Tackling Bufferbloat in 3G/4G Mobile Networks

Haiqing Jiang, Yaogong Wang, Kyunghan Lee and Injong Rhee
North Carolina State University, Raleigh, NC, USA
hjiang5, ywang15, klee8, rhee@ncsu.edu

ABSTRACT

With the exponential growth of hand-held devices like smart phones and tablet computers, a deep investigation of TCP performance in cellular networks is becoming increasingly important. In this paper, we conducted extensive measurements over the 3G/4G networks of four major U.S. carriers as well as the largest carrier in Korea and discovered a significant problem: the bufferbloat in cellular networks nullifies loss-based congestion control and allows excessive growth of the TCP congestion window, resulting in extremely long delays and throughput degradation. To alleviate this issue, smart phone vendors put an upper bound on the sender's congestion window by advertising a static receive window smaller than the actual receive buffer size. This simple trick helps mitigate the problem, but has fundamental limitations due to its static nature. In this paper, we propose a dynamic receive window adjustment (DRWA) algorithm to improve TCP performance over bufferbloated cellular networks. According to our extensive real-world tests, DRWA may reduce the delay by 25 $\sim$ 49% in general cases and increase TCP throughput by up to 51% in some specific scenarios. Since DRWA requires modification at the client side (e.g., smart phones) only and is fully compatible with existing TCP protocol, it is immediately deployable.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Design, Measurement, Performance

Keywords

TCP, Cellular networks, Bufferbloat, Receive window adjustment

1. INTRODUCTION

TCP is the dominant transport layer protocol of the current Internet, carrying around 90% of the total traffic [10,14]. Hence, the performance of TCP is of utmost importance to the well-being of the Internet and has direct impacts on user experience. Although TCP is well-studied in traditional networks, its performance over cellular networks has not been given adequate attention.

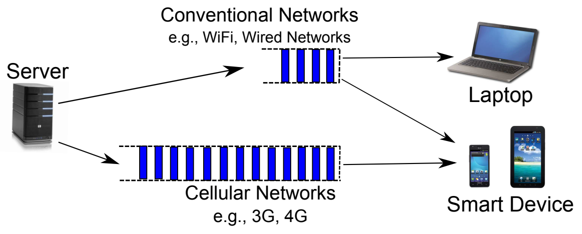


Figure 1: Over-buffering has been widely observed in the current Internet [6] but is especially severe in cellular networks, resulting in up to several seconds of round trip delay and even throughput degradation.

According to our measurements, TCP has a number of performance issues in this relatively new environment, including extremely long delays and sub-optimal throughput in certain scenarios. The reasons behind such performance degradations are two-fold. First, most of the widely deployed TCP implementations use loss-based congestion control where the sender will not slow down its sending rate until it sees packet loss. Second, most cellular networks are over-buffered to accommodate bursty traffic and channel variability [6, 12] as depicted in Figure 1. The exceptionally large buffer along with link layer retransmission conceals packet losses from TCP senders. The combination of these two facts leads to the following phenomenon: the TCP sender continues to increase its sending rate even if it has already exceeded the bottleneck link capacity since all of the overshot packets are absorbed by the buffers. This results in up to several seconds of round trip delays.

To solve this problem, smart phone vendors have devised with a small trick: they set a relatively small value for TCP maximum receive buffer size although the physical buffer size is much larger. Since the advertised receive window ($rwnd$) cannot exceed the receive buffer size and the sender cannot send more than what is allowed by the advertised receive window, this limit effectively prevents the TCP congestion window ($cwnd$) from excessive growth and controls the RTT (round trip time) experienced by the flow in a reasonable range. However, since the limit is statically configured, it is sub-optimal in many scenarios, especially considering the dynamic nature of the wireless mobile environment. In high speed long distance networks (e.g., downloading from an overseas server over 4G LTE (Long Term Evolution) network), the static value is too small to saturate the link and results in severe throughput degradation. On the other hand, in small bandwidth-delay product (BDP) networks, the static value is too large and the flow experiences excessively long RTT.

There are many possible ways to solve this problem, ranging from modifying TCP congestion control algorithm at the sender to adopting Active Queue Management (AQM) at the base station. However, all of them incur considerable deployment cost. In this paper, we propose dynamic receive window adjustment (DRWA), a light-weight, receiver-based solution that is immediately deployable. Since DRWA requires modification at the receiver side only and is fully compatible with existing TCP protocol, carriers or device manufacturers can simply issue an OTA (over the air) update to smart phones so that they can immediately enjoy better performance even when interacting with existing TCP servers.

DRWA is similar in spirit to delay-based congestion control algorithms but runs on the receiver side. It modifies the existing receive window adjustment algorithm of TCP to indirectly control the sending rate.. Roughly speaking, DRWA increases the advertised window when the current RTT is close to the minimum RTT we have observed so far and decreases it when RTT becomes larger due to queuing delay. With proper parameter tuning, DRWA could keep the queue size at the bottleneck link small yet not empty so that throughput and delay experienced by the TCP flow are both optimized. Our extensive experiments show that DRWA keeps RTT 25 $\sim$ 49% lower than the current TCP implementations in smart phones while achieving similar throughput in ordinary cases. In large BDP networks, DRWA can achieve up to 51% throughput improvement.

In summary, the key contributions of this paper include:

- We report extensive observations of TCP's behavior in a range of various cellular networks and point out its negative impacts on user experience.
- We anatomize the TCP implementation in state-of-the-art smart phones and locate the root causes of its performance issue in cellular networks.
- We propose a simple and backward-compatible remedy that is experimentally proven to be safe and effective. It provides substantial fixes to TCP performance issues and is immediately deployable.

The rest of the paper is organized as follows. Section 2 introduces the bufferbloat problem in cellular networks and the current TCP receive window adjustment algorithm. Section 3 visualizes TCP's performance issue in cellular networks via extensive measurements over the cellular networks of the four major U.S. carriers as well as the largest cellular carrier in Korea. We then anatomize the root causes of the problem and propose our solution in Section 4. Finally, we show the experimental performance of DRWA in comparison to the current implementation in Section 5 and conclude our work in Section 6.

2. PRELIMINARIES

2.1 Bufferbloat in Cellular Networks

Bufferbloat, as termed by Gettys [6] in late 2010, is an abnormal phenomenon in current Internet experience where excessive buffering of packets causes unnecessarily high end-to-end latency and jitter, as well as throughput degradation. It is not specific to cellular networks but might be most prominent in this environment. These excessive buffers were originally introduced into cellular networks for a number of reasons. First, the channel status of cellular links fluctuates rapidly and the corresponding channel rate varies from dozens of Kbps to tens of Mbps. Second, the data traffic over such links is highly bursty. To absorb such bursty traffic over such a variable channel, the simple yet effective approach adopted by current cellular networks is to provide large buffers. These buffers smooth the bursty traffic and reduce the packet loss rate in cellular networks. Further, due to the relatively high bit error rate over the wireless channel, link layer retransmission is typically performed in cellular networks, this also requires large buffers in the routers or base stations to store the unacknowledged packets. Finally, some carriers configure middleboxes for the purpose of deep packet inspection. A recent finding [15] pointed out that these middleboxes also buffer a large amount of out-of-order packets.

Although providing large buffers seems to be a viable solution at Layer 2, it has an undesirable interaction with TCP congestion control at Layer 4. TCP relies mostly on packet loss to detect network congestion. Although other variants exist such as delay-based congestion control, most of the widely deployed TCP implementations (e.g., Newreno [4], BIC [16], CUBIC [7]) still use loss-based congestion control [17]. Excessive buffers in cellular networks prevent packet losses from occurring even if TCP's sending rate far exceeds the bottleneck link capacity. This "hides" the network congestion from the TCP sender and causes its congestion control algorithm to malfunction. It eventually results in a number of TCP performance degradations which we detail in Section 3.

2.2 TCP Receive Window Adjustment (Auto-tuning)

As we know, in TCP flow control, the receive window was originally designed to prevent a fast sender from overwhelming a slow receiver with limited buffer space. It reflects the available buffer size on the receiver side so that the sender will not send more packets than the receiver can accommodate. The combination of this flow control and TCP congestion control ensures that neither the receiver nor any intermediate router along the path will be overloaded.

With the advancement in storage technology, memories are becoming increasingly cheaper. Currently, it is not uncommon to find a computer equipped with several gigabytes of memory and even smart phones are now equipped with 1GB of RAM (e.g. Motorola Droid Razr, Samsung Galaxy S2). Hence, buffer space on the receiver side is hardly the bottleneck in the current Internet. To improve TCP throughput, a receive buffer auto-tuning technique called Dynamic Right-Sizing (DRS [3]) was proposed. In DRS, instead of determining the receive window based by the available buffer size, the receive buffer size is dynamically adjusted in order to suit the connection's demand. Specifically, in each RTT, the receiver estimates the sender's congestion window and then advertises a receive window which is *twice the size* of the estimated congestion window. The fundamental goal of DRS is to allocate enough buffer (as long as we can afford it) so that the throughput of the TCP connection is never limited by the receive window size but only constrained by network congestion. Meanwhile, DRS tries to avoid allocating more buffers than necessary.

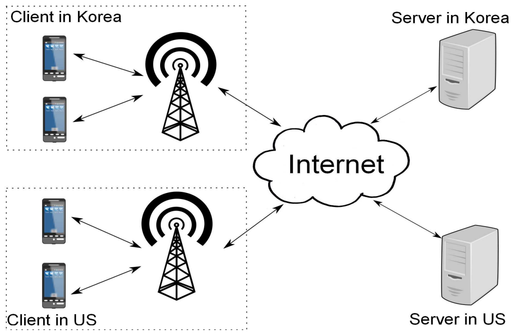
Linux adopted a receive buffer auto-tuning scheme similar to DRS since kernel 2.4.27. Since Android is based on Linux, it inherits the same receive window adjustment algorithm. Other major operating systems also implemented customized TCP buffer auto-tuning (Windows since Vista, Mac OS X since 10.5, FreeBSD since 7.0). This implies a significant role change for the TCP receive window. Although the functionality of flow control is still preserved, most of the time the receive window as well as the receive buffer size is undergoing dynamic adjustments. However, this dynamic adjustment is unidirectional: DRS increases the receive window size only when it might potentially limit the congestion window growth but never decreases it. In this paper, we modify the receive window adjustment algorithm to be bidirectional in order to improve TCP performance in bufferbloated cellular networks. Details of our algorithm can be found in Section 4.

3. OBSERVATIONS

In this section, we report several interesting findings in our extensive real-world measurements of TCP over various cellular networks. We then diagnose the their root causes and point out their potential impacts on user experience.

3.1 Measurement Setup

Figure 2 gives an overview of our measurement setup. We have both clients and servers in U.S. and Korea so that we can evaluate different scenarios where clients download files from nearby servers or remote servers over various cellular networks operated by different carriers. Our servers run Ubuntu 10.04 (with 2.6.35.13 kernel) and use its default TCP congestion control algorithm CUBIC unless otherwise noted. The signal strength during our tests is between



(a) Experiment Architecture

(b) Experiment Phones

Figure 2: Our test environment

-75dBm and -87dBm (typically considered as good signal condition in daily life). We developed simple applications on the client side to download files from servers with different traffic patterns. Different phone models are used for different carriers. Refer to Table 1 for details.

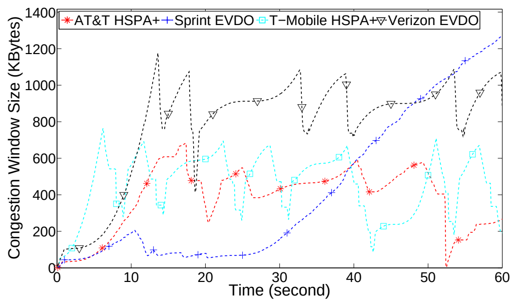
3.2 TCP Anomaly in Cellular Networks

In the past few decades, TCP has been extensively studied in traditional networks, especially wired networks. With the exponential growth of hand-held devices like smart phones and tablet computers, TCP performance in cellular networks is becoming increasingly important. Unfortunately, TCP behavior of smart mobile devices over cellular networks lacks deep investigation despite a number of measurement studies [2, 9, 11, 12]. During our extensive real-world measurements over various cellular networks, we found that the current TCP implementation exhibits abnormal behavior in bufferbloated cellular networks leading to a number of performance issues including long delays and sub-optimal throughput.

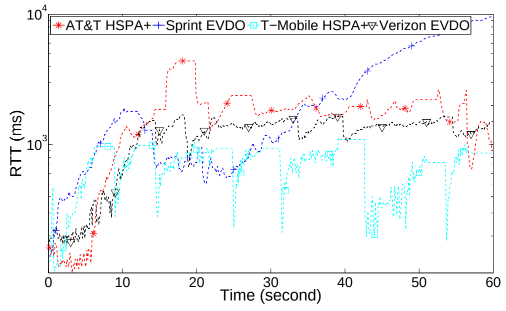
3.2.1 *A Reality Check of Bufferbloat in Cellular Networks*

The potential problem of over-buffering in cellular networks was first pointed out by Ludwig et al. [13] as early as 1999 when researchers were focusing on GPRS networks. However, over-buffering still prevails in cellular networks today. To estimate the buffer space in current cellular networks, we set up the following test: we used a laptop running

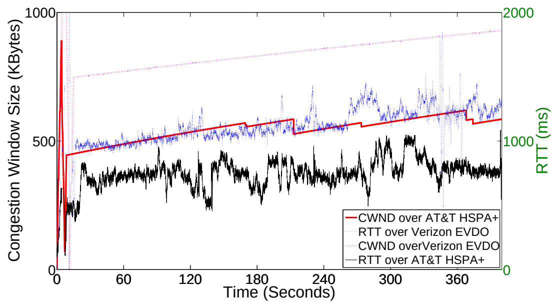
Ubuntu 10.04 to download a large file from our U.S. server over the 3G networks of the four major U.S. carriers. By default, Ubuntu sets both maximum receive buffer size and maximum send buffer size greater than 3MB so that the flow will not be limited by the buffer size and server applies TCP CUBIC. Figure 3 depicts the measurement results. As we can see from Figure 3(a), $cwnd$ probes to more than 600KB across all cellular networks, which is far beyond the BDP of the underlying network. For instance, the peak downlink rate for EVDO is 3.1Mbps. If the RTT of the underlying path is 150ms (thats the minimum RTT we observed in Figure 3(b)), the actual BDP is only around 58KB. With so much overshooting, it is not surprising that high end-to-end latency (up to 10 seconds) is observed.



(a) Congestion Window Size



(b) Round Trip Time



(c) Bufferbloat with TCP NewReno

Figure 3: Bufferbloat prevails across the 3G networks of all four major U.S. carriers, resulting in extremely long RTT.

To further confirm that this anomaly is not unique to TCP CUBIC, we vary the TCP congestion control algorithm from TCP CUBIC to TCP NewReno on the server side and repeat the test in AT&T and Verizon's 3G networks. As shown in the Figure 3(c), the same problem is also observed for TCP NewReno. At the initial phase, $cwnd$ rapidly increases by applying slow start because the slow start threshold (ssthresh) is set as a large value in Linux implementation. But later on, we observe that $cwnd$ keeps increasing during the congestion avoidance phase without any packet loss event for a long period of time.

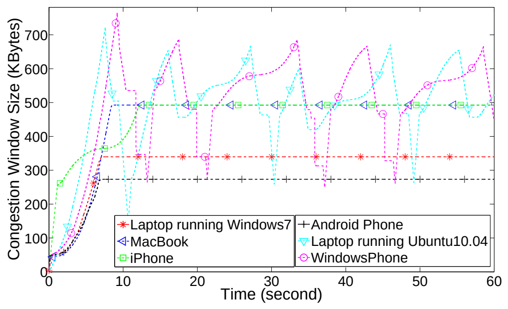


Figure 4: TCP behavior over AT&T HSPA+ network: (a) two patterns, namely "flat TCP" and "fat TCP" are observed depending on the client platform. (b) The problem is not unique to TCP CUBIC.

As discussed in Section 2.1, the large buffers are introduced for reasons. Simply removing them is not a viable option. Adding AQM would definitely help, but is expensive to deploy. Hence, we turn our attention to end-to-end solutions that are easier to deploy.

3.2.2 *TCP Behavior in Cellular Networks*

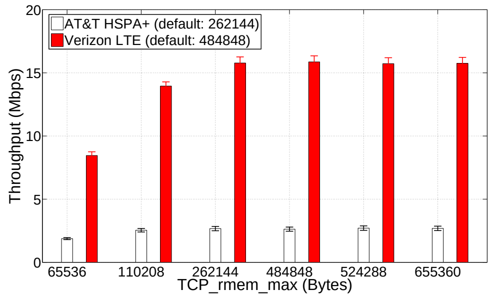
Figure 4 depicts the evolution of TCP congestion window when the clients of various platforms download a large file from a server over AT&T HSPA+ network. In the test, the platforms applied consist of Android phones, iPhone, Windows Phone 7, laptop running Ubuntu 10.04, Macbook and laptop running Windows 7. To our surprise, two types of $cwnd$ patterns are observed: "flat TCP" and "fat TCP". Flat TCP, such as observed in Android phones, is the phenomenon where the TCP congestion window grows to a static value and stays there until the session ends. On the other hand, fat TCP such as observed in Windows Phone and Linux laptop is the phenomenon that packet loss events do not occur until the congestion window grows to a significantly large value far beyond the BDP. Fat TCP can easily be explained by the bufferbloat in cellular networks. But the abnormal flat TCP behavior caught our attention and revealed an untold story of TCP over cellular networks which we detail in the next subsection.

3.2.3 *Understanding the Anomaly*

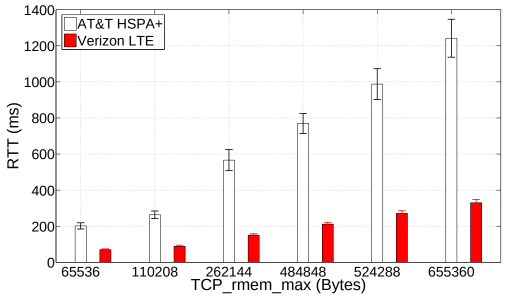
How could the TCP congestion window stay at a static

	Samsung Galaxy S2 (AT&T)	HTC EVO Shift (Sprint)	Samsung Droid Charge (Verizon)	LG G2x (T-Mobile)
Wi-Fi	110208	110208	393216	393216
UMTS	110208	393216	196608	110208
EDGE	35040	393216	35040	35040
GPRS	11680	393216	11680	11680
HSPA+	262144	N/A	N/A	262144
WiMAX	N/A	524288	N/A	N/A
LTE	N/A	N/A	484848	N/A
Default	110208	110208	484848	110208

Table 1: Maximum TCP receive buffer size (*tcp_rmem_max*) in bytes on different Android phones for different carriers. Note that these values may vary on customized ROMs and can be looked up by looking for "setprop net.tcp.buffersize.*" in the init.rc file of Android phones. Also note that different values are set for different carriers even if the network types are the same. We guess that these values are experimentally determined based on each carrier's network conditions and configurations.



(a) Throughput Performance



(b) RTT Performance

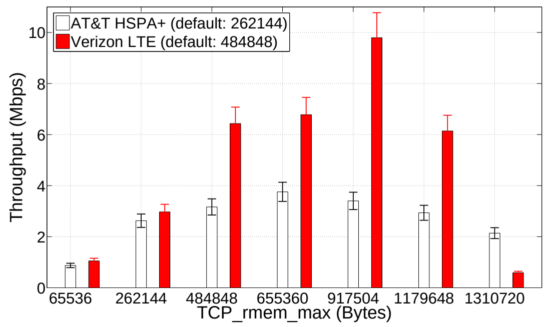
Figure 5: Throughput and RTT performance measured for a whole day when downloading from a nearby server over LTE and HSPA+ networks with various *tcp_rmem_max* values. For this test environment, 110208 may work better than the default 262144 in AT&T HSPA+ network. Similarly, 262144 may work better than the default 484848 in Verizon LTE network. However, the optimal value depends on the environment and is hard to set statically in advance.

value? The static $cwnd$ first indicates that no packet loss is observed by the TCP sender (otherwise the congestion window should have decreased multiplicatively at any loss event). This is due to the large buffers in cellular networks and its link layer retransmission mechanism as discussed earlier. Measurement results from [9] also confirm that cellular networks typically experience close-to-zero packet loss rate.

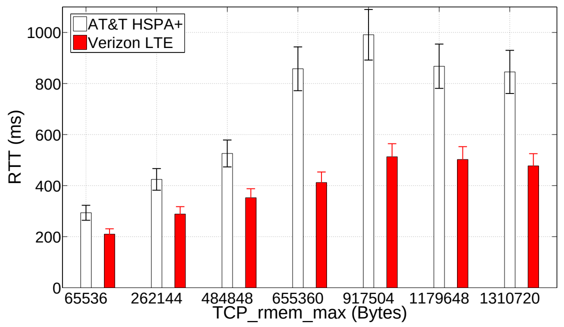
If packet losses are perfectly concealed, the congestion window may not drop but it will persistently grow as fat TCP does. However, it unexpectedly stops at a certain value and this static value is different for each cellular network or client platform. Our deep inspection into the TCP implementation in Android phones (since it is open-source) reveals that the value is determined by a parameter called *tcp_rmem_max* that specifies the maximum receive window advertised by an Android phone. This gives the answer to flat TCP behavior: the receive window advertised by the receiver crops the congestion windows in the sender. By inspecting various Android phone models, we found that *tcp_rmem_max* has diverse values for different types of networks as shown in Table 1. Generally speaking, larger values are assigned to faster communication standards (e.g., LTE).

To understand the impact of *tcp_rmem_max*, we compared the TCP performance under various *tcp_rmem_max* settings for Verizon's LTE and AT&T's HSPA+ networks in Figure 5. Obviously, a larger *tcp_rmem_max* value allows the congestion window of the TCP sender to grow to a larger size and hence leads to higher throughput. But this throughput improvement will flatten out once the link capacity is saturated. Further increase of *tcp_rmem_max* brings nothing but longer queuing delay. For instance, when downloading from a nearby server, the end-to-end latency is relatively small and hence the BDP is small. The default values for both LTE and HSPA+ are large enough to achieve full bandwidth utilization as shown in Figure 5(a) but trigger excessive packets in network and thus result in unnecessarily large RTT as shown in Figure 5(b). This demonstrates the fundamental limitations of the static parameter setting: it mandates one specific trade-off point in the system which may be sub-optimal for other applications. Two realistic scenarios are discussed in the next section.

In summary, both flat TCP and fat TCP have performance issues. On one hand, in small BDP networks, fat/flat TCP will bring unnecessarily long end-to-end latency due to excessive queuing. On the other hand, flat TCP will suffer from significant throughput degradation in large BDP links.



(a) Throughput Performance



(b) RTT Performance

Figure 7: Throughput and RTT performance measured for a whole day when downloading from a remote server in Korea via LTE and HSPA+ networks with various *tcp_rmem_max* values. The static setting results in sub-optimal throughput performance since it fails to probe maximum available bandwidth of the long fat pipe. 655360 for AT&T and 917504 for Verizon provided much higher throughput than their default values.

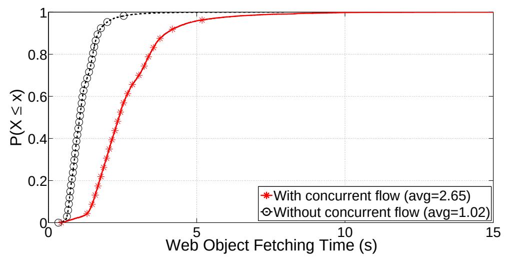


Figure 6: Web object fetching performance with and without a concurrent long-term flow. The time taken to fetch the same web objects becomes 2.6 times longer if a file download coexists. If TCP maintains a smaller queue in the network, the drastic performance degradation can be mitigated.

3.3 Impact on User Experience

Web Browsing with Concurrent Flows: Top lines of smart phones scheduled to be launched during 2012 are typically equipped with a quad core CPU of about 1.5GHz per core, more than 1GB RAM, and a high resolution screen (e.g., 1280×720 pixels). Due to their significantly improved capability, the phones are expected to perform multi-tasking more often. For instances, people will enjoy web browsing or online gaming while downloading files such as books, musics, movies or applications from on-line markets in the background. In such cases, we found that the current TCP implementation incurs long delays for the interactive flow (Web browsing or online gaming) since the buffer is filled with packets belonging to the concurrent flows.

Figure 6 shows that the Web object fetching time is severely degraded when a concurrent long-term flow is under way. Since Web objects are typically small (for instance, we use 8KB, 16KB, 32KB and 64KB in this test), their fetching time mainly depends on RTT rather than throughput. When a concurrent long-term flow causes long queues to be built up at the base station, the Web objects will be severely delayed. As the figure shows, average Web object fetching time is 2.6 times longer with a concurrent background download[1].

Throughput from Servers with Long Latency: The sites that smart phone users visit are diverse. Some contents are well maintained and CDNs (content delivery networks) are assisting them to get "closer" to their customers via replication. In such cases, the throughput performance can be well-supported by the static setting of *tcp_rmem_max*. However, there are still many websites or files showing long latencies due to their remote locations, such as the most common usage of smart phones: web browsing, market application download and streaming. In such cases, the static setting of *tcp_rmem_max* (which is tuned for moderate latency case) fails to provide the maximum possible throughput since it cannot fill the long fat pipe. Figure 7 shows that when downloading contents from a server abroad, the client suffers from sub-optimal throughput performance under the default setting. A larger *tcp_rmem_max* can achieve higher throughput, but if it is too large, packet loss will eventually occur in which case throughput degradation will also occur.

4. DYNAMIC RECEIVE WINDOW ADJUSTMENT (DRWA)

4.1 Candidate Solutions and Our Approach

In order to address TCP's problem in buffer-bloated cellular networks, there are a few possible solutions. One obvious solution is to reduce the buffer size in cellular networks so that TCP can function the same way as it does in wired networks. However, as explained earlier these extra buffers are essential to ensure the performance of cellular links under dynamic conditions and cannot be easily removed. Further, the modification of TCP protocols could be simpler than

[1]Note that the fetching times of multiple small objects in parallel do not affect each other since the bandwidth is not saturated.

Algorithm 1 DRWA

```
Initialization:
$RTT_{min} \leftarrow \infty$;
$cwnd_{est} \leftarrow data\_rcvd$ in the first $RTT_{est}$;
$rwnd \leftarrow 0$;

RTT and Minimum RTT Estimation:
$RTT_{est} \leftarrow$ the time between when a byte is first acknowledged and the receipt of data that is at least one window beyond the sequence number that was acknowledged;

if TCP timestamp option is available then
    $RTT_{est} \leftarrow$ averaging the RTT samples obtained from the timestamps within the last RTT;
end if

if $RTT_{est} < RTT_{min}$ then
    $RTT_{min} \leftarrow RTT_{est}$;
end if

DRWA:
if data is copied to user space then
    if $elapsed\_time < RTT_{est}$ then
        return;
    end if

    $cwnd_{est} \leftarrow \alpha * cwnd_{est} + (1 - \alpha) * data\_rcvd$;
    $rwnd \leftarrow \lambda * \frac{RTT_{min}}{RTT_{est}} * cwnd_{est}$;
    Advertise $rwnd$ as the receive window size;
end if
```

modification of network infrastructure.

An alternative to this solution is to employ certain AQM schemes like RED [5] or REM [1]. By randomly dropping or marking certain packets before the buffer is full, we can notify TCP sender in advance and avoid the excessively long delay. However, despite being studied extensively in the literature, few AQM schemes are actually deployed over the Internet due to the complexity of their parameter tuning, the extra packet losses introduced by them and the limited performance gains provided by them.

Another possible solution to this problem is the modification of the TCP congestion control algorithm at the sender side. As shown in Figure 8, delay-based TCP congestion control algorithm (e.g., TCP Vegas) perform normally while all of the loss-based TCP congestion control algorithms (e.g., CUBIC, BIC [16], NewReno) face the flat TCP problem. Since delay-based congestion control backs off when RTT starts to increase rather than waiting until packet loss happens, they may serve the over-buffered cellular networks better than loss-based congestion control. However, as Figure 9 shows, although delay-based TCP congestion control decreases RTT to a great extent, they suffer from throughput degradation. This agrees with the observation over a cellular

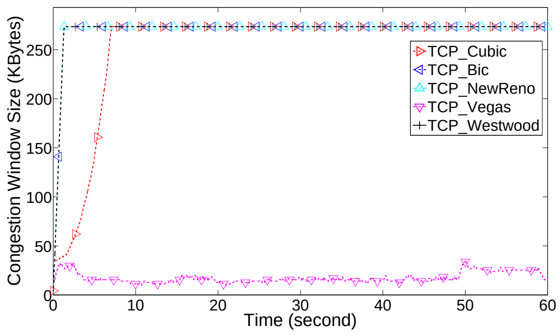


Figure 8: Delay-based TCP variants are free from bufferbloat problem.

network in [12]. Further, adopting delay-based congestion control requires modifications on the sender side (typically large-scale servers) which may incur considerable deployment cost and affect both wired and wireless users.

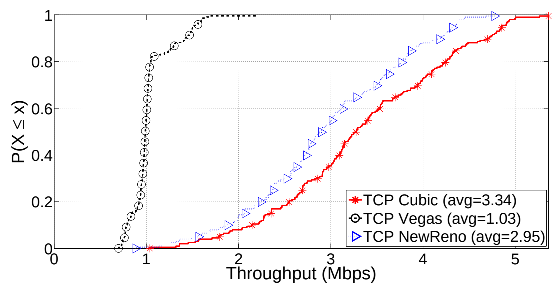
In light of the problems with the above-mentioned solutions, we suggest to handle the problem on the receiver side by changing the static setting of *tcp_rmem_max*. That is because receiver (mobile device) side modification has minimum deployment cost. Vendors may simply issue an OTA update to the protocol stack of the mobile devices so that they can enjoy a better TCP performance without affecting other wired users. It is a light-weight, effective and immediately deployable solution to the problem. Therefore, we propose a dynamic receive window adjustment (DRWA) algorithm to ensure full utilization of the available bandwidth while maintaining a small RTT.

4.2 Algorithm of DRWA

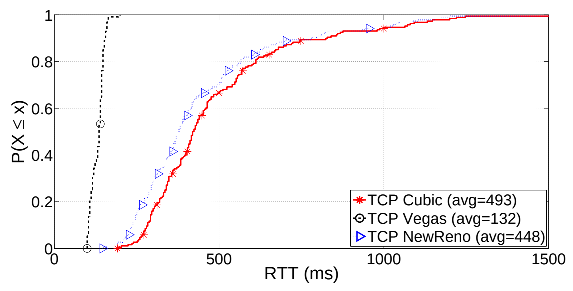
The aim of DRWA is to adaptively set the receive window to a proper size in different environment. Sometimes, it should be larger than the current static limit to achieve more throughput and other times it should become smaller than the current value to avoid unnecessary queues in the link. The challenges in this work consist of three parts. First, DRWA should remove the static setting of a relatively small maximum receive buffer size. Second, DRWA should bear the capability to estimate the proper pipe size of a link via a new window adjustment algorithm. Finally, DRWA should be compatible with the current TCP protocol and easy to deploy.

DRWA is built on top of DRS. Instead of an unidirectional adjustment where the advertised window is non-decreasing, we need a bidirectional adjustment algorithm to rein TCP in the buffer-bloated cellular networks but at the same time to ensure full utilization of the link. To accomplish that, DRWA needs to keep the queue size within a proper range dynamically. Algorithm 1 gives the details.

DRWA uses the same technique as DRS to measure RTT on the receiver side if TCP timestamp option is unavailable. However if TCP timestamp option is available, DRWA uses



(a) Throughput



(b) RTT

Figure 9: Throughput and RTT performance of TCP Vegas in cellular networks: although delay-based congestion control reduces the RTT, it suffers from throughput degradation.

it to obtain a more accurate estimation of the RTT (Timestamps can provide multiple RTT samples within an RTT whereas the traditional DRS way provides only one sample per RTT). We surveyed the support for TCP timestamp option in Windows Server and Linux and found that when DRWA runs on Android phones, it could turn on timestamp regardless of whether talks to a Linux server or a Windows server. With the assistance of timestamps, DRWA is able to achieve robust RTT measurement on the receiver side and thus conquers the well-known problem of accurately measuring RTT in dynamic networks, as shown in Figure 10. In addition to RTT measurement, DRWA also records the minimum RTT ever seen in this connection and uses it later to determine the receive window size. Since the minimum RTT approximates the round-trip propagation delay between the two hosts when no queue is built up in the intermediate routers especially in the cellular base station, we use it as an indication of what the network and channel conditions are.

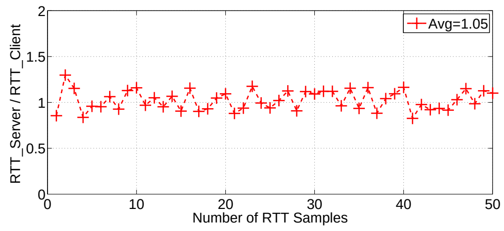


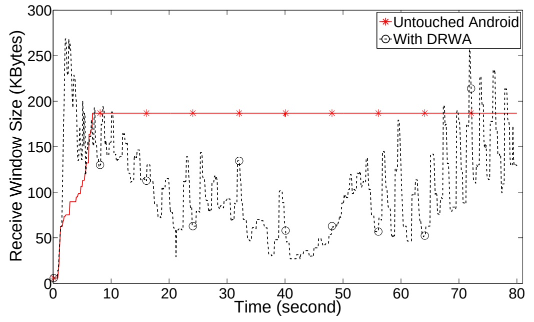
Figure 10: With timestamp option, DRWA is able to achieve robust RTT measurement on the client side. The testing was conducted over AT&T HSPA+ network by using Samsung Galaxy S2 phone. The two RTT measurements are consistent though there exists minor deviation.

After knowing the RTT, DRWA counts the amount of data received within one RTT in the same way as DRS. However, DRWA further smooths the estimated congestion window by using a moving average with a low-pass filter (α is set to 7/8 in our current implementation). This smoothed value is used to determine the receive window we advertise. In contrast to DRS who always sets $rwnd$ to $2 * cwnd_{est}$, DRWA sets it to $\lambda * \frac{RTT_{min}}{RTT_{est}} * cwnd_{est}$. When RTT_{est} is close to RTT_{min}, implying the network is not congested, $rwnd$ will increase quickly to give the sender enough space to probe the available bandwidth. As RTT_{est} increases, we gradually slow down the increment rate of $rwnd$ to stop TCP from overshooting. The operation of taking the maximum of the newly calculated $rwnd$ and the previous $rwnd$ in DRS is also removed so that DRWA makes bidirectional adjustment of the advertised window and controls the RTT_{est} to stay around $\lambda * RTT_{min}$. More detailed explanation of λ will be given in the following section.

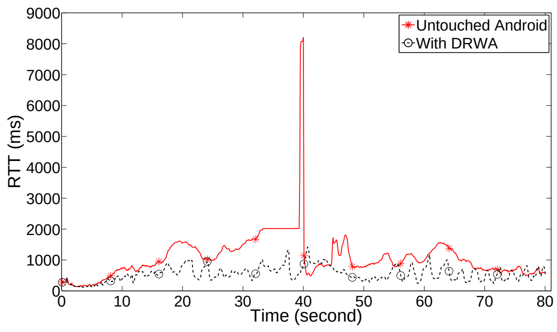
This algorithm is simple yet effective. Its ideas stem from delay-based congestion control algorithms but work better than they do for two reasons. First, since DRWA only *guides* the TCP congestion window by advertising an adaptive $rwnd$, the bandwidth probing responsibility still lies with the TCP congestion control algorithm at the sender side. Therefore, typical throughput loss seen from using delay-based TCP will not appear. Also, due to some unique characteristics of cellular networks, RTT based control can work more effectively. In wired networks, a router may handle hundreds of TCP flows at the same time and they may share the same output buffer. That makes RTT measurement noisier and delay-based congestion control less reliable. However, in cellular networks, a base station typically has separate buffer space for each user and a mobile user is unlikely to have many simultaneous TCP connections. This makes RTT measurement a more reliable signal for network congestion.

4.3 Adaptive Nature of DRWA

DRWA allows a TCP receiver to report a proper receive window size to its sender in every RTT rather than advertising a static limit. Due to its adaptive nature, DRWA is able to track the variability of channel conditions. Figure 11 shows the evolution of the receive window size and the corresponding RTT performance. During this test, we moved the Android phone from a good signal area to a weak signal area (from 0 second to 40 second) and then returned it to the good signal area (from 40 second to 80 second). As shown in Figure 11(a), the receive window size dynamically adjusted by DRWA well demonstrates the signal change in-

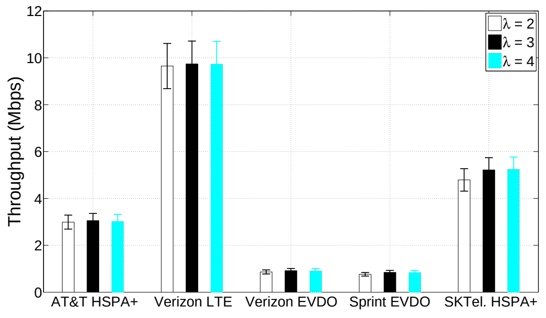


(a) Receive Window Size

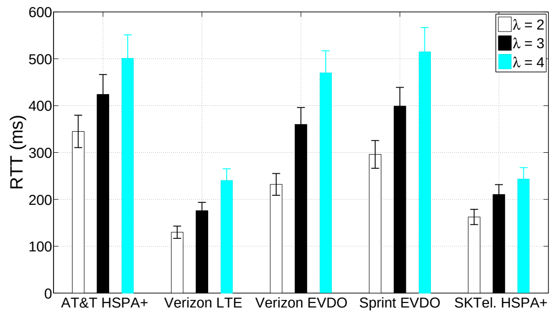


(b) RTT Performance

Figure 11: TCP behavior comparison between untouched Android phones and phones with DRWA: in this test the phones are moved from an area with good signal to an area with weak signal and then moved back again. In contrast to the static setting of the receive window in untouched Android phones, DRWA nicely tracks the variation of the channel conditions and dynamically adjusts the receive window. Due to the dynamic adjustment, DRWA is able to keep the RTT constantly low while the untouched Android phone experiences drastic increase in RTT under weak signal.



(a) Throughput Performance



(b) RTT Performance

Figure 12: Impact of λ on the throughput and RTT performance of TCP with DRWA in different cellular networks. $\lambda = 3$ gives a good balance between throughput and RTT in four major U.S carriers as well as the largest Korean carrier.

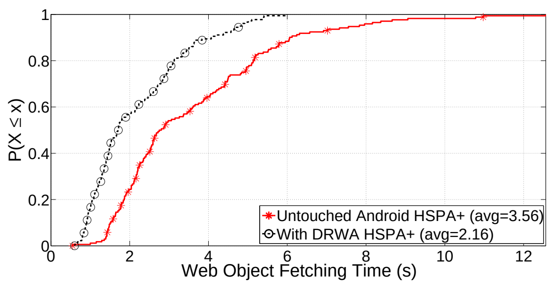
curred by the movement. This leads to a steadily low RTT while the static setting of untouched Android results in an ever increasing RTT as the signal strength decreases and the RTT blows up in the area of the weakest signal strength.

4.4 Impact of λ on TCP Performance

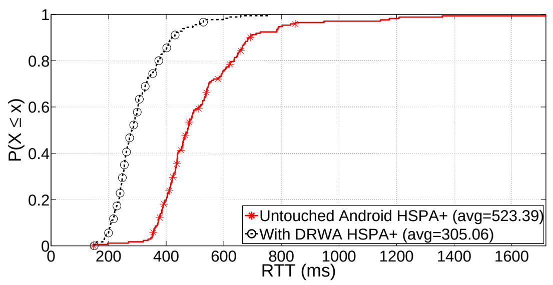
λ is a key parameter in DRWA. It tunes the operation region of the algorithm and reflects the trade-off between throughput and delay. Note that when RTT_{est}/RTT_{min} equals to λ, the advertised receive window will be equal to its previous value, leading to a steady state. Therefore, λ reflects the target RTT of DRWA. If we set λ to 1, that means we want RTT to be around RTT_{min} so that almost no queue is built up. This ideal case only guarantees high throughput if 1) the traffic has constant bit rate, 2) the available bandwidth is also constant and 3) the constant bit rate equals to the constant bandwidth. In practice, Internet traffic is bursty and the channel condition varies over time. Both necessitate the existence of some buffers to absorb the temporarily excessive traffic and drain the queue later on when the load becomes lighter or the channel condition becomes better. Otherwise, we cannot fully utilize the link. λ determines how aggressive we want to be in keeping the link busy and how much delay penalty we can tolerate. The larger λ is, the more aggressive the algorithm is. It will guarantee the throughput of TCP to be saturated all of the time but at the same time introduce extra delays. Figure 12 gives the comparison of performance among different values of λ. This test combines multiple scenarios ranging from local to remote access, good to weak signal. Each has been repeated 400 times over the span of 24 hours in order to find the optimal parameter setting. In our current implementation, we set λ to 3 which works very well for most cellular networks. However, a better approach may exist and may make this parameter adaptive. We leave this as our future work.

4.5 Improvement in User Experience

Section 3.3 lists two scenarios where existing TCP implementation may have a negative impact on user experience. In this section, we demonstrate that, by applying DRWA, we can drastically improve user experience in such scenar-



(a) Web page fetching time



(b) RTT experienced by the Web browsing TCP flow

Figure 13: Web browsing with a concurrent long-term flow over AT&T HSPA+: DRWA reduces the RTT experienced by the TCP flows and hence improves Web browsing performance.

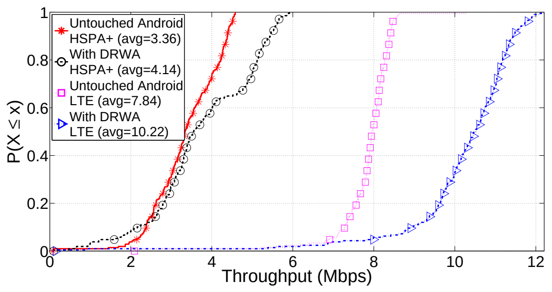


Figure 14: Throughput improvement brought by DRWA when clients in U.S. download from a server in Korea: Each test lasts for 24 hours. The improvement ratios are 23% in AT&T HSPA+ network and 30% in Verizon LTE network.

ios. More comprehensive experiment results are provided in Section 5.

Figure 13 shows Web object fetching performance with a concurrent long-term flow. Since DRWA reduces the length of the queue built up in the cellular networks, it brings on average 42% reduction in the RTT experienced by all the TCP flows coming down to the receiver. This translates into 39% speed-up in Web object fetching since the download completion time of (typically small) Web pages and the embedded objects (e.g., images, flash clips) are mainly determined by the RTT.

Figure 14 shows the scenario where a mobile user in the U.S. downloads from a remote server in Korea. Since the RTT is very long in this scenario, the BDP of the underlying network is fairly large. The static setting of *tcp_rmem_max* is too small to fill the long, fat pipe and results in throughput degradation. With DRWA, we are able to fully utilize the available bandwidth and achieve 23-30% improvement in throughput.

5. MORE EXPERIMENTS

We implemented DRWA in Android phones by patching their kernels. It turned out to be fairly simple to implement DRWA in the Linux/Android kernel. It only takes around 100 lines of code. We downloaded the original kernel source codes of different Android models from their manufacturers' website, patched the kernels with DRWA and recompiled them. Finally, the phones were flashed with our customized kernel images. We provided a procfs entry for users to easily turn on or off DRWA. We did head-to-head comparisons between untouched Android phones (without DRWA) and Android phones with DRWA. The test environment is illustrated by Figure 2.

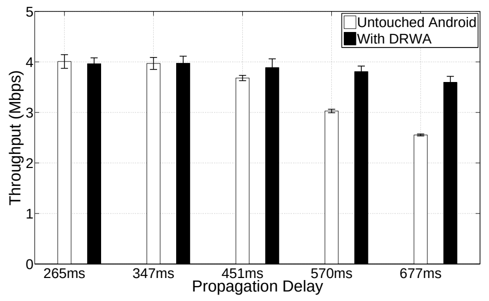
5.1 Throughput Improvement

Figure 15 shows the throughput improvement of Android phones with DRWA over untouched Android phones. The test involved file downloading (file size is 100MB) via different cellular networks operated by various carriers. For each network we ran the test for 24 hours. During the test, we applied *netem*, the built-in network emulator in Linux [8] on the server side to emulate the scenarios of different propagation delay. From the figure, we see that Android phones with DRWA significantly improve the throughput in all cellular networks as the propagation delay increases. The scenario over the Sprint EVDO network with the propagation delay of 754 ms shows the largest improvement (as high as 51%). In LTE networks, the phones with DRWA show throughput improvement up to 39% under the latency of 219 ms.

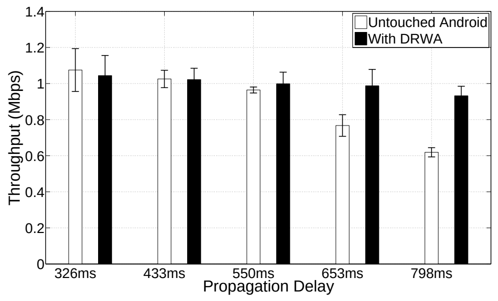
The reason behind the improvement is obvious. When the latency increases, the static values fail to saturate the pipe, resulting in throughput degradation. In contrast, networks with small latencies do not show such degradation. According to our experiences, RTTs between 400 ms and 700 ms are easily observable in cellular networks, especially when using services from foreign servers. In the LTE networks, TCP throughput is even more sensitive to *tcp_rmem_max* setting. The BDP can be dramatically increased by a slight RTT increase. Therefore, the static configuration easily becomes far from optimal. However, DRWA is able to keep pace with the increasing BDP without any problem.

5.2 End-to-End Latency Reduction

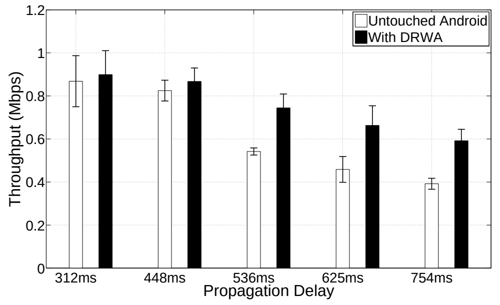
In networks with small BDP, the static *tcp_rmem_max* set-



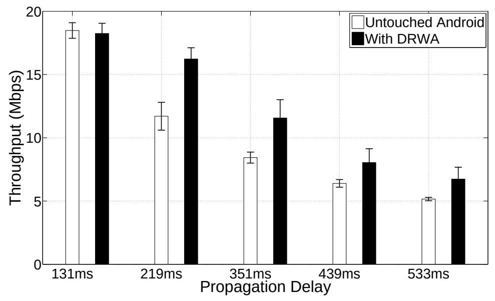
(a) Improvement Ratio in AT&T HSPA+: -1%, 0.1%, 6%, 26% and 41%



(b) Improvement Ratio in Verizon EVDO: -3%, -0.3%, 4%, 29% and 51%



(c) Improvement Ratio in Sprint EVDO: 4%, 5%, 37%, 45% and 51%



(d) Improvement Ratio in Verizon LTE: -1%, 39%, 37%, 26% and 31%

Figure 15: Throughput improvement provided by DRWA for various cellular networks under different network latencies: we see significant throughput improvement when the end-to-end latency is long.

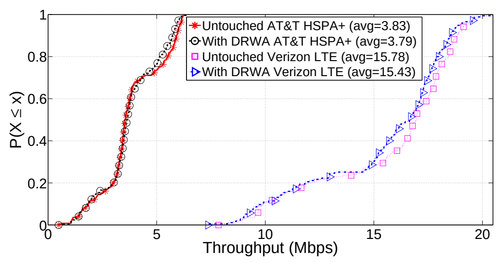
ting is sufficient to fully utilize the bandwidth of the network. However, it has a side effect of long RTT. In such networks, the static receive window reported by current implementations misleads a TCP sender to put excessive packets in the network, resulting in unnecessarily long RTT. However, DRWA manages the RTT to be λ times of the RTT_{min}, which is substantially smaller than that of current implementations in networks with small BDP. Figure 16 shows the improvement in RTT brought by DRWA while throughput is preserved.

In Figure 16, we downloaded a file from a nearby server installed at a university campus in U.S. to Android phones in U.S. to explore the throughput and end-to-end delay performance in small BDP networks. We measured the performance for a whole day per each carrier and compared the performance between Android phones with and without DRWA. During the entire day run, each round of file downloading took three minutes, resulting in over 400 runs within a day. From Figure 16, we can verify that remarkable reduction of RTT up to 49% is achieved while the throughput is guaranteed in a similar level (4% difference at maximum).

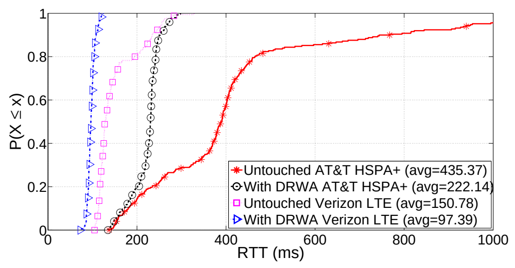
Another important observation from the experiments is that the current implementation with a static receive window experiences much larger RTT variation than DRWA. As Figures 16(a) and 16(c) show, the RTT values of untouched Android phones are distributed over a much wider range than that of phones with DRWA. The reason is clear because DRWA intentionally enforces the RTT to remain around the target value of $\lambda * RTT_{min}$. This property of DRWA will potentially benefit jitter sensitive applications such as live video communications and voice chats.

6. CONCLUSION

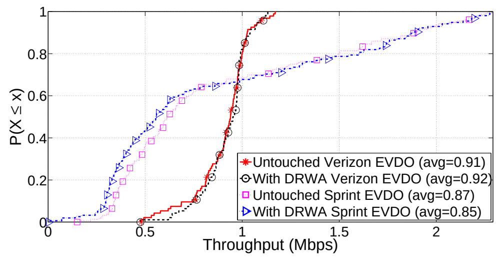
In this paper, we thoroughly investigated TCP's behavior and performance over cellular networks. We reveal that the excessive buffers available in existing cellular networks void the loss-based congestion control algorithms and the naive solution adopted of setting a static *tcp_rmem_max* is sub-optimal. Built on top of our observations, a dynamic receive window adjustment algorithm is proposed. This solution requires modifications only on the receiver side and is backward-compatible as well as incrementally deployable. We ran extensive experiments over various cellular networks to evaluate the performance of our proposal and compare it with the current implementation. Experiment results show that our scheme makes RTT 24 $\sim$ 49% lower than the current implementation of TCP while throughput is guaranteed to be the same in general cases or up to 51% higher in a high speed network with long latency. The bufferbloat problem is becoming more and more prevalent in the Internet. It is not specific to cellular networks although it might be the most prominent in this environment. A more fundamental solution to this problem may be needed. Our work provides a good starting point and is an immediately deployable solu-



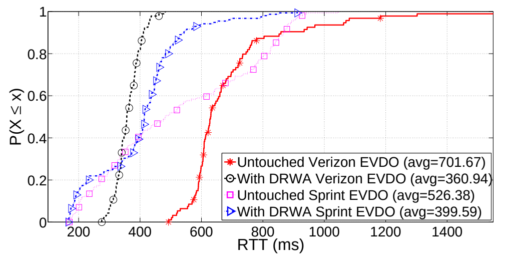
(a) Throughput: Verizon LTE and AT&T HSPA+



(b) RTT: Verizon LTE and AT&T HSPA+



(c) Throughput: Verizon EVDO and Sprint EVDO



(d) RTT: Verizon EVDO and Sprint EVDO

Figure 16: RTT improvement in networks with small BDP: DRWA provides huge RTT reduction without throughput loss across different cellular networks. The RTT reduction ratios are 49%, 35%, 49% and 24% for AT&T HSPA+, Verizon LTE, Verizon EVDO and Sprint EVDO networks respectively.

tion for smart phone users.

7. REFERENCES

[1] S. Athuraliya, S. Low, V. Li, and Q. Yin. REM: Active Queue Management. *IEEE Network*, 15:48–53, May 2001.

[2] M. C. Chan and R. Ramjee. TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation. In *Proceedings of ACM MobiCom*, 2002.

[3] W.-c. Feng, M. Fisk, M. K. Gardner, and E. Weigle. Dynamic Right-Sizing: An Automated, Lightweight, and Scalable Technique for Enhancing Grid Performance. In *Proceedings of the 7th IFIP/IEEE International Workshop on Protocols for High Speed Networks (PIHSN)*, pages 69–83, 2002.

[4] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 2582, April 1999.

[5] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, August 1993.

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

[7] S. Ha, I. Rhee, and L. Xu. CUBIC: a New TCP-friendly High-speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42:64–74, July 2008.

[8] S. Hemminger. Netem - emulating real networks in the lab. In *Proceedings of the Linux Conference*, 2005.

[9] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing Application Performance Differences on Smartphones. In *Proceedings of ACM MobiSys*, 2010.

[10] K.-c. Lan and J. Heidemann. A Measurement Study of Correlations of Internet Flow Characteristics. *Computer Networks*, 50:46–62, January 2006.

[11] Y. Lee. Measured TCP Performance in CDMA 1x EV-DO Networks. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, 2006.

[12] 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 ACM MobiCom*, pages 211–222, 2008.

[13] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph. Multi-layer tracing of tcp over a reliable wireless link. In *SIGMETRICS*, pages 144–154, New York, NY, USA, 1999. ACM.

[14] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger. TCP Revisited: a Fresh Look at TCP in the Wild. In *Proceedings of the 9th ACM SIGCOMM IMC*, pages 76–89, 2009.

[15] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *Proceedings of the ACM SIGCOMM*, 2011.

[16] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-distance Networks. In *Proceedings of IEEE INFOCOM*, 2004.

[17] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu. TCP Congestion Avoidance Algorithm Identification. In *Proceedings of ICDCS*, 2011.