90 dBm and minimum RTT and maximum throughput are about 50 ms and 150 Mbps. Both ExLL implementations have similar basic operations except for calculations of MTE, so similar behaviors are indeed observed. They both exit from Cubic at the similar moment and show similar CWND control resulted from Eq. 3. Unlike BBR or PropRate, CWND fluctuation from intentional overbuffering and queue draining does not exist, and thus RTT stays very closely at around its minimum value. Figure 12 (b) summarizes RTT against throughput for both ExLL implementations tested with an eNB with different RTT and throughput variation. It confirms that both implementations perform comparably, though receiver-side ExLL performs a little better as expected. Figure 12 (c) shows another comparison made in a mobile channel whose bandwidth swings between 100 Mbps to 50 Mbps. Thanks to its responsive cellular bandwidth estimation, receiver-side ExLL adapts to the channel very smoothly, hence it shows highly suppressed RTT close to its minimum RTT. Sender-side ExLL also performs quite closely to receiver-side ExLL. Figure 12 (d) summarizes the performance of both ExLL implementations in mobile channels with an eNB with different mobility scenarios, and it confirms they are comparable. For brevity, we only present receiver-side ExLL in the remaining evaluations.

5.2 Performance in Static Channel

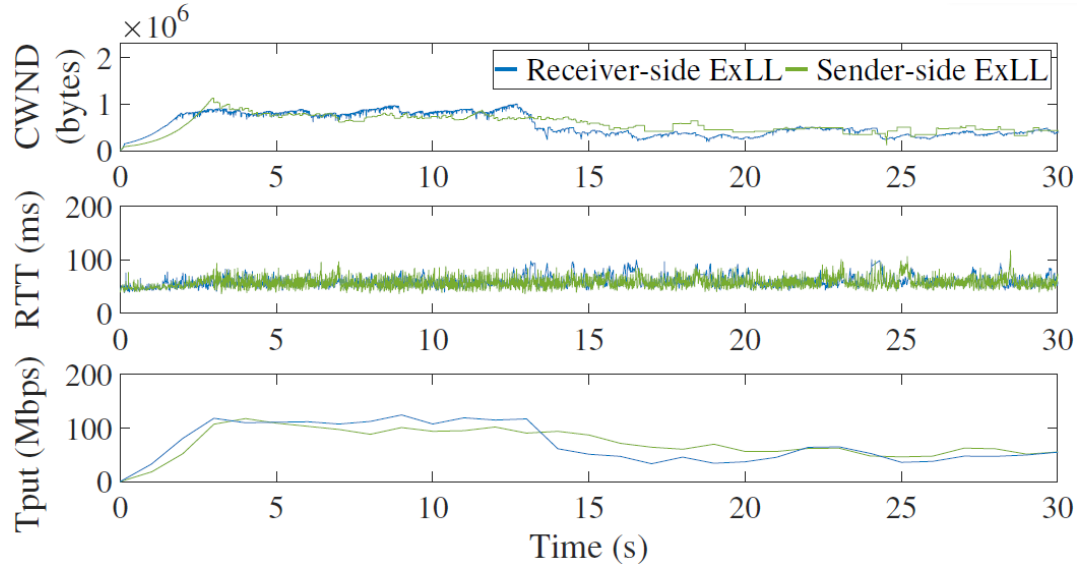
In a static LTE network with 90 Mbps bandwidth and 50 ms minimum RTT, we compare CWND, RTT, and throughput recorded while downloading data from a UE with ExLL and BBR in Figure 13 (a). ExLL



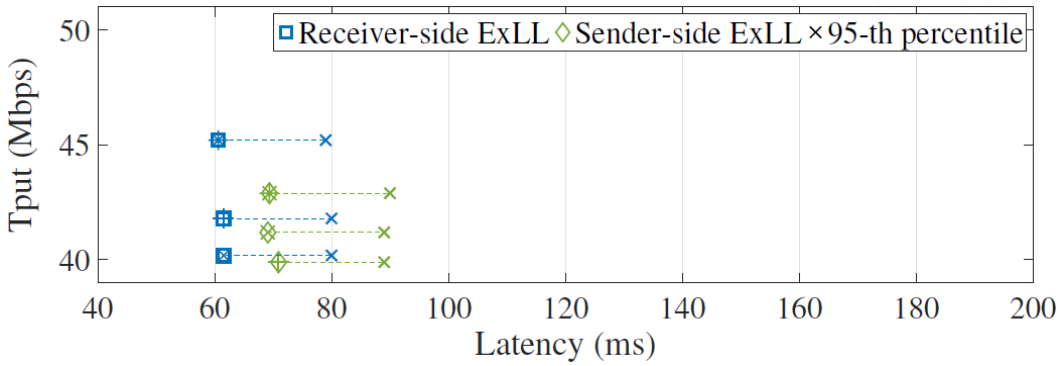
(a) CWND, RTT, and throughput measured in a stationary cellular receiver in a real LTE network



(b) ExLL comparison in a stationary cellular receiver from three LTE eNBs that give different latency and throughput



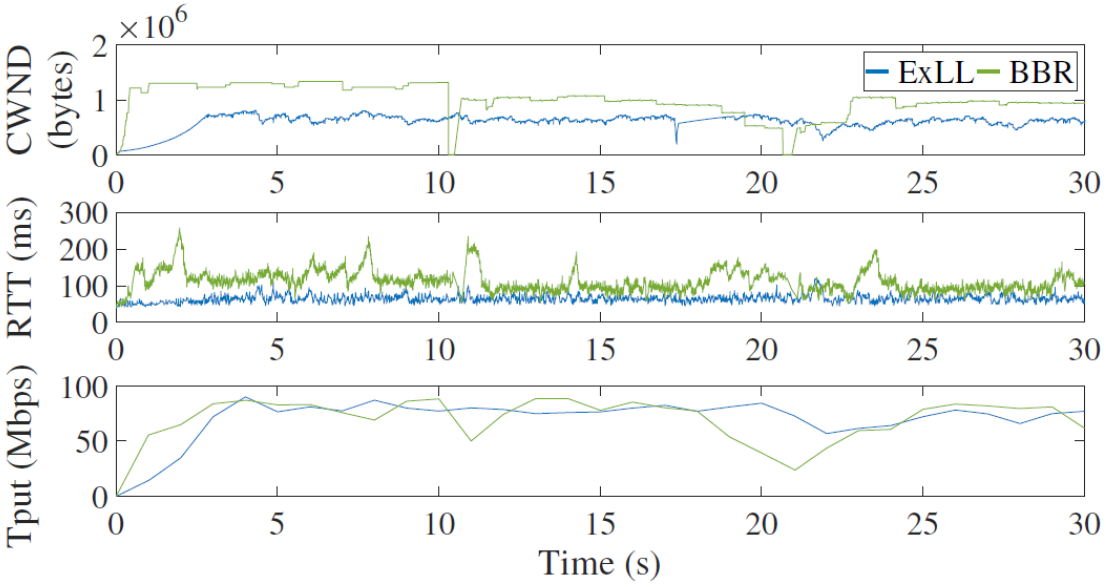
(c) CWND, RTT, and throughput measured in a mobile cellular receiver in a real LTE network



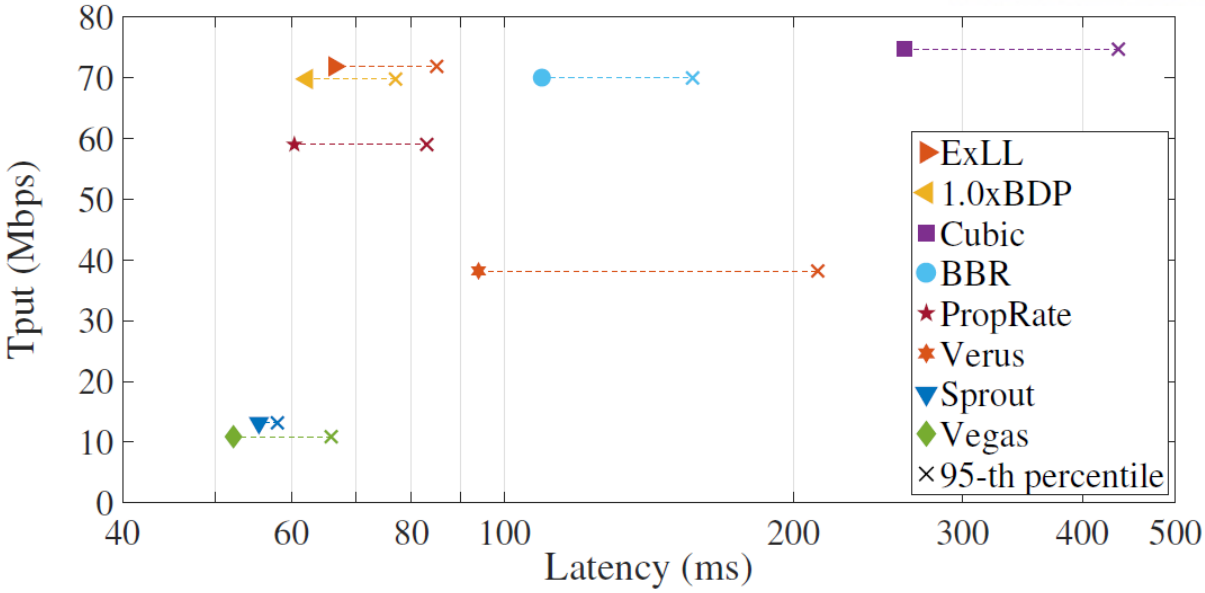
(d) ExLL comparison in a mobile cellular receiver from an LTE eNB with three different mobility scenarios

Figure 12: Congestion control behaviors and performance of receiver-side and sender-side ExLL compared in real LTE networks while the tested cellular receiver is (a) stationary or (c) mobile. (b) and (d) summarize the comparison of throughput and RTT between two implementations of ExLL.

shows much lower average RTT as well as much lower RTT variance compared to BBR. We repeat the same experiment multiple times with other protocols and summarize the throughput and latency performance in a scatter plot, Figure 13 (b). Each protocol has a marking in the graph that represents its average throughput and RTT and another marking connected by a dotted line, which presents the average throughput and the 95-th percentile RTT. As Figure 13 (b) tells, ExLL outperforms others. It provides full throughput as well as a lower and more constrained RTT than other protocols. Also, the result from sending static amounts of congestion window as 1.0 ×BDP that characterizes the ideal performance boundary confirms that ExLL operates nearly ideally in a static LTE channel.



(a) CWND, RTT, and throughput of ExLL and BBR measured in a stationary UE in a real LTE network



(b) Mean and 95-th percentile RTT against the average throughput from ExLL and other congestion controls in a real LTE network

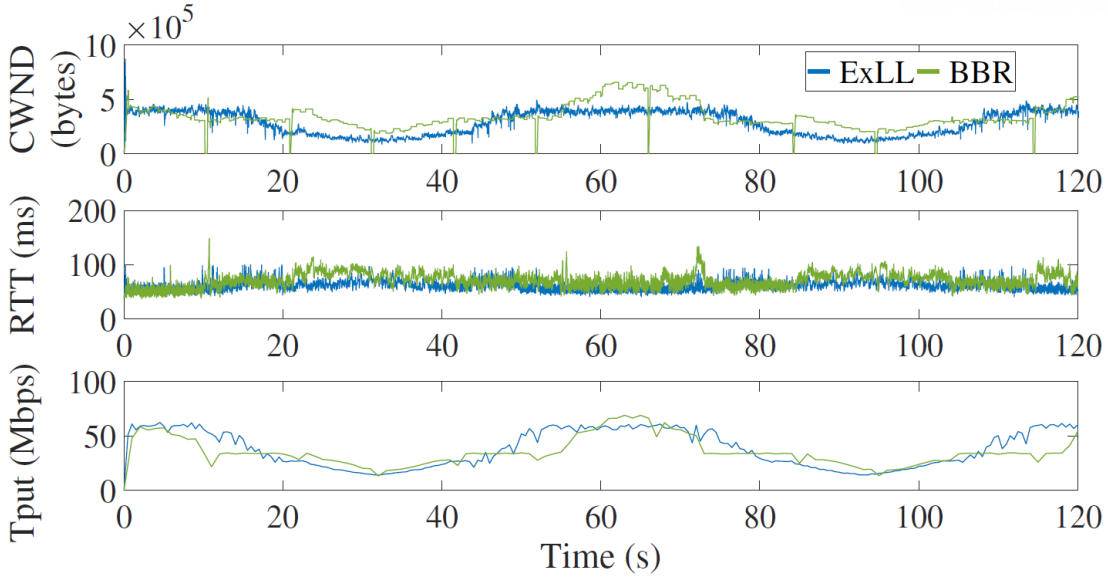
Figure 13: (a) A comparison between ExLL and BBR in a stationary LTE channel, (b) RTT and throughput performance comparison between ExLL and other protocols. ExLL outperforms other low-latency protocols and operates very closely to the ideal performance characterized by sending 1.0 × BDP of the network.

5.3 Performance in Mobile Channel

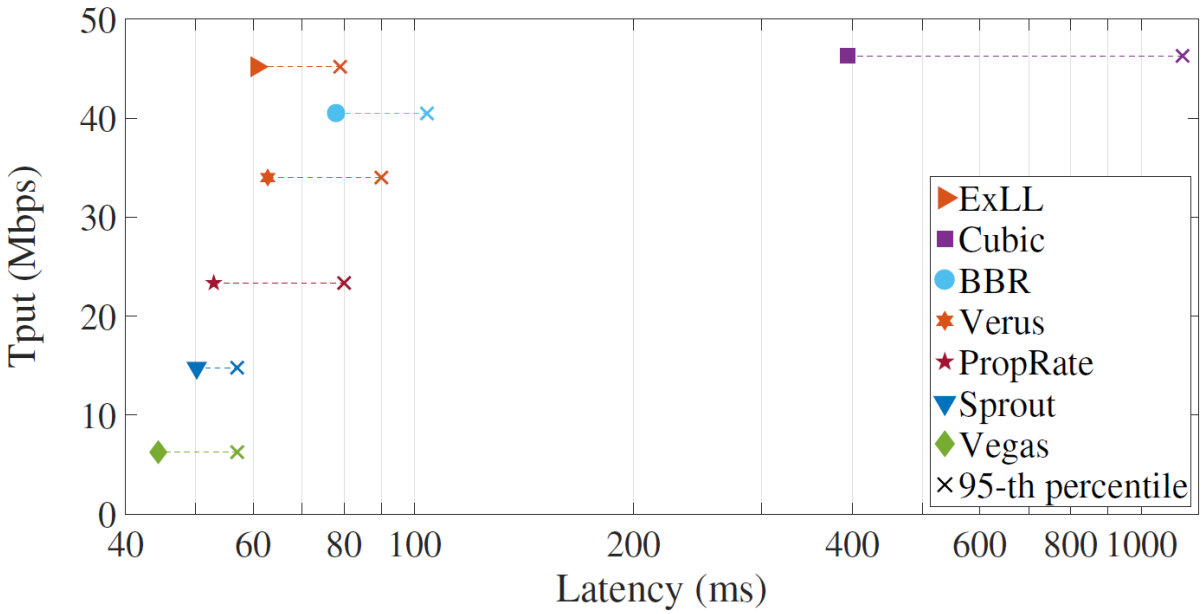
We then conduct similar experiment over a mobile LTE network whose bandwidth bounces between 65 Mbps and 15 Mbps while the minimum RTT stays at around 45 ms. For fair comparison, instead of testing over real LTE networks, we program the signal attenuator of our in-lab LTE testbed and apply exactly the same mobile channel to all protocols. Figure 14 (a), capturing CWND, RTT, and throughput of ExLL and BBR, demonstrates that ExLL has smaller RTT compared to BBR. Especially whenever the channel gets worse, ExLL shows extremely efficient adaptation to the dynamic channel by suppressing RTT significantly better than BBR while not losing throughput. A scatter plot summarizing the statistics from repeated runs in the same mobile scenario with various protocols, Figure 14 (b), evidences that ExLL outperforms other low-latency protocols by non-negligible margins in all aspects: average RTT, 95-th percentile RTT, and throughput.

5.4 Performance with Multiple Flows

In section 2, we show that cellular devices in LTE networks are served by separate bearers. To see the impact of self-inflicted delay (i.e., queueing made in its own bearer), we evaluate the performance of



(a) CWND, RTT, and throughput of ExLL and BBR measured in a UE in a mobile channel

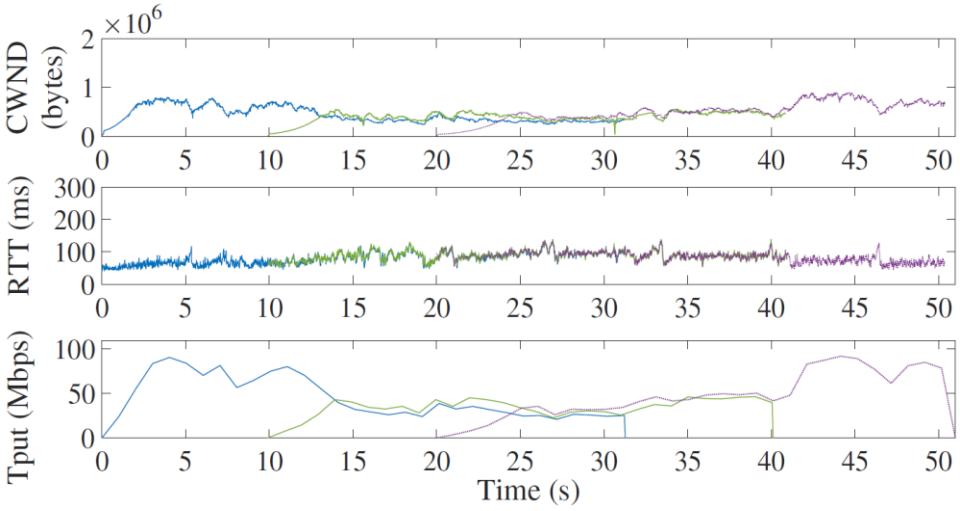


(b) Mean and 95-th percentile RTT against the average throughput from ExLL and other congestion controls in mobile channels

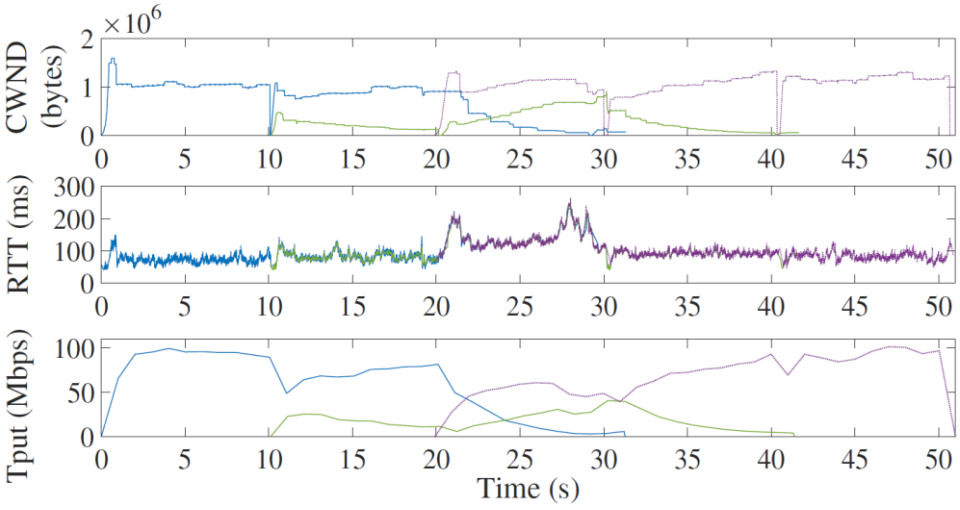
Figure 14: (a) A comparison between ExLL and BBR in the same mobile channel generated by in-lab LTE testbed, (b) RTT and throughput performance comparison between ExLL and other protocols in a mobile channel. ExLL shows nearly 20 ms less RTT compared to BBR while getting throughput as much as Cubic.

multiple flows running between a server and a cellular device over an LTE network of 90 Mbps and 50 ms. Note that BBR is controlled by the server while ExLL here is controlled by the receiver. Since receiver-side ExLL can turn Cubic flows from any servers into ExLL-controlled ones, it guarantees that the receiver has only ExLL-controlled flows in its corresponding bearer. Therefore, based on the

equilibrium data rate of ExLL provided in section 4.1, we can expect throughput fairness between ExLL-controlled flows. Figure 15 shows how ExLL and BBR behave when running three flows that start from different moments. As shown in the figure, ExLL shows much fairer throughput sharing than BBR. Also, ExLL shows much more constrained RTT compared to BBR especially when three flows coexist. ExLL shows about 93 ms while BBR exhibits around 137 ms. We further experiment multi-flow scenarios with different number of flows (n) ranging from 2 to 4 under the same LTE network. We configure that flows start together and last for either 30 or 3000 seconds. Table 3 summarizes the result of throughput fairness between flows by calculating Jain's fairness index [20] that ranges from $1/n$ (unfairest) to 1 (fairest). The fairness of ExLL outperforms Cubic and BBR for short flows. For long flows, ExLL shows comparable fairness performance with Cubic and outperforms BBR.



(a) ExLL: 3 Flow Fairness



(b) BBR: 3 Flow Fairness

Figure 15: ExLL (a) persistently maintains lower latency while giving throughput fairness to second and third flows compared to BBR (b).

Table 3. Jain's Fairness Index from Multi-flow Scenarios

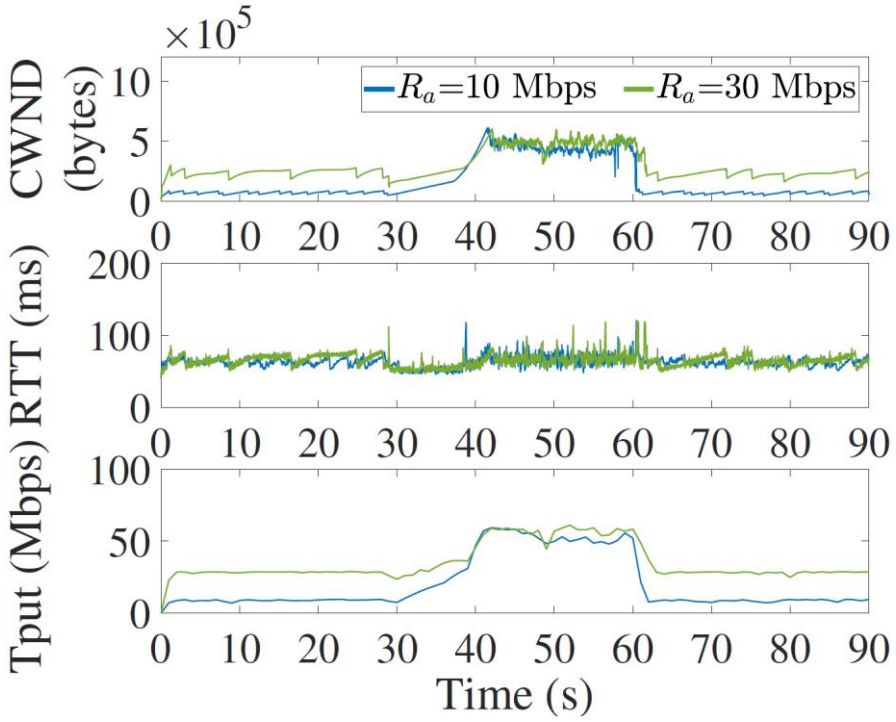
Flow duration: 30 seconds				Flow duration: 3000 seconds			
# of flows	ExLL	Cubic	BBR	# of flows	ExLL	Cubic	BBR
2	0.992	0.919	0.809	2	0.990	0.997	0.846
3	0.985	0.891	0.799	3	0.979	0.988	0.882
4	0.979	0.836	0.807	4	0.978	0.976	0.833

5.5 Non-Cellular Bottleneck Adaptation

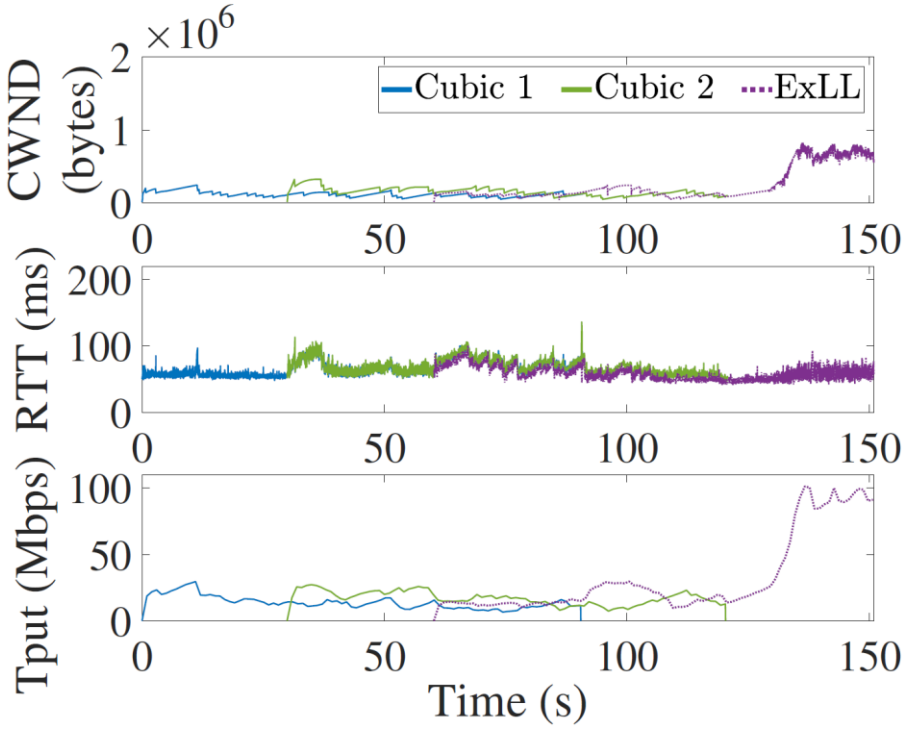
We first demonstrate the adaptability of ExLL to non-cellular bottleneck via the following experiment: we throttle the bandwidth of access link from our server to the Internet by R_a Mbps for 30 seconds, 500 Mbps for 30 seconds, and R_a Mbps again for 30 seconds via *netem* [16] and let a cellular receiver connected to 60 Mbps channel download data with ExLL. We present CWND, RTT, and throughput from the experiment in Figure 16 (a) with R_a of 10 Mbps and 30 Mbps. The figure shows that for the first 30 seconds where non-cellular bottleneck exists, ExLL does not exit from its observation mode and lets Cubic in the server govern the control. For the next 30 seconds where the bottleneck moves to cellular link, ExLL takes control over Cubic. Then, for the last 30 seconds where the bottleneck moves back to the non-cellular link from which packet drops occur, ExLL stops controlling (i.e., stops updating RWND) and lets Cubic take control as explained in Section 4.6. Figure 16 (a) confirms that ExLL adapts well to non-cellular or cellular bottleneck irrespective of R_a values.

We then test the adaptability of ExLL in more complicated scenarios in which 1) a cellular receiver downloads from our server using ExLL via a non-cellular bottleneck link (40 Mbps) that is shared with two Cubic flows (say Cubic 1 and 2) connecting different wired servers and via a cellular link (100 Mbps) and the non-cellular bottleneck later disappears, and 2) a cellular receiver downloads one ExLL flow (say ExLL 1) from our server which experiences a non-cellular bottleneck (40 Mbps) and downloads two more ExLL flows (say ExLL 2 and 3) from another server whose bottleneck forms at the cellular link (100 Mbps). In both scenarios, we let flows start sequentially with the interval of 30 seconds and last commonly for 90 seconds as shown in Figure 16 (b) and Figure 16 (c), respectively with CWND, RTT, and throughput information. Figure 16 (b) shows that when the ExLL flow joins the non-cellular bottleneck which was already occupied by two Cubic flows, it runs as Cubic to keep the fairness, and as soon as the bottleneck disappears, it turns into ExLL and fully exploits the cellular

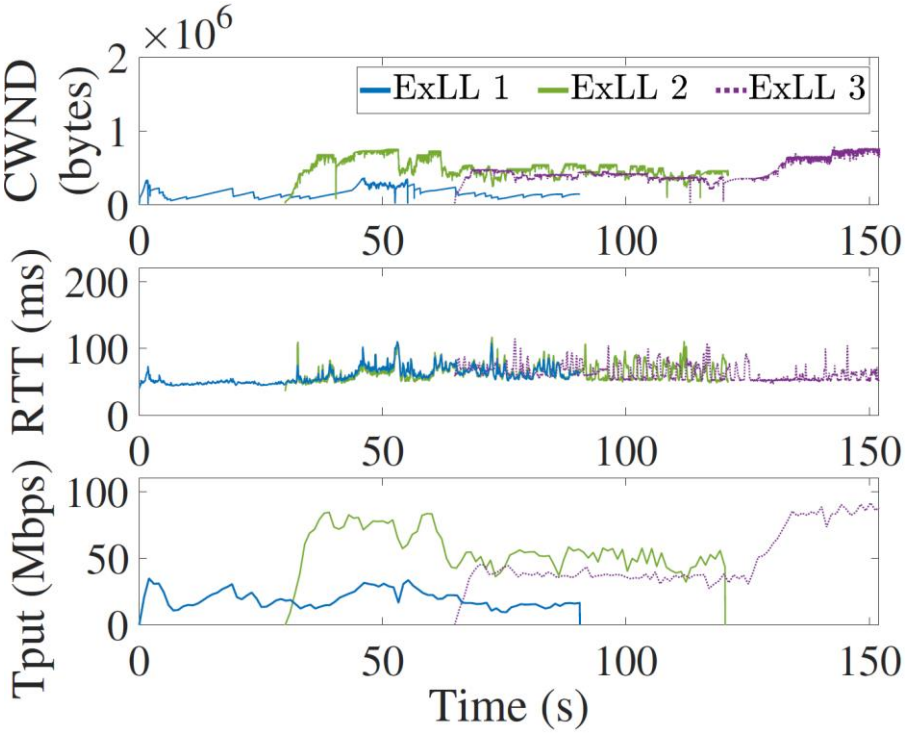
link with low latency. Figure 16 (c) shows that the ExLL flow experiencing a non-cellular bottleneck (ExLL 1) runs as Cubic, but it does not starve by other ExLL flows (ExLL 2 and 3) competing with ExLL 1 in the cellular link as well as it does not make those ExLL flows (ExLL 2 and 3) to starve.



(a) One ExLL flow adapting to non-cellular or cellular bottleneck over time



(b) One ExLL flow and two Cubic flows competing in a non-cellular bottleneck link



(c) One ExLL flow experiencing non-cellular bottleneck competing with two ExLL flows, with no non-cellular bottleneck, in a cellular link

Figure 16: (a) ExLL takes control or hands it over to Cubic adaptively under cellular or non-cellular bottleneck for coexistence with Cubic. (b) ExLL runs as Cubic and achieves fairness when competing with Cubic flows under non-cellular bottleneck. (c) An ExLL flow experiencing non-cellular bottleneck coexists with two ExLL flows with no non-cellular bottleneck in a cellular link.

6. Application Performance

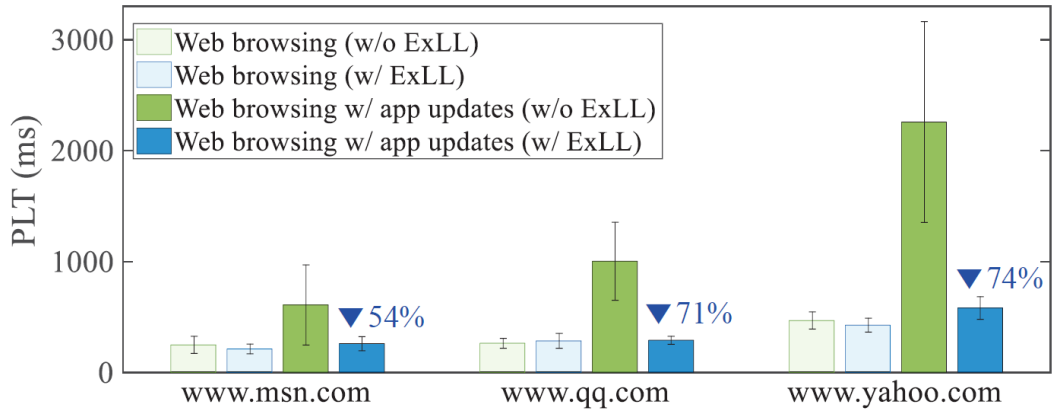
In this section, we evaluate the improvement in quality of experience (QoE) from adopting ExLL in web browsing scenario through QoE metrics: page loading time (PLT) and Speed Index [31]. Unlike other low-latency congestion controls, thanks to its receiver-side design, ExLL allows us to immediately test such improvement from commercial servers running Cubic without touching the servers. In order to measure PLT in a systematic way, we extract the event timing information from Android Chrome browser using Chrome developer tool [11].We define PLT by the time interval from the moment of requesting a new page, $T_{navigationStart}$, to the moment of receiving the last byte of the requested page, $T_{responseEnd}$. Speed Index is a popular page load performance metric that quantifies how fast contents of a page are visibly populated over time. Speed Index can also be obtained from Android Chrome browser using Chrome developer tool [11].

In a similar manner, using the Youtube player library of Android, we develop an Android application that measures the time interval between the moment that a new video is requested by a user while playing another video and the moment that the requested video starts to play after satisfying its required buffering. For the precise measurement of video CST, we record the time stamp from the Android Youtube player API's state change listener, *onVideoStarted()* capturing the first moment of playing the requested video and compare it with the moment of requesting a new video in our test application. We divide types of videos to test as (i) videos with low accessibility (i.e., overseas videos with very low view counts) and (ii) videos with high accessibility (i.e., popular videos on domestic contents with high view counts).

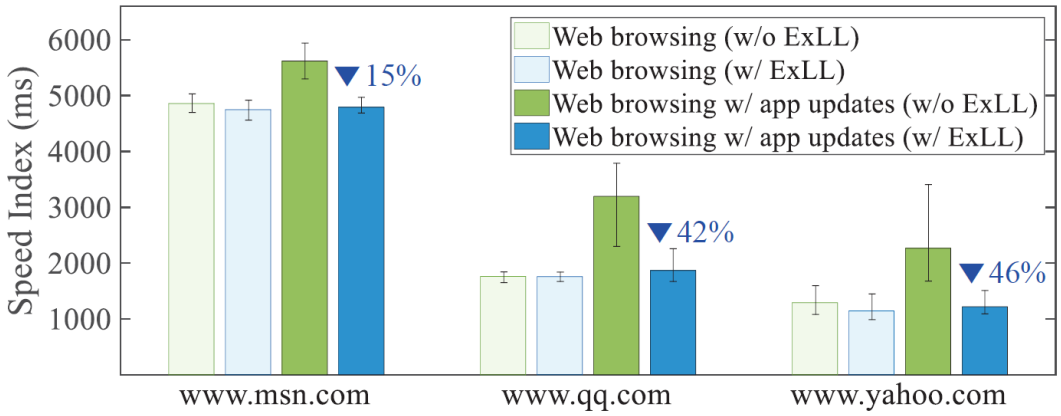
It is known that low-latency congestion controls do not reduce the download duration of a given data because the duration depends more on throughput rather than latency [21]. However, low-latency congestion control can still demonstrate its benefit when the bottleneck experiences bufferbloat. To capture such benefit, in Figure 17 (a), we compare PLT in an Android device for loading three popular web sites with and without ExLL while updating or not updating applications from Google Play Store. When browsing runs only, improvement in PLT with ExLL is minor. But when update coexists, ExLL manages PLT nearly the same as that with no concurrent update. In such a case, ExLL reduces PLT significantly by about 54%, 71%, and 74% from three websites respectively. As shown in Figure 17(b), we can also observe the improvement of ExLL with Speed Index, which is by about 15%, 42%, and 46% from three websites, respectively.

Figure 18 shows the average video CST with and without ExLL. ExLL records 37% and 25% shorter CST for the videos of low and high accessibility. It is impressive to observe that ExLL with a background download session experiences virtually no performance degradation in the CST compared

to the scenario with no background session. This is because ExLL resolves bufferbloat in the network and makes a new video delivered much more quickly to the user device.



(a) PLT as $T_{responseEnd} - T_{navigationStart}$



(b) Speed Index

Figure 17: (a) Average PLT and (b) Speed Index with 95% confidence interval measured from three popular web sites with or without application updates. ExLL substantially improves PLT and Speed Index especially when application updates coexist.

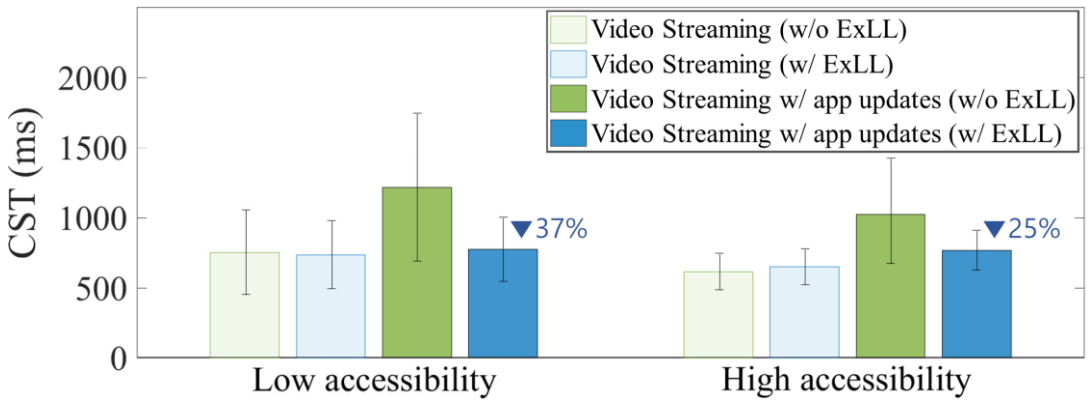


Figure 18: Video channel switching time. ExLL switches to a new video much faster than TCP Cubic when background traffic exists.

7. Related Work

There have been many proposals aiming at achieving low latency and high throughput together. Below we categorize them into three groups by their targeting networks: 1) the Internet, 2) datacenter, and 3) cellular networks.

Low latency for the Internet: There have been delay-based congestion control protocols such as Vegas [8] and FAST [32], and they shed light on designing a protocol that achieves low latency. However, the difficulty involved in tuning parameters across different networks as well as their coexistence issue with loss-based congestion controls have limited their wide deployment. One of the most recent proposals, BBR [10] implements the adaptability by introducing four modes of operation: start-up, drain, probe-bandwidth, and probe-RTT, to estimate time-varying network bandwidth. By controlling its CWND to be close around the estimated BDP, BBR gets substantial improvement in the packet latency. Recently proposed Copa [6] adjusts its CWND towards the target rate which is based on the observed queueing delay under Markovian packet arrival assumption for a range of networks. Its throughput, however, is shown to be much lower than those of both CUBIC and BBR in cellular networks.

Low latency for datacenter networks: In datacenter networks, pHost [12] uses pull-based packet scheduling using token packets generated from receivers in order to minimize the flow completion time. However, it assumes that the congestion in the core is free and the size of flows activated is known in advance. ExpressPass [19] proposes a receiver-driven congestion control in which a sender explicitly controls packet transmission depending on credit packets sent by a receiver. This requires not only network switches with a specific function to adjust the credit packets to be transmitted to the available bandwidth but also additional overhead for sending the credit packets.

Low latency for cellular networks: DRWA [21] proposes a receiver-based congestion control for cellular devices, which roughly controls RWND inversely proportional to the ratio between the current RTT and minimum RTT so that the CWND of TCP sender is cropped by the RWND of TCP receiver. CQIC [24] presents a cross-layer congestion control by directly estimating the channel capacity based on physical layer information (i.e., channel quality indication (CQI) and discontinuous transmission ratio) of the cellular device. Sprout [33] models the cellular network bandwidth as random walk and performs a short-term prediction on the number of packets that can be transferred by the network without incurring additional queueing delays. Verus [36] devises a curve fitting-based delay profiling which maps resulting RTT values into the corresponding CWNDs. Verus finds a relationship between RTT increase and its CWND changes to decide operating points of achieving lower RTTs. CLAW [34] harnesses limited PHY-layer statistics available from LTE smartphones with an analytical model to estimate the achievable cellular bandwidth and uses it to reduce Web loading

time rather than minimizing transport-layer latency. PropRate [23] proposes to directly monitor the bottleneck buffer size in cellular networks by referring to the increase of one-way delay. PropRate also utilizes two modes of operation like BBR which fill up and drain the bottleneck queue to balance latency and throughput.

8. Conclusion

In this work, we proposed ExLL, a new low-latency congestion control tailored for cellular networks, that closely achieves minimum possible latency even in dynamic cellular channels while retaining throughput as much as Cubic. ExLL not only leverages cellular network characteristics such as RB allocation patterns in downlink scheduling and SR periodicity in uplink scheduling but also suggests a refined equation-based congestion control from FAST. Our implementation using Android smartphones shows that ExLL outperforms existing low-latency congestion control algorithms in both static and dynamic channels of LTE networks.

REFERENCES

[1] 3GPP. 2017. Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description (TS 36.300 v14.6.0 Release 14). http://www.3gpp.org/dynareport/36300.htm.

[2] 3GPP. 2017. Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation (TS 36.211 v14.6.0 Release 14). http://www.3gpp.org/dynareport/36211.htm.

[3] 3GPP. 2017. Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures (TS 36.213 v14.6.0 Release 14). http://www.3gpp.org/dynareport/36213.htm.

[4] 3GPP. 2017. General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access (TS 23.401 v14.7.0 Release 14). http://www.3gpp.org/dynareport/23401.htm.

[5] N. A. Ali, A. E. M. Taha, and H. S. Hassanein. 2013. Quality of Service in 3GPP R12 LTE-Advanced. IEEE Communications Magazine 51, 8 (2013), 103–109.

[6] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. In Proc. of USENIX NSDI.

[7] Praveen Balasubramanian. 2017. Updates on Windows TCP. https://datatracker.ietf.org/meeting/100/materials/slides-100-tcpm-updates-on-windows-tcp.

[8] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In Proc. of ACM SIGCOMM.

[9] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda. 2013. Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey. IEEE Communications Surveys Tutorials 15, 2 (2013), 678–700.

[10] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. ACM Queue 14, 5 (2016), 50.

[11] Google Developers. 2017. Chrome DevTools. https://developers.google.com/web/tools/chrome-devtools/.

[12] Peter X. Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed Near-Optimal Datacenter Transport Over Commodity Network Fabric. In Proc. of ACM CoNEXT.

[13] J. Gettys. 2011. Bufferbloat: Dark Buffers in the Internet. IEEE Internet Computing 15, 3 (May-June 2011), 96.

[14] Yihua Guo, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding On-device Bufferbloat for Cellular Upload. In Proc. of ACM IMC.

[15] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a New TCP-friendly High-speed TCP Variant. ACM SIGOPS Operating Systems Review 42 (July 2008), 64–74. Issue 5.

[16] S. Hemminger. 2005. Netem - emulating real networks in the lab. In Proc. of the Linux Conference.

[17] Ravi Netravali Hongzi Mao and Mohammad Alizadeh. 2005. Netem - emulating real networks in the lab. In Proc. of the Linux Conference.

[18] Junxian Huang, 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.

[19] Keon Jang Inho Cho and Dongsu Han. 2017. Credit-Scheduled Delay-Bounded Congestion Control for Datacenters. In Proc. of ACM SIGCOMM.

[20] Raj Jain. 1990. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley and Sons.

[21] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. 2012. Tackling bufferbloat in 3G/4G networks. In Proc. of ACM IMC.

[22] Chris Johnson. 2012. Long Term Evolution In Bullets. CreateSpace Independent Publishing Platform.

[23] Wai Kay Leong, Zixiao Wang, and Ben Leong. 2017. TCP Congestion Control Beyond Bandwidth-Delay Product for Mobile Cellular Networks. In Proc. of ACM CoNEXT.

[24] Feng Lu, 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.

[25] S. Mohan, R. Kapoor, and B. Mohanty. 2011. Latency in HSPA Data Networks. Technical Report. Qualcomm. https://goo.gl/kiEQrJ

[26] NextEPC. 2017. Open source implementation of EPC. https://www.nextepc.org.

[27] Afif Osseiran, Jose F. Monserrat, and Werner Mohr. 2011. Mobile and Wireless Communications for IMT-Advanced and Beyond. Wiley, West Sussex, United Kingdom.

[28] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis. 2016. 5G-Enabled Tactile Internet. IEEE Journal on Selected Areas in Communications 34, 3 (March 2016), 460–473.

[29] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li, and Songwu Lu. 2018. Supporting Mobile VR in LTE Networks: How Close Are We? Proc. ACM Meas. Anal. Comput. Syst. 2, 1 (April 2018), 8:1–8:31.

[30] Jeanette Wannstrom. 2013. Carrier Aggregation explained. http://www.3gpp.org/technologies/keywords-acronyms/101-carrier-aggregation-explained.

[31] WebPagetest. 2018. WebPagetest Documentation. https://sites.google.com/a/webpagetest.org/docs/.

[32] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. 2006. FAST TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Transactions on Networking 14 (December 2006), 1246–1259. Issue 6.

[33] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic forecasts achieve high throughput and low delay over cellular networks. In Proc. of USENIX NSDI.

[34] Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. 2017. Accelerating Mobile Web Loading Using Cellular Link Information. In Proc. of ACM MobiSys.

[35] Swarun Kumar Xiufeng Xie, Xinyu Zhang and Li Erran Li. 2015. piStream: Physical Layer Informed Adaptive Video Streaming Over LTE. In Proc. of ACM MobiCom.

[36] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In Proc. of ACM SIGCOMM.

[37] Xincheng Zhang. 2018. LTE Optimization Engineering Handbook. Wiley, West Sussex, United Kingdom.