Receiver-side TCP Countermeasure to Bufferbloat in Wireless Access Networks

Heesu Im, *Student Member, IEEE,* Changhee Joo, *Senior Member, IEEE,* Taeseop Lee, *Student Member, IEEE,* and Saewoong Bahk, *Senior Member, IEEE*

***Abstract*—Bufferbloat has drawn much attention in the network community for its negative impact on TCP delay performance and user QoE. Recently, it has been more commonly noted in wireless access networks, in part, due to over-provisioned buffer space. Previous solutions that focused only on bufferbloat prevention have suffered from either deployment or fairness problems when coexisting with conventional TCP flows. In this paper, we address the 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 solution competes for shared resource in a fair manner with conventional TCP flow control methods and prevents bufferbloat. We implement our solution in commercial smart devices and verify its performance through real experiments in LTE and Wi-Fi networks.**

***Index Terms*—Bufferbloat, TCP, AQM, fairness, LTE, Wi-Fi**

I. Introduction

Long delays in accessing the Internet through wireless mobile networks have been commonly witnessed [3]. 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 [4]. 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 [6], [7], [8], [9], [10]. 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 [6] 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. Loss-based TCP congestion control has governed network congestion since the beginning of the Internet. 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 [30] that is 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 decreases its *cwnd* that determines the number of bytes in-flight, and increases it if a packet is successfully delivered to the receiver. On occurrence of bufferbloat, packets will not be lost owing to the large-size buffer, so the TCP sender will keep increasing the amount of in-flight data.

There have been many efforts to tackle the persistent queueing and bufferbloat problem. The solutions can be categorized into three types according to the location where they work: sender-oriented, network-oriented, and receiver-oriented [6].

1) Sender-oriented solutions: 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 [11] and Fast TCP [12] can detect the congestion based on round trip time (RTT) and other delay information. They do not suffer from long delay due to the bufferbloat because they decrease *cwnd* when long RTT's are observed [6]. Mo and Walrand [13] have developed an interesting congestion measure, named the decoupled fairness criteria that use *cwnd* and the measured bandwidth-delay product (BDP). Roughly, they have shown that setting *cwnd* to the BDP (and thus controlling *cwnd* irrespective of loss detection) maximizes a variety of utility functions and achieves fairness. A main weakness of delay-based approaches is that they suffer from bandwidth starvation when they coexist with loss-based approaches [14].

2) Network-oriented solutions: Active queue management (AQM) at intermediate routers can prevent the buffer from bloating [5], [6]. AQM schemes such as random early detection (RED) [15], exponential RED (E-RED) [16], and random exponential marking (REM) [17] discard (or mark if explicit congestion notification (ECN) is enabled [18]) in-

H. Im, T. Lee, and S. Bahk are with the Department of Electrical and Computer Engineering, Seoul National University, Seoul, 151-742, Korea. E-mail: hsim, tslee, sbahk@netlab.snu.ac.kr.

C. Joo is with the School of Electrical and Computer Engineering, UNIST, Ulsan 689-798, Korea.
E-mail: cjoo@unist.ac.kr.

coming packets in a probabilistic manner before the buffer becomes full, and can inform TCP senders of incipient congestion such that they reduce their $cwnd$'s. Despite many advantages of queue management, few intermediate routers enable AQM in practice by default due to the difficulty in parameter settings. Recently, a sojourn-time based AQM scheme named CoDeL has been developed to simplify the configuration while solving the bufferbloat problem [5].

3) Receiver-oriented solutions: 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 and hardly need flow control, and set *rwnd* sufficiently large by default, which is named Auto-tuning [6]. On the other hand, the receiver can utilize *rwnd* to restrict the amount of in-flight data [19]. For example, most smartphone manufacturers limit *rwnd* to some threshold values to prevent the potential bufferbloat problem. The threshold values depend on underlying networks and remain static. Dynamic Receive Window Adjustment (DRWA) [6] tackles the bufferbloat problem by exploiting dynamic control of *rwnd*. In DRWA, the receiver adjusts *rwnd* such that the estimated RTT remains close to the minimum RTT over time, which is, in principle, similar to TCP-Vegas. However, the resemblance to TCP-Vegas also implies that DRWA is likely to have a fairness problem when loss-based TCP flows coexist in competing environments, resulting in its performance degradation. Through experimental results, we show that the performance of DRWA indeed significantly degrades when competition for shared buffers exists, particularly in Wi-Fi networks.

In this paper, we propose a receiver side solution, called Receiver-side TCP Adaptive queue Control (RTAC), to tackle the bufferbloat problem. Unlike DRWA, RTAC can coexist with conventional TCP flows without performance degradation. In RTAC, using the estimated *cwnd* and the measured RTT, 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.

One of the main advantages of the receiver-oriented approaches is that they can prevent bufferbloat without the intervention of service providers, and can be quickly and easily deployed by updating the firmware. Through experiments in public long term evolution (LTE) and Wi-Fi networks, we show that RTAC successfully prevents the bufferbloat and achieves very good delay performance. In resource competing environments, TCP flows of RTAC achieve throughput performance compatible with those of conventional TCP receivers, and they coexist in a fair manner. The contributions of this paper are summarized as follows:

- We identify the bufferbloat problem *in 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* solution named RTAC to tackle the bufferbloat problem based on the dynamics of TCP and AQM. It can successfully prevent the bufferbloat and achieves fair resource sharing with TCP flows of conventional receivers.
- We evaluate the performance of our solution through experiments. Implementing RTAC on commercially available Android phones (Samsung Galaxy S2 (E120K)), we evaluate its effectiveness in solving the bufferbloat problem in both LTE and Wi-Fi networks.

The rest of the paper is organized as follows. We first overview the dynamics of TCP and AQM in Section II. In Section III, we develop a receiver-side solution that aims to prevent bufferbloat while achieving fairness under the coexistence with various types of TCP flows. Section IV describes experimental setup and configuration. Experimental results to evaluate our solution in comparison with the state-of-the-art follow in Section V. We conclude our paper in Section VI.

II. 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 [16], [20], [21]:

$$\begin{array}{ll} \text{Maximize} & \sum_{s \in S} U_s(x_s) \\ \text{subject to} & y_l \le c_l, \quad \text{for all links } l. \end{array} \tag{1}$$

For example, TCP-Reno is a solution to (1) when the utility function is

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

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

We consider the standard dual problem of (1), and obtain the Lagrangian function as

$$\begin{aligned} L(x,p) &:= \sum_{s\in S} U_s(x_s) - \sum_l p_l(y_l - c_l) \\ &= \sum_{s\in S} (U_s(x_s) - x_s q_s) + \sum_l p_l c_l, \end{aligned} \tag{3}$$

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), the optimal utility sum is achieved when, for all s,

$$U_s'(x_s) = q_s. \tag{4}$$

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 [6]. Hence, in wireless access networks, we approximate the sum price as the price of an access link, i.e.,

$$q_s \approx p_{last\_hop} = f(Q_s), \tag{5}$$

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 [16], [17]. Throughout this paper, 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

$$f(Q_s) = K(Q_s + 1), \tag{6}$$

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 $Q_s = 0$.

III. Receiver-side TCP Adaptive Queue Control against Bufferbloat

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.

A. Receiver-side window control

Most existing TCP congestion controls show throughput performance compatible with TCP-Reno [30], [31], [32], and a flow is called TCP-Friendly if it achieves throughput compatible with TCP-Reno [22]. In this regard, we try to control the TCP sender to transmit at a rate compatible with TCP-Reno using its utility function (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* [23]. The latter can be exploited to control the transmission rate from the receiver side [6], [24]. 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 $rwnd$, i.e., if *cwnd* $<$ *rwnd*, from (2) and (4), we have

$$\frac{cwnd}{RTT} = \frac{\sqrt{a}}{RTT_m}\frac{1}{\sqrt{q_s}}, \tag{7}$$

where the coefficient a is known to be $\frac{3}{2}$ for TCP-Reno [25]. 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 (6), the receiver calculates the loss probability (link price) $\hat{q}_s$, and advertises *rwnd* as

$$rwnd = \sqrt{a}\frac{RTT}{RTT_m}\frac{1}{\sqrt{\hat{q}_s}}, \tag{8}$$

where $\hat{q}_s = K(Q_s + 1)$ and $a = \frac{3}{2}$. Then from (7) and (8), the transmission rate is controlled as

$$x_s = \frac{w_s}{RTT} = \frac{\sqrt{a}}{RTT_m} \cdot \min\left\{\frac{1}{\sqrt{q_s}}, \frac{1}{\sqrt{\hat{q}_s}}\right\}. \tag{9}$$

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.

B. Delay measurement and queue length estimation

Our receiver-side solution (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 easily 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 [26] 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. As noted, the access link queueing delay commonly dominates the total queueing delay over the path in wireless access networks [6], 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 the estimated queue length $\hat{Q}_s$ as

$$\hat{Q}_s = (\widehat{RTT} - \widehat{RTT}_m)\frac{\hat{w}_s}{\widehat{RTT}}, \tag{10}$$

where the estimated transmission window size $\hat{w}_s$ can be obtained at the receiver using the method in [27].

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.

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

On receiving a packet:

if (measured RTT is valid) **then**
  $\widehat{RTT} \leftarrow (1-\beta)\cdot\widehat{RTT} + \beta\cdot$ (measured RTT)
  Update $\widehat{RTT}_m$ if necessary.
end if

On copying data from receiver buffer to user space:

$pkts \leftarrow pkts + copied\_data\_in\_packets$
if $(current\_time - last\_update\_time) \geq \widehat{RTT}$ **then**
  $\hat{w}_s \leftarrow \alpha\cdot\hat{w}_s + (1-\alpha)\cdot pkts$
  $\hat{Q}_s \leftarrow (\widehat{RTT} - \widehat{RTT}_m)\frac{\hat{w}_s}{\widehat{RTT}}$
  $\hat{q}_s \leftarrow K(\hat{Q}_s + 1)$
  rwnd $\leftarrow \sqrt{a}\frac{\widehat{RTT}}{\widehat{RTT}_m}\frac{1}{\sqrt{\hat{q}_s}}$
  $pkts \leftarrow 0$
  $last\_update\_time \leftarrow current\_time$
end if

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 [6]. 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.

C. Configuration of RTAC

From (8) and (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 [28]. The precise setting of K is beyond the scope of this paper 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. Then *rwnd* needs to be set no smaller than the BDP and no greater than 1.5 times BDP considering the sawtooth-like behavior of TCP. 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 (6), (8), and (10):

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

which leads to

$$K = \frac{a}{\theta-1}\left(\frac{1}{BDP}\right)^3. \tag{11}$$

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 $\theta \in [1.3, 1.4]$.

IV. 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 CISCO AP AIR-SAP1602I-K-K9. We use Samsung Galaxy S2 (E120K) and modify the manufacturer's open source code [1]. 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

TABLE I
EXPERIMENTAL SETUP

TCP Sender	CUBIC with the timestamp option on MSS = 1448 Bytes
Server	Ubuntu 12.04 LTS (Kernel 3.2.0-54) Intel(R) Xeon(R) CPU E3-1270 V2 (3.50GHz)
Smartphone	Samsung Galaxy S2 (Model E120K) OS: Kernel version 3.0.8, Android Icecream Sandwich (4.0.3)
LTE networks	Operator: Korea Telecom (KT) Bandwidth: 100 Mbps
Wi-Fi networks	Access Point: CISCO AIR-SAP1602I-K-K9 Bandwidth: 72 Mbps (802.11n)
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

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 [2]. The detailed settings are summarized in Table I.

A. 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[1] 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

$$rwnd = \min\{2\hat{w}_s^*,\ tcp\_rmem\_max\}, \qquad (12)$$

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

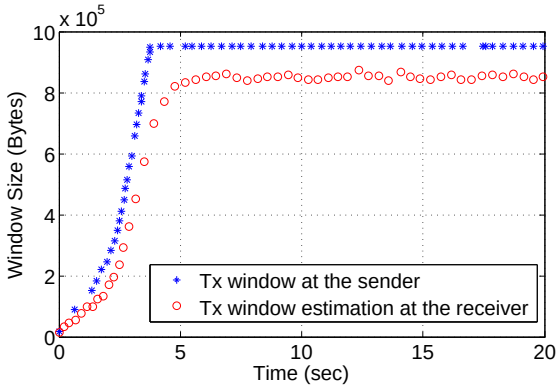
$$rwnd = \lambda \frac{\widehat{RTT}_m}{\widehat{RTT}} \hat{w}_s, \qquad (13)$$

where λ is a configuration parameter set to 3 in most scenarios [6].

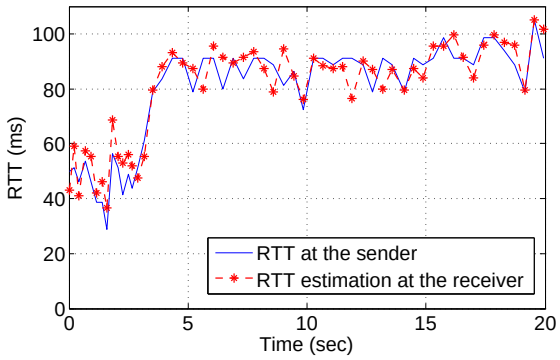
RTAC controls the receive window according to the dynamics of TCP and AQM. From (6), (8), and (10), it sets the receive window as

$$rwnd = \sqrt{\frac{3}{2}} \frac{\widehat{RTT}}{\widehat{RTT}_m} \frac{1}{\sqrt{K\hat{w}_s\left(1-\frac{\widehat{RTT}_m}{\widehat{RTT}}\right)+1}}. \qquad (14)$$

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



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



(b) Round-trip time

Fig. 1. Transmission window and RTT at the sender and their measurements at the receiver. The receiver underestimates $\hat{w}_s$ by about 11% compared to the sender. Regarding the RTT, the sender and the receiver show similar results.

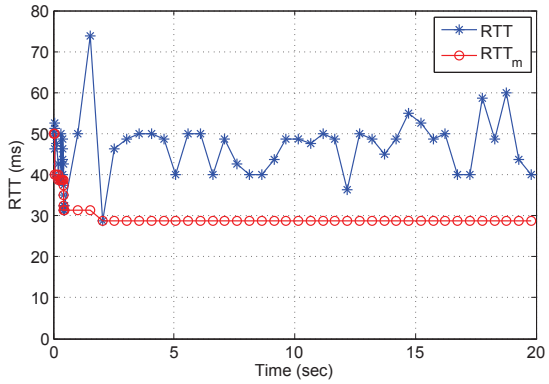
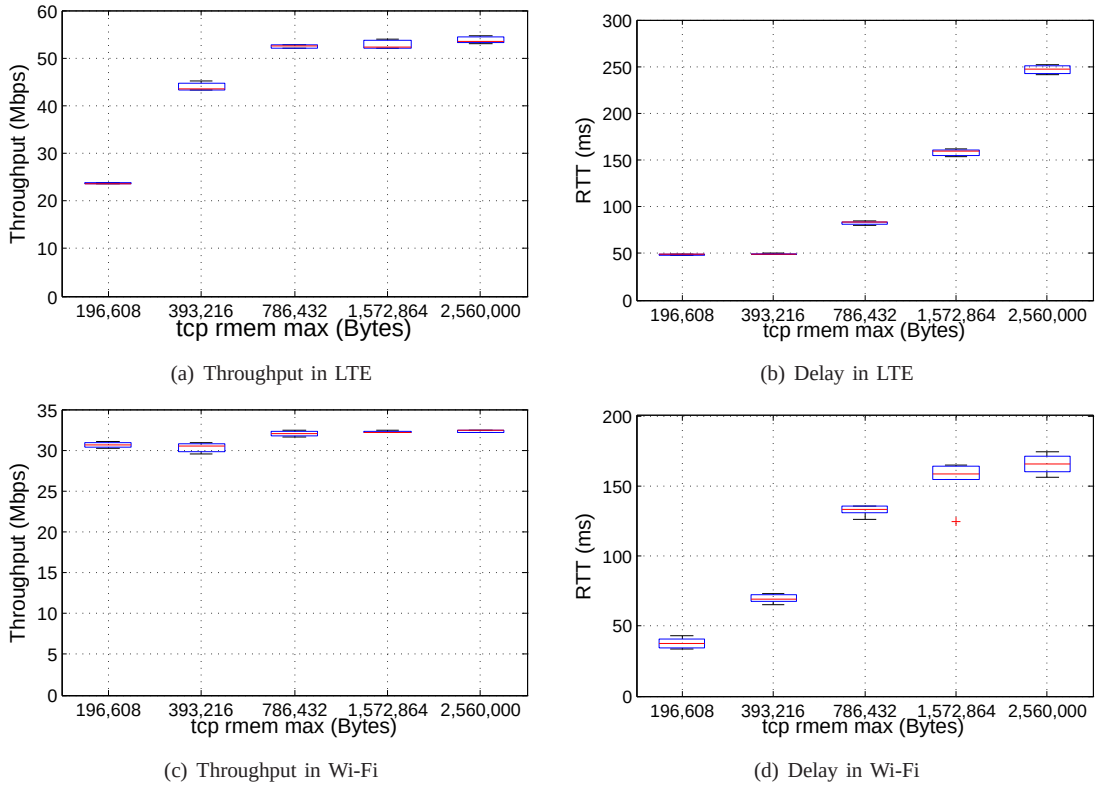


Fig. 2. RTT and RTT_m in the LTE network with low rate. With *tcp_rmem_max* (i.e., 50KBytes) which does not incur bufferbloat, peak RTT almost reaches double RTT_m.

Using (11) with $\theta = 1.38$, we set K as $3 \cdot 10^{-7}$ for the LTE network, and $5 \cdot 10^{-5}$ 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. For example, if the receiver has a greater RTT estimation than the sender, it may unnecessarily throttle the sender, resulting in underutilization of the wireless capacity. Hence,



(a) Throughput in LTE

(b) Delay in LTE

(c) Throughput in Wi-Fi

(d) Delay in Wi-Fi

Fig. 3. TCP performance with Auto-tuning receiver in LTE and Wi-Fi networks for different *tcp_rmem_max* values. As *tcp_rmem_max* increases, the throughput performance becomes saturated and the delay performance gets worse. The optimal setting of *tcp_rmem_max* depends on various factors of the underlying network. Rectangles are used to indicate the 25th and 75th percentiles.

it is important that the sender and the receiver have the same estimation of the transmission window (i.e., $\hat{w}_s$) and the round-trip time (i.e., $\widehat{RTT}$), and that the receiver has accurate RTT_m estimation. Since these measurements are critical for appropriate operation of the receiver-side control, we first evaluate their accuracy.

Fig. 1 shows the measurements of $\hat{w}_s$ and $\widehat{RTT}$ at the receiver (after taking the running averages with parameters α and β) along with the sender's *cwnd* and RTT. Though the receiver underestimates w_s by 11% compared to the sender in a saturated region, and there exists 4.4% mismatch in RTT, they are overall alike. This is because the receiver-side estimation of $\hat{w}_s$ does not take into account lost or delayed packets [27]. In contrast, 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 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. We set $\Delta_{LTE} = 18$ msec for LTE and $\Delta_{Wi-Fi} = 8$ msec for Wi-Fi.

V. Experimental Results

In this section, we present experimental results to observe bufferbloat in WiFi and cellular networks under various scenarios that include flow competition cases for shared resource. We also consider fairness performance and TCP variants for performance evaluation.

A. Bufferbloat in wireless access networks

The bufferbloat in public cellular networks has been noted in the literature [6], [7]. We confirm the previous results in LTE networks and observe that the problem also occurs in Wi-Fi networks. To the best of our knowledge, this is a first academic observation of the bufferbloat problem in Wi-Fi networks.

The severity of 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 [6]. We first measure the throughput and RTT at Auto-tuning receiver in the LTE network for 30 seconds. Figs. 3(a) and 3(b) show that the setting of

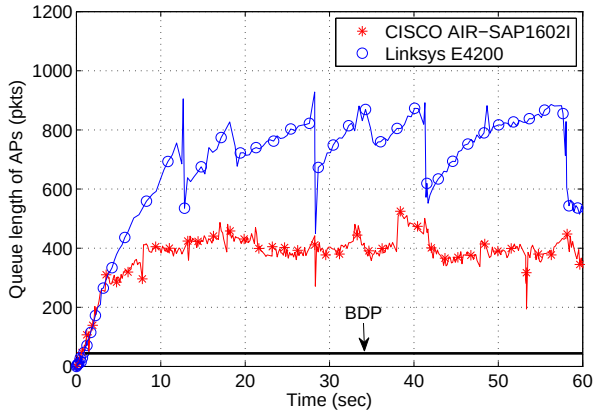


Fig. 4. Queue length measurements in the Wi-Fi network with two different AP types. Depending on the AP type, the AP queue length reaches hundreds of packets while the network BDP is 44 packets, resulting in large queueing delay.

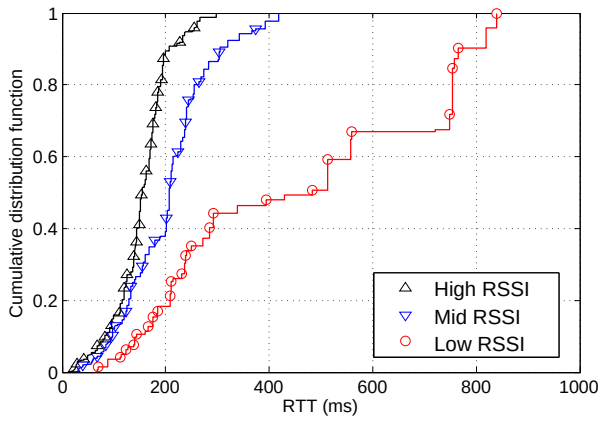


Fig. 5. Delay distribution in the Wi-Fi network with three different distance settings from AP. In general, with the signal strength weaker, packets experience longer delay. The longest delay of 773 msec is observed when the device is located the farthest where the average RSSI is -73 dbm.

tcp_rmem_max has a critical impact on performance. A too small value leads to low throughput (i.e., underutilization) and a too large 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. 3(c) and 3(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* much smaller than 200 KBytes.

The degree of RTT increase in the Wi-Fi network is strongly related with the buffer size of the AP. We have measured queue lengths of commercial APs (i.e., CISCO AIR-SAP1602I and Linksys E4200) by using (10) at the receiver. Fig. 4 shows queue length measurements with Auto-tuning receiver. Although the network BDP is 44 packets, the AP queue length reaches hundreds of packets which is much more than the BDP, leading to unnecessarily long RTT.

The delay performance due to bufferbloat can deteriorate when the signal strength is weak. We have evaluated the performance of Auto-tuning receiver located at three different distances from the AP. For the duration of 30 seconds, we recorded both average received signal strength indicator (RSSI) value and instantaneous RTT. The measured average RSSIs at those locations are -53 dbm (closest), -68 dbm (medium), and -73 dbm (farthest), respectively. Fig. 5 shows the cumulative RTT distribution of received packets at each location. It is clear that the lower the signal strength, the higher the delay packets suffer from. For the low RSSI case, the largest RTT measurement was 773 msec. The results stem from frequent retransmissions and low bandwidth. When RSSI is low, packets are more likely to experience corruption and will be retransmitted by automatic repeat request (ARQ). Also, low RSSI often leads to low transmission rate and accordingly long transmission delay. The throughputs achieved at each location are 30 Mbps (closest), 19 Mbps (medium), and 7.1 Mbps (farthest), respectively.

Recently, the factory default setting of *tcp_rmem_max* has been increasing. For example, the setting of Galaxy S2 (E120K) in the LTE network is $1,220$ 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 is related to the increased capacity in wireless access links. Accordingly 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. In addition, we set *tcp_rmem_max* to the same value for DRWA and RTAC because it is sufficiently high not to limit $rwnd$.

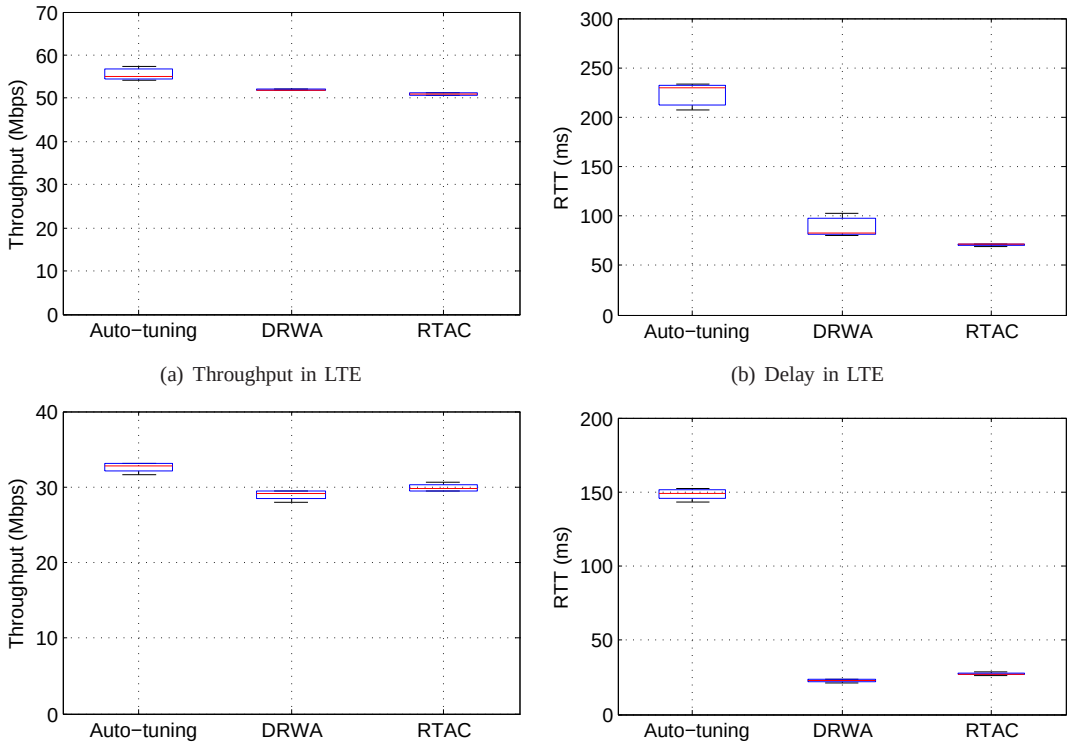
B. Prevention of bufferbloat

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

Fig. 6 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 33 Mbps and 148 msec in Wi-Fi. DRWA achieves similar throughput as Auto-tuning but much better delay performance. It shows 52 Mbps and 89 msec in LTE, and 29 Mbps and 23 msec in Wi-Fi. Also, RTAC shows 51 Mbps and 71 msec in LTE, and 30 Mbps and 27 msec in Wi-Fi. Both DRWA and RTAC maintain low queue length well without underutilization of the wireless link by controlling *rwnd*, thereby leading to prevention of bufferbloat.

C. Fairness of TCP flows with various receiver types

For a single flow, both DRWA and RTAC are successful in preventing bufferbloat. We now consider the scenarios where multiple TCP flows coexist and compete for the resource of the BS or AP. It has been known that the BS



(a) Throughput in LTE

(b) Delay in LTE

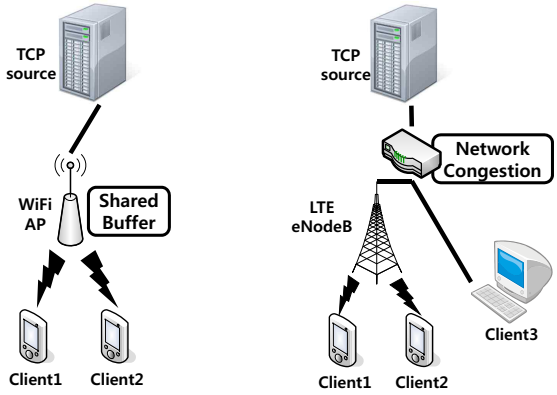
(c) Throughput in Wi-Fi

(d) Delay in Wi-Fi

Fig. 6. Prevention of bufferbloat in LTE and Wi-Fi networks. Both DRWA and RTAC prevent bufferbloat in LTE and Wi-Fi networks so that they achieve high throughput and low delay, while Auto-tuning suffers from long delay. Rectangles are used to indicate the 25th and 75th percentiles.

in a cellular network maintains a separate buffer for each flow [29], 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. 7(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. 7(b), which creates a wired bottleneck link just before the server. The additional client runs Auto-tuning and competes with the wireless clients at the wired link. We denote this scenario as Indirect Competition (IC).

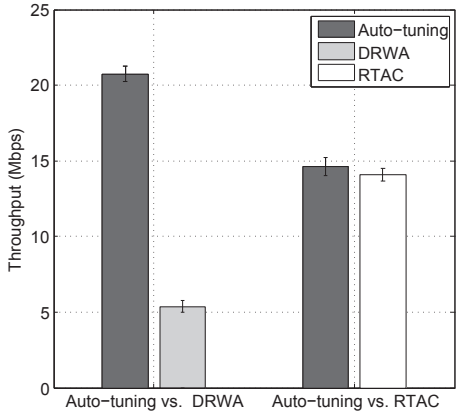


(a) Direct competition (DC) in Wi-Fi

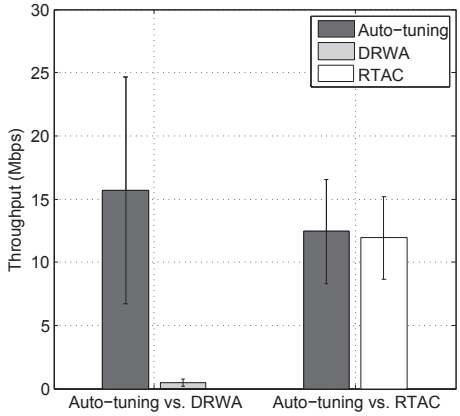
(b) Indirect competition (IC) in LTE

Fig. 7. Resource competition scenarios with two wireless clients. In LTE, since the clients do not directly compete for buffers at the BS owing to the use of per-flow queueing, we create a bottleneck link in the wired network by adding Client 3.

In the DC scenario in Wi-Fi, we first run Auto-tuning for one client and DRWA for the other. Fig. 8(a) illustrates throughput performance of the two TCP flows, in which DRWA achieves only 26% 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., 95% throughput of Auto-tuning), sharing buffers of the AP in a fair manner. In the IC scenario of LTE, we observe similar results as shown in Fig. 8(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-



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



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

Fig. 8. Coexitence of RTAC and Auto-tuning in LTE and Wi-Fi networks. RTAC successfully coexists with Auto-tuning and achieves 95% throughput of Auto-tuning in Wi-Fi and LTE networks while DRWA achieves only 26% throughput of Auto-tuning in Wi-Fi DC scenario and 3% in LTE IC scenario when competing with Auto-tuning.

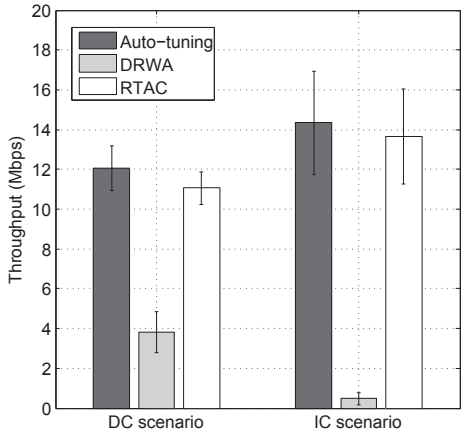


Fig. 9. Fairness among Auto-tuning, DRWA, and RTAC receivers. When three TCP flows are competing together, RTAC and Auto-tuning achieve similar performances while DRWA suffers from starvation.

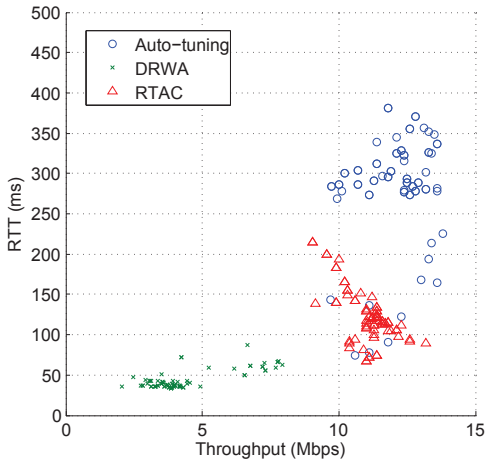


Fig. 10. Throughput and RTT performance among Auto-tuning, DRWA, and RTAC receivers in the Wi-Fi network. RTAC shows better throughput performance than DRWA, and better RTT performance than Auto-tuning.

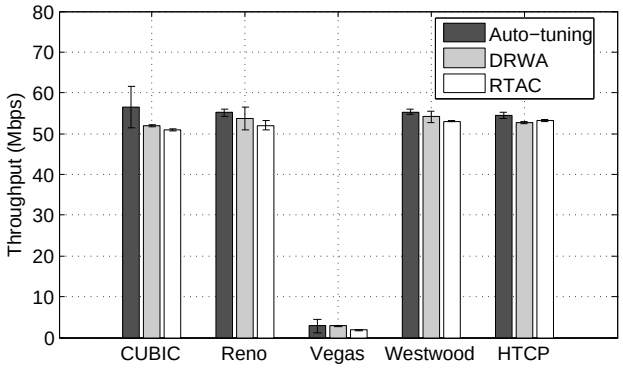
tuning.

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. 7 by adding one more wireless client. Fig. 9 shows the throughputs of the three receivers in Wi-Fi and LTE networks. Again, in both cases, RTAC is compatible with Auto-tuning while DRWA suffers from severe starvation. In addition, we measure RTT performance both in LTE and Wi-Fi networks. In the LTE network, RTT performances of Auto-tuning, DRWA, and RTAC receivers are similar because network congestion happens at the same router. On the other hand, RTT performance in Wi-Fi shows an unexpected result. As shown in Fig. 10, RTAC achieves better throughput performance than DRWA while showing better RTT performance than Auto-tuning. It seems that the latest Wi-Fi AP such as CISCO AIR-SAP1602I-K-K9 does not manage its buffers in a shared manner. Nevertheless, RTAC successfully coexists with Auto-tuning in terms of throughput and RTT.

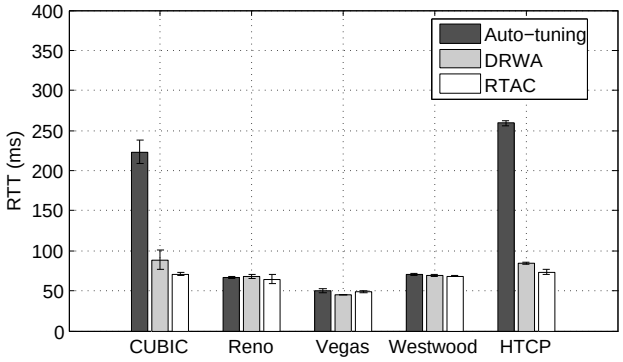
D. The impact of TCP variants

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 [31], 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 [32].

Fig. 11 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 bufferbloat prevention of a single TCP flow, but not TCP-Vegas. TCP-Vegas suffers from low throughput regardless of TCP receiver types, which comes from inaccurate estimation on RTT_m. As we discussed before, RTT_m changes over time in the LTE network, and

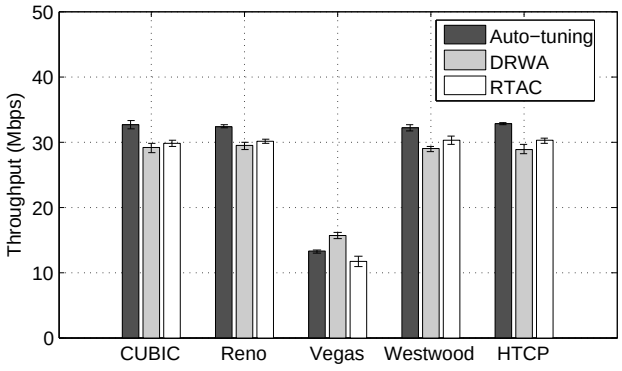


(a) Throughput

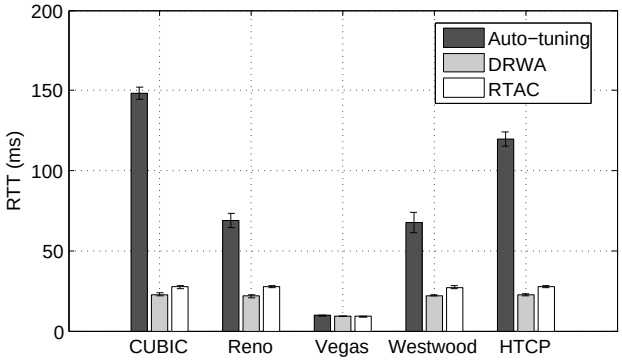


(b) Delay

Fig. 11. Performance of a single TCP flow with various TCP sender types in the LTE network. High-speed TCP variants such as CUBIC and HTCP aggravate the bufferbloat in Auto-tuning, and TCP-Vegas suffers from low throughput due to incorrect estimation of the minimum RTT at the sender.



(a) Throughput



(b) Delay

Fig. 12. Performance of a single TCP flow with various TCP sender types in the Wi-Fi network.

thus without appropriate compensation, the sender will have $\widehat{RTT}_m < RTT_m$ most of the time, which leads to an unnecessarily small *cwnd* .

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. 12 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 bufferbloats of Auto-tuning with TCP-Reno and TCP Westwood become more striking.

Finally, we evaluate the fairness of Auto-tuning, DRWA, and RTAC, under different TCP variants at the sender. We consider the same three client scenarios of DC in Wi-Fi and IC in LTE. Fig. 13(a) depicts the throughput performance of the three clients for various TCP sender types in Wi-Fi. It shows that RTAC achieves more than 92% throughput of Auto-tuning for all TCP variants except TCP-Vegas. This means that RTAC and Auto-tuning can coexist well. In contrast, DRWA achieves only 15–37% throughput of Auto-tuning .

In LTE, Fig. 13(b) shows that Auto-tuning and RTAC achieve similar throughputs while DRWA achieves much lower throughput except with 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.

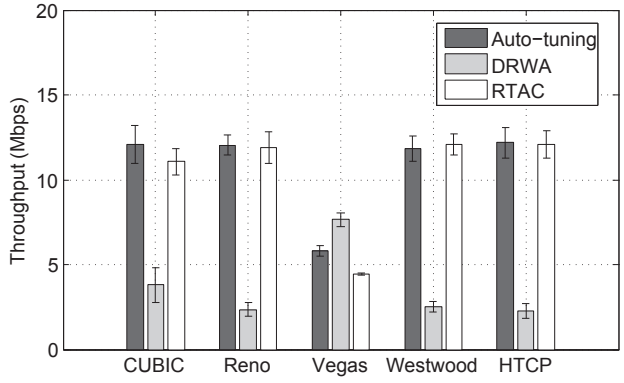
E. Discussion

Our receiver-side solution of RTAC performs well with accurate estimation of BDP. Overestimation leads to failure in bufferbloat prevention, and underestimation can result in underutilization of network capacity or instability. In general, precise estimation of BDP is difficult, in particular, in wireless access networks due to the dynamic nature of wireless channel, traffic, and user mobility.

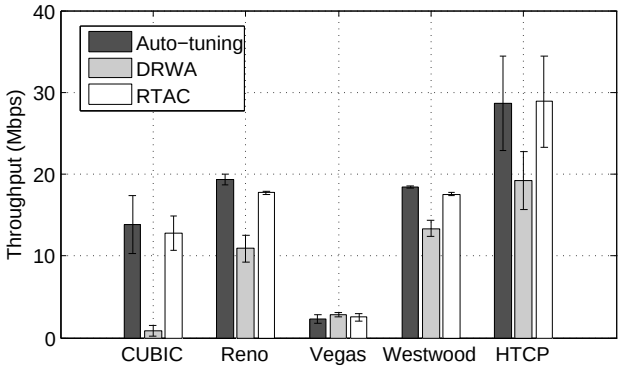
There have been a few bandwidth estimation techniques that have been proposed, for example, [33] for cellular networks and [34], [35] for Wi-Fi networks. They can be incorporated with RTAC to improve the accuracy of BDP estimation. Similarly, precise estimation on RTT is of great importance, which we leave as future work.

VI. Conclusion

In this paper, we have presented a receiver-side countermeasure, named RTAC, to solve bufferbloat in wireless access networks such as LTE and Wi-Fi networks. According to TCP dynamics, a RTAC receiver estimates



(a) Direct competition in Wi-Fi



(b) Indirect competition in LTE

Fig. 13. Performance under competing environments with various TCP sender types. Overall RTAC achieves throughput performance compatible with Auto-tuning for various TCP types, while DRWA fails to share the buffering resource in a fair manner except with TCP-Vegas.

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. Through extensive experimental measurements, we have verified that RTAC successfully prevents bufferbloat, achieving good delay performance without sacrificing throughput performance, and that it achieves fair resource sharing with compatible TCP flows, outperforming the state-of-the-art schemes.

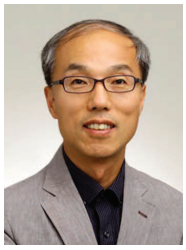
REFERENCES

[1] http://opensource.samsung.com/
[2] http://sourceforge.net/projects/iperf/
[3] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating The Edge Network," in *Proceedings of ACM IMC*, 2010.
[4] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," ACM Queue, vol. 9, issue 11, pp. 40-54, Nov. 2011.
[5] K. Nichols and V. Jacobson, "Controlling Queue Delay," ACM Queue, vol. 10, issue 5, pp. 20-34, May 2012.
[6] H. Jiang, Y. Wang, K. Lee, and I. Rhee, "Tackling Bufferbloat in 3G/4G Networks," in *Proceedings of ACM IMC*, 2012.
[7] 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.
[8] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroiu,"Characterizing Residential Broadband Networks," in *Proceedings of ACM IMC*, 2007.
[9] 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.
[10] M. Allman, "Comments on Bufferbloat," ACM SIGCOMM Computer Communication Review (CCR), vol. 43, no. 1, pp. 31-37, Jan. 2013.
[11] 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.
[12] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proceedings of INFOCOM*, 2004.
[13] 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.
[14] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and Comparison of TCP Reno and Vegas," in *Proceedings of INFOCOM*, 1999.
[15] 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.
[16] 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.
[17] 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.
[18] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001.
[19] 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
[20] F. Kelly, "Charging and Rate Control for Elastic Traffic," European Transactions on Telecommunications, vol. 8, no. 1, pp. 33-37, Jan. 1997.
[21] 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.
[22] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 5348, Sep. 2008.
[23] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, Sept. 2009.
[24] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming," Technical report, NCSU, 2000.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] 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.
[31] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proceedings of PFLDNet Workshop*, 2004.
[32] 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.
[33] A. Chakraborty, V. Navda, V. N. Padmanabhan, and R. Ramjee, "Coordinating Cellular Background Transfers using LoadSense," in *Proceedings of Mobicom*, 2013.
[34] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye, "Bandwidth Estimation in Broadband Access Networks," in *Proceedings of ACM IMC*, 2004.
[35] M. Li, M. Claypool, and R. Kinicki, "Wbest: a bandwidth estimation tool for IEEE 802.11 wireless networks," in *Proceedings of IEEE LCN*, 2008.

Heesu Im received B.S. and M.S. degrees in the school of Electrical Engineering & Computer Science, Seoul National University in 2008 and 2010, respectively. He is currently a Ph.D. candidate in the school of Electrical Engineering & Computer Science, Seoul National University. His research interests include network protocol design, wireless sensor networks, and P2P networks.

Changhee Joo (S??8-M'05-SM'13) received his Ph.D degree from Seoul National University, Korea. From 2005, he had been with Purdue University and the Ohio State University, USA. In 2010, he joined Korea University of Technology and Education, Korea, and now he is with Ulsan National Institute of Science and Technology (UNIST), Korea. His research interests include resource allocation in wireless networks across layers, performance optimization of the Internet protocols, and the design of wireless sensor networks and vehicular networks. He is an editor of Journal of Communications and Networks, and has served on the technical committees of several primary conferences. He is a member of IEEE, and a recipient of the IEEE INFOCOM 2008 best paper award.

Taeseop Lee is currently a Ph.D. candidate in the School of Electrical Engineering and Computer Science, Seoul National University. He received his B.S. degree from Seoul National University, in 2011. His research interests include fourth-generation wireless networks, Massive MIMO, and multi-path TCP.



Saewoong Bahk (M'94-SM'06) received B.S. and M.S. degrees in Electrical Engineering from Seoul National University in 1984 and 1986, respectively, and the Ph.D. degree from the University of Pennsylvania in 1991. From 1991 through 1994 he was with AT&T Bell Laboratories as a member of technical staff where he worked for network management. In 1994, he joined the school of electrical engineering at Seoul National University and currently serves as a professor. He has been serving as TPC members for various conferences including ICC, GLOBECOM, INFOCOM, PIMRC, WCNC, etc. He was TPC chair for IEEE Vehicular Technology Conference (VTC) - 2014 Spring. He was awarded for excellent teaching and excellent performance from the engineering college of Seoul National University in 2010 and 2013, respectively. He is on the editorial boards of IEEE Transaction on Wireless Communications (TWireless), Computer Networks Journal (COMNET), and Journal of Communications and Networks (JCN). His areas of interests include performance analysis of communication networks and network security. He is an IEEE Senior Member and a Member of Who's Who Professional in Science and Engineering.