Understanding Bufferbloat in Cellular Networks

Haiqing Jiang, Zeyu Liu, Yaogong Wang, Kyunghan Lee and Injong Rhee
Dept. of Computer Science, North Carolina State University
Raleigh, NC, U.S.A
hjiang5, zliu8, ywang15, klee8, rhee@ncsu.edu

ABSTRACT

Bufferbloat is a prevalent problem in the Internet where excessive buffers incur long latency, substantial jitter and sub-optimal throughput. This work provides the first elaborative understanding of bufferbloat in cellular networks. We carry out extensive measurements in the 3G/4G networks of the four major U.S. carriers to gauge the impact of bufferbloat in the field. Due to the bufferbloat problem, several pitfalls of current TCP protocols have been proposed in this paper. We also discover a trick employed by smart phone vendors to mitigate the issue and point out the limitations of such ad-hoc solutions. Our measurement study is coupled with theoretical analysis using queuing models. Finally, we comprehensively discuss candidate solutions to this problem and argue for a TCP-based end-to-end solution.

Categories and Subject Descriptors

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

Keywords

Bufferbloat, Cellular Networks, TCP Performance

1. INTRODUCTION

Mobile data traffic from hand-held devices (e.g., smartphones, tablets) is growing at an unprecedented rate. A forecast report from CISCO [4] shows that the global mobile data traffic of 2016 will be 18 times of that of 2011 due to proliferation of smart mobile devices and data-demanding applications and services for such devices. In order to provide better user experiences to mobile data consumers, we pay our attention to the performance of cellular networks, the only ubiquitous data network as of now. Given that TCP is the dominant transport layer protocol of the current Internet, carrying around 90% of the total traffic [12], we specifically focus on the performance of TCP over cellular networks. Surprisingly, the performance of TCP over cellular networks has been under-explored compared to its importance to mobile data traffic, probably due to the closed nature of cellular networks.

CellNet'12, August 13, 2012, Helsinki, Finland.

According to our extensive measurement of TCP performance over four major cellular networks in U.S, we observe a number of performance degradation issues, including long delays and throughput degradation in certain scenarios, which have been also reported by previous measurement work on cellular networks [10,13,14]. In this paper, we pinpoint that "bufferbloat (excessive buffer spaces)" [7] is one of the most critical reasons behind the performance degradation. To support our claim, we provide an in-depth reality check of bufferbloat in four major carriers in U.S. Our results clearly shows that all of them are severely over-buffered. The large buffer in cellular networks may be the engineering choices for bursty traffic, user fairness and channel variability. However, the choice unfortunately turns out to mislead loss-based TCP congestion control algorithms to overshoot the proper amount of packets to be put to pipe. Consequently, the misbehavior of TCP results in long queuing delay. The over-buffering of packets also results in large jitter of end-to-end latency, which potentially brings in *time-out* events and eventually leads throughput degradation.

Although bufferbloat problem is prevalent in current cellular networks, the practical solutions have not been deployed. During our further reality check, we reveal an untold story about current implementation of TCP in smartphones and tablets (Android platform): to solve bufferbloat problem, smartphone vendors apply a small trick that they set the maximum TCP receive buffer size to be a relatively small value. Note that TCP will follow the two logics: advertise window cannot exceed maximum receive buffer size and sending window is the minimum between congestion window and advertise window. Therefore, the limitation of maximum receive buffer size will eventually prevent TCP sender from over-shooting and avoid the excessive growth of end-to-end latency (e.g., RTT). However, this simple hack brings in sub-optimal performance issues due to the highly dynamic nature of mobile data access. The reason is obviously that there is no static value which is able to perfectly capture the diversity of the link capacity in various mobile environments. Simply speaking, the static value could be too small in large BDP (Bandwidth-Delay Product) links but too large in small BDP links.

There exist many potential solutions to bufferbloat in cellular networks ranging from removing extra buffers within networks to applying Active Queue Management (AQM). We provide a comprehensive discussion about the potential solution spaces and finally propose a new direction, TCP-based end-to-end solution, which is highly practical.

Our contributions in this paper are in three-folds: First, we are first to show that bufferbloat problem is prevalent in cellular networks based on the real-world measurement in the four largest carriers in U.S. Second, we further argue that high speed TCP may aggravate the performance degradation in bufferbloated networks.

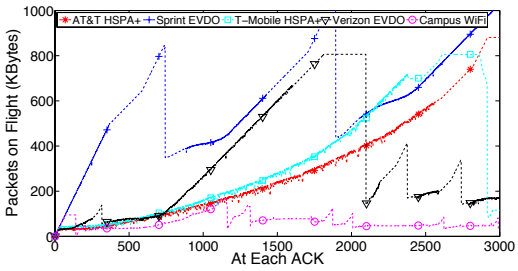


Figure 1: Surprisingly, we observed a fat pipe in all four major U.S. carriers. This observation verifies the prevalent bufferbloat problem in cellular networks.

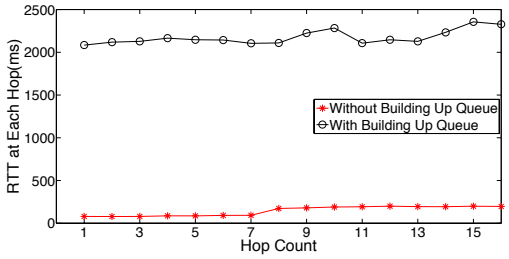


Figure 2: We verify that the bottleneck is within the cellular network (the first IP hop).

Client Location	Server Location	Network	Signal
Raleigh, US	NCSU, Princeton, KAIST	Sprint, AT&T, T-Mobile, Verizon	Good, Weak
Seoul, KR	NCSU, Princeton, KAIST	SKTelecom	Good, Weak
Chicago, US	NCSU, Princeton, KAIST	AT&T	Good, Weak

Table 1: The various scenarios where the bufferbloat problem has been verified. The network model for each carrier is: Sprint-EVDO, AT&T-HSPA+, T-Mobile-HSPA+, Verizon-EVDO/LTE and SKTelecom-HSPA+.

Finally, we present a sketch of TCP-based end-to-end solutions to address the problem in a pragmatic way (in both engineering and business perspectives).

The rest of the paper is organized as follows. Section 2 and Section 3 introduces bufferbloat problem in cellular networks. Section 4 visualizes TCP performance issues in the bufferbloated cellular networks. We examine the current implementation of TCP in smartphones in Section 5 and provide the candidate solutions in Section 6. Finally, we conclude our work in Section 7.

2. BUFFERBLOAT IN CELLULAR NETWORKS

"Bufferbloat" [7], as termed by Gettys in late 2010, is an undesirable phenomenon prevalent in the Internet where excessive buffering of packets within the network results in unnecessarily large end-to-end latency, jitter as well as throughput degradation in certain scenarios. With the exponential growth of mobile data traffic, the performance of cellular networks is of grwoing importance to mobile applications. However, to the best of our knowledge, the impact of bufferbloat to the performance of cellular networks has not been studied although it may critically affect the performance.

The potential problem of over-buffering in cellular networks was first pointed out by Ludwig et al. [15] as early as 1999 when researchers were focusing on GPRS networks. The large buffer was essential in cellular networks for the link layer in which packets experiencing channel errors are buffered. However, since the side-effects to TCP brought by over-buffering has not been well understood, the problem still prevails in today's cellular networks. To estimate the buffer space in current cellular networks, we set up the following test: we launch a bulk-data transfer between a laptop (receiver) and a server (sender) over the 3G networks of the four major U.S. carriers. The server is connected to a compus 100M ethernet. The laptop uses USB Modems to access 3G mobile data. Both the sender and the receiver runs Linux (Ubuntu 10.04) which uses CUBIC [8] as the TCP congestion control algorithm. By default, Ubuntu sets both maximum receive buffer size and maximum send buffer size to a large value (greater than 3 MB). Hence, the flow will never be limited by the buffer size of end points. We use tcpprobe to monitor the variables in kernel space. Due to the closed nature of cellular networks, we are not able to know the detailed queue size within networks. Instead, we measure the size of packets on flight on sender side to estimate the buffer space within the networks, although they are not exactly the same. Figure 1 shows our measurement results. Surprisingly, we observe exceptionally *fat pipes* in all four major U.S. cellular carriers. For instance, Sprint is able to bear more than 1000 KB of packets in flight which is clearly beyond the reasonable range of BDP in its EVDO network.

As a baseline comparison, we also repeat the experiment between a client in U.S and a remote server in Korea over our campus WiFi network. Due to the long distance of the link and the ample bandwidth of WiFi, the corresponding pipe size is expected to be large. However, according to Figure 1, even in such a scenario, the size of in-flight packets is still much smaller than the ones we observe in cellular networks.

We extend the measurement to a number of scenarios (36 cases) in the field to verify that the above observation is universal in current cellular networks. As shown in Table 1, we have clients and servers in various locations, over various cellular networks in various signal conditions. All the scenarios prove the existence of extremely "fat pipe" and Figure 1 depicts one of those representative samples.

To further confirm that the bufferbloat is within the cellular network rather than the Internet part, the location where the long queue is built up should be identified. To this purpose, we designed the following experiment: we use *traceroute* to measure the RTT for each hop along the path and compare the results with or without background download traffic. Figure 2 shows the results. If the queue is built up at hop x, the queuing delay will increase significantly at that hop when background traffic is in place. The results show that the queue is mainly in the first IP hop which definitely contains the cellular link. We believe that the *traceroute* packets are buffered on the way back due to the long queue built up by the background download traffic.

3. ANALYSIS OF BUFFERBLOAT

We develop a realistic model based on some simplifications to analyze TCP performance behavior in bufferbloated networks.

Traffic in 3G networks involves large portion of heavy traffic, like video streaming, web browsing and file transfer, etc. On the other hand, the mobile device traffic contains long period of inactive phase. In other words, in high speed 3G networks pack-

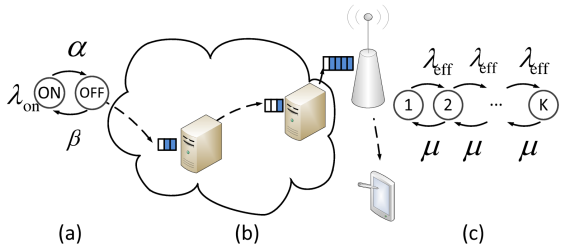


Figure 3: Modeling for end-to-end delay analysis: (a) IPP with on-off period for network source simulation, (b) a sequence of queuing system for network route simulation, (c) *M/M/1/K* finite storage system for bottleneck hop buffer simulation.

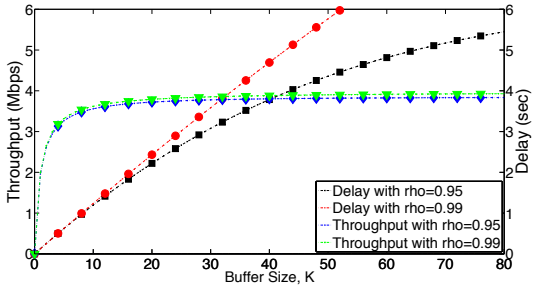


Figure 4: Matlab simulation for throughput and delay with network utilization ratio $\rho = 0.95, 0.99$ respectively.

ets usually arrive in a bursty fashion, to present this feature of the traffic, we adopted a simplied model with varied packet arrival rate. Specifically, we simulate the network source by an interrupted Poisson process with on-off periods, where the arrival rate at state *on* is λ_{on} and the transition rates between *on* and *off* state are α and β, respectively, as shown in Figure 3 (a). In this paper, in order to keep the theoretical analysis simple as well as maintaining the bursty source simulation accuracy, we adopt the equivalent representation (described in [1]) of IPP by a poisson process with effective arrival rate $\lambda_{eff} = \frac{\beta\lambda_{on}}{\alpha+\beta}$. We model each route in the network as a sequence of multiple processors with different processing abilities, as shown in Figure 3 (b).

3.1 End-to-end Delay Analysis

Note the facts that the processing ability in each of the multiple hops differ in a large extent, and in a practical situation in cellular networks, the last hop from the base station to mobile devices is usually the bottleneck of the route, in terms of congestion. Consequently, the expectation of packets RTT could be written as follows:

$$E[RTT] = 2 \times \left(\sum_{i=1}^{N-1} E[D_i] + E[D_{bottleneck}] \right). \quad (1)$$

where D_i denotes the delay in the i-th hop of the route.

As the bottleneck hop provides the smallest bandwidth, when network gets congested $E[D_{bottleneck}]$ will be the largest delay among those at all hops and therefore of most importance to investigate the end-to-end latency. In the following discussion, we analyze $E[D_{bottleneck}]$ in particular to shed a light on the latency issue in cellular networks.

Let us denote the buffer size at the bottleneck hop as K, in term of the number of packets can be cached. Note that the arrival rate of packets at the bottleneck hop is equivalent to the packets output rate at the previous hops. Using Little's Law we have:

$$E[D_{bottleneck}] = \frac{E[Q]}{\lambda_{eff}}, \quad (2)$$

where Q is the average queue size. Consider the bottleneck hop as a *M/M/1/K* finite storage system, (whose state transition diagram shown as Figure 3 (c), where λ_{eff} represents the arrival rate and μ denotes the processing rate), then the average queue size could be given as follows:

$$E[Q] = \frac{\lambda_{eff}}{\mu(1-\rho)} + (K+1)\left(1 - \frac{1}{1-\rho^{K+1}}\right), \quad (3)$$

where $\rho = \frac{\lambda_{eff}}{\mu}$ is defined as the utilization ratio of the bottleneck hop. Hence, $E[D_{bottleneck}]$ can be given by combining (2) and (3):

$$E[D_{bottleneck}] = \frac{1}{\mu(1-\rho)} + \frac{(K+1)}{\lambda_{eff}}\left(1 - \frac{1}{1-\rho^{K+1}}\right), \quad (4)$$

Note the fact that in a perfect situation to satisfy user's demand on both short end-to-end delay and high data delivery rate, the network should be designed to obtain both short RTT and full bandwidth utilization in TCP congestion control. To this end, the processor in the bottleneck hop would be nearly fully-utilized at all time, which indicates $\frac{\lambda_{eff}}{\mu}$ is nearly 1. Therefore, $E[D_{bottleneck}]$ grows faster than the buffer size, K. This behavior reveals the extremely long end-to-end delay problem that lies in the bufferbloated networks.

3.2 Throughput Analysis

Using the modeling discussed above, the throughput could be evaluated as the following

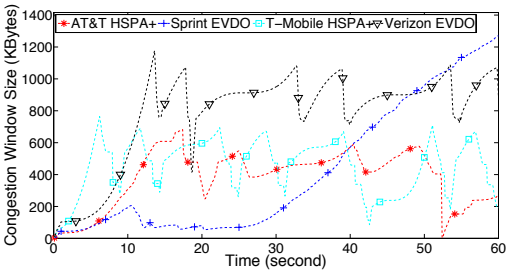
$$E[B] = \mu P_{working} = \mu + \frac{\lambda_{eff} - \mu}{1-\rho^{K+1}}. \quad (5)$$

where $P_{working}$ denotes the probability of the working period of the bottleneck hop. This formula shows that the throughput will quickly reach a limit as buffer size K increases. In other words, the large buffer size does not improve the TCP throughput performance greatly.

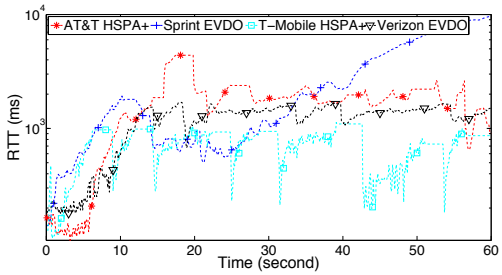
To illustratively understand Equation (4) and (5), we plot their curves on the same graph using buffer size as the input variable (as shown in Figure 4). As depicted in this plot, this simple analysis model suggests that the delay grows proportionally with the increasing of buffer size, (up to 6 seconds, with the shown parameter setting), while the throughput reaches an upper-bound (4Mbps) quickly at a small value of buffer size (10-20) and then remains almost the same. Note that this extracted model simplifies realistic conditions to a large extent, the actual number may not be perfectly inline with the practical situation, but the trend of the curves sheds a light on how the buffer size influences the performance of both delay and throughput. With this suggestion of our model, we further performed the experimental results in the following section and they validate the modeling analysis well.

4. TCP OVER BUFFERBLOATED CELLULAR NETWORKS

Bufferbloat directly affects TCP behavior and the resulting performance. Therefore, we set up the same experiment as Figure 1 in order to explore the detailed TCP behavior over bufferbloated

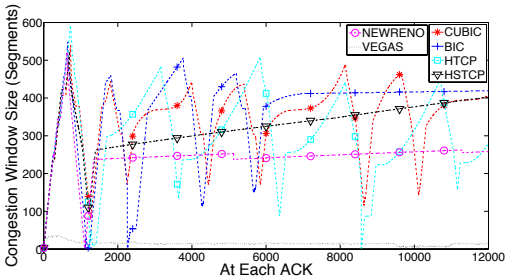


(a) Congestion Window Size

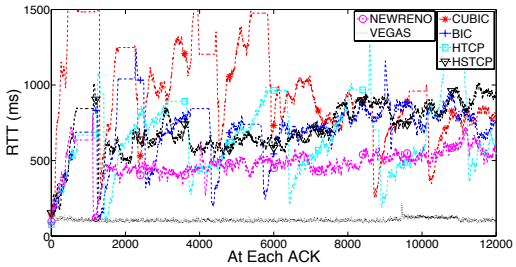


(b) Round Trip Time

Figure 5: TCP performance suffers from bufferbloat in current cellular networks, resulting in over-shooting of packets on sender side and thus extremely long end-to-end latency.



(a) Congestion Window Size



(b) Round Trip Time

Figure 6: All these loss-based high speed TCP variants suffer from bufferbloat problem. Comparing to TCP NewReno, the aggressive nature of high speed TCP variants boosts over-shooting of packets. But TCP Vegas, a delay-based TCP variant, is capable to resist bufferbloat.

cellular networks. Figure 5 shows the detailed behavior of TCP CUBIC in various cellular networks. The figure shows that TCP congestion window ($cwnd$) keeps probing even if its size is far beyond the BDP of the underlying networks. For instance, the peak downlink rate for EVDO is 3.1 Mbps and the RTT is around 150 ms (observed minimum RTT). Therefore, the BDP should be around 58 KB. However, the actual pipe size inferred by the congestion window size is much larger than that. Intuitively, the large pipe size indicates the severe over-buffering in cellular networks. With so much overshooting of TCP congestion window, the exceptionally high end-to-end latency (up to 10 seconds!) as shown in Figure 5(b) is expected. This observation exactly validates equation 4 in Section 3.

4.1 Pitfalls of High-Speed TCP Variants in Cellular Networks

According to [20], a large portion of the Web servers in the current Internet use high-speed TCP variants such as BIC [19], CUBIC [8] and CTCP [17]. In bufferbloated cellular networks, the nearly zero packet loss rate [10] combined with the aggressive nature of high speed TCP variants result in severe congestion window overshooting. To further investigate this problem, we compared the performance of several high speed TCP variants with NewReno [5] and TCP Vegas [3] over AT&T HSPA+ network. Figure 6 shows that all these loss-based high speed TCP variants overshoots more often than NewReno. In contrast, TCP Vegas is resistive to bufferbloat as it uses a delay-based congestion control algorithm. Because of the closed nature of Windows systems, we cannot explore the behavior of CTCP. But we believe that it may suffer from similar problems.

These high speed variants were originally designed for efficient probing of the available bottleneck capacity in large BDP networks. But in bufferbloated cellular networks, they only make the problem worse by constant overshooting. Therefore, the bufferbloat problem adds a new dimension in the design of a new efficient TCP, especially for cellular networks. We will discuss this comprehensively in Section 6.

5. CURRENT IMPLEMENTATION IN SMART PHONES

Figure 7 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. To our surprise, two types of $cwnd$ patterns are observed: “flat TCP" and “fat TCP". Flat TCP, such as observed in Android phone and iPhone, 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 7 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 implementation in smartphones which we detail in the next subsection.

	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
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 2: Maximum TCP receive buffer size (*tcp_rmem_max*) in bytes on different Android phones for different carriers.

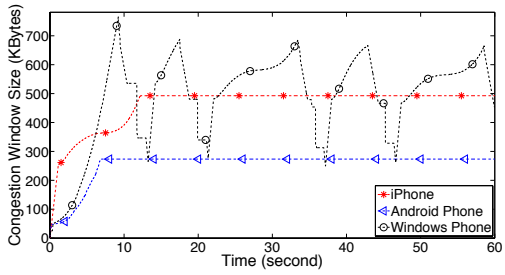


Figure 7: We observed two types of $cwnd$ patterns: "flat TCP" and "fat TCP". The abnormal flat TCP behavior will reveal an untold story of TCP implementation in smartphones.

5.1 An Untold Story

How could the TCP congestion window stay at a static 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 [10] 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 2. 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. Generally speaking, larger values are assigned to faster communication standards (e.g., LTE). Although we cannot access to the implementation details of iPhone (iOS is closed), we can infer that iPhone might have adopted a similar scheme.

Actually, this simple trick smartphone vendors apply will lead to sub-optimal performance in many scenarios. On one hand, for links with small BDP, the static value may be too high comparing to actual link bottleneck capacity. In this case, we will see unexpectedly long RTT because of excessive queuing delay although the link bandwidth can be fully utilized. On the other hand, when users encounter large BDP connections due to long propagation delays, e.g., accessing an oversea Web server, throughput degradation may be observed if the static value of tcp_rmem_max is not large enough to fill the long fat pipe.

6. CANDIDATE SOLUTIONS

To address the bufferbloat problem in cellular networks, there are a few possible solutions. In this section, we first introduce two existing solutions proposed in the literature and then propose other possible directions to solve this problem. To the best of our knowledge, we are the first to suggest TCP-based end-to-end solutions to address the bufferbloat problem. Note that these solutions are not specific to cellular networks and can be applied to general bufferbloated networks.

6.1 Existing Solutions

Reducing in-network buffers: One obvious solution is to reduce the buffer size in cellular networks so that TCP can function the same way as it does in networks without bufferbloat, e.g., wired networks. However, these buffers were introduced into cellular networks for a number of reasons. First, to absorb the bursty traffic over the variable wireless channel, the simple yet effective approach adopted by current cellular networks is to provide large buffers. Second, the link layer retransmission 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 [18] which essentially relies on a large buffer. It is not easy to remove the extra buffres in cellular networks because the operations have to be done by these commercial carriers. Therefore, we argue that the modifying network protocols to take bufferbloat into consideration is better than simply removing the buffers.

Applying AQM in routers: An alternative to is to employ certain Active Queue Management (AQM) schemes like RED [6] or REM [2]. By randomly dropping or marking certain packets before the buffer becomes full, TCP senders can be notified to avoid excessive growth of queues in buffers. 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 early drop and the limited performance gains provided by them. More recently, Nichols et al. proposed CoDel [16], a parameterless AQM that aims at handling bufferbloat. However, whether it will be adopted by carriers and network operators remains to be seen.

6.2 Our Suggested Solutions

Modifying TCP: In fact, RED is a congestion avoidance scheme implemented in routers. Inspired by that as well as in light of the problems with the above-mentioned solutions, we suggest a new direction to address bufferbloat: modification of the TCP protocol.

Compared to the other two solutions, handling bufferbloat at TCP layer has a number of advantages.

- An end-to-end solution is more feasible and light-weight compared to solutions that modify routers.
- Considering deployment issues, an end-to-end solution is easier to deploy and has lower deployment cost.
- The solution can be either server-based or client-based or both. This potential flexibility provides a larger solution space to researchers.

However, there are still some design challenges. First, although delay-based TCP algorithms (e.g., TCP Vegas) performs normally in bufferbloated cellular networks, they are proved to suffer from throughput degradation in cellular networks [14]. Therefore, a more sophisticated TCP algorithm that combines the good properties of both loss-based and delay-based congestion control is needed. Second, the new TCP protocol should be able to maintain good performance across wired networks, WiFi networks and cellular networks.

In light of the challenges with the server-based solutions, the potential receiver-based solutions should be seriously considered. That is because receiver side (mobile devices) 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. Further, since the receiver has the most knowledge of the last-hop wireless link, it could make more informed decisions than the sender. Note that a receiver-centric solution is not an ad-hoc hack to TCP. It is a systematic and in fact preferred approach to transport protocol design for mobile hosts as argued by [9]. We have proposed, implemented and tested a light-weight, receiver-based and immediately deployable solution, named "Dynamic Receive Window Adjustment (DRWA)", in Android smartphones. The extensive experiment results in the field strongly prove that DRWA our proposal may reduce the latency experienced by TCP flows by 25% $\sim$ 49% and increase TCP throughput by up to 51% in some specific scenarios. Because of the limited space of this paper, one can refer to [11] to find more details.

7. CONCLUSION

In this paper, we target a real and prevalent problem in cellular networks. We are first to conduct a reality check of bufferbloat in the four major carriers in U.S. and provide a solid modeling analysis of bufferbloat in cellular networks. Further, we point out TCP performance degradation due to bufferbloat and the pitfalls of high-speed TCP variants' design. We reveal that smartphones' vendors are applying a simple hack of limiting receiver buffer space which is sub-optimal in many scenarios. In the light of problems in bufferbloated cellular networks, we comprehensively discuss the potential solutions. Although the existing solution space is lying on IP layer, such as applying Active Queue Management, we argue that the solutions of TCP protocols can be a good new direction to target the problem by considering the practical issues. Our work involkes more research efforts on new TCP protocols design to solve bufferbloat in the current Internet, especailly cellular networks.

8. ACKNOWLEDGMENTS

This research is supported in part by Samsung Electronics, Mobile Communication Division.

9. REFERENCES

[1] M. Alwakeel. Equivalent Poisson Process for Interrupted Poisson Process with on-off Periods. *J. Sci. Med. Eng.*, 19:103–112, 2007.

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

[3] 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.

[4] CISCO. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2011 - 2016. *White Paper*, Februry 2012.

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

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

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

[8] 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.

[9] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. MobiCom '03, pages 1–15, New York, NY, USA, 2003. ACM.

[10] 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.

[11] H. Jiang and I. Rhee. Tackling bufferbloat in 3g/4g mobile networks. *NCSU Computer Science Department Technical Report*, March 2012.

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

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

[14] 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.

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

[16] K. Nichols and V. Jacobson. Controlling queue delay. *Queue*, 10(5):20:20–20:34, May 2012.

[17] K. Tan, J. Song, Q. Zhang, and M. Sridharan. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. In *Proceedings of PFLDnet*, 2006.

[18] 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.

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

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