공학박사 학위논문

TCP Performance Enhancement in Wireless Networks

무선 네트워크에서의 TCP 성능 향상 기법

2015년 8월

서울대학교 대학원

전기·컴퓨터 공학부

임 희 수

Abstract

TCP (Transmission Control Protocol), one of the most essential protocol for the Internet, has carried the most of the Internet traffic since its birth. With the deployment of various types of wireless networks and proliferation of smart devices, a rapid increase in mobile data traffic volume has been observed and TCP has still carried the majority of mobile traffic, thus leading to huge attention again on TCP performance in wireless networks. In this dissertation, we tackle three different problems that aim to improve TCP performance in wireless networks.

Firstly, we dealt with the downstream bufferbloat problem in wireless access networks such as LTE and Wi-Fi. We clarify the downstream bufferbloat problem in resource competitive environments such as Wi-Fi, and design a receiver-side countermeasure for easy deployment that does not require any modification at the sender or intermediate routers. Exploiting TCP and AQM dynamics, our scheme competes for shared resource in a fair manner with conventional TCP flow control methods and prevents bufferbloat. We implement our scheme in commercial smart devices and verify its performance through real experiments in LTE and Wi-Fi networks.

Secondly, we consider the upstream bufferbloat problem in LTE networks. We clarify that the upstream bufferbloat problem can significantly degrade multitasking users' QoE in LTE networks and design a packet scheduler that aims to separate delay-sensitive packets from non delay-sensitive packets without computational overhead. We implement the proposed packet scheduler in commercial smart devices and evaluate the performance of our proposed scheme through real experiments in LTE networks.

Lastly, we investigate the TCP fairness problem in low-power and lossy networks (LLNs). We confirm severe throughput unfairness among nodes with different hop counts and propose dynamic TX period adjustment scheme to enhance TCP fairness in LLNs. Through experiments on the testbed, we evaluate how much the proposed scheme enhances fairness index.

Keywords: TCP, AQM, bufferbloat, LTE, Wi-Fi, LLN

Student Number: 2010-30229

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

TCP (Transmission Control Protocol), firstly designed to provide a reliable data transfer across the ARPANET in the early 1970s, is a transport layer protocol that the majority of current Internet traffic such as WWW (World Wide Web), e-mail, FTP (File Transfer Protocol), and streaming service relies on. Since its birth, TCP has been evolved along with the advancement of communication technology and the evolution of the network and TCP has still governed the Internet traffic.

TCP has a few key features as follows: i) retransmission of lost packets, ii) ordered delivery, iii) flow control, and iv) congestion control. The essential element is congestion control that is a distributed algorithm to determine how much data each source can inject to the network. TCP congestion control fundamentally aims to achieve high link utilization while coexisting with other flows in a fair manner with the assistance of AQM (Active Queue Management) schemes at intermediate routers. For several decades, various TCP congestion control schemes have been proposed and verified their performances in terms of throughput and fairness.

With the deployment of various types of wireless networks and proliferation of smart devices, a rapid increase in mobile data traffic volume has been observed [1] and TCP has still carried the majority of mobile traffic, thus leading to huge attention again on TCP performance in wireless networks. In early days, TCP congestion control schemes for wireless networks mainly focus on how to distinguish between buffer overflow and random loss due to wireless channel error because a TCP source does not need to decrease its rate when a random loss happens for fully utilizing the link. On the other hand, recent wireless technologies provide very low random loss rate with the assistance of robust link layer retransmission. As a result, queue management rather than classification between random loss and buffer overflow have been drawn much attention in the network community.

Recently, the bufferbloat problem, unnecessary long delay experience due to over-provisioned buffer space, has attracted significant attention since it severely degrades the quality of experience (QoE) of users, especially in multi-core multitasking smartphone systems. In addition, connecting low-power and lossy networks (LLNs) to the

Internet have drawn another attention with the emerging concept of IoT (Internet of Things). Due to the fact that LLNs typically have different characteristics such as low throughput, high packet loss, and frequent topology changes, many technical challenges including TCP will be encountered.

In this dissertation, we dealt with three different TCP problems that can arise in wireless networks. Throughput dissertation, we identify problems based on measurement and evaluate our solutions in real wireless networks such as LTE, Wi-Fi, and LLNs.

1.2 Background and Related Work

There have been several efforts to tackle the persistent queueing, or the bufferbloat problem. The solutions can be categorized into three types according to the location where they work [7]:

Replacing the loss-based congestion detection with the delay-based congestion detection can solve the bufferbloat problem. It has been shown that delay-based congestion controllers like TCP-Vegas [12] and Fast TCP [13] can detect the congestion based on round trip time (RTT) and other delay information. They do not suffer from long delay due to bufferbloat because they throttle *cwnd* before the RTT becomes too large [7]. A main weakness of delay-based approaches is that they suffer from bandwidth starvation when they coexist with loss-based approaches [15]. Mo and Walrand [14] have developed an interesting congestion measure, Decoupled fairness Criteria, using *cwnd* and the measured bandwidth-delay product (BDP). Roughly, they have shown that by setting *cwnd* to the BDP (and thus by controlling *cwnd* irrespective of loss detection) a variety of utility functions can be maximized.

Active queue management (AQM) at intermediate routers can prevent the buffer from bloating [6, 7]. AQM schemes such as random early detection (RED) [16], exponential RED (E-RED) [17], and random exponential marking (REM) [18] discard (or mark if explicit congestion notification (ECN) is enabled [19]) incoming packets in a probabilistic manner before the buffer becomes full, and can inform the TCP senders of incipient congestion such that they can reduce their transmission rate before they experience long queueing delay. Despite many advantages of the queue management, few intermediate routers enable AQM in practice by default due to difficulty in its parameter settings. Recently, a sojourn-time based AQM scheme named CoDeL [6] and lightweight AQM scheme named PIE [45] have been developed to address the bufferbloat problem. However, it remains still unclear how quickly these AQM schemes will be deployed in practice including mobile networks, e.g., LTE networks.

Flow control at the receiver that adjusts its advertised receive window (*rwnd*) can provide an alternative way to prevent bufferbloat. Originally, *rwnd* has been introduced to limit the amount of in-flight data for the receiver to receive it in a reliable manner. However, most modern handheld devices are equipped with enough buffer space, hardly need flow control, and set *rwnd* sufficiently large by default, which is commonly denoted by Auto-tuning [7]. On the other hand, the receiver can utilize *rwnd* to restrict the amount of in-flight data [20]. Most smartphones regulate the maximum amount of in-flight data to prevent the bufferbloat problem by placing a cap on *rwnd* . One of the main advantages of the receiver-oriented approach is that it can prevent the bufferbloat without intervention of service providers and can quickly and easily deployed by updating the firmware of user device. However, since the setting is static and the performance highly depends on the underlying network environment, the problem would linger especially when the user switches the operators or roams. There have been several dynamic controls of *rwnd* in the literature to throttle TCP transmission rate for traffic prioritizing [40], service differentiation [41], quick transfer of background traffic [42], and prefetching web contents [43]. They aim to provide preference to certain flows but are unable to prevent bufferbloat. Recently, a scheme termed DRWA that dynamically changes *rwnd* to solve the bufferbloat problem has been proposed [7]. In DRWA, the receiver adjusts *rwnd* to keep RTT close to its minimum RTT, which is, in principle, similar to TCP-Vegas. Although DRWA successfully prevents bufferbloat, it suffers from significant performance degradation when it competes with other competitive schemes.

In addition, there have been many efforts to improve TCP performance under two-Way TCP traffic in asymmetric networks [46, 47, 53, 54]. *Ack compression*, gives rise to rapid queue fluctuation, was known as the cause of download throughput degradation [62]. Ack prioritization for reducing *ack compression* was proposed [53, 54]. In addition, connection-level bandwidth allocation was proposed [54]. *Data pendulum* is another reason of performance degradation in asymmetric links in the way that only one of buffers are fully utilized at a given time [46]. [47] designed asymmetric queueing (AQ) to improve TCP performance in residential access networks .

In cellular networks (3G/HSPA), proxy-based solutions to tackle two-way TCP traffic in asymmetric links were proposed [48, 49]. Y. Xu *et al.* proposed receiver-side flow control (RFRS) [48] which regulates upload traffic by flow control. TCP receiver-rate estimation (TCP-RRE) [49] was designed to increase download throughput by injecting as much as packets to the mobile devices.

Fair queueing (FQ) was designed to share the link capacity fairly at the gateway. Examples of FQ implementation are stochastic fair queueing (SFQ) [50], weighted fair queueing (WFQ) [51], and class-based queueing (CBQ) [52]. These FQ algorithms were designed to work at the routers or switches; therefore, elaborate parameter settings are necessary to achieve good performance.

1.3 Outline

This dissertation is organized as follows.

In Chapter 2, we consider the downstream bufferbloat and propose a reciever-side scheme to takcle the downstream bufferbloat problem. Based on dynamics of TCP and AQM, RTAC dynamically determines *rwnd* to prevent bufferbloat. Furthermore, RTAC can coexist well with other conventional TCP schemes in a fair manner. We show that RTAC successfully prevents bufferbloat and achieves good performance in LTE and Wi-Fi networks.

In Chapter 3, we deal with the upstream bufferbloat problem in LTE networks. We clarify the negative impact of upload bufferbloat on multitasking users' QoE in LTE networks and design a packet scheduler that aims to separate delay-sensitive packets from non delay-sensitive packets. We also implement the proposed packet scheduler on Android devices and verify its effectiveness through real experiments in LTE networks.

In Chapter 4, we dealt with TCP performance in terms of fairness in low-power and lossy networks. We confirm that TCP throughput unfairness between LLN nodes are severe and propose an algorithm to adjust TX period dynamically to enhance fairness. Through experiments on our testbed, we evaluate the effectiveness of our solution.

We conclude the dissertation in Chapter 5.

Chapter 2

Receiver-side TCP Countermeasure to Bufferbloat in Wireless Access Networks

2.1 Introduction

Long delays in accessing the Internet through wireless mobile networks have been commonly witnessed [4]. One of the main reasons is persistent queues at intermediate routers, in particular, at wireless edge routers such as cellular base stations (BSs) due to their excessively large-size buffers [5]. Low price of memory contributes to the installation of such large-size buffers. Due to the very large buffer space, a TCP connection rarely observes a loss even if it fully utilizes the bandwidth, which causes more packets to be injected into the network through its congestion window (*cwnd*) size.

Extra packets beyond capacity pile up at the buffer and cause excessive delays. This phenomenon, called *bufferbloat*, has been observed empirically in both cellular and wired network environments [7, 8, 9, 10, 11]. Recently, the bufferbloat problem has attracted significant attention since it severely degrades the quality of experience (QoE) of users, especially in multi-core multitasking smartphone systems [7] that are already popular. For example, long-lived TCP flows for such as file downloads give rise to negative impact on delay-sensitive flows for such as mobile games and streaming services because they pile up packets at BSs or APs with oversized buffers.

Bufferbloat can be considered as self-generated congestion that occurs due to a fundamental mismatch between buffer sizes at bottleneck links and loss-based TCP congestion control approaches that have governed the Internet since its birth. Although there have been many advances in the details, some original features such as window-based ACK clocking and loss-based congestion detection still remain effective. For example, TCP CUBIC [31] that has been recently adopted in Linux Kernel 2.6.19 and above still controls data transmission rate upon a packet loss. In general, upon a packet loss, the TCP sender shrinks its congestion window that determines the number of bytes in-flight, and inflates it if a packet is successfully delivered to the receiver. On occurrence of bufferbloat, however, packets will not be lost owing to large-size buffer, so the TCP sender will keep increasing the amount of in-flight data.

In this chapter, we propose a receiver-oriented scheme, named Receiver-side TCP Adaptive queue Control (RTAC), to tackle the downstream bufferbloat problem. Unlike DRWA, RTAC can coexist well with other conventional TCP schemes without performance degradation. In RTAC, the receiver controls *rwnd* in a TCP-Friendly manner according to the dynamics of TCP and AQM. In this sense, RTAC can be regarded as a realization of AQM at the receiver side. We show that RTAC successfully prevents bufferbloat and achieves good performance under resource competing environments. The contributions of this chapter can be summarized as follows:

- We clarify the bufferbloat problem in a resource shared environment like Wi-Fi networks. Different from LTE networks, TCP flows in Wi-Fi networks share the buffer space at access points (APs) and directly compete with each other for the buffer resource.

- We develop a *receiver-oriented* scheme to the bufferbloat problem based on the dynamics of TCP and AQM. It successfully prevents bufferbloat and achieves fair resource sharing with TCP flows of conventional receivers.

- We evaluate the performance of our scheme through experiments. Implementing RTAC on commercially available Android phones (Samsung Galaxy S2 (E120K)), we evaluate its effectiveness on the bufferbloat prevention in both LTE and Wi-Fi networks.

The rest of this chapter is organized as follows. We first overview the dynamics of TCP and AQM in Section 2.2. In Section 2.3, we develop a receiver-side scheme that aims to prevent bufferbloat while achieving fairness under the coexistence with various types of TCP flows. Section 2.4 describes experimental setup and we provide experimental results to evaluate our scheme in Section 2.5. Finally, we conclude this chapter in Section 2.6.

2.2 Dynamics of TCP and AQM

We first describe the dynamics of TCP and AQM in a general network setting. Then we focus on the scenarios where users receive data packets through wireless access links such as LTE and Wi-Fi networks.

We consider a network with a given set of nodes and (wired or wireless) links. Each traffic flow is a TCP connection between a source and a destination. Let S denote the set of all traffic flows in the network. The source of flow $s \in S$ injects data packets into the network at rate x_s, which traverses the pre-determined path that consists of a subset of links L_s. Each flow s is associated with utility function $U_s(x_s)$ that is assumed to be concave and differentiable. Each link has capacity c_l. We denote the set of flows that pass through link l by S_l, and the total packet arrival rate at link l by y_l, i.e., $y_l := \sum_{s \in S_l} x_s$.

It has been known that TCP and its variants are solutions to the following Network Utility Maximization (NUM) problem with different utility functions in [17, 21, 22]:

$$\begin{aligned} &\text{Maximize } \sum_{s \in S} U_s(x_s) \\ &\text{subject to } y_l \le c_l, \quad \text{for all links } l. \end{aligned} \tag{2.1}$$

For example, TCP-Reno is a solution to (2.1) when the utility function is

$$U_s^{Reno}(x_s) = -\frac{a}{RTT_m^2}\frac{1}{x_s}, \tag{2.2}$$

where RTT_m is the RTT without queueing delay and a is a coefficient. Similarly, TCP-Vegas solves this problem with the utility function $U_s^{Vegas}(x_s) = \eta \cdot RTT_m \cdot \log(x_s)$, where η is a coefficient.

We consider the standard dual problem of (2.1), and obtain the Lagrangian function as

$$\begin{aligned} L(\mathbf{x}, p) &:= \sum_{s \in S} U_s(x_s) - \sum_l p_l(y_l - c_l) \\ &= \sum_{s \in S} (U_s(x_s) - x_s q_s) + \sum_l p_l c_l, \end{aligned} \tag{2.3}$$

where p_l denotes the Lagrangian multiplier of link l and q_s denotes the sum of p_l over path L_s, i.e., $q_s := \sum_{l \in L_s} p_l$. Note that the Lagrangian multiplier p_l is often interpreted as the price of link l or the link congestion information such as queue length and loss probability. From the Karush-Kuhn-Tucker conditions (KKT) [39], the optimal utility sum is achieved when $\nabla L(\mathbf{x}, p) = 0$, i.e.,

$$U_s'(x_s) = q_s, \quad \text{for all } s, \tag{2.4}$$

where $U'(x) = \frac{\mathrm{d}U(x)}{\mathrm{d}x}$.

We now focus on typical wireless access network scenarios, where users are connected through LTE and Wi-Fi networks. In this case, wireless links often become a bottleneck due to their limited bandwidth, which explains why bufferbloat typically occurs at edge routers or wireless access links [7]. Hence, in wireless access networks, we approximate the sum price as the price of an access link, i.e.,

$$q_s \approx p_{last\_hop} = f(Q_s), \tag{2.5}$$

where Q_s denotes the queue length at the wireless link of flow s, and $f(\cdot)$ denotes the loss probability function under a given queueing discipline at the BS or AP. For example, the DropTail queueing discipline with maximum buffer space $\bar{Q}_s$ has $f(Q_s) = 0$ if $Q_s < \bar{Q}_s$ and $f(Q_s) = 1$ if $Q_s \geq \bar{Q}_s$.

Each AQM scheme has its own mapping function. For instance, random early detection (RED) has a function that linearly maps the queue length in $[min_{th}, max_{th}]$ into the drop probability in $[p_{min}, p_{max}]$.

Random early marking (REM) and E-RED have an exponential mapping function, which has a good property of presenting the summation of the link prices over the path [17, 18]. Throughout this chapter, we consider a linear mapping function similar to RED, not only because it is easier to understand the dynamics but, more importantly, it has a smaller approximation error than non-linear functions when implemented in the Linux Kernel where floating-point operations are not allowed. To elaborate, we use the following mapping function

$$f(Q_s) = K(Q_s + 1), \tag{2.6}$$

where K denotes the slope of the linear mapping function and will be discussed in detail later, and the extra '1' is added to avoid the potential divide-by-zero problem when $Q_s = 0$.

2.3 Receiver-side TCP Adaptive Queue Control

In this section, we propose our receiver-side solution that aims to coexist with other types of TCP flows in a fair manner as well as prevent bufferbloat. It does not require changes at either TCP senders or bottleneck routers, and thus can be quickly deployed in real networks. We first explain its operations based on TCP and AQM dynamics.

2.3.1 Receiver-side Window Control

Most existing TCP congestion controls show throughput performance compatible with TCP-Reno [31, 32, 33], and a flow is called TCP-Friendly if it achieves throughput compatible with TCP-Reno [23]. In this regard, we try to control the TCP sender to transmit at a rate compatible with TCP-Reno using its utility function (2.2). A different utility function results in a different controller, and may fail to achieve fairness when conventional TCP flows coexist.

We first note that the TCP sender determines its transmission rate by taking the minimum of *cwnd* and *rwnd* [24]. The latter can be exploited to control the transmission rate from the receiver side [7, 25, 40, 41, 42, 43]. Let x_s denote the transmission rate, i.e., $x_s := \frac{w_s}{RTT}$, where $w_s := \min\{cwnd, rwnd\}$. If the transmission rate is constrained by *cwnd*, i.e., if $cwnd < rwnd$, from (2.2) and (2.4), we have

$$\frac{cwnd}{RTT} = \frac{\sqrt{a}}{RTT_m} \frac{1}{\sqrt{q_s}}, \tag{2.7}$$

where the coefficient a is known to be $\frac{3}{2}$ for TCP-Reno [26]. We interpreted the price q_s as the loss probability at the wireless access link. If the wireless access router has the DropTail queueing discipline with excessively large buffer space, the loss probability remains close to 0, which in turn results in an excessively large *cwnd*, i.e., the bufferbloat problem.

To prevent bufferbloat without the modification at the sender and the router, we apply AQM at the receiver side. Suppose that the receiver knows RTT, the minimum RTT (RTT_m), and the queue length Q_s at the access link. From (2.6), the receiver calculates the loss probability (link price) $\hat{q}_s$, and advertises *rwnd* as

$$rwnd = \sqrt{a}\frac{RTT}{RTT_m}\frac{1}{\sqrt{\hat{q}_s}}, \tag{2.8}$$

where $\hat{q}_s = K(Q_s + 1)$ and $a = \frac{3}{2}$. Then from (2.7) and (2.8), the transmission rate is controlled as

$$x_s = \frac{w_s}{RTT} = \frac{\sqrt{a}}{RTT_m} \cdot \min\left\{\frac{1}{\sqrt{q_s}}, \frac{1}{\sqrt{\hat{q}_s}}\right\}. \tag{2.9}$$

Since the actual loss probability q_s is close to 0 due to the large buffer size, the emulated loss probability $\hat{q}_s$ dominates the equation and controls the transmission rate.

Note that the receiver can control the sender effectively only when $cwnd > rwnd$, and thus it can selectively prevent bufferbloat without weakening the sender's capability of congestion control. In other words, when the network is congested, the sender reduces *cwnd* to a smaller value than *rwnd*, and controls its transmission rate. Also since our proposal is based on the dynamics of TCP and AQM, and compatible with previous TCP-Reno variants, it can coexist with other conventional TCP flows in a fair manner. On the other hand, it assumes that the receiver knows the queue length Q_s of the wireless access router, which is not available in practice. In the following, we design the receiver to estimate Q_s using the measured RTT and the estimated transmission window size.

2.3.2 Delay Measurement and Queue Length Estimation

Our receiver-side solution (2.8) requires information about RTT, RTT_m, and the queue length Q_s of the wireless access link, along with appropriate configuration of the parameter K. In this subsection, we describe how the receiver estimates the queue length Q_s from RTT measurement. The configuration of K will be discussed at the end of this section.

The receiver can measure the round-trip time RTT, since TCP connection is full-duplex by default. We denote the measured RTT at the receiver by $\widehat{RTT}$. The RTT without queueing delay can be also obtained by taking the minimum of $\widehat{RTT}$ values over time, as in TCP-Vegas. We denote the estimated minimum RTT as $\widehat{RTT}_m$. The measurement accuracy can be further improved by the TCP timestamp option that is widely used in wireless networks [27] and set on in many Android devices by default.

We now estimate the access link queue length Q_s from the delay information. To elaborate, we use the RTT difference with respect to RTT_m. Under bufferbloat, the access link queueing delay commonly dominates the total queueing delay over the path in wireless access networks, and thus it can be estimated from RTT measurements as $(\widehat{RTT} - \widehat{RTT}_m)$. Then multiplying the queueing delay by the transmission rate, we can approximately obtain[1] the estimated queue length $\hat{Q}_s$ as

$$\hat{Q}_s = (\widehat{RTT} - \widehat{RTT}_m)\frac{\hat{w}_s}{\widehat{RTT}}, \tag{2.10}$$

where the estimated transmission window size $\hat{w}_s$ can be obtained at the receiver using the method in [28]. Note that if we can estimate one-way forward delay from the sender to the receiver as in LEDBAT [38], the estimation can be precise.

The overall procedures in our receiver-side solution, called RTAC, are provided in Algorithm 1, where α and β are the parameters for a running average.

Remarks: RTAC sets *tcp_rmen_max* sufficiently large to remove an artificial constraint on *rwnd* and fully utilizes the link capacity. It is necessary in particular for so-called Long-Fat networks [7]. The receiver's estimations of the transmission window size and RTT may vary a lot, so it is necessary to take their averages with parameters α and β respectively. Interestingly, we observed that $\widehat{RTT}_m$ also varies over time and needs some compensation, which will be discussed separately.

[1]Our end-to-end delay estimation includes all the queueing delay sum over the path, and thus the resultant queue length estimation will be no smaller than the actual queue length at the bottleneck. In a certain environment like with multiple congested links, the estimation could be inaccurate.

Algorithm 1 Receiver-side TCP Adaptive Queue Control (RTAC)

```
On receiving a packet:
1: if (measured $RTT$ is valid) then
2:     $\widehat{RTT} \leftarrow (1-\beta)\cdot\widehat{RTT} + \beta\cdot$(measured $RTT$)
3:     Update $\widehat{RTT}_m$ if necessary.
4: end if

On copying data from receiver buffer to user space:
1: $pkts \leftarrow pkts + copied\_data\_in\_packets$
2: if $(current\_time - last\_update\_time) \geq \widehat{RTT}$ then
3:     $\hat{w}_s \leftarrow \alpha\cdot\hat{w}_s + (1-\alpha)\cdot pkts$
4:     $\hat{Q}_s \leftarrow (\widehat{RTT} - \widehat{RTT}_m)\frac{\hat{w}_s}{\widehat{RTT}}$
5:     $\hat{q}_s \leftarrow K(\hat{Q}_s + 1)$
6:     $rwnd \leftarrow \sqrt{a}\frac{\widehat{RTT}}{\widehat{RTT}_m}\frac{1}{\sqrt{\hat{q}_s}}$
7:     $pkts \leftarrow 0$
8:     $last\_update\_time \leftarrow current\_time$
9: end if
```

2.3.3 Configuration of RTAC

From (2.8) and (2.10), RTAC determines *rwnd* from the three measurement values of $(\widehat{RTT}, \widehat{RTT}_m, \hat{w}_s)$, and a configuration parameter K. The parameter K represents the slope of the linear mapping function of AQM, and affects the operating point of TCP and AQM dynamics. A low value of K results in a large queue length and unnecessary delay, while a high value causes underutilization and even instability [29]. The precise setting of K is beyond the scope of this chapter and needs further study with rigorous analysis. However, we briefly address this issue to provide a rough idea of its setting.

Basically, it is desirable that the solution achieve as small a queue length as possible without underutilizing capacity. This can be achieved by maintaining the *cwnd* between BDP and $2BDP$, which means that the average *cwnd* is around $1.5BDP$ due to its sawtooth-like behavior.

We obtain BDP from multiplying the wireless link capacity by $\widehat{RTT}_m$. Let $\widehat{RTT}^*$ denote the long-term average RTT, i.e., operating point, and let $\theta := \frac{\widehat{RTT}^*}{\widehat{RTT}_m}$. We configure RTAC such that the amount of in-flight data is θ times BDP, i.e., $rwnd = \theta \cdot BDP = \hat{w}_s$. To this end, we obtain the following from (2.6), (2.8), and (2.10):

$$\begin{aligned} \theta \cdot BDP = rwnd &\approx \theta\sqrt{a}\tfrac{1}{\sqrt{K \cdot Q_s}} \\ &= \theta\sqrt{a}\tfrac{1}{\sqrt{K(\theta-1)BDP}}, \end{aligned}$$

which leads to

$$K = \frac{a}{\theta - 1}\left(\frac{1}{BDP}\right)^3. \tag{2.11}$$

Hence, in our settings, the configuration of K depends on θ, which can be considered as the ratio of the amount of total in-flight data to BDP, and needs to be minimized as close as to 1 for better delay performance, subject to full utilization of the wireless capacity. In our experiments, the best performance has been observed for $\theta \in [1.3, 1.4]$.

2.4 Experimental Setup and Configuration

We implement RTAC on Android devices and conduct experiments over two different wireless access networks. One is a public LTE network operated by Korea Telecom (KT), the second largest cellular operator in Korea, and the other is an university Wi-Fi network with Qualcomm Atheros AR93xx. We use Samsung Galaxy S2 (E120K) and modify the manufacturer's open source code [2]. We also implement DRWA on the same platform and compare their performance.

We consider traffic downloading scenarios from a server to users via wireless access networks. The server runs Ubuntu 12.04 LTS over octa-core 3.5 GHz PC and uses the default TCP implementation, i.e., TCP CUBIC with the TCP timestamp option on. Traffic is generated by Iperf [3]. The detailed settings are summarized in Table 3.1.

2.4.1 Receiver Measurement Errors and Configuration

We overview the operation of three different TCP receiver types and discuss their parameter configurations; they are the factory default TCP receiver of Galaxy S2, denoted by Auto-tuning, and the state-of-the-art receiver-side window controller, denoted by DRWA, and our proposed RTAC.

In Auto-tuning, the receiver advertises the *rwnd* as the minimum of twice the measured transmission window size over time and *tcp_rmem_max*[2] which limits the maximum advertised receive window. Letting $\hat{w}_s^*$ denote the largest transmission window estimated by the receiver of flow s during a session, it can be expressed as

$$rwnd = \min\{2\hat{w}_s^*,\ tcp\_rmem\_max\}, \tag{2.12}$$

[2]*tcp_rmem_max* can be checked via 'sysctl net.ipv4.tcp_rmem' command.

Table 2.1: Experimental Setup

TCP Server	Ubuntu 12.04 LTS (Kernel 3.2.0-54)
	Intel(R) Xeon(R) CPU E3-1270 V2 (3.50GHz)
	CUBIC with the timestamp option on
	MSS = 1448 Bytes
	TSO (tcp-segmentation-offload): OFF
	TX-ring = 256 packets

Smartphone	Samsung Galaxy S2 (Model E120K) OS: Kernel version 3.0.8, Android Icecream Sandwich (4.0.3) TSO (tcp-segmentation-offload): OFF TX-ring = 256 packets
LTE networks	Operator: Korea Telecom (KT) Bandwidth: 100 Mbps
Wi-Fi networks	Access Point: Qualcomm Atheros AR93xx Bandwidth: 54 Mbps (802.11g)

TCP receiver settings	
Auto-tuning	$tcp\_rmem\_max = \begin{cases} 2,560 \text{ KBytes for LTE} \\ 2,560 \text{ KBytes for Wi-Fi} \end{cases}$
DRWA	$tcp\_rmem\_max = 2,560$ KBytes $\lambda = 3$
RTAC	$tcp\_rmem\_max = 2.560$ KBytes $\alpha = 1/8, \beta = 1/8$ $\theta = 1.38$ for both LTE and Wi-Fi

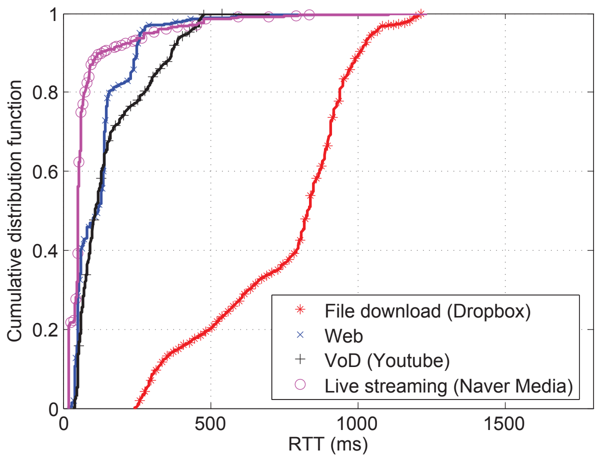


Figure 2.1: The severity of bufferbloat according to applications.

where *tcp_rmem_max* has a different value for LTE and Wi-Fi networks.

DRWA adjusts the receiver window dynamically according to the RTT measurement and the transmission window size $\hat{w}_s$ as

$$rwnd = \lambda \frac{\widehat{RTT}_m}{\widehat{RTT}} \hat{w}_s, \tag{2.13}$$

where λ is a configuration parameter set to 3 in most scenarios [7].

RTAC controls the receive window according to the dynamics of TCP and AQM. From (2.6), (2.8), and (2.10), it sets the receive window as

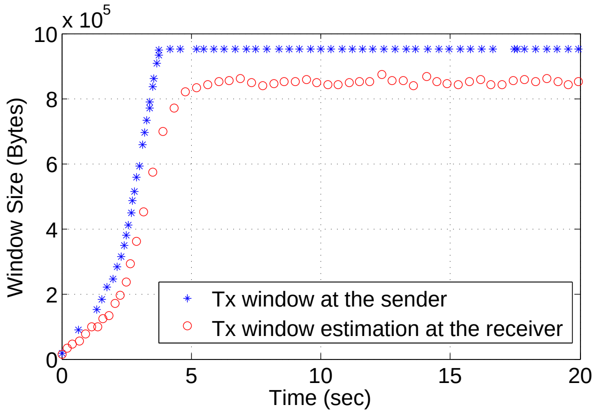
$$rwnd = \sqrt{\frac{3}{2}\frac{\widehat{RTT}}{\widehat{RTT}_m}} \frac{1}{\sqrt{K\hat{w}_s\left(1-\frac{\widehat{RTT}_m}{\widehat{RTT}}\right)+1}}. \tag{2.14}$$

Using (2.11) with $\theta = 1.38$, we set K as $3 \cdot 10^{-7}$ for the LTE network, and $15 \cdot 10^{-5}$ for the Wi-Fi network.

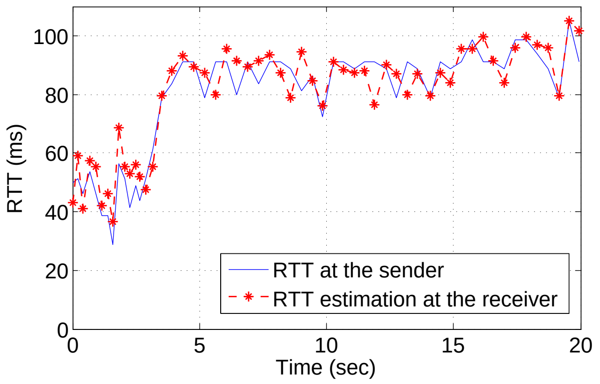
Note that both DRWA and RTAC detect the incipience of queueing at the BS or AP by measuring $\widehat{RTT}$, $\widehat{RTT}_m$, and $\hat{w}_s$, and throttle the transmission window by controlling *rwnd*. If the measurements at the receiver are not accurate, both schemes will suffer in their performance. In practice, RTT estimation at the receiver can be inaccurate in particular when the sender has an application-limited amount of data. In this chapter, we focus on the bufferbloat problem caused by commercial services over LTE/Wi-Fi systems. Fig. 2.1 shows the RTT distribution of TCP flows with different applications: web, VoD streaming (Youtube), live streaming (Naver Media) and file transfer (Dropbox). As shown in Fig. 2.1, it is clear that the long-lived flow such as Dropbox (file transfer) suffers from bufferbloat while the other long-lived flows such as Youtube[3] (VoD), Naver Media (live streaming) and short-lived flows (web) that finishes data transfer with small *cwnd* do not incur bufferbloat. Hence, we focus on long-lived TCP flows like file transfer that is likely to fully utilize link, and assume that the receiver can perform reasonably good estimation on RTT.

Since these measurements are critical for appropriate operation of the receiver-side control, we first evaluate their accuracy. Fig. 2.2 shows that the measurements of $\hat{w}_s$ and $\widehat{RTT}$ at the receiver (after taking the running averages with parameters α and β) are well aligned along with the sender's *cwnd* and RTT. The receiver underestimates w_s by 11% compared to the sender in a saturated region, and makes

[3]The reason that Youtube does not experience large RTT is related to queue discipline (i.e., sched_fq) for scheduling traffic.



(a) Transmission window $\hat{w}_s$



(b) Round-trip time

Figure 2.2: Transmission window and *RTT* at the sender and their measurements at the receiver.

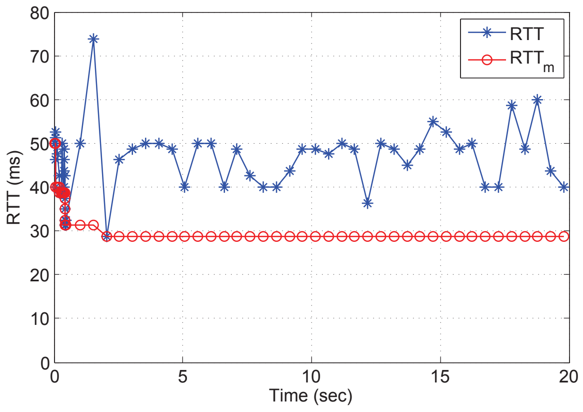


Figure 2.3: RTT and RTT_m in the LTE network with low rate.

similarly accurate estimation on RTT as the sender (4.4% mismatch).

On the other hand, we found that the estimation of RTT without queueing delay (i.e., RTT_m) is a bit troublesome. We inject data packets at a very low rate of 8 Mbps (a seventh of the nominal link capacity) such that packets do not experience any queueing, and measure RTT using the TCP timestamp option. Fig. 2.3 shows the measurements of $\widehat{RTT}$ and $\widehat{RTT}_m$ in the LTE network. Since there is no queueing, the measured $\widehat{RTT}$ is indeed RTT_m. Interestingly, the empirical results show that RTT_m varies over time. We conjecture that the variation is incurred by frame synchronization in LTE, retransmission at the wireless link, deep inspection at the base station, and/or other processing delays. In our measurements, RTT_m is about 50% higher than the minimum RTT measurement ever seen, i.e., $\widehat{RTT}_m$. Since RTT_m is an important factor that determines the BDP and the queueing-delay estimation, such a large mismatch can cause unexpected behavior. To this end, we compensate RTT without queueing delay as $\widehat{RTT}^*_m := \widehat{RTT}_m + \Delta$ and replace $\widehat{RTT}_m$ with $\widehat{RTT}^*_m$. The compensation delay Δ needs to be determined according to the measurements for each wireless access network. In practice, average Δ seems static and it can be derived from measurement statistics, e.g., from [37]. We set $\Delta_{LTE} = 18$ msec for LTE and $\Delta_{Wi-Fi} = 8$ msec for Wi-Fi.

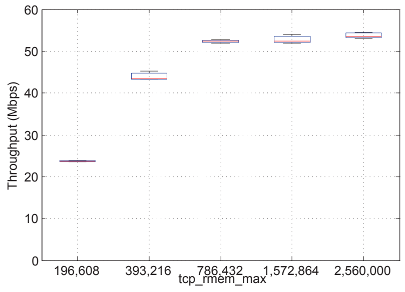
2.5 Experimental Results

In this section, we evaluate the performance of RTAC and other schemes through experiments in WiFi and cellular networks under various scenarios.

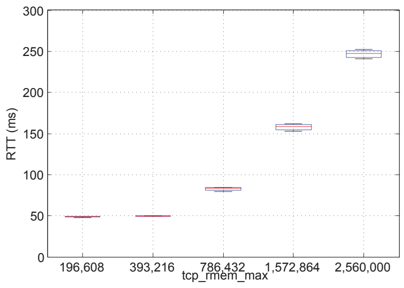
2.5.1 Bufferbloat in Wireless Access Networks

The bufferbloat in cellular and Wi-Fi networks has been noted in the literature [5, 7, 8]. We firstly confirm the previous results in LTE networks and Wi-Fi networks.

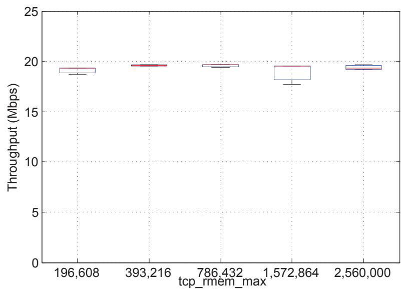
The bufferbloat is closely related to the setting of *tcp_rmem_max* of the TCP receiver as well as the buffer size of the BS or AP [7]. We first measure the throughput and RTT at Auto-tuning receiver in the LTE network for 30 seconds. Figs. 2.4(a) and 2.4(b) show that the setting of *tcp_rmem_max* has a critical impact on the performance. Too small a value leads to low throughput (i.e., underutilization) and



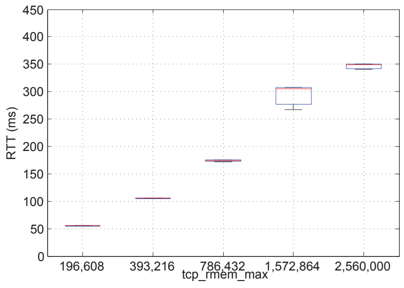
(a) Throughput in LTE



(b) Delay in LTE



(c) Throughput in Wi-Fi



(d) Delay in Wi-Fi

Figure 2.4: TCP performance with Auto-tuning receiver in LTE and Wi-Fi networks for different *tcp_rmem_max* values.

too large a value causes high delay (i.e., bufferbloat). From the results, we can say that the best performance can be achieved when *tcp_rmem_max* is in the range $[400, 800]$ KBytes. However, the optimal value depends on various factors of the underlying access network. Figs. 2.4(c) and 2.4(d) depict the TCP performance in the Wi-Fi network. A similar trend is observed as in the LTE network, with the best setting of *tcp_rmem_max* around 200 KBytes.

We conduct additional measurements to identify the bufferbloat problem in the LTE network. Three client devices are associated with the same LTE eNodeB, and each client measures the delay toward the google server (i.e., www.google.co.kr) by pinging periodically. Two out of three devices initiate downloading using iperf to build up the queue at LTE eNodeB; one starts at 10 seconds, and the other starts at 20 seconds. Fig. 2.5 shows the results. As Client 2 starts downloading at 10 seconds, it experiences additional delay, but the other flows do not, implying that their queues are managed separately. Similarly, when Client 3 starts downloading at 20 seconds, the delay increase of Client 3 shows its queues building up. Note that Client 1 still maintains the lowest delay. Around 20 seconds, the delay of Client 2 is almost doubled since its bandwidth share is halved. These results imply the system performs per-flow queueing for a small number of users.

Wireless available bandwidth can play a role. We place an Autotuning receiver at three different locations. At each location, we mea-

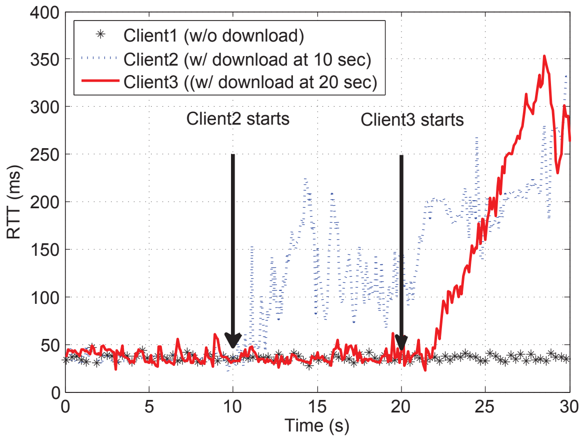


Figure 2.5: Bufferblot in the LTE network.

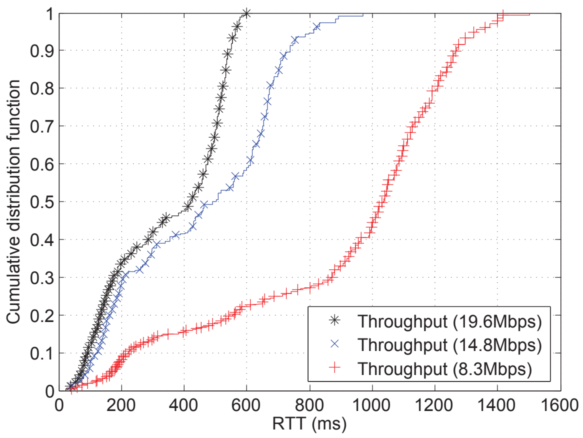


Figure 2.6: Delay distribution in the Wi-Fi network with three different distance settings from AP.

sure the throughput, which is 19.6 Mbps, 14.8 Mbps, and 8.3 Mbps, respectively. Fig. 2.6 shows the cumulative *RTT* distribution of received packets at each location. It is clear that the lower the throughput is, the higher delay the packets suffer from. For the low throughput case, the largest *RTT* measurement was 1.5 sec. Throuhgout this chapter, we conduct most of the experiments in high throughput environment.

Recently, the parameter *tcp_rmem_max* tends to be set higher as new devices are released. For example, the setting of Galaxy S2 (E120K) in the LTE network has $1,220$ KBytes, while Galaxy S3 (E210K) and Galaxy S4 (E330K) set it to $2,560$ KBytes. For the Wi-Fi network, *tcp_rmem_max* for each device is set to 196 (Galaxy S2), $2,097$ (Galaxy S3), and $2,560$ (Galaxy S4) KBytes, respectively. The increasing trend in new devices seems to be related to the increasing capacity of wireless access links. In the sequel, we use *tcp_rmem_max* $= 2,560$ KBytes for Auto-tuning receiver, since it is the default setting of the most recent device of Galaxy S4 for both LTE and Wi-Fi networks. For DRWA and RTAC, we also set *tcp_rmem_max* to the same value.

2.5.2 Prevention of Bufferbloat

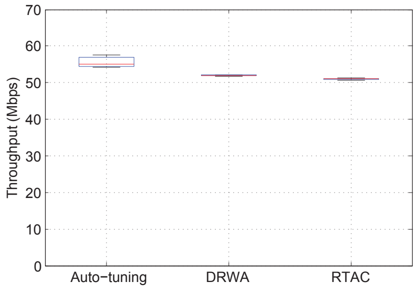
We now investigate the effectiveness of our proposed scheme in bufferbloat prevention. We establish a single downstream TCP connection, and evaluate its performance with different receivers of Auto-tuning, DRWA, and RTAC for 30 seconds.

Fig. 2.7 shows their throughput and delay performance in LTE and Wi-Fi networks. On average, Auto-tuning achieves a throughput of 55 Mbps and a delay of 223 msec in LTE, and 18.9 Mbps and 337 msec in Wi-Fi. DRWA achieves 52 Mbps and 89 msec in LTE, and 18.5 Mbps and 32 msec in Wi-Fi. Also, RTAC shows 51 Mbps and 71 msec in LTE, and 18.8 Mbps and 37.8 msec in Wi-Fi. Both DRWA and RTAC maintain low queue length and fully utilize of the wireless link by controlling *rwnd*, accordingly preventing the downstream bufferbloat.

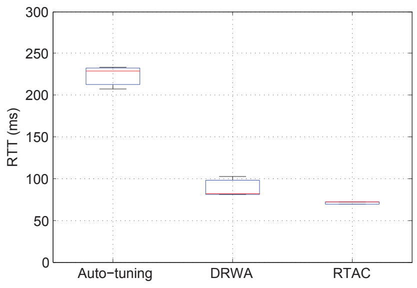
We also compare the performance of Auto-tuning and RTAC with different RSSI scenarios by placing an Auto-tuning receiver at three different locations. At each location, we measure average received signal strength indicator (RSSI) value, which is -51 dbm (high RSSI), -62 dbm (mid RSSI), and -70 dbm (low RSSI), respectively. Since the signal strength changes the link capacity, we have different Ks for RTAC, i.e., $K = 15 \cdot 10^{-5}$ for high RSSI, $37 \cdot 10^{-5}$ for mid RSSI, and $12 \cdot 10^{-4}$ for low RSSI, which has been obtained from (2.11) with $\theta = 1.38$. Fig. 2.8 shows the results, in which Auto-tuning suffers from larger delay in accordance with the signal weaker, while RTAC successfully removes bufferbloat with any signal strengths.

2.5.3 Fairness of TCP Flows with Various Receiver Types

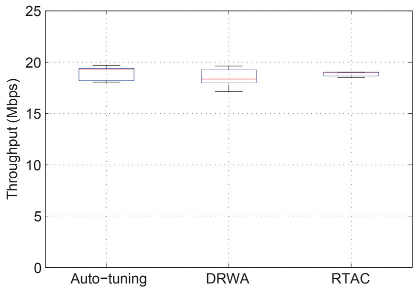
For a single flow, both DRWA and RTAC are successful in preventing the downstream bufferbloat. We now consider the scenarios where



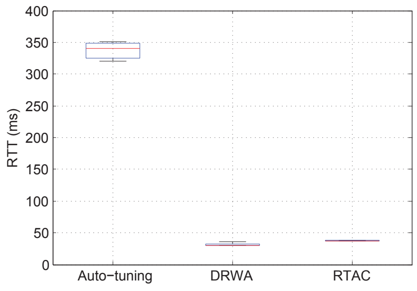
(a) Throughput in LTE



(b) Delay in LTE



(c) Throughput in Wi-Fi



(d) Delay in Wi-Fi

Figure 2.7: Prevention of bufferbloat in LTE and Wi-Fi networks.

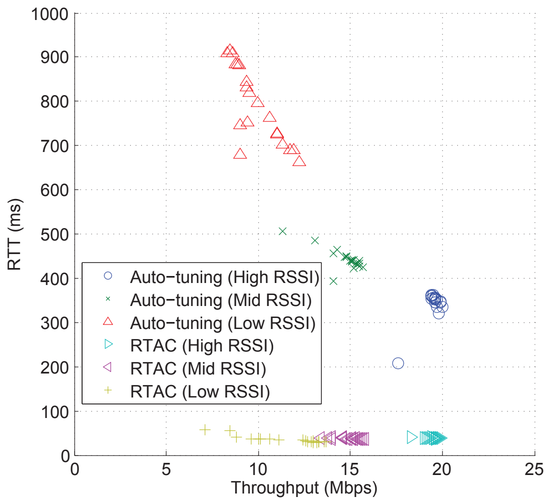
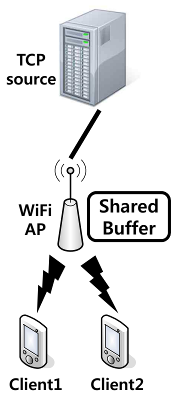


Figure 2.8: Performance of Auto-tuning and RTAC under different Wi-Fi signal strengths.

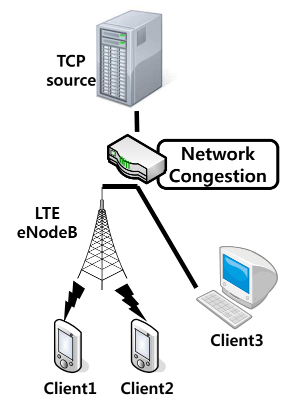
multiple TCP flows coexist and compete for the shared resource. It has been known that the BS in 3G network maintains a separate buffer for each flow [30], while the AP in a Wi-Fi network uses shared buffers for all the flows. Thus, we consider different resource competition scenarios for LTE and Wi-Fi networks, respectively.

We let two wireless clients download data from the server via a common wireless access link. In the Wi-Fi network, the two clients are connected to a single AP as shown in Fig. 2.9(a), and compete for its buffer space. We denote this Wi-Fi scenario as Direct Competition (DC). For the LTE network, however, we cannot make two wireless clients compete directly with each other, since queues at the BS are separately managed for each flow. So, we foster a resource-competitive environment in the wired network part by adding an additional client as shown in Fig. 2.9(b), which creates a wired bottleneck link just before the server. The additional client runs Auto-tuning and competes with the wireless clients through the wired link. We denote this scenario as Indirect Competition (IC).

In the DC scenario in Wi-Fi, we first run Auto-tuning for one client and DRWA for the other. Fig. 2.10(a) illustrates throughput performance of the two TCP flows, in which DRWA achieves only 6% throughput of Auto-tuning. Under the same environment, we replace DRWA with RTAC. The results show that both Auto-tuning and RTAC achieve similar throughput performance (i.e., 96% throughput of Auto-tuning), sharing buffers of the AP in a fair manner. In the IC



(a) Direct competition (DC) in Wi-Fi

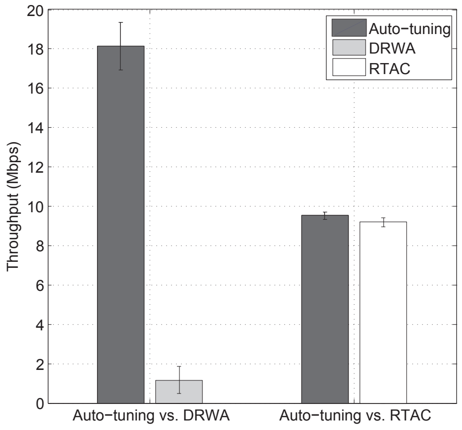


(b) Indirect competition (IC) in LTE

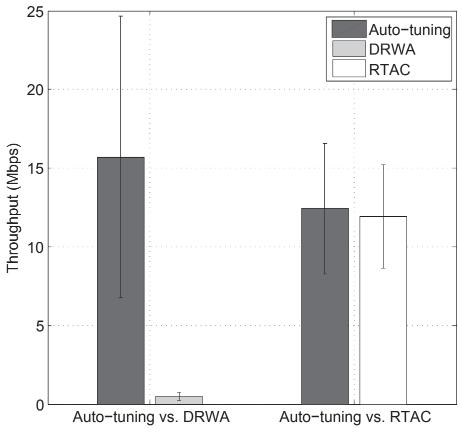
Figure 2.9: Resource competition scenarios with two wireless clients.

scenario of LTE, we observe similar results as shown in Fig. 2.10(b). RTAC achieves 96% throughput of Auto-tuning while DRWA starves severely (i.e., only 2% of Auto-tuning) due to the addition of Client 3 in the wired bottleneck link. From our experimental results, we conclude that DRWA indeed works similar to TCP-Vegas and has the same fairness problem due to its conservative behavior, which makes it less attractive in resource-competitive environments. On the other hand, RTAC always works well owing to utilizing the TCP and AQM dynamics and achieves throughput performance compatible with Auto-tuning.

We also perform experimental measurements of the scenario where three TCP flows (each with Auto-tuning, DRWA, and RTAC receiver, respectively) coexist in LTE and Wi-Fi networks. We extend the DC and IC scenarios in Fig. 2.9 by adding one more wireless client.



(a) Throughput performance in Wi-Fi direct competition (DC)



(b) Throughput performance in LTE indirect competition (IC)

Figure 2.10: Coexitence of RTAC and Auto-tuning in LTE and Wi-Fi networks.

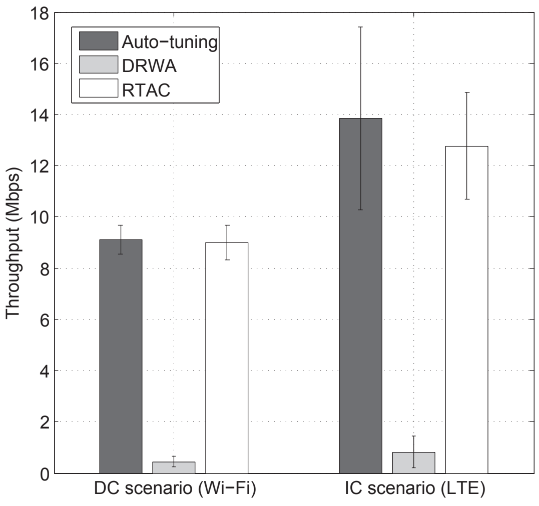


Figure 2.11: Fairness among Auto-tuning, DRWA, and RTAC receivers.

Fig. 2.11 shows the throughput of the three receivers in Wi-Fi and LTE network. Again, in both cases, RTAC is compatible with Auto-tuning while DRWA suffers from severe starvation.

Fig. 2.12 depicts the throughput and RTT measurements of the three different schemes (i.e., Auto-tuning, DRWA, and RTAC) in the DC scenario under the shared-buffer AP. These results show that both RTAC and Auto-tuning achieve similar throughput performance while DRWA suffers from starvation due to non-negligible queueing delay that RTAC and Auto-tuning flows contribute to. Also, the three schemes achieve similar delay performance due to the shared buffer.

Further, we tested different combinations of a RTAC flows + b Auto-tuning flows with (a,b) = (3,0), (2,1), (1,2), (0,3). The results in Fig. 2.13 show the performance difference between RTAC and Auto-

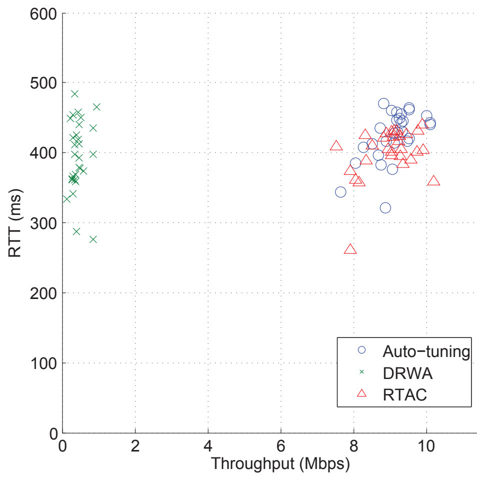
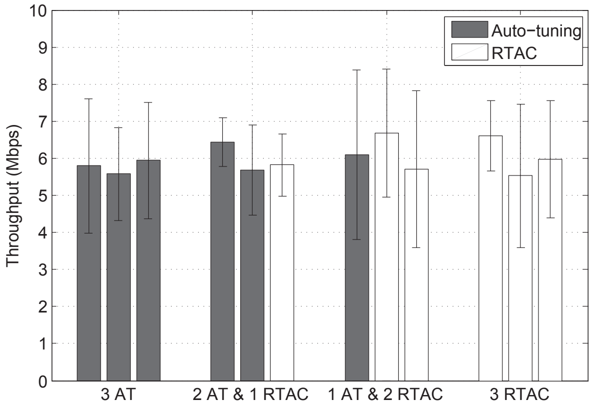


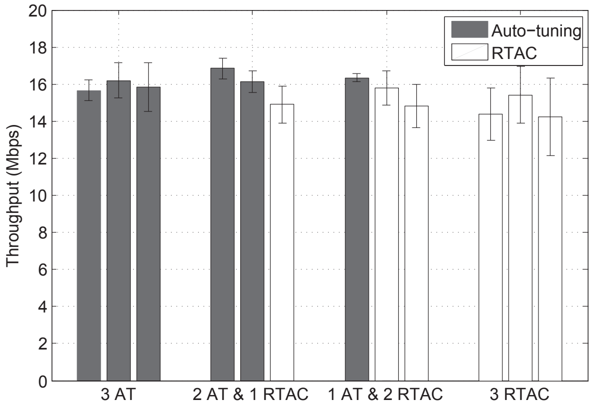
Figure 2.12: Performance of an AP with shared buffer.

tuning flows is marginal, and RTAC coexists with Auto-tuning in a fair manner in both LTE and Wi-Fi networks.

We prepare a Wi-Fi AP (i.e., CISCO AIR-SAP1602I-K-K9) and conduct the same experiments. In the LTE and Wi-Fi network with shared buffer, the delay performance of Auto-tuning, DRWA, and RTAC is similar because the network congestion happens at the same router. On the other hand, *RTT* measurements in CISCO AP shows an unexpected result. As shown in Fig. 2.14, Auto-tuning experiences longer delay than RTAC and DRWA, which is unexpected, since they are supposed to share a single buffer space and experience the same delay. Though additional experiments, we found that the latest Wi-Fi AP such as CISCO AIR-SAP1602I-K-K9 does not manage its buffer space in a shared manner.



(a) Direct competition in Wi-Fi



(b) Indirect competition in LTE

Figure 2.13: Performance with aggregated traffic from different number of Auto-tuning and RTAC devices.

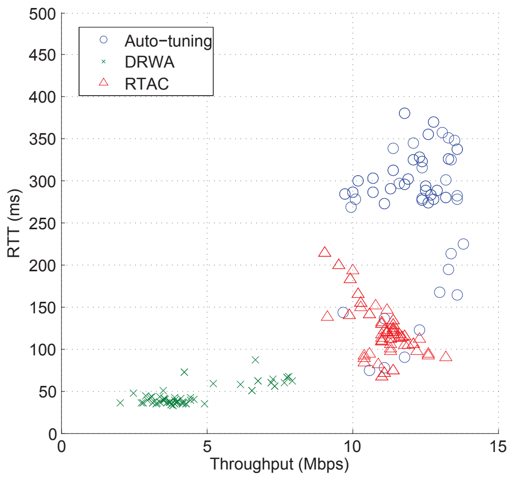
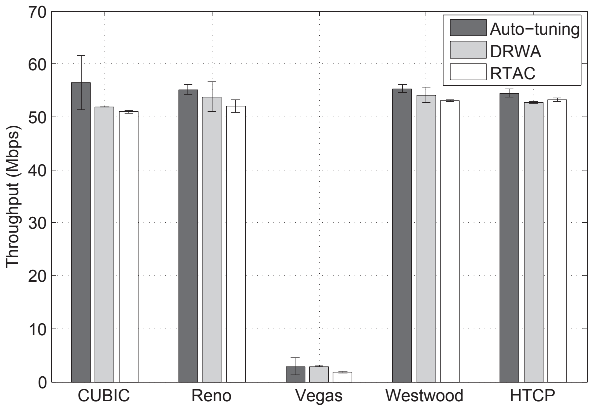


Figure 2.14: Throughput and *RTT* performance among Auto-tuning, DRWA, and RTAC receivers with CISCO AP.

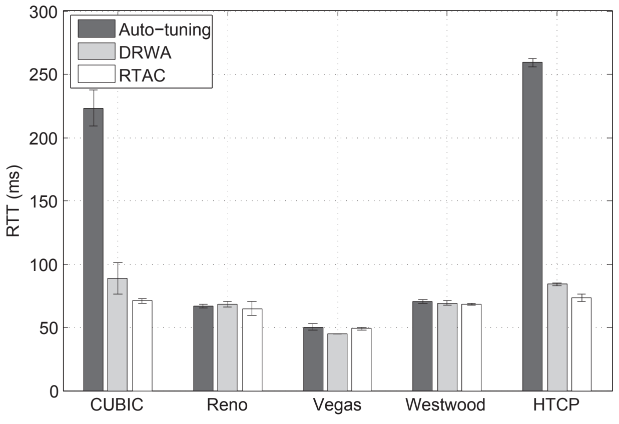
2.5.4 The Impact of TCP Variants

So far, we performed experiments with TCP CUBIC at the sender. In this subsection, we evaluate different receiver-side controllers along with TCP variants at the sender, including TCP-Reno, HTCP, TCP-Vegas, and TCP Westwood. TCP-Reno is a well-known traditional loss-based congestion controller, HTCP is a high-speed variant of TCP (like TCP CUBIC) for Long-Fat Networks [32], TCP-Vegas is a delay-based congestion controller, and TCP Westwood can handle wireless loss and load dynamics by adaptively setting the congestion control parameters using information obtained from the ACK stream [33].

Fig. 2.15 shows the throughput and delay performance for a single TCP flow under various TCP sender types in the LTE network. It demonstrates that DRWA and RTAC are successful in prevent-

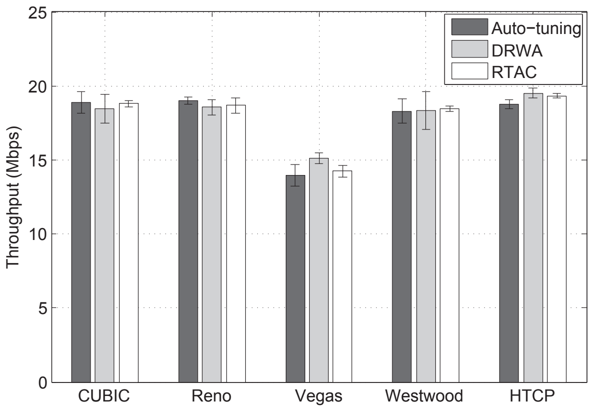


(a) Throughput

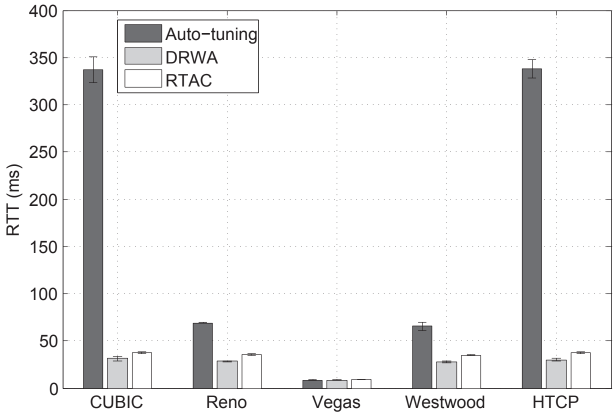


(b) Delay

Figure 2.15: Performance of a single TCP flow with various TCP sender types in LTE network.



(a) Throughput



(b) Delay

Figure 2.16: Performance of a single TCP flow with various TCP sender types in Wi-Fi network.

ing bufferbloat by a single TCP flow, except the TCP-Vegas sender. TCP-Vegas suffers from low throughput regardless of TCP receiver types, which may come from inaccurate estimation on RTT_m. As we discussed before, RTT_m changes over time in the LTE network, and thus without appropriate compensation, the sender will have $\widehat{RTT}_m < RTT_m$ most of the time, which leads to an unnecessarily small *cwnd* for TCP-Vegas.

Another interesting observation is that Auto-tuning suffers from bufferbloat, especially for CUBIC and HTCP that are designed for Long-Fat Networks and thus quickly increases *cwnd* when there is no loss. The results confirm again that an aggressive TCP sender can aggravate bufferbloat.

Fig. 2.16 shows the performance of a single TCP flow in the Wi-Fi network. The overall results are similar to those in the LTE network. TCP-Vegas suffers less throughput degradation than in the LTE network due to a smaller variation of RTT_m, and the bufferbloat of Auto-tuning with TCP-Reno and TCP Westwood becomes more striking.

Finally, we evaluate the fairness between Auto-tuning, DRWA, and RTAC receivers, under different TCP variants at the sender. We consider the same three client scenarios of DC in Wi-Fi and IC in LTE. Fig. 2.17(a) depicts the throughput performance of the three clients for the TCP variants in Wi-Fi. It shows that RTAC achieves more than 94% throughput of Auto-tuning for all TCP variants (except

TCP-Vegas). This means that RTAC and Auto-tuning coexist well. In contrast, DRWA achieves only lower than 13% throughput of Auto-tuning for all TCP variants except TCP-Vegas.

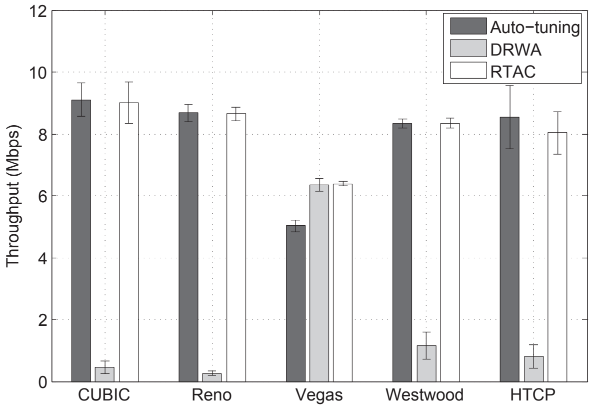
In LTE, Fig. 2.17(b) shows that Auto-tuning and RTAC achieve similar throughputs while DRWA achieves much lower throughput except TCP-Vegas. DRWA works well with TCP-Vegas which is too conservative in competing environments, resulting in very low throughput. Overall throughput performance in the IC scenario is lowered due to the congestion in the wired link.

2.5.5 The Impact of Upload Bufferbloat

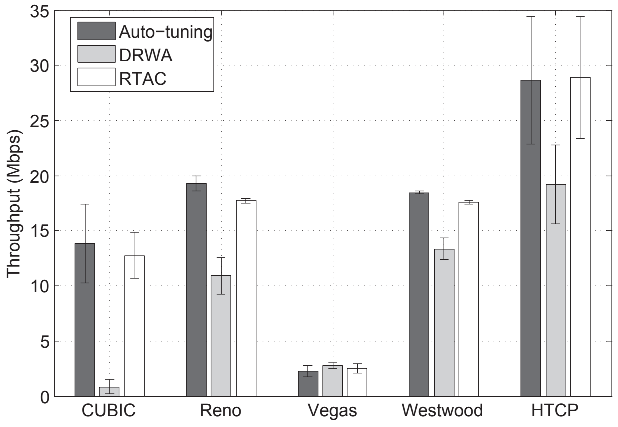
For downstream bufferbloat, RTAC significantly reduces unnecessary RTT increase by controlling *rwnd* dynamically. On the other hand, there can be another problem in the upstream when upload traffic exists. Specifically, upload traffic can build up a large queue at the receiver, which hinders ACKs of download traffic from returning and deteriorates the performance of download traffic [46, 47, 48, 49]. This problem can be solved by applying multi-queue schemes such as fair queueing [44, 50, 51, 52] and priority queueing [53, 54] at user devices.

To elaborate, we conduct additional experiments[4], in which a user device first generates upload traffic toward a server using iperf, and after 10 sec, starts to download TCP traffic using iperf in the LTE

[4]We have used Galaxy S2 with kernel version 3.0, in which Byte Queue Limits (BQL) is not implemented, and we have used soft rate limiting with Hierarchy Token Bucket (HTB).

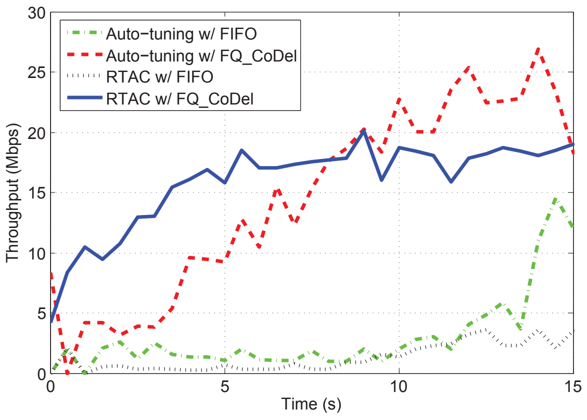


(a) Direct competition in Wi-Fi

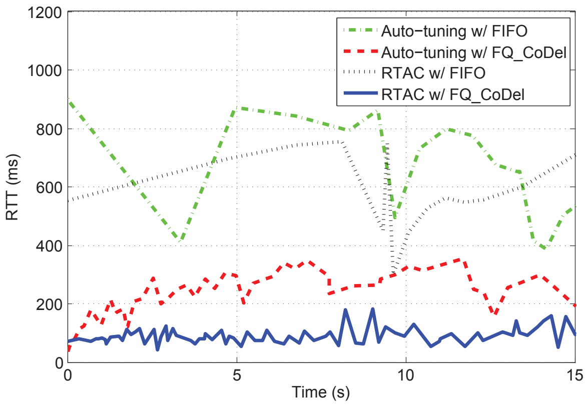


(b) Indirect competition in LTE

Figure 2.17: Performance under competing environments with various TCP sender types.



(a) Download Throuhgput



(b) Download Round-trip time

Figure 2.18: Download performance of RTAC with FQ_CoDel in LTE network.

network. Fig. 2.18 shows the performance of the download traffic in terms of throughput and RTT under each combination of Auto-tuning/RTAC and FIFO/FQ_CoDel. In a nutshell, the upload traffic builds up a large queue at the user device with FIFO queue due to limited wireless capacity, which suppresses the download traffic by delaying returning ACKs, resulting in low throughput and high delay. A multi-class queue discipline like FQ_CoDel [44] can solve the problem by processing the returning ACKs separately from the upload traffic as shown in Fig. 2.18.

Note that, however, FQ_CoDel at the user device does not completely remove the delay as shown in Fig. 2.18(b). Auto-tuning with FQ_CoDel still has a large delay of up to 380 ms, which is indeed due to the bufferbloat at eNodeB. Our results demonstrate that RTAC can be used along with FQ_CoDel, further improving the delay performance by removing the bufferbloat over the download path.

2.5.6 Coexistence with the Unlimited Sender

When an unlimited TCP flow coexists with limited TCP flows (i.e., flows with $tcp\_rmem\_max = 2,560\ KBytes$), the unlimited sender fills up the buffer space, and overwhelms other limited flows. To clarify its impact, we experimented with two flows: one RTAC (limited) flow and one unlimited flow, where the unlimited flow is prepared by setting its *tcp_rmem_max* to a large value (i.e., 7.6 MBytes). We evaluate their performance in both DC (Wi-Fi) and IC (LTE) scenarios. Fig. 2.19

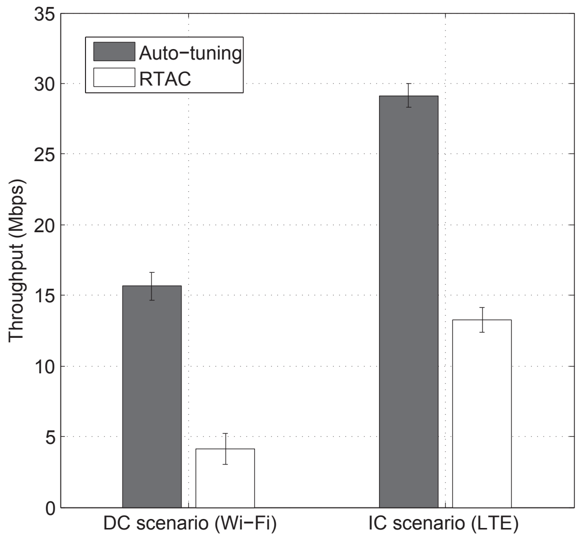


Figure 2.19: Impact of a flow with unlimited *rwnd*.

shows their throughput performance. In the IC (LTE) and DC (Wi-Fi) scenarios, the unlimited sender pushes RTAC aside by injecting more packets aggressively, as we expected.

2.6 Summary

In this chapter, we have presented a receiver-side countermeasure, named RTAC, to address the downstream bufferbloat problem in wireless access networks such as LTE and Wi-Fi networks. According to TCP dynamics, a RTAC receiver estimates an appropriate amount of in-flight data for a wireless access link, and controls the transmission rate via advertised receive window. We have implemented RTAC on commercial smartphones and conducted extensive experimental measurements. We have verified that RTAC successfully prevents bufferbloat, and achieves good delay performance without sacrificing throughput performance nor fairness with conventional TCP flows, outperforming the state-of-the-art schemes. Furthermore, we have showed that RTAC can be incorporated with per-flow queueing like FQ_CoDel to alleviate the upstream bufferbloat problem. There still remain interesting open problems. For instance, a TCP connection may be established between a far-away sender and a receiver across the Internet. In this case, packets go through multiple hops, which results in noise on our queue length estimation and performance degradation. Hence, an accurate estimation on the bottleneck queue length in various scenarios will be necessary to extend our scheme. In addition, precise estimation on network status such as BDP and RTT should be investigated to cope with a rapid change of wireless available bandwidth.

Chapter 3

Dual Queue Approach for Improving User QoE in LTE Networks

3.1 Introduction

With the evolution of hardware and software technology, smart devices such as smartphone and tablet based on multi-core architecture enable users to experience multitasking. For example, it is hardly surprising that smart device users are able to enjoy mobile messenger services while they are uploading data to social network service.

Recently, users tend to generate more upload traffic than before [59]. For example, the statistics of the most-used smartphone applications in 2013 [60] shows that smartphone users frequently use social network services (Facebook, Google+, and Twitter), mobile messenger services (WeChat and Whatsapp), m-VoIP service (Skype), which incur a large volume of upload traffic. In addition, the increasing popularity of the cloud storage service (e.g. Google Drive and Dropbox) has caused more upload traffic.

Performance like throughput and delay in multitasking scenario is affected by network resource allocation. Specifically, one of the resource allocation is related to queuing discipline (qdisc), which determines how to enqueue and dequeue packets on each application at end hosts or intermediate nodes such as routers and cellular base stations (BSs). Previous works have pointed out that performance degradation in multitasking scenarios can happen in both wired [46, 47, 53, 54] and cellular networks [7, 48, 49]. Fundamental reason of performance degradation in multitasking scenarios lies in qdisc based on single queue architecture. In other words, background traffic is able to occupy most of the buffer, thus leading to unnecessary large delay on foreground traffic. For example, when a smartphone user is running a mobile game and uploading data to the cloud storage server simultaneously, packets of the mobile game is delayed due to the stacked packets of upload data in the transmit queue.

In this chapter, we propose a new metric, called sendbuffer occupancy ratio (SOR), for packet classification to improve user QoE in multitasking scenarios. The proposed scheme basically adopts dual transmit queue like [46, 47, 53, 54] and can achieve better delay performance by classifying between delay-sensitive packets and non delay-sensitive packets using SOR without computational overhead. Through extensive measurements in LTE networks, we show that the proposed scheme significantly reduces the delay of delay-sensitive applications.

The contributions of this chapter are summarized as follows:

- We clarify the severity of delay increase of multitasking users in LTE networks through extensive measurements.
- We develop a SOR based packet classification algorithm (SORPC) that aims to improve multitasking users' QoE. Based on priority queueing, the proposed scheme achieves better delay performance by separating delay-sensitive packets from non delay-sensitive packets.
- We evaluate the performance of the proposed scheme through experiments. We implement the proposed scheme on commercial Android phones (i.e., Galaxy S2 (E120K)) and verified its effectiveness with commercial applications in LTE networks.

The rest of this chapter is organized as follows. We first present the performance degradation in multitasking scenarios in LTE networks in Section 3.2. Section. 3.3 describes our proposed scheme that aims to improve user QoE in multitasking scenarios. Experiment results to evaluate the proposed scheme follow in Section 3.4. Finally we conclude this chapter in Section 3.5.

3.2 User QoE Degradation in Multitasking Scenarios

In this section, we first present the severity of upload queueing delay and briefly show its negative impact on users' QoE in multitasking scenarios.

3.2.1 Unnecessary Large Upload Queueing delay

Unnecessary large delay in traffic downloading scenarios from a server to users in cellular networks have been observed [7, 8] due to large size buffer at base stations, and it is known as *bufferbloat* [5]. On the other hand, we consider traffic uploading scenarios from a user to a server and conduct measurements over LTE network operated by Korea Telecom (KT) using iperf [3] as a traffic generator with TCP as transport layer protocol.

We first measure the upload throughput and RTT for 30 seconds for different TCP sendbuffer of which maximum value is limited by system parameter *tcp_wmem_max*[1]. Fig. 3.1 shows throughput and RTT according to *tcp_wmem_max*. As *tcp_wmem_max* increases, throughput stays around 20 Mbps; however, RTT increases more than 400 msec when *tcp_wmem_max* is set as 2,560 KBytes. The optimal throughput and delay performance can be achieved when *tcp_wmem_max* is smaller than 200 KBytes. However, there is a ten-

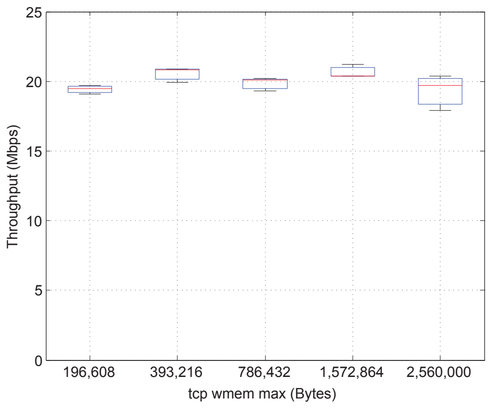
[1] *tcp_wmem_max* limits sendbuffer for each TCP connection and *wmem_max* limits sendbuffer for all types of connections

dency that default *tcp_wmem_max* is set higher value as new smartphone model is released. For example, *tcp_wmem_max* is set as 1,220 KBytes for Galaxy S2 (E120K), 2,560 KBytes for Galaxy S3 (E210K), and 2,560 KBytes for Galaxy S4 (E330K). This is related with uplink bandwidth increase of cellular networks for achieving higher throughput. We conduct additional experiments to identify the relation between uplink bandwidth and the degree of *RTT* increase. Fig. 3.2 shows that the upload queueing delay get worsen as uplink bandwidth shrinks. For instance, *RTT* increase reaches more than 900 msec when uplink bandwidth is lower than 10 Mbps.

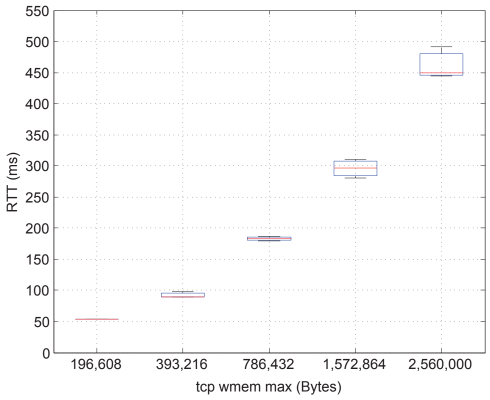
This measurement study indicates that the cause of unnecessary large delay in traffic uploading is due to not low link speed of cellular networks [48] but large sendbuffer of mobile devices. The best setting of *tcp_wmem_max* changes depending on network status; therefore, it is difficult to achieve the optimal performance with the fixed *tcp_wmem_max* setting.

3.2.2 Negative Impact on Performance in Multitasking Scenarios

Unnecessary large sendbuffer may have little impact on user experience in singletasking environments. However, a large number of stacked packets in sendbuffer give the negative impact on multitasking scenarios that there are more than 2 running applications simultaneously. Although each application has its own sendbuffer shown



(a) Upload throughput



(b) Upload RTT

Figure 3.1: TCP upload performance for different *tcp_wmem_max*.

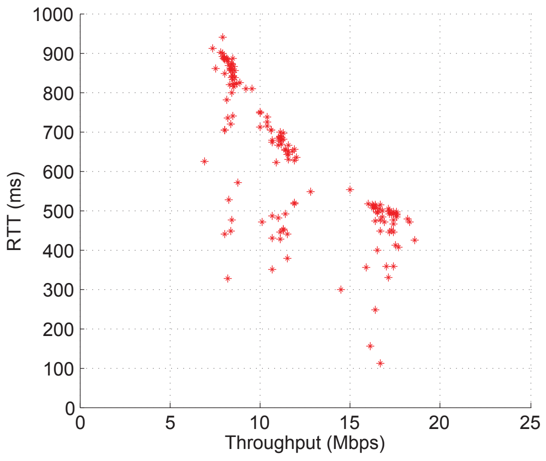


Figure 3.2: Upload delay performance for different upload throughput.

in Fig. 3.5, they share transmit queue which consist of a single queue. As a result, packets from applications requiring high QoS can be delayed in the transmit queue due to the stacked packets from other applications.

Fig. 3.3 shows the *RTT* increase of foreground ping application from a user to a server (i.e., www.google.com) while background upload traffic is generated by iperf. *RTT* of foreground ping application can be reached more than 1.8 sec when background upload throughput is 9 Mbps, which is critical value for delay-sensitive applications. In addition, we measure the degree of download performance degradation with upload traffic. Both traffics are generated by iperf with TCP CUBIC [31] and download starts 10 sec after upload begins. Fig. 3.4 shows that throughput degradation is inevitable when measurement duration is short (i.e., 10 sec) although uplink bandwidth is high. Low

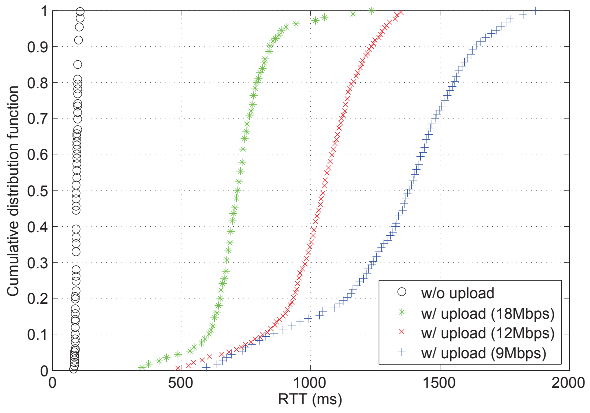


Figure 3.3: Delay Performance of ping application with background upload traffic.

throughput is caused by late arrival of download ACKs and it results in slow *cwnd* increase. On the other hand, performance degradation of download is not that severe with long measurement duration (i.e., 60 sec).

Our observation from the measurements shows that critical performance degradation in multitasking scenarios can happen in LTE networks and it is rooted in a single transmit queue which is shared by whole applications in end devices.

3.3 SOR based Packet Classification with Multiple Transmit Queue

In this section, we describe the proposed scheme that aims to improve user QoE. We present considering queue structure and propose a new

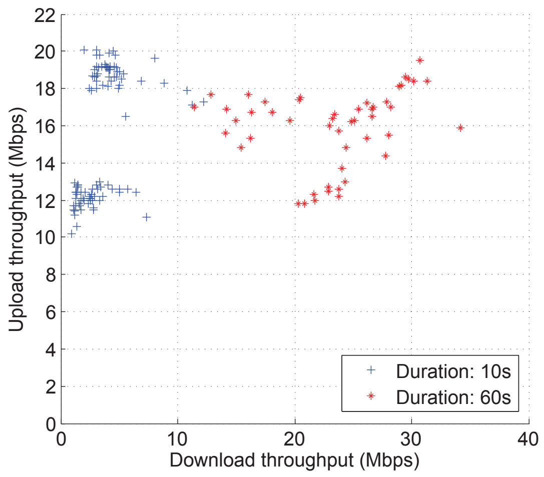


Figure 3.4: Throughput performance of download traffic with background upload traffic.

metric for separating delay-sensitive packets from non delay-sensitive packets.

3.3.1 Dual Transmit Queue

Large sendbuffer environment with a single transmit queue have a bad influence on multitasking scenarios in the way that it incurs unnecessary large upload queueing delay. Reducing sendbuffer is one of the most simple way to tackle this problem. However, it is not easy to find the best value of *tcp_wmem_max* as network status changes.

We adopt dual transmit queue architecture like [53, 54] to improve user QoE in multitasking scenarios. The primary goal of the proposed queue management scheme is to separate delay-sensitive applications (e.g. VoIP, mobile messenger, and mobile game) from non delay-sensitive applications (e.g. FTP). For example, when a user begins m-VoIP while uploading data to the cloud storage server, we want to process packets from m-VoIP in advance of packets from cloud storage service. Different with previous works [53, 54] that focused only on improving download performance, we consider QoE of interactive applications such as mobile messenger services, online games, and m-VoIP as well as download performance. Hence, it is the key for the performance how to classify packets from delay-sensitive and non delay-sensitive applications which will be explained in Sec. 3.3.2.

The structure of dual transmit queue is depicted in Fig. 3.5. FIFO0 has higher priority than FIFO1; therefore, FIFO1 can be dequeued only after FIFO0 becomes empty. FIFO1 mainly serves non delay-sensitive applications like file transfer while FIFO0 serves delay-sensitive applications according to packet classification.

Although linux-based android smartphones adopt *pfifo_fast* qdisc as a default, which maintains 3 different priority transmit queue, which is called band[2], priority queueing of *pfifo_fast* rarely used because it classifies packets according to type of service (ToS) field in IP header of each packet. Most of popular applications are not likely to use ToS (i.e., ToS = 0x00); therefore, *pfifo_fast* operates like a single transmit queue.

[2]*pfifo_fast* has 3 bands. Band 0 is processed with the highest priority and band 2 with the lowest priority

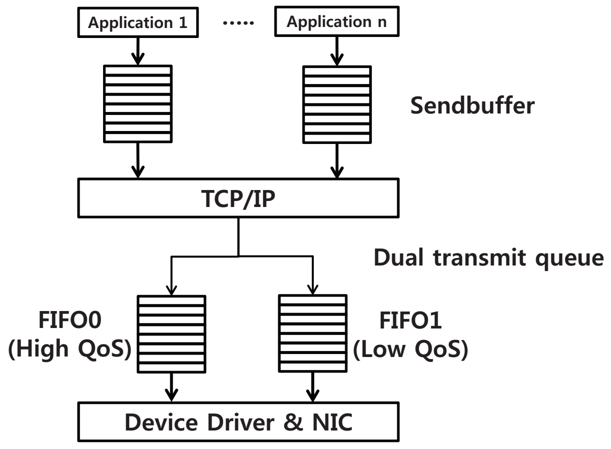


Figure 3.5: Dual transmit queue architecture.

3.3.2 SOR based Packet Classification Algorithm

Based on the dual transmit queue architecture, packet classification before enqueuing a packet to the transmit queue is the key performance factor. Using well-known port number (ex. FTP (21), HTTP (80), and HTTPS (443)), which is registered in Internet Assigned Numbers Authority (IANA) [61] in one of the simplest way to classify packets. However, its low accuracy of port number based classification may cause false results [63, 64]. In addition, it is not adequate for mobile devices to adopt more sophisticated classification techniques [65, 66, 67] which require high computational complexity.

We propose a sendbuffer occupancy ratio based packet classification algorithm, called SORPC, to improve user QoE in multitasking scenarios without computational overhead. From the measurements which will be shown in Sec. 3.4.1, we found that many commercial applications supporting file transfer fill their sendbuffer very quickly. As a result, we define a packet classification metric as sendbuffer occupancy ratio (SOR) as follow:

$$SOR_s(t) = \frac{Q_s(t)}{sndbuf(t)}, \tag{3.1}$$

where $Q_a(t)$ denotes the amount of data piled up in the sendbuffer of session s at time t and $sndbuf(t)$ denotes the size of the sendbuffer at time t. $Q_s(t)$ and $sndbuf(t)$ can be easily obtained from the socket structure without overhead; therefore, $SOR_s(t)$ can be calculated whenever there are packets to transmit. To make a enqueuing decision, SORPC sets the same threshold SOR_{th} for all the sessions. When $SOR_s(t) \geq SOR_{th}$ at time t, SORPC refers session s as a file transfer application and it enqueues a packet to FIFO1 which serves non delay-sensitive applications. Otherwise, SORPC enqueues a packet to FIFO0 to transmit.

In addition to SOR criterion, SORPC gives higher priority on UDP packets to provide better QoE becasue UDP is widely used by delay-sensitive applications such as video streaming, video phone call, and m-VoIP. SORPC follows the same technique with previous works [53, 54] in the way that it gives more priority download ACKs than upload data.

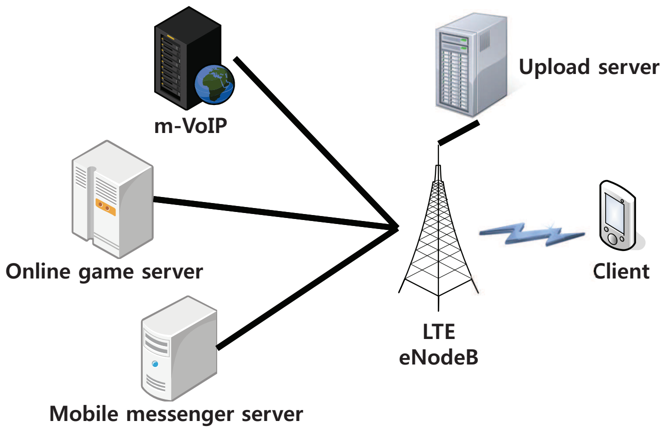


Figure 3.6: Experimental scenarios.

3.4 Experiment Results

We implement SORPC on commercial android devices (i.e., Samsung Galaxy S2 (E120K)) and conduct experiments in LTE network operated by KT (Korea Telecom). All experiments is done with background upload traffic generated by iperf or Dropbox. We compared SORPC with a single transmit queue scheme (Single Queue) throughput whole experiments. For the interactive applications, we use KakaoTalk[3] as mobile messenger service, Marble for all[4] as mobile game, and skype as m-VoIP. Youtube and Amazon.com has been chosen for download applications. The detailed setup are summarized in Table. 3.1.

[3]KakaoTalk is the most popular mobile messenger applications in Korea and it has been downloaded more than 100 millions.

[4]Marble for all is a mobile board game like blue marble and it has recorded more than 10 million downloads.

Table 3.1: Experimental Setup

Smartphone	Samsung Galaxy S2 (Model E120K) OS: Kernel version 3.0.8, Android Icecream Sandwich (4.0.3) Chipset: Qualcomm Snapdragon S3 MSM8660, 1500 MHz (Dual-Core) Congestion control: CUBIC tcp_wmem_max: 2,560,000 Bytes SOR_{th}: 0.5
LTE networks	Operator: Korea Telecom (KT) Download Bandwidth: 100 Mbps Upload Bandwidth: 25 Mbps
Applications	Upload traffic generator: iperf, Dropbox Download traffic generator: iperf Mobile messenger: KakaoTalk Online game: Marble for all m-VoIP: SkypeStreming: Youtube Web: www.amazon.com

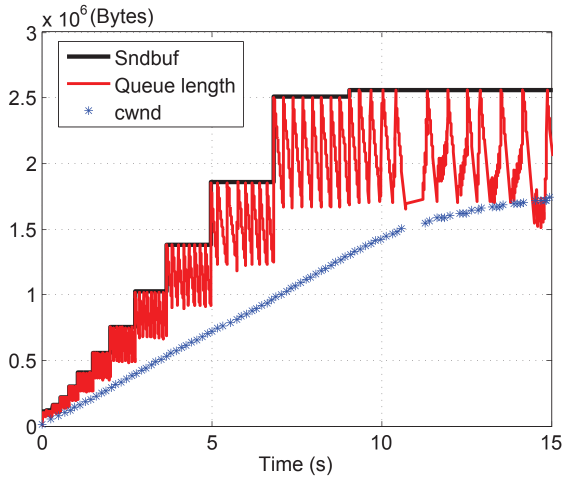
3.4.1 Packet Classification Metric: Sendbuffer Occupancy Ratio (SOR)

The degree of upload queueing delay is related to both *tcp_wmem_max* and upload bandwidth as presented in Sec. 3.2. We first briefly overview sendbuffer auto-tuning technique. Sendbuffer auto-tuning is similar to advertised receive window (*rwnd*) auto-tuning which determines *rwnd* as $\min\{2 * cwnd_{est},\ tcp\_rmem\_max\}$, where $cwnd_{est}$ denotes the estimated *cwnd* at the receiver side [28] and *tcp_rmem_max* denotes the maximum receive buffer space. Sendbuffer (*sndbuf*) is also determined as follow:

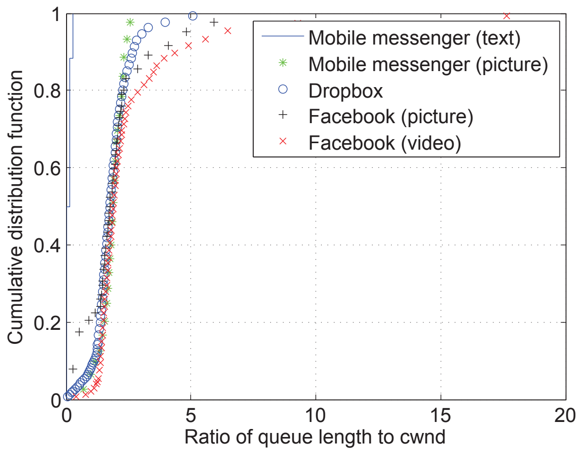
$$sndbuf = \min\{2 * cwnd,\ tcp\_wmem\_max\}. \tag{3.2}$$

Fig. 3.7(a) shows the evolution of sendbuffer, *cwnd*, and queue length using iperf. As *cwnd* increases, sendbuffer is automatically expanded based on (3.2) until it reaches *tcp_wmem_max*. In addition, sendbuffer never decreases and can only be expanded when there are no memory pressure. We also measure the ratio of queue length to *cwnd* (QCR). If the ratio is larger than 1, it means that there are more packets than network BDP which is approximately estimated from *cwnd*; otherwise, there are less packets than network capacity. Fig. 3.7(b) shows the distribution of QCR. Regardless of applications, QCR is almost larger than 1 all the time. In other words, there are always more packets than network capacity for file transfer applications. In the case of text service through mobile messenger, QCR is always less than 1 due to large initial *cwnd* (i.e., 10) of Android devices.

With the assumption that file transfer applications enqueue more packets than network BDP, we measure *SOR* of popular file transfer applications. Fig. 3.8 shows *SOR* distributions of popular file upload applications (except text in mobile messenger)). Facebook and Dropobx shows similar distributions in the way that they mostly fill their sendbuffers more than half. When a user transfers a picture via mobile messenger, *SOR* distribution is also similar. However, *SOR* of



(a) Sendbuffer autotuning



(b) Ratio of queue length to *cwnd*

Figure 3.7: Sendbuffer auto-tuning with queue length.

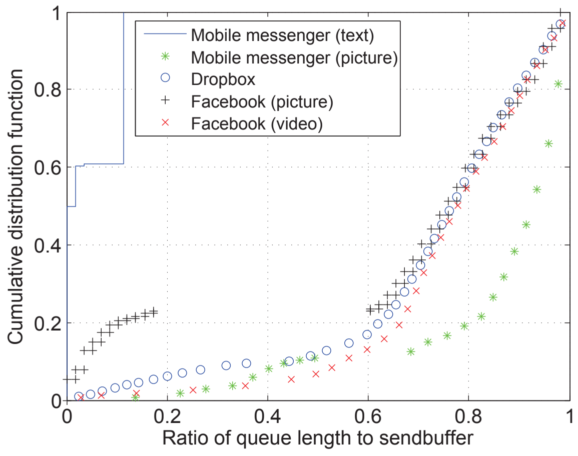


Figure 3.8: Distribution of SOR for popular upload applications.

text service through mobile messenger shows different distribution due to the small size of data packets. The threshold for SOR can affect the performance of SORPC. Throughout this chapter, we use 0.5 for SOR_{th}.

Fig. 3.9 shows the time to reach SOR_{th} with Dropbox and Facebook. It takes only few hundred milliseconds to reach SOR_{th} for both applications. This characteristic has the strength and weakness. Fast decision of packet classification is the strength; on the other hand, the weakness is the possibility of unfairness among competing upload file transfer applications which will be presented in Sec. 3.4.4.

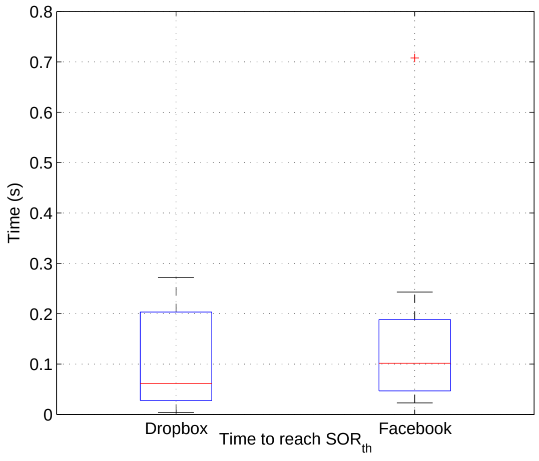


Figure 3.9: Time to reach SOR_{th}.

3.4.2 Improving RTT performance of Interactive Applications

We conduct experiments to verify the effectiveness of SORPC using three popular interactive applications: mobile messenger service (KakaoTalk), mobile game (Marble for All), and m-VoIP (Skype). Iperf is used as an upload traffic generator and conduct the same experiments for different uplink bandwidth. The delay performances of mobile messenger service and mobile game can be obtained from TCP layer; however, we use "display technical call info" option to get RTT performance for Skpye which mainly uses UDP as a transport layer protocol.

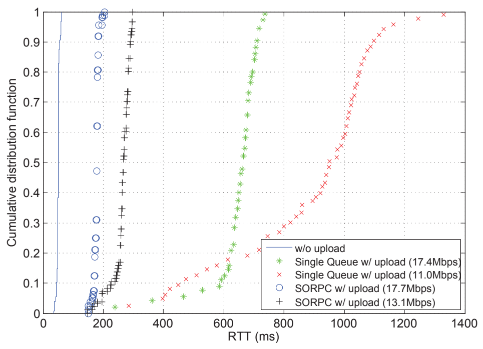
Fig. 3.10(a) shows RTT distribution of mobile messenger application. Without upload traffic, average RTT is around 50 msec.

Single Queue shows deteriorated *RTT* performance with background upload traffic and *RTT* increase get worse as uplink bandwidth becomes smaller. On the other hand, SORPC shows very low *RTT* performance less than 300 msec regardless of uplink bandwidth. It is because SORPC can automatically separate mobile messenger packets from background upload traffic using *SOR* criterion. SORPC also improves the *RTT* performance of mobile game as shown in Fig. 3.10(b). At the worst case, a packet of Single Queue is delayed by 1.5 sec while SORPC achieves the *RTT* performance lower than 418 msec. In the case of m-VoIP, SORPC is also effective to reduce the delay with background upload traffic by processing UDP packets first while *RTT* of Single Queue is critical to use m-VoIP as shown in Fig. 3.10(c).

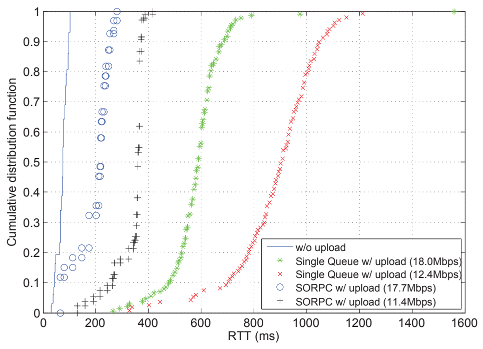
There is a gap between SORPC and without upload although *SOR* criterion classifies a packet well. Hardware limitation that we use (E120K with a dual-core Snapdragon) for supporting multitasking is the main reason of that gap.

3.4.3 Improving Download Performance

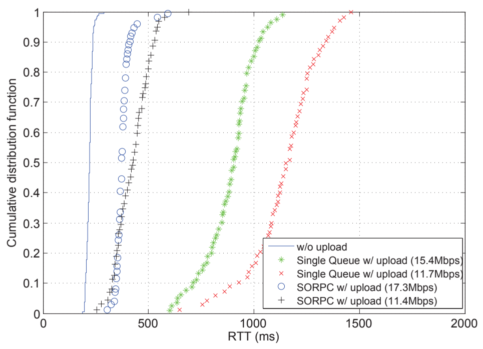
Severe performance degradation on download with simultaneous upload has been observed in both wired [53, 54] and 3G/HSPA networks [48, 49]. We firstly measure the impact of simultaneous upload on download performance in LTE network with popular applications such as Youtube streaming and Web loading which use TCP as a transport layer protocol. File upload to a Dropbox server is chosen as



(a) Mobile messenger (KakaoTalk)



(b) Mobile game (Marble for All)



(c) m-VoIP (Skype)

Figure 3.10: *RTT* performance in multitasking scenarios. The numbers in parentheses refer to throughput of background uploading traffic.

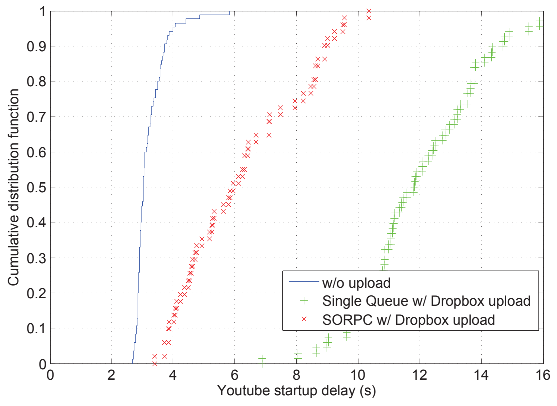
a background upload traffic.

Fig. 3.11(a) shows youtube startup delay performance. When there are no background upload traffic, average start delay is around 4.4 sec while Single Queue shows worse performance of 8 sec in average and 15 sec in the worst case. Web loading also shows the similar performance as shown in Fig. 3.11(b). Average web loading time is 3.2 sec without upload traffic; however, the performance of Single Queue is 12 sec in average. SORPC achieves better performances than Single Queue by transmitting download ACKs of Youtube and Web in advance of Dropbox data packets like [53, 54]. In average, SORPC achieves 6.3 sec startup delay with Youtube and 6.2 sec web loading time. Fluctuation of the download performance is much severe due to the bottleneck at the server. When we experiment with servers of popular applications, the performance variation happens although wireless link capacity is enough.

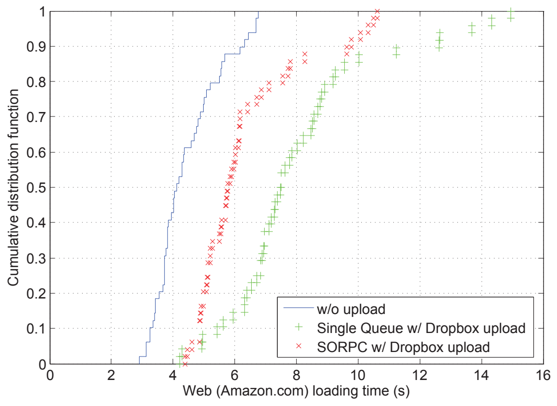
Short-term throughput degradation shown in Fig. 3.4 is also alleviated by prioritizing download ACKs first. We use iperf as a traffic generator for both upload and download and experiment duration is 10 sec. Fig. 3.12 shows that average short-term throughput of SORPC increases from 3.2 Mbps to 19.1 Mbps.

3.4.4 Fairness among Competing Upload Flows

SORPC's effectiveness to improve user QoE in multitasking scenarios is presented in Sec. 3.4.2 and 3.4.3. However, when there are compet-



(a) Youtube



(b) Web

Figure 3.11: QoE of download applications with upload traffic.

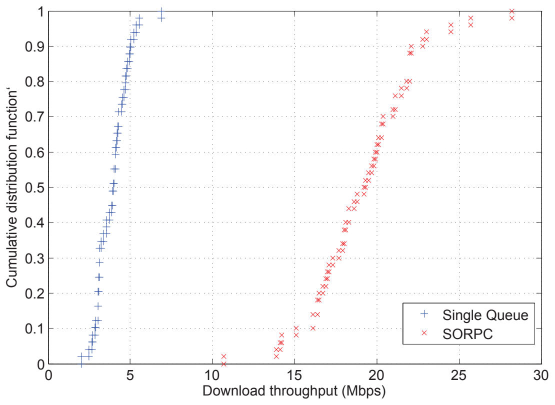


Figure 3.12: Throughput enhancement of SORPC.

ing upload file transfers, SORPC incurs short-term unfairness among competing upload flows. We conduct experiments with two upload flows using iperf to identify the degree of unfairness. There is a 10 sec gap between two upload flows.

Fig. 3.13 shows that fairness is achieved between two flows when the duration is 60 sec and it shows 0.94 of Jain's fairness index [68]. On the other hand, unfairness between two flows happens when the duration is 10 sec with 0.64 of fairness index. The reason of unfairness is that the time to fill sendbuffer is very short; therefore, late upload flows can only enqueue a small amount of packets to FIFO0. We remain a fairness issue among competing flows as a future work.

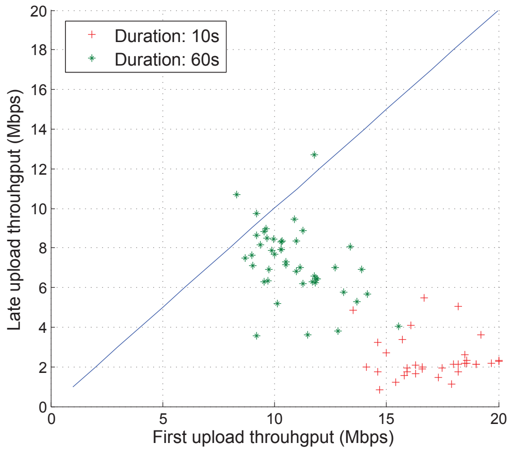


Figure 3.13: Unfairness among competing upload traffics.

3.5 Summary

In this chapter, we have presented a packet classification algorithm in transport layer to improve multitasking user's QoE in LTE networks. Based on dual transmit queue architecture, the proposed scheme effectively separates delay-sensitive packets from non delay-sensitive packets without computational overhead. We have implemented the proposed scheme on commercial available smartphones and conducted extensive experiments. We have verified that the proposed packet classification technique performs well and result in better delay performance for delay-sensitive applications.

Chapter 4

Uplink Congestion Control in Low-power and Lossy Networks

4.1 Introduction

Low-power and lossy networks (LLNs), consist of many embedded networking devices which formed in wireless multi-hop networks, have recently drawn huge attention due to its various applications such as smart grid automated metering infrastructures (AMIs) [69, 70], monitoring [71, 72], and wireless sensor networks (WSNs) [73, 74]. In addition, current LLNs has been employing IPv6-based architecture to be connected to the Internet. For example, the IETF has recently standardized protocols such as RPL [78] and 6LoWPAN [77, 79, 80] for connecting LLNs to the Internet with the concept of the Internet of Things (IoT).

A move that makes LLNs be part of the Internet naturally gives rise to many challenges about the possibility of interoperation between LLNs and existing Internet. To be connected to the Internet, LLNs performs wireless multi-hop communication from LLN nodes to LBR (LLN Border Gateway) as depicted in Fig. 4.1. Due to the fact that LLNs typically have different characteristics such as low memory, high packet loss, and frequent topology changes from wired and wireless access networks such as LTE and Wi-Fi, many technical challenges will be encountered.

One of the main challenges in connecting LLNs to the Internet is about TCP for compatibility reason because TCP has provided a reliable data transfer for the most of the Internet traffic since its birth. To reflect this trend, BLIP, IPv6 stack in TinyOS, have recently released experimental version of TCP in TinyOS by providing fundamental TCP features such as 3-way handshake and congestion control including TCP retransmission schemes.

Main difference between LLNs and the Internet is the role of each node. A LLN node can become a data source and a router for its child nodes simultaneously like wireless ad-hoc network. Therefore, LLN nodes fundamentally contend with their child nodes that potentially have packets to transmit through their parent. In addition, LLN nodes (e.g. wireless sensor motes) typically have equipped with low memory and results in frequent packet loss due to buffer overflow when congestion happens. These characteristics of LLNs give negative impact on TCP performance in temrs of throughput and fairness.

Previous works that aim to enhance TCP performance have been mainly done in 802.11 based wireless ad-hoc networks [81, 82, 83]. However, they have some practical problems to be adopted in LLNs in the way that they incur additional control overhead and are too heavy to be implemented in LLN nodes. In addition, transport layer protocols for wireless sensor networks, one of the main applications of LLNs, have been developed to enhance performance at transport layer [84, 85, 86]. However, there remain an interoperability issue with existing Internet because their main target is limited to themselves.

In this chapter, we propose an uplink TCP congestion control for LLNs in order to improve fairness. Firstly, we confirm that throughput among nodes with different hop counts shows severe unfairness due to frequent buffer overflow of LLN nodes and propose dynamic TX period adjustment (DTPA) to overcome throughput unfairness. To the best of our knowledge, this it the first work that handle TCP fairness problem in LLNs. Through experiments on our testbed with 30 TelosB sensor motes, we evaluate the effectiveness of the proposed scheme.

The rest of this chapter is organized as follows. We first describe the system model in Section 4.2. In Section 4.3, we develop a distributed algorithm that aims to mitigate TCP unfairness in LLNs.

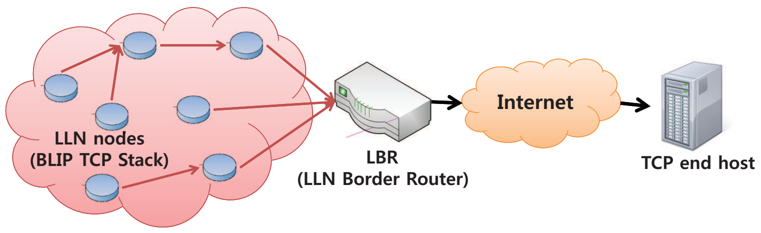


Figure 4.1: A scenario of LLN connected to the Internet through LBR.

Section 4.4 describes experimental setup and experimental results to evaluate our solution. Finally, we conclude this chapter in Section 4.5.

4.2 System Model

We consider a scenario for LLN as shown in Fig. 4.1. LLN nodes can communicate each other by utilizing LLN interface and is connect to the LBR using unicast routing protocol like RPL [78]. Each LLN node can be connected to the external TCP end host via LBR (LLN Border Router) which performs a gateway between LLN and the wide area network (WAN) which can either be the public Internet or a private IP-based network [91]. Both and LLN nodes and TCP end host use TCP/IPv6 stack to communicate each other at the network and transport layers. We mainly consider a traffic upload scenario from each LLN nodes to the TCP end host. We assume that the network from LBR to the TCP end host is quite stable and round-trip time (RTT) from the LBR to the TCP end host is relatively small compared to the RTT from each LLN nodes to the LBR.

4.3 Proposed Scheme

In this section, we introduce the new concept of TX period (TXP) for improving TCP fairness among LLN nodes and propose dynamic TX period adjustment (DTPA) with throughput analysis in LLN.

4.3.1 Tx Period

LLNs are fundamentally different from wired and wireless access networks in the way that each node in LLNs can be a both data source and router simultaneously. Different with 802.11 based wireless adhoc networks, LLN nodes typically have very low memory[1] which is the performance bottleneck in multi-hop communications. When we consider uplink TCP traffic from LLN nodes to the external TCP end host, TCP performance of a LLN node far from the LBR get worsen. Not only uplink DATA packets but also TCP ACKs from the external TCP end host frequently get lost due to the small queue length of intermediate LLN nodes although its data is successfully delivered to the TCP end host, thus leading to unfairness among LLN nodes with different hop counts from the LBR.

We introduce the concept of TX period (TXP) to enhance fairness among LLN nodes. An intuition on TXP is 'wait and transmit strategy (WSS)'. Because a node close to the LBR is likely to experience low packet loss rate and low RTT compared to a node far away from the LBR, a node close to the LBR should wait their own data

[1]For example, TelosB only provides 48 KBytes ROM and 10 KBytes RAM.

transfer although they have something to transmit for TXP. After TXP timer fires, it can transmit its own data. Therefore, parent nodes only perform packet forwarding on behalf of their child nodes during TXP.

4.3.2 Dynamic TX Period Adjustment

Basic intuition on TXP is that parent nodes yield their TX opportunity to their child nodes for improving TCP fairness. We propose dynamic TX period adjustment (DTPA) that determines TXP dynamically in a distributed manner. We first explain TCP throughput analysis of LLN nodes.

TCP congestion control is a distributed algorithm to share network resources. and is known to an optimal solution of network utility maximization problem [17, 21, 22]. In equilibrium point, [75, 76] showed that throughput of TCP Reno is

$$x_{Reno} = \frac{MSS}{RTT} \frac{a}{\sqrt{p}}, \tag{4.1}$$

where p denotes loss probability and the coefficient a is known to be $\sqrt{\frac{3}{2}}$.

DTPA basically follows the same AIMD policy with TCP Reno as follows:

$$cwnd = \begin{cases} cwnd + \frac{1}{cwnd} & \text{on every ACK reception} \\ \frac{1}{2}cnwd & \text{on a packet loss} \end{cases} \tag{4.2}$$

Similar to (4.1), throughput of LLN nodes with TCP Reno can be modeled as

$$x_{DTPA} = \frac{MSS}{TXP + RTT}\frac{a}{\sqrt{p}}. \tag{4.3}$$

To differentiate TXP among LLN nodes with different hop counts, we set TXP as a function of p and RTT as

$$TXP_i(RTT_i, p_i) = h(p_i) - RTT_i, \tag{4.4}$$

where i denote LLN node index and $h(\cdot)$ denotes the mapping function from packet loss rate to TXP. Due to the fact that DTPA mainly focuses on improvement of TCP throughput fairness, DTPA sets the same bandwidth to all LLN nodes as follows:

$$\frac{1}{TXP_i + RTT_i}\frac{a}{\sqrt{p_i}} = c, \tag{4.5}$$

where c is a system parameter that depends on network topology. From (4.5), DTPA finally determines TXP as

$$TXP_i = \begin{cases} \frac{a}{c\sqrt{p_i}} - RTT_i & if \frac{a}{c\sqrt{p_i}} - RTT_i > 0 \\ 0 & o.w \end{cases} \tag{4.6}$$

Our intuition on (4.6) makes a node close to the LBR wait longer because it is likely to experience low packet loss rate and low RTT compared to nodes far away from the LBR. If c is set as an infinite value, DTPA shows the same operation of TCP Reno. With small value of c, every LLN nodes should waste too much time for unnecessary large TXP. Proper setting of c is important issue and we remain is as a future work. The overall procedures of DTPA are provided in Algorithm 2.

4.4 Experimental Results

4.4.1 Experimental Setup

We configure a testbed environment as depicted in Fig. 4.2. There are 30 LLN nodes and one LBR in an office environment. The LBR consists of a Linux desktop PC and an LLN interface. The LBR forwards packets to/from the Linux PC through UART and LLN interface uses the *ppprouter* stack in TinyOS. The desktop PC uses TCP/IP to exchange data with multiple LLN devices where each LLN node is a TelosB clone device [87] with an MSP430 microcontroller and a CC2420 radio. We use TinyOS as the embedded software in our experiments [88]. In TinyOS, we use berkeley low-power IP stack

Algorithm 2 Dynamic TX Period Adjustment (DTPA)

```
Initialization:
1: $RTOcounter \leftarrow 0$
2: $FRcounter \leftarrow 0$
3: $TotalTxcounter \leftarrow 0$

On receiving a packet:
1: **if** (measured $RTT$ is valid) **then**
2:     $\widehat{RTT} \leftarrow (1-\beta) \cdot \widehat{RTT} + \beta \cdot (\text{measured } RTT)$
3:     $p = (RTOcounter + FRcounter)/TotalTxcounter$
4:     $TXP = \frac{a}{c \cdot \sqrt{p}} - \widehat{RTT}$
5:     **if** $TXP < 0$ **then**
6:         $TXP \leftarrow 0$
7:     **end if**
8: **end if**

On retransmission timer fired:
1: $RTOcounter \leftarrow RTOcounter + 1$

On receiving 3 DUPACKs:
1: $FRcounter \leftarrow FRcounter + 1$
```

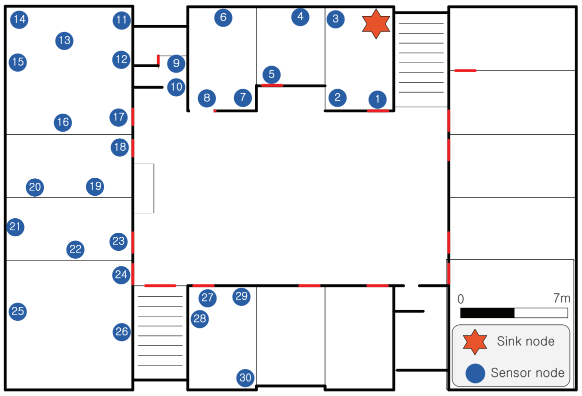
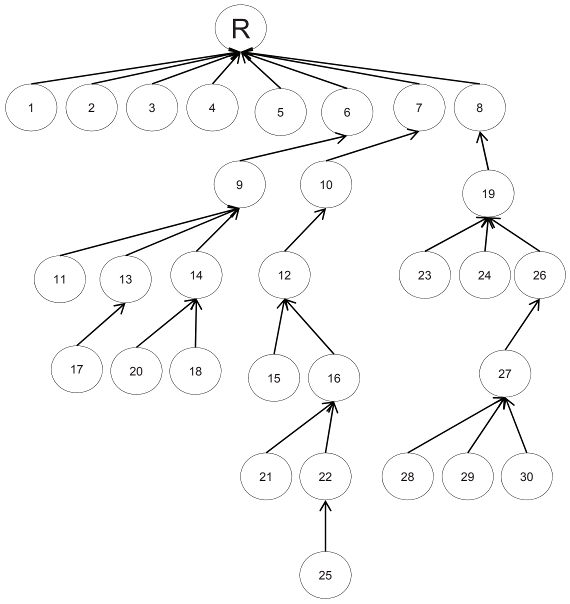


Figure 4.2: Testbed topology in an office environment.

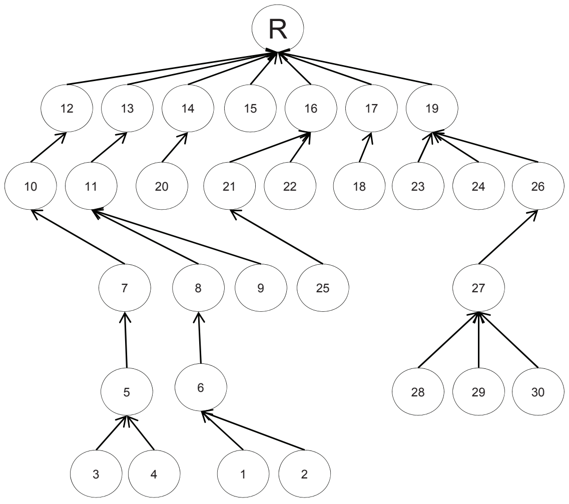
(BLIP) as a TCP/IPv6 stack including 6LoWPAN that allows IPv6 to operate efficiently over IEEE 802.15.4. We have not turned on a low power listening (LPL) duty cycling mechanism [89, 90], and each node employs a FIFO transmit queue of which size is 15 packets and we generate traffic 2 packets/s for all LLN nodes. We have implemented RTT estimator in BLIP TCP stack due to the fact that DTPA determines its TXP based on RTT.

4.4.2 Throughput analysis vs. Measurement

We first compare throughput analysis in (4.3) with real measurement results. We consider 2 different topologies as shown in Fig. 4.3. Node ID is correspond to the number in Fig. 4.2. Fig. 4.4 shows the result between analysis and measurement. Real measurement results show that throughput analysis of DTPA is almost similar to experimental results. Due to the small queue length at each node, network per-

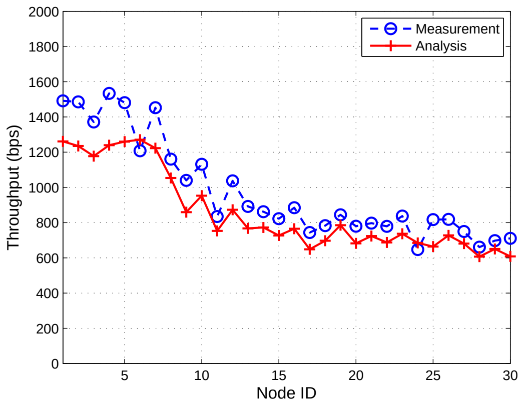


(a) 6hop topology with 30 nodes

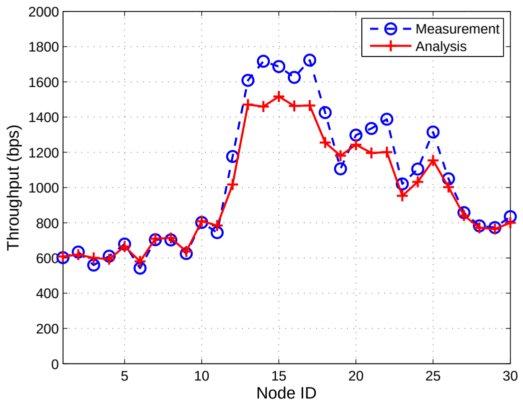


(b) 5hop topology with 30 nodes

Figure 4.3: Different topologies with different maximum hop count.



(a) 6hop topology



(b) 5hop topology

Figure 4.4: Throughput analysis vs. Measurement.

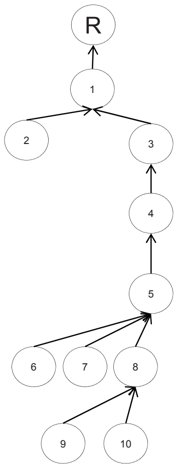
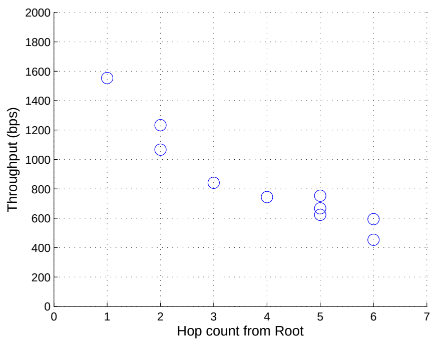


Figure 4.5: 6 hop topology with 10 nodes.

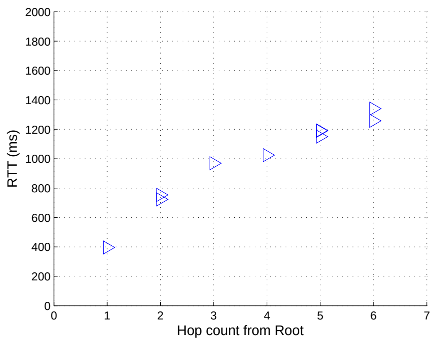
formance converges fast in terms of packet loss rate and RTT, thus leading to the small gap between analysis and experimental results.

4.4.3 TCP Performance in Low-power Lossy Networks

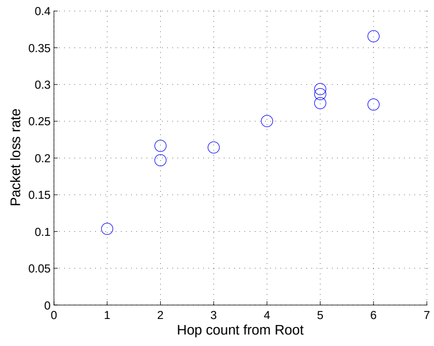
We firstly show that TCP throughput unfairness in LLNs. Fig. 4.5 shows 6 hop topology with 10 nodes. Fig. 4.6 shows TCP performance according to hop count from the LBR. Throughput significantly decreases as hop count increases. Because RTT and packet loss rate also increases according to hop count, throughput of a node in 6 hop from the LBR only achieves only 20 % of a node in 1 hop from the LBR. This results are directly related to the limited queue length of LLN nodes. Fig. 4.7 shows that the number of buffer overflow at each nodes. As shown in Fig. 4.7, buffer overflow at node 1 almost reaches 2000 times while other nodes only shows smaller buffer overflow. It



(a) Throughput



(b) RTT



(c) Packet loss rate

Figure 4.6: Throughput unfairness among nodes with different hop count.

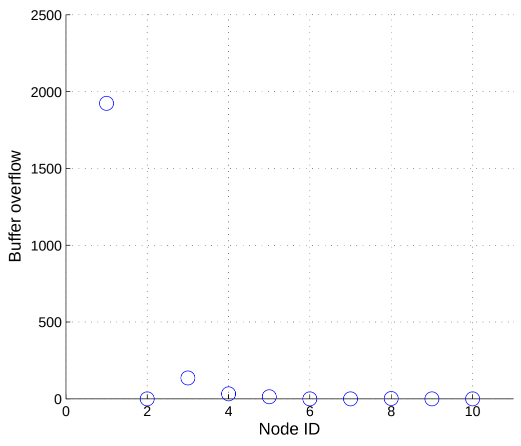
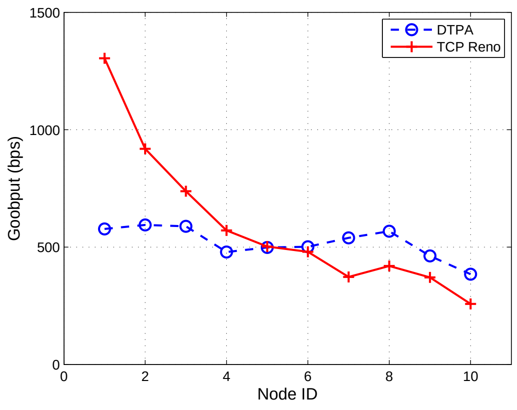


Figure 4.7: 6 hop topology with 10 nodes.

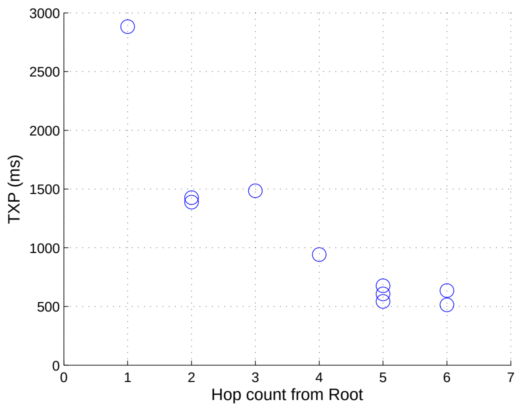
means that the bottleneck of this topology lies in node 1. TCP DATA and ACKs that passes node 1 experience high loss rate, thus leading to throughput degradation as shown in Fig. 4.6(a).

4.4.4 Fairness improvement of DTPA

We now verify the performance of DTPA in various topology scenarios. Firstly, we consider 6 hop topology with 10 nodes as shown in Fig. 4.5. Fig. 4.8(a) shows that DTPA shows better performance in terms of fairness while overall goodput decreases by 23 %. Because TX period is dynamically determined based on (4.5), nodes close to the LBR shows higher TXP than nodes far away from the LBR as shown in Fig. 4.9(b). During TXP, each node cannot send any data and only can forward packets for their child, thus leading to throughput reduction of node close to the LBR.

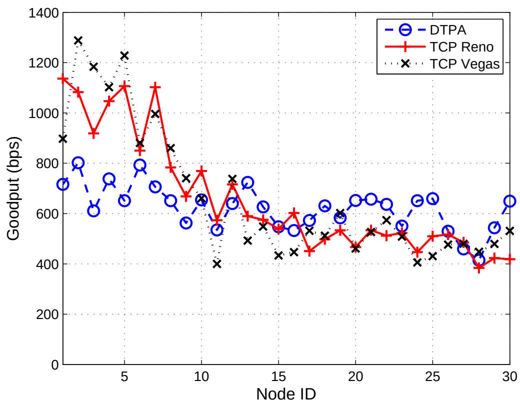


(a) Fairness Improvement

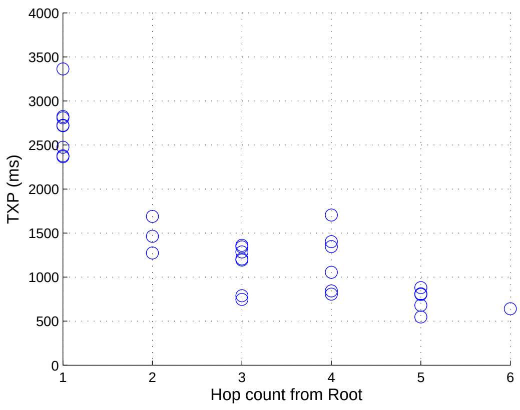


(b) TX period

Figure 4.8: Performance of DTPA in 6 hop topology with 10 nodes.



(a) Fairness Improvement



(b) TX period

Figure 4.9: Performance of DTPA in 6 hop topology with 30 nodes.

Fig. 4.9 shows the performance of DTPA in the topology with 30 nodes as presented in Fig. 4.3(a). Similar to results of 6 hop topology with 10 nodes, DTPA shows better performance in terms of fairness by setting TXP of nodes close to the LBR as a high value. Interestingly, TCP-Vegas [12] suffers from TCP unfairness similar to TCP-Reno due to inaccurate end-to-end queue length estimation in LLNs. Frequent loss makes it hard for TCP-Vegas to estimate queue length.

4.5 Summary

In this chapter, we have confirmed TCP throughput unfairness among LLN nodes. Due to the limited memory, each node can only hold small number of packets and results in frequent packet loss when congestion happens. We introduce the concept of TX period and propose dynamic TX period adjustment that aims to enhance TCP fairness in LLNs. Through extensive experiments using testbed with 30 nodes, we verify the effectiveness of our solution.

Chapter 5

Conclusion

5.1 Research Contributions

In this dissertation, we dealt with three different problems that aim to improve TCP performance in wireless networks.

First, we have presented a receiver-side countermeasure, named RTAC, to address the downstream bufferbloat problem in wireless access networks such as LTE and Wi-Fi networks. According to TCP and AQM dynamics, a RTAC receiver estimates an appropriate amount of in-flight data for a wireless access link, and controls the transmission rate through advertised receive window. We have implemented RTAC on commercial smartphones and conducted extensive experimental measurements. We have verified that RTAC successfully prevents bufferbloat, and achieves good delay performance without sacrificing throughput performance nor fairness with conventional

TCP flows, outperforming the state-of-the-art schemes.

Next, we have presented a new packet classification algorithm, named SORPC ,to improve multitasking user's QoE in LTE networks. SORPC adopts multiple transmit queue architecture and effectively separates delay-sensitive packets from non delay-sensitive packets using SOR. We have implemented the proposed scheme on Android device and verified that SOR performs well in terms of packet classification, thus leading to better delay performance for delay-sensitive applications.

Thirdly, we dealt with TCP performance in low-power and lossy networks. We firstly have shown severe TCP throughput unfairness among LLN nodes due to the limited memory. We propose dynamic TX period adjustment that aims to enhance TCP fairness by yielding each node's transmission opportunity to its child. Through extensive experiments on the testbed, we have verified the effectiveness of our proposed scheme.

To summarize, TCP in wireless networks has drawn attention in the network community for not only poor delay performance but also an interoperability issue of LLNs with existing Internet. There still remains some issues to resolve, it is anticipated that our proposed schemes can be a used as a guideline to improve TCP performance in wireless networks. Besides the three problems considered in this dissertation, many interesting problems that need deeper investigation of TCP will appear in wireless networks.

5.2 Future Research Directions

RTAC performs better with a precise estimation on BDP. In wireless networks, however, the dynamic nature of wireless channel, traffic, and user mobility makes accurate estimation very difficult. There have been several estimation techniques to tackle the problem in cellular and Wi-Fi networks [34, 35, 36], which can be incorporated with RTAC to operate in an auto-adaptive manner. To this end, developing precise estimation schemes for BDP, RTT, and wireless capacity, is of great interest and remains as a future work.

RTAC can be considered as the emulation of RED at the receiver-side like PERT [58] that uses only smoothed RTT with high weight to predict congestion accurately. In addition, CoDel [6] drops a packet based on its sojourn time and PIE [45] calculates the drop probability from delay. To incorporate these schemes with RTAC, there should be an investigation on how smoothed RTT can reflect congestion over the entire paths, and how the receiver estimates a sojourn time at the wireless last hop, and how one can achieve fairness under the existence of other competing TCP flows, which is an interesting open problem.

Finally, it is worth noting that controlling bufferbloat exacerbates the inherent fairness problem of TCP due to different $RTTs$ [55, 56]. Hence, the removal of the downstream bufferbloat at the receiver-side may bring back the fairness problem in certain scenarios. The problem can be mitigated by applying TCP congestion control such as TCP

Libra [57] that is designed to enhance RTT fairness. Furthermore, the fairness problem can be addressed by adopting per-flow queueing like FQ_CoDel [44] at bottleneck routers as in LTE and Wi-Fi networks.

Bibliography

[1] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[2] http://opensource.samsung.com/

[3] M. Gates, A. Tirumala, J. Dugan, and K. Gibbs, Iperf version 2.0.0, Part of Iperf's source code distribution, NLANR applications support, University of Illinois at Urbana-Champaign, Urbana, IL, USA, May 2004. [Online]. Available: http://iperf.sf.net

[4] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating The Edge Network," in *Proceedings of ACM IMC*, 2010.

[5] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," ACM Queue, vol. 9, issue 11, pp. 40-54, Nov. 2011.

[6] K. Nichols and V. Jacobson, "Controlling Queue Delay," ACM Queue, vol. 10, issue 5, pp. 20-34, May 2012.

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

[8] S. Alfredsson, G. D. Giudice, J. Garcia, A. Brunstrom, L. D. Cicco, and S. Mascolo, "Impact of TCP Congestion Control on Bufferbloat in Cellular Networks," in *Proceedings of WoWMoM*, 2013.

[9] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroiu,"Characterizing Residential Broadband Networks," in *Proceedings of ACM IMC*, 2007.

[10] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescape, "Broadband Internet Performance: A View From the Gateway," in *Proceedings of ACM SIGCOMM*, 2011.

[11] M. Allman, "Comments on Bufferbloat," ACM SIGCOMM Computer Communication Review (CCR), vol. 43, no. 1, pp. 31-37, Jan. 2013.

[12] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proceedings of ACM SIGCOMM*, 1994.

[13] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proceedings of INFOCOM*, 2004.

[14] J. Mo, and J. Walrand, "Fair End-to-End Window-Based Congestion Control," IEEE/ACM Transactions on Networking (ToN), vol. 8, no. 1, pp. 556-567, Oct. 2000.

[15] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," in *Proceedings of INFO-COM*, 1999.

[16] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking (ToN), vol. 1, issue. 4, 397-413, Aug. 1993.

[17] S. Liu, T. Basar, and R. Srikant, "Exponential-RED: A Stabilizing AQM Scheme for Low- and High-Speed TCP Protocols," IEEE/ACM Transactions on Networking, vol. 13, no. 5, pp. 1068-1081, Oct. 2005.

[18] S. Athuraliya , S. H. Low, V. H. Li, and Q. Yin, "REM: Active Queue Management," IEEE Network, vol. 15, issue 3, pp. 48-53, May 2001.

[19] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001.

[20] T. Goff, J. Moronski, D. S. Phatak, and V. Gupta, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," in *Proceedings of INFOCOM*, 2000

[21] F. Kelly, "Charging and Rate Control for Elastic Traffic," European Transactions on Telecommunications, vol. 8, no. 1, pp. 33-37, Jan. 1997.

[22] S. H. Low, F. Paganini, and J. C. Doyle, "Internet Congestion Control," IEEE Control Systems Magazine, vol. 22, issue 1, pp. 28-43, Feb. 2002.

[23] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348, Sep. 2008.

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

[25] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming," Technical report, NCSU, 2000.

[26] S. H. Low, "A Duality Model of TCP and Queue Management Algorithms," IEEE/ACM Transactions on Networking (ToN), vol. 11, no. 4, pp. 525-536, Aug. 2003.

[27] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, and Z. M. Mao, "An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance," in *Proceedings of ACM SIGCOMM*, 2013.

[28] W. Feng, M. Fisk, M. Gardner, and E. Weigle, "Dynamic Right-Sizing An Automated, Lightweight, and Scalable Technique for Enhancing Grid Performance," in *Proceedings of PfHSN*, 2002.

[29] C. Joo, S. Bahk, and S. S. Lumetta, "A Hybrid Active Queue Management for Stability and Fast Adaptation," Journal of Communications and Networks (JCN), vol. 8, no. 1, Mar. 2006.

[30] X. Liu, A. Sridharan, S. Machiraju, M. Seshadri, and H. Zang, "Experiences in a 3G Network: Interplay between the Wireless Channel and Applications," in *Proceedings of Mobicom*, 2008.

[31] S. Ha, I. Rhee, and L. Xu, "CUBIC: a New TCP-friendly High-speed TCP Variant," ACM SIGOPS Operating Systems Review, vol. 42, issue 5, pp. 64-74, Jul. 2008.

[32] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proceedings of PFLDNet Workshop*, 2004.

[33] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *Proceedings of Mobicom*, 2001.

[34] A. Chakraborty, V. Navda, V. N. Padmanabhan, and R. Ramjee, "Coordinating Cellular Background Transfers using LoadSense," in *Proceedings of Mobicom*, 2013.

[35] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye, "Bandwidth Estimation in Broadband Access Networks," in *Proceedings of ACM IMC*, 2004.

[36] M. Li, M. Claypool, and R. Kinicki, "Wbest: a bandwidth estimation tool for IEEE 802.11 wireless networks," in *Proceedings of IEEE LCN*, 2008.

[37] http://www.speedtest.net/

[38] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)," RFC 6817, Dec. 2012.

[39] S. Boyd and L. Vandenberghe, "Convex Optimization," Cambridge University Press, Cambridge, UK, 2004; available online from http://www.stanford.edu/?boyd/cvxbook.html.

[40] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad, "Receiver based management of low bandwidth access links," in *Proceedings of ACM INFOCOM*, 2000.

[41] P. Mehra, A. Zakhor, and C. De Vleeschouwer, "Receiver-driven band- width sharing for TCP," in *Proceedings of ACM INFOCOM*, 2003.

[42] P. Key, L. Massoulie, and B. B. Wang, "Emulating low-priority transport at the application layer: a background transfer service," in *Proceedings of ACM SIGMETRICS*, 2004.

[43] M. Crovella and P. Barford, "The network effects of prefetching," in *Proceedings of ACM INFOCOM*, 1998.

[44] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "FlowQueue-Codel," Internet-Draft, Sep. 2014.

[45] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. V. Steeg, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem," Internet-Draft, Feb. 2014.

[46] M. Heusse, S. A. Merritt, T. X. Brown, and A. Duda, "Two-way TCP Connections: Old Problem, New Insight," ACM SIGCOMM Computer Communication Review (CCR), vol. 41, issue 2, pp. 5-15, Apr. 2011.

[47] M. Podlesny and C. Williamson, "Improving TCP Performance in Residential Broadband Networks: A Simple and Deployable Approach," ACM SIGCOMM Computer Communication Review (CCR), vol. 42, issue 1, pp. 61-68, Jan. 2012.

[48] Y. Xu, W. K. Leong, B. Leong, and A. Razeen, "Dynamic Regulation of Mobile 3G/HSPA Uplink Buffer with Receiver-Side Flow Control," in *Proceedings of ICNP*, 2012.

[49] W. K. Leong, Y. Xu, B. Leong, and Z. Wang, "Mitigating Egregious ACK Delays In Cellular Data Networks by Eliminating TCP ACK Clocking," in *Proceedings of ICNP*, 2013.

[50] P. E. McKenney, "Stochastic Fairness Queueing," in *Proceedings of INFOCOM*, 1990.

[51] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," Internetworking: Research and Experience, vol. 1, pp. 3-26, 1990.

[52] S. Floyd and V. Jacobson, "Link-sharing and Resource Management Models for Packet Networks," IEEE Transactions on Networking, vol. 3, no. 4, pp. 365-386, Aug. 1995.

[53] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The Effects of Asymmetry on TCP Performance," in *Proceedings of Mobicom*, 1997.

[54] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Improving TCP Throughput over Two-Way Asymmetric Links: Analysis and Solutions," in *Proceedings of SIGMETRICS*, 1998.

[55] L. Xu, K. Harfoush, and I. Rhee, " Binary Increase Congestion Control for Fast, Long Distance Networks ," in *Proceedings of INFOCOM*, 2004.

[56] T. V. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random

Loss," IEEE/ACM Transactions on Networking (ToN), vol. 5, no. 3, pp. 336-350, Jun. 1997.

[57] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "TCP Libra: Exploring RTT-Fairness for TCP," UCLA Computer Science Department, Tech. Rep. UCLA-CSD TR-050037, 2005.

[58] S. Bhandarkar, A. L. N. Reddy, Y. Zhang, and D. Loguinov, "Emulating AQM from end hosts," in *Proceedings of SIGCOMM*, 2007.

[59] AT&T. A Different Take on the Big Game - Stats from the Stands. http://www.attinnovationspace.com/innovation/story/a7780988, Feb. 2012.

[60] Mashable. The 10 Most Frequently Used Smartphone Apps. http://mashable.com/2013/08/05/most-used-smartphone-apps, Aug. 2013.

[61] Internet Assigned Numbers Authority (IANA). Service Name and Transport Protocol Port Number Registry. http://www.iana.org/assignments/port-numbers, Dec. 2014.

[62] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," in *Proceedings of Sigcomm*, 1991.

[63] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, “Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification,” in *Proceedings of IMC*, 2004.

[64] A. Moore and K. Papagiannaki, “Toward the accurate identification of network applications,” in *Proceedings of Passive and Active Measurement Workshop (PAM)*, 2005.

[65] G. Mnz, H. Dai, L. Braun, and Georg Carle, “TCP traffic classification using markov models,” in *Proceedings of Traffic Monitoring and Analysis (TMA)*, 2010.

[66] A. Dainotti, W. de Donato, A. Pescape, and P. S. Rossi, “Classification of Network Traffic via Packet-Level Hidden Markov Models,” in *Proceedings of Globecom*, 2008.

[67] L. Bernaille, R. Teixeira, and K. Salamatian, “Early application identification,” in *Proceedings of CoNEXT*, 2006.

[68] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, “A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System,” DEC Research Report TR-301, 1984.

[69] Cisco, “Connected Grid Networks for Smart Grid - Field Area Network,” http://www.cisco.com/web/strategy/energy/field area network.html.

[70] E. Ancillotti, R. Bruno, and M. Conti, “The role of the rpl routing protocol for smart grid communications,” Communications Magazine, IEEE, vol. 51, no. 1, pp. 75-83, Jan. 2013.

[71] German Federal Ministry of Education and Research, “Project of the Future: Industry 4.0,” http://www.bmbf.de/en/19955.php.

[72] V. Gungor and G. Hancke, “Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches,” Industrial Electronics, IEEE Transactions on, vol. 56, no. 10, pp. 4258-4265, Oct. 2009.

[73] J. Paek, B. Greenstein, O. Gnawali, K.-Y. Jang, A. Joki, M. Vieira, J. Hicks, D. Estrin, R. Govindan, and E. Kohler, “The Tenet Architecture for Tiered Sensor Networks,” ACM Transactions on Sensor Networks, vol. 6, no. 4, pp. 34:1-34:44, 2010.

[74] J. Paek, J. Hicks, S. Coe, and R. Govindan, “Image-based environmental monitoring sensor application using an embedded wireless sensor network,” Sensors, vol. 14, no. 9, pp. 15981-16002, 2014. [Online]. Available: http://www.mdpi.com/1424-8220/14/9/15981

[75] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The macroscopic behavior of the TCP congestion avoidance algorithm,” ACM Computer Communication Review, vol. 27, no. 3, July 1997.

[76] T. V. Lakshman and U. Madhow, “The performance of TCP/IP for networks with high bandwidthdelay products and random loss,” IEEE/ACM Transactions on Networking, vol. 5, no. 3, pp. 336-350, June 1997.

[77] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, J.-P. Vasseur, M. Durvy, A. Terzis, A. Dunkels, and D. Culler, “Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks,” in *Proceedings of ACM SenSys*, 2011.

[78] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,” RFC 6550, Mar. 2012.

[79] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks,” RFC 4944, Sep. 2007.

[80] J. Hui and P. Thubert, “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks,” RFC 6282, Sep. 2011.

[81] J. Liu and S. Singh, “ATCP: TCP for Mobile Ad Hoc Networks,” IEEE Journal on Selected Areas in Communications (JSAC), vol. 19, no. 7, July 2001.

[82] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, M. Gerla, “The Impact of Multihop Wireless Channel on TCP Throughput and Loss,” in *Proceedings of IEEE INFOCOM*, 2003.

[83] K. Xu, M. Gerla, L. Qi, and Y. Shu, “Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED,” in *Proceedings of ACM Mobicom*, 20014.

[84] C.-Y. Wan, S. B. Eisenman, and A. T. Campbell, “CODA: congestion detection and avoidance in sensor networks,” in *Proceedings of ACM SenSys*, 2003.

[85] F. Stann and J. Heidemann, “RMST: reliable data transport in sensor networks,” in *Proceedings of IEEE SNPA (Sensor Network Protocols and Applications)*, 2003.

[86] J. Paek and R. Govindan, “RCRT: Rate-Controlled Reliable Transport Protocol for Wireless Sensor Networks,” ACM Transactions on Sensor Netowkrs (TOSN), vol. 7, Issue 3, Sep. 2010.

[87] J. Polastre, R. Szewczyk, and D. Culler, “Telos: Enabling Ultra-Low Power Wireless Research,” in Proceedings of IPSN/SPOTS, Apr. 2005.

[88] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, A. Terzis, A. Dunkels, and D. Culler, “ContikiRPL and TinyRPL: Happy Together,” in *Proceedings of IPSN Workshop*, Apr. 2011.

[89] D. Moss, J. Hui, and K. Klues, “Low power listening,” TinyOS TEP 105.

[90] K. T. Cho and S. Bahk, “Duty cycle optimization for a multi hop transmission method in wireless sensor networks,” IEEE Communications Letters, vol. 14, no. 3, Mar. 2010.

[91] Y. Z. et al, “On deploying relays for connectd indoor sensor networks,” Journal of Communications and Networks, vol. 16, no. 3, June 2014.

초 록

TCP는 인터넷을 이루는 필수적인 프로토콜 중 하나로써, 현재 대부분의 인터넷 트래픽에서 사용되고 있다. 다양한 무선 네트워크의 배치와 스마트 단말의 보급으로 인해 모바일 데이터 트래픽이 폭증한 현 시점에서 TCP는 여전히 대부분의 모바일 트래픽에서 사용되며, 이는 무선 네트워크에서의 TCP 성능에 대한 큰 관심을 불러일으키고 있다. 본 논문에서는 무선 네트워크 환경에서 TCP 성능 향상을 위한 세 가가지 문제를 다룬다.

첫째, LTE와 Wi-Fi와 같은 무선 네트워크에서의 다운스트림 bufferbloat 문제를 다룬다. LTE 뿐만이 아니라 Wi-Fi와 같이 자원 경쟁적인 환경에서의 bufferbloat 문제를 확인하고, 이를 해결하기 위한 수신단 윈도우 조절 기법을 제안한다. 제안하는 기법을 스마트 단말에 구현하고, 실제 LTE 및 Wi-Fi 네트워크에서의 실험을 통해 제안 기법의 성능을 평가한다.

둘째, LTE 네트워크에서의 업스트림 bufferbloat 문제를 고려한다. LTE 네트워크에서 멀티태스킹 사용자의 QoE가 업스트림 bufferbloat으로 인해 크게 저하됨을 확인하고, 사용자 QoE 향상을 위한 패킷 스케쥴러를 제안한다. 제안하는 패킷 스케쥴러를 실제 스마트 단말에 구현하고, 실제 LTE 네트워크에서의 실험을 통해 제안하는 기법의 효과를 평가한다.

셋째, 저전력의 손실이 많이 발생하는 네트워크에서의 TCP 공평성 문제를 고려한다. 저전력의 손실이 많이 발생하는 네트워크에서 노드 간 수율 불공평성이 심각하게 나타남을 확인하고, TCP 공평성 향상을 위한 동적 전송 주기 결정 기법을 제안한다. 제안하는 기법이 자원이 한정되어 있는 저전력의 손실이 많이 발생하는 네트워크

노드에 적용될 수준의 가벼운 프로토콜임을 실제 구현을 통해 확인하고, 테스트베드 실험을 통해 기존 기법과의 공평성 지표를 비교한다.

주요어 : TCP, AQM, bufferbloat, LTE, Wi-Fi, 저전력의 손실이 많이 발생하는 네트워크
학 번 : 2010-30229