Wireless Networks 11, 81–97, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation

MUN CHOON CHAN *
Department of Computer Science, National University of Singapore, 3 Science Drive 2, Singapore 117543

RAMACHANDRAN RAMJEE
Bell Labs, Lucent Technologies, 101 Crawfords Corner Road, Room 4G-526, Holmdel, NJ 07733, USA

Abstract. Wireless link losses result in poor TCP throughput since losses are perceived as congestion by TCP, resulting in source throttling. In order to mitigate this effect, 3G wireless link designers have augmented their system with extensive local retransmission mechanisms. In addition, in order to increase throughput, intelligent channel state based scheduling have also been introduced. While these mechanisms have reduced the impact of losses on TCP throughput and improved the channel utilization, these gains have come at the expense of increased delay and rate variability. In this paper, we comprehensively evaluate the impact of variable rate and variable delay on long-lived TCP performance. We propose a model to explain and predict TCP's throughput over a link with variable rate and/or delay. We also propose a network-based solution called *Ack Regulator that* mitigates the effect of variable rate and/or delay without significantly increasing the round trip time, while improving TCP performance by up to 100%.

Keywords: TCP, 3G wireless, link delay and rate variation, throughput model, ack regulator

1. Introduction

Third generation wide-area wireless networks are currently being deployed in the United States in the form of 3G1X technology [29] with speeds up to 144 Kbps. Data-only enhancements to 3G1X have already been standardized in the 3G1X-EVDO standard (also called High Data Rate or HDR) with speeds up to 2 Mbps [6] and is currently under going field trials. UMTS [32] is the third generation wireless technology in Europe and Asia with deployments planned this year. As these 3G networks provide pervasive Internet access, good performance of TCP over these wireless links will be critical for end user satisfaction.

While the performance of TCP has been studied extensively over wireless links [3,4,20,26], most attention has been paid to the impact of wireless channel losses on TCP. Losses are perceived as congestion by TCP, resulting in source throttling and very low net throughput.

In order to mitigate the effects of losses, 3G wireless link designers have augmented their system with extensive local retransmission mechanisms. For example, link layer retransmission protocols such as RLP and RLC are used in 3G1X [30] and UMTS [28], respectively. These mechanisms ensure in-order packet delivery and loss probability of less than 1% on the wireless link using link layer retransmission, thereby mitigating the adverse impact of loss on TCP. However, while these mechanisms mitigate losses, the use of link layer retransmission and in-order delivery result in substantial packet delay variation when there are packet losses. As we shall see in section 3, ping latencies vary between 179 ms to over 1 second in a 3G1X system.

In addition, in order to increase throughput, intelligent channel state based scheduling have also been introduced. Channel state based scheduling [7] refers to scheduling techniques which take the quality of wireless channel into account while scheduling data packets of different users at the base station. The intuition behind this approach is that since the channel quality varies asynchronously with time due to fading, it is preferable to give priority to a user with better channel quality at each scheduling epoch. While strict priority could lead to starvation of users with inferior channel quality, a scheduling algorithm such as proportional fair [6] can provide long-term fairness among different users. However, while channel-state based scheduling improves overall throughput, it also increases rate variability.

Thus, while the impact of losses on TCP throughput have been significantly reduced by local link layer mechanisms and higher raw throughput achieved by channel-state based scheduling mechanisms, these gains have come at the expense of increased delay and rate variability. These rate and delay variations cause the bandwidth-delay product of the wireless channel to vary significantly, resulting in frequent buffer overflow loss and smaller average TCP congestion window sizes. In addition, this variability also translates to bursty ack arrivals (also called ack compression) at the TCP source which gives rise to multiple packet loss events, resulting in further back-off and timeouts. The combination of smaller average window sizes and frequent/multiple packet drops result in significant degradation of TCP throughput.

In this paper, we make three main contributions. First, we comprehensively evaluate the impact of variable rate and vari-

* Corresponding author.
E-mail: chanme@comp.nus.edu.sg

able delay on long-lived TCP performance. Second, we propose a model to explain and predict TCP's throughput over a link with variable rate and/or delay. Third, we propose a network-based solution called *Ack Regulator* that mitigates the effect of variable rate and/or delay without significantly increasing the round trip time, thereby improving TCP performance.

The remaining sections axe organized as follows. In section 2, we discuss related work. In section 3, we present the motivation for our work using traces from a 3G1X system. In section 4, we describe a model for computing the throughput of a long-lived TCP flow over links with variable rate and variable delay. We then present a simple network-based solution, called *Ack Regulator*, to mitigate the effect of variable rate/delay in section 5. In section 6, we present extensive simulation results that compare TCP performance with and without Ack Regulator, highlighting the performance gains using the Ack Regulator when TCP is subjected to variable rate and delay. Finally, in section 7, we present our conclusions.

2. Related work

In this section, we review prior work on improving TCP performance over wireless networks. Related work on the modeling of TCP performance is presented in section 4.

A lot of prior work has focused on avoiding the case of a TCP source misinterpreting packet losses in the wireless link as congestion signals. In [4], a snoop agent is introduced inside the network to perform duplicate ack suppression and local retransmissions on the wireless link to enhance TCP performance. In [3], the TCP connection is split into two separate connections, one over the fixed network and the second over the wireless link. The second connection can recover from losses quickly, resulting in better throughput. More recent work on applying a split-TCP approach to GPRS can be found in [10]. They measure the channel characteristics of an operational GPRS network and show that substantial delay variations are observed in such a network. The proposed solution, call *ATCP*, aggregates all TCP flows to a mobile into a single TCP flow, eliminates slow start (assume no congestion loss in the Radio Access Network) and uses a separate flow control and error detection and recovery scheme that takes into account the channel characteristics. Link-layer enhancements for reducing wireless link losses including retransmission and forward error correction have been proposed in [26]. Link layer retransmission is now part of both the CDMA2000 and UMTS standards [29,32]. In order to handle disconnections (a case of long-lived loss), M-TCP has been proposed [8]. The idea is to send the last ack with a zero-sized receiver window so that the sender can be in persist mode during the disconnection. Link failures are also common in Ad Hoc networks and techniques to improve TCP performance in the presence of link failures have been proposed in [14]. Note that none of these approaches address specifically the impact of delay and rate variation on TCP, which is the focus of this paper.

Several generic TCP enhancements with special applicability to wireless links are detailed in [15,18]. These include enabling the Time Stamp option, use of large window size and window scale option, disabling Van Jacobson header compression, and the use of Selective Acknowledgments (Sack). Large window size and window scaling are necessary because of the large delay of wireless link while Sack could help TCP recover from multiple losses without the expensive timeout recovery mechanism.

Another issue with large delay variation in wireless links is spurious timeouts where TCP unnecessarily retransmits a packet (and lowers its congestion window to a minimum) after a timeout, when the packet is merely delayed. In [18], the authors refer to rate variability due to periodic allocation and de-allocation of high-speed channels in 3G networks as Bandwidth Oscillation. Bandwidth Oscillation can also lead to spurious timeouts in TCP because as the rate changes from high to low, the rtt value increases and a low Retransmission Timeout (RTO) value causes a spurious retransmission and unnecessarily forces TCP into slow start. A similar problem is also described in [13] where the author analyzed the behavior of TCP over a Wideband CDMA (UMTS) channel which exhibits variable delay and rate. In [20], the authors conduct experiments of TCP over GSM circuit channels and show that spurious timeouts axe extremely rare. However, 3G wireless links can have larger variations than GSM due to processing delays and rate variations due to channel state based scheduling. Given the increased variability on 3G packet channels, the use of TCP time stamp option for finer tracking of TCP round trip times and possibly the use of Eifel retransmission timer [21] instead of the conventional TCP timer can help avoid spurious timeouts.

As mentioned earlier, the effect of delay and rate variability is ack compression and this results in increased burstiness at the source. Ack compression can also be caused by bidirectional flows over regular wired networks or single flow over networks with large asymmetry. This phenomenon has been studied and several techniques have been proposed to tackle the burstiness of ack compression. In order to tackle burstiness, the authors in [24] propose several schemes that withholds acks such that there is no packet loss at the bottleneck router, resulting in full throughput. However, the round trip time is unbounded and can be very large. In [31], the authors implement an ack pacing technique at the bottleneck router to reduce burstiness and ensure fairness among different flows. In the case of asymmetric channels, solutions proposed [5] include ack congestion control and ack filtering (dropping of acks), reducing source burstiness by sender adaptation and giving priority to acks when scheduling inside the network. However, the magnitude of asymmetry in 3G networks is not large enough and can be tolerated by TCP without ack congestion control or ack filtering according to [15].

Note that, in our case, ack compression occurs because of link variation and not due to asymmetry or bidirectional flows. Thus, we require a solution that specifically adapts to link variation. Moreover, the node at the edge of the 3G

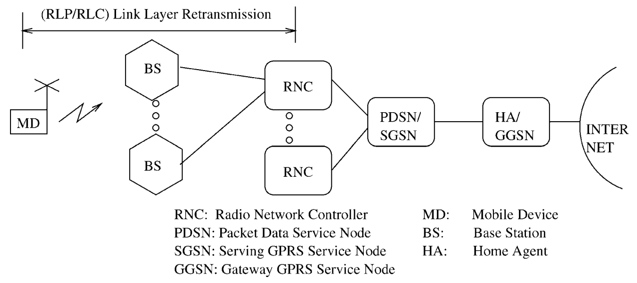

(RLP/RLC) Link Layer Retransmission
MD
BS
BS
RNC
RNC
PDSN/ SGSN
HA/ GGSN
INTER NET
RNC: Radio Network Controller
PDSN: Packet Data Service Node
SGSN: Serving GPRS Service Node
GGSN: Gateway GPRS Service Node
MD: Mobile Device
BS: Base Station
HA: Home Agent

Figure 1. 3G network architecture.

wireless access network is very likely to be the bottleneck router (given rates of 144 Kbps to 2 Mbps on the wireless link) and is the element that is exposed to varying delays and service rates. Thus, this node is the ideal place to regulate the acks in order to improve TCP performance. This is discussed in more detail in the next section.

3. Motivation

A simplified architecture of a 3G wireless network is shown in figure 1. The base stations are connected to a node called the Radio Network Controller (RNC). The RNC performs CDMA specific functions such as soft handoffs, encryption, power control, etc. It also performs link layer retransmission using RLP (RLC) in 3G1X (UMTS) system. In the 3G1X system, the RNC is connected to a PDSN using a GRE tunnel (one form of IP in IP tunnel) and the PDSN terminates PPP with the mobile device. If Mobile IP service is enabled, the PDSN also acts as a Foreign Agent and connects to a Home Agent. In the UMTS system, the RNC is connected to a SGSN using a GTP tunnel (another form of IP in IP tunnel); the SGSN is connected to a GGSN, again through a GTP tunnel. Note that the tunneling between the various nodes allows for these nodes to be connected directly or through IP/ATM networks.

In this architecture, the RNC receives a PPP/IP packet through the GRE/GTP tunnel from the PDSN/SGSN. The RNC fragments this packet into a number of radio frames and then performs transmission and local retransmission of these radio frames using the RLP (RLC) protocol. The base station (BS) receives the radio frames from the RNC and then schedules the transmission of the radio frames on the wireless link using a scheduling algorithm that takes the wireless channel state into account. The mobile device receives the radio frames and if it discovers loss of radio frames, it requests local retransmission using the RLP (RLC) protocol. Note that, in order to implement RLP (RLC), the RNC needs to keep a per-user queue of radio frames. The RNC can typically scale up to tens of base stations and thousands of active users.

In order to illustrate the variability seen in a 3G system, we obtained some traces from a 3G1X system. The system consisted of an integrated BS/RNC, a server connected to the RNC using a 10 Mbps Ethernet and a mobile device connected to the BS using a 3G1X link with 144 Kbps downlink in infinite burst mode and 8 Kbps uplink. The infinite burst

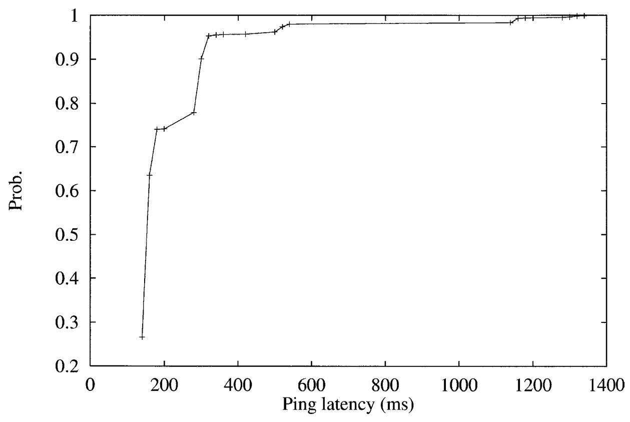


Figure 2. CDF of ping latencies.

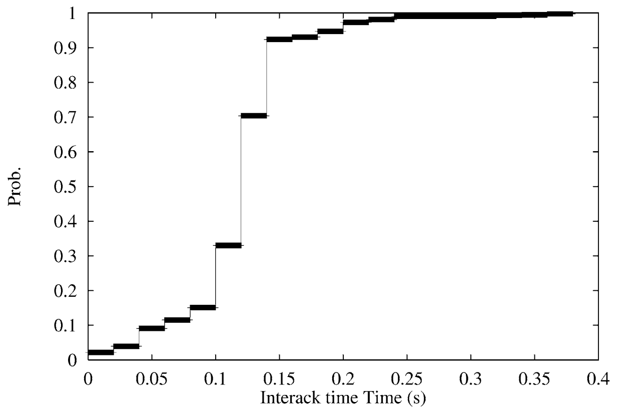
mode implies that the rate is fixed and so the system only had delay variability.

Figure 2 plots the cumulative distribution function (cdf) of ping latencies from a set of 1000 pings from the server to the mobile device (with no observed loss). While about 75% of the latency values are below 200 ms, the latency values go all the way to over 1 s with about 3% of the values higher than 500 ms.

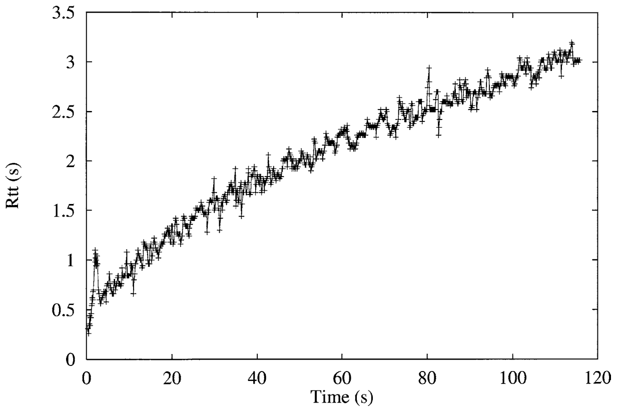
In the second experiment, a TCP source at the server using Sack with timestamp option transferred a 2 MB file to the mobile device. The MTU was 1500 bytes with user data size of 1448 bytes. The buffer at the RNC was larger than the TCP window size,[1] and thus, the transfer resulted in no TCP packet loss and a maximal throughput of about 135 Kb/s. The transmission time at the bottleneck link is $1.448 \cdot 8/135 = 86$ ms. If the wireless link delay were constant, the TCP acks arriving at the source would be evenly spaced with a duration of 172 ms because of the delayed ack feature of TCP (every 2 packets are acked rather than every packet). Figure 3(a) plots the cdf of TCP ack inter-arrival time (time between two consecutive acks) at the server. As can be seen, there is significant ack compression with over 10% of the acks arriving within 50 ms of the previous ack. Note that the ack packet size is 52 bytes (40 + timestamp) and ack transmission time on the uplink is $52 \cdot 8/8 = 52$ ms; an interack spacing of less then 52 ms is a result of uplink delay variation.

Note that the delay variability and the resulting ack compression did not cause any throughput degradation in our system. This was due to the fact that the buffering in the system was greater than the TCP window size resulting in no buffer overflow loss. Figure 3(b) depicts the TCP round trip time (rtt) values over time. Since the buffer at the RNC is able to accommodate the whole TCP window, the rtt increases to over 3 s representing a case of over 30 packets in the buffer at the RNC ($30 \cdot 0.086 = 2.5$ s). Given an average ping latency of 215 ms and a transmission time of 86 ms for a 1500 byte packet, the bandwidth delay product of the link is approximately $(0.215 + 0.86) \cdot 135 = 5$ KB or about 3 packets. Thus, the system had a buffer of over 10 times the bandwidth delay product. Given that we had only one TCP flow in the system,

[1] We did not have control over the buffer size at the RNC in our system.



(a) TCP ack inter-arrival.



(b) TCP rtt.

Figure 3. 3G link delay variability.

a buffer of over 64 KB is not a problem. But, if every TCP flow is allocated a buffer of 64 KB, the buffer requirements at the RNC would be very expensive, since the RNC supports thousands of active users.

Even discounting the cost of large buffers, the inflated rtt value due to the excessive buffering has several negative consequences as identified in [20]. First, an inflated rtt implies a large retransmission timeout value (rto). In the case of multiple packet losses (either on the wireless link or in a router elsewhere in the network), a timeout-based recovery would cause excessive delay, especially if exponential backoff gets invoked. Second, if the timestamp option is not used, the rtt sampling rate is reduced and this can cause spurious timeouts. Third, there is a higher probability that the data in the queue becomes obsolete (for, e.g., due to user aborting the transmission), but the queue will still have to be drained resulting in wasted bandwidth.

Thus, while excessive buffering at the RNC can absorb the variability of the wireless links without causing TCP throughput degradation, it has significant negative side effects, making it an undesirable solution.

4. Model

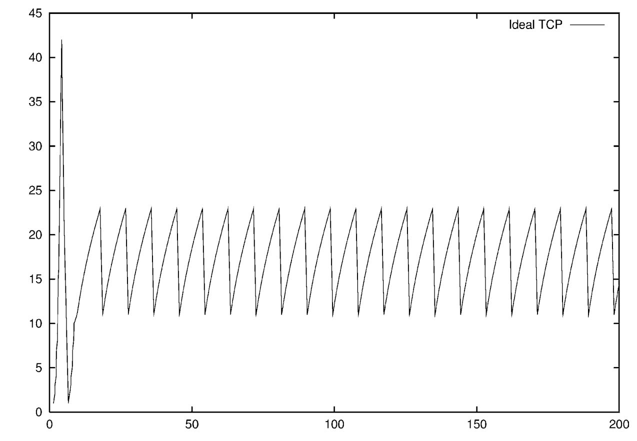
In this section, we model the performance of a single long-lived TCP flow over a network with a single bottleneck server that exhibits rate variation based on a given general distribution and a single wireless link attached to the bottleneck server that exhibits delay variation based on another given distribution.

We use a general distribution of rate and delay values for the discussion in this section since we would like to capture the inherent variation in rate and delay that is a characteristic of the 3G wireless data environment. Given that the wireless standards are constantly evolving, the actual rate and delay distribution will vary from one standard or implementation to another and is outside the scope of this paper. Later, in section 6, we will evaluate TCP performance over a specific wireless link, the 3G1X-EVDO (HDR) system, using simulation.

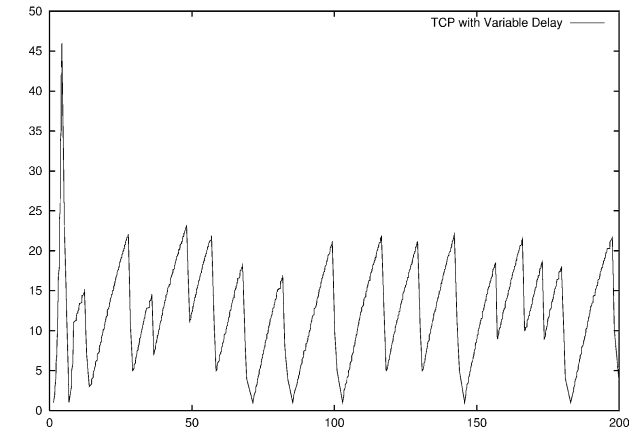
We would like to model TCP performance in the case of variable rate and delay for two reasons. One, we would like to understand the dynamics so that we can design an appropriate mechanism to improve TCP performance. Two, we would like to have a more accurate model that specifically takes the burstiness caused by ack compression due to rate/delay variability into account.

TCP performance modeling has been extensively studied in the literature [1,2,9,19,22,23]. Most of these models assume constant delay and service rate at the bottleneck router and calculate TCP throughput in terms of packet loss probability and round trip time. In [23], the authors model TCP performance assuming deterministic time between congestion events [1]. In [22], the authors improve the throughput prediction of [23] assuming exponential time between congestion events (loss indications as Poisson). In our case, ack compressions and link variation causes bursty losses and the deterministic or Poisson loss models are not likely to be as accurate. In [9], the authors model an UMTS wireless network by extending the model from [23] and inflating the rtt value to account for the average additional delay incurred on the wireless link. However, we believe this will not result in an accurate model because (1) the rtt value in [23] is already an end-to-end measured value and (2) the loss process is much more bursty than the deterministic loss assumption in [23]. In [2], the authors observe that mean values are not sufficient to predict the throughput when routers have varying bandwidth and show that increasing variance for the same mean service rate decreases TCP throughput. However, the approach is numerical, and provides little intuition in the case of delay variance.

Our approach starts with the model in [19] which describes how TCP functions in an “ideal” environment with constant round trip time, constant service rate and suffers loss only through buffer overflow. A brief summary of the result from [19] is presented here before we proceed to our model, which can be seen as an extension. We chose to extend the model in [19] since it makes no assumption about the nature of loss event process (which is highly bursty in our case) and explicitly accounts for link delay and service rate (which are variable in our case). For simplicity, we will only discuss the analysis of TCP Reno. TCP Sack can be analyzed similarly. We also assume that the sender is not limited by the maxi-



(a) Constant delay.



(b) Variable delay.

Figure 4. TCP congestion window evolution over time.

mum receiver window; simple modifications can be made to the analysis for handling this case.

Figure 4(a) shows how the TCP congestion window varies in a constant rate and delay setting. The initial phase where TCP tries to probe for available bandwidth is the *Slow Start* phase. After slow start, TCP goes to *Congestion avoidance* phase. In the case of long-lived TCP flow, one can focus only on the congestion avoidance phase. Let μ be the constant service rate, τ the constant propagation delay, T the minimum round trip time $\tau + 1/\mu$ and B the buffer size. The congestion window follows a regular saw-tooth pattern, going from W_0 to $W_{\max}$, where $W_0 = W_{\max}/2$ and $W_{\max} = \mu\tau + B + 1$. Due to the regularity of each of the saw-tooth, consider one such saw-tooth. Within a single saw-tooth, the congestion avoidance phase is divided into two epochs. In the first epoch, say epoch A, the congestion window increases from W_0 to μT, in time t_A with number of packets sent n_A. In the second epoch, say epoch B, the congestion window increases from μT to $W_{\max}$, in time t_B with number of packets sent n_B. TCP throughput (ignoring slow start) is simply given by $(n_A + n_B)/(t_A + t_B)$ where

$$t_A = T(\mu T - W_0), \tag{1}$$

$$n_A = \frac{W_0 t_A + t_A^2/(2T)}{T}, \tag{2}$$

$$t_B = \frac{W_{\max}^2 - (\mu T)^2}{2\mu}, \tag{3}$$

$$n_B = \mu t_B. \tag{4}$$

This model, while very accurate for constant μ and T, breaks down when the constant propagation and service rate assumptions are not valid. Figure 4(b) shows how the congestion window becomes much more *irregular* when there is substantial variation in the wireless link delay. This is because the delay variation and ack compression result in multiple packet losses.

There are three main differences in the TCP congestion window behavior under variable rate/delay from the traditional saw-tooth behavior. First, while the traditional saw-tooth behavior always results in one packet loss due to buffer overflow, we have possibilities for multiple packet losses due to link variation. To account for this, we augment our model with parameters $p1$, $p2$, $p3$ representing respectively the conditional probability of a single packet loss, double packet loss, and three or more packet losses. Note that, $p1 + p2 + p3 = 1$ by this definition. Second, while the loss in the traditional saw-tooth model always occurs when window size reaches $W_{\max} = \mu\tau + B + 1$, in our model losses can occur at different values of window size, since μ and τ are now both variables instead of constants. We capture this by a parameter $W_f = \sqrt{\sum_{i=1}^{N} W_{\max_i}^2/N}$, that is the square root of the second moment of the $W_{\max}$ values of each cycle. The reason we do this instead of obtaining a simple mean of $W_{\max}$ values is because throughput is related to W_f quadratically (since it is the area under the curve in the congestion window graph). Third, due to the fact that we have multiple packet losses in our model, we need to consider timeouts and slow starts in our throughput calculation. We represent the timeout duration by the T_0 parameter which represents the average timeout value, similar to the timeout parameter in [23].

We now model the highly variable congestion window behavior of a TCP source under rate/delay variation. We first use W_f instead of $W_{\max}$. We approximate τ (the propagation delay) by $\hat{\tau}$, the average link delay in the presence of delay variability. We replace μ (the service rate) by $\hat{\mu}$, the average service rate in the presence of rate variability. Thus, T becomes $\hat{T} = \hat{\tau} + 1/\hat{\mu}$. Now consider three different congestion window patterns: with probability $p1$, single loss followed by congestion avoidance, with probability $p2$, double loss followed by congestion avoidance, and with probability $p3$, triple loss and timeout followed by slow start and congestion avoidance.[2]

First, consider the single loss event in the congestion avoidance phase. This is the classic saw-tooth pattern with two epochs as identified in [19]. Lets call these $A1$ and $B1$ epochs. In epoch $A1$, window size grows from W_{01} to $\hat{\mu}\hat{T}$ in time, t_{A1}, with number of packets transmitted, n_{A1}. In epoch $B1$, window size grows from $\hat{\mu}\hat{T}$ to W_f in time, t_{B1}, with

[2] We assume that three or more packet losses result in a timeout; this is almost always true if the source is TCP Reno.

number of packets transmitted, n_{B1}. Thus, *with probability $p1$, $n_{A1}+n_{B1}$ packets are transmitted in time $t_{A1}+t_{B1}$* where

$$W_{01} = \frac{(int)W_f}{2}, \tag{5}$$

$$t_{A1} = \hat{T}\left(\hat{\mu}\hat{T} - W_{01}\right), \tag{6}$$

$$n_{A1} = \frac{(W_{01}t_{A1} + t_{A1}^2/(2\hat{T})}{\hat{T}}, \tag{7}$$

$$t_{B1} = \frac{W_f^2 - (\hat{\mu}\hat{T})^2}{2\hat{\mu}}, \tag{8}$$

$$n_{B1} = \hat{\mu}t_{B1}. \tag{9}$$

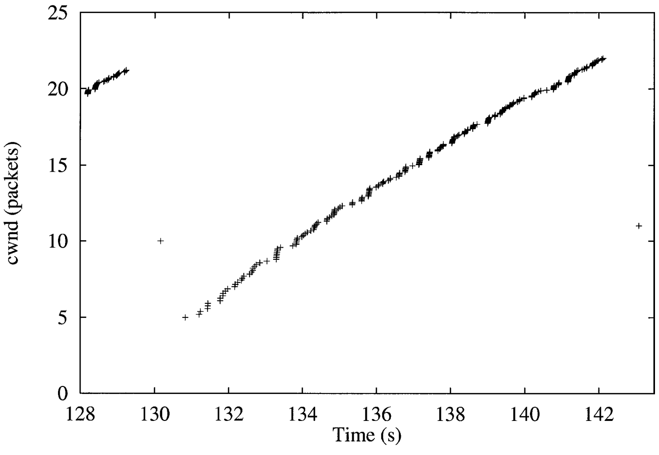
Next, consider the two loss event. An example of this event is shown in figure 5(a). The trace is obtained using ns-2 simulation described in section 6. In this case, after the first fast retransmit (around 130 s), the source receives another set of duplicate acks to trigger the second fast retransmit (around 131 s). This fixes the two losses and the congestion window starts growing from W_{02}. The second retransmit is triggered by the new set of duplicate acks in response to the first retransmission. Thus, the duration between the first and second fast retransmit is the time required for the first retransmission to reach the receiver (with a full buffer) plus the time for the duplicate ack to return to the sender. In other words, this duration can be approximated by the average link delay with a full buffer, $\hat{T} + B/\hat{\mu} = t_R$. We have three epochs now, epoch t_R (time 130–131 s) with one retransmission and zero new packet, epoch $A2$ (131–137 s) with window size growing from W_{02} to $\hat{\mu}\hat{T}$ in time, t_{A2}, with number of packets transmitted, n_{A2}, and epoch $B1$ (137–143 s) as before. Thus, *with probability $p2$, $n_{A2}+n_{B1}$ packets are transmitted in time $t_R + t_{A2} + t_{B1}$* where

$$W_{02} = \frac{(int)W_{01}}{2}, \tag{10}$$

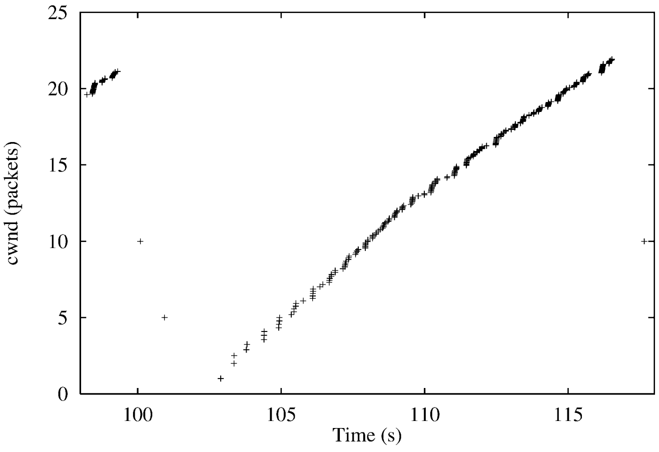
$$t_R = \hat{T} + \frac{B}{\hat{\mu}}, \tag{11}$$

$$t_{A2} = \hat{T}\left(\hat{\mu}\hat{T} - W_{02}\right), \tag{12}$$

$$n_{A2} = \frac{W_{02}t_{A2} + t_{A2}^2/(2\hat{T})}{\hat{T}}. \tag{13}$$



(a) Two packet loss.



(b) Three packet loss.

Figure 5. Congestion window with multiple losses.

Finally, consider the three loss event. An example of this event is shown in figure 5(b). In this case, after the first fast retransmit, we receive another set of duplicate acks to trigger the second fast retransmit. This does not fix the three losses and TCP times out. Thus, we now have five epochs: first is the retransmission epoch (100–101 s) with time t_R and zero new packet, second is the timeout epoch (101–103 s) with time T_0 and zero new packet, third is the slow start epoch (103–106 s) where the window grows exponentially up to previous ssthresh value of W_{03} in time t_{ss} (equation (15)) with number of packets transmitted n_{ss} (equation (16)),[3] fourth is epoch $A3$ (106–111 s) where the window size grows from W_{03} to $\hat{\mu}\hat{T}$ in time t_{A3} (equation (17)) with number of packets transmitted n_{A3} (equation (18)), and fifth is epoch $B1$ (111–118 s) as before. Thus, *with probability $p3$, $n_{ss}+n_{A3}+n_{B1}$ packets are transmitted in time $t_R + T_0 + t_{ss} + t_{A2} + t_{B1}$* where

$$W_{03} = \frac{(int)W_{02}}{2}, \tag{14}$$

$$t_{ss} = \hat{T}\log_2(W_{03}), \tag{15}$$

$$n_{ss} = \frac{W_{03}}{\hat{T}}, \tag{16}$$

$$t_{A3} = \hat{T}\left(\hat{\mu}\hat{T} - W_{03}\right), \tag{17}$$

$$n_{A3} = \frac{W_{03}t_{A3} + t_{A3}^2/(2\hat{T})}{\hat{T}}. \tag{18}$$

Given that the different types of packet loss events are independent and using $p1 + p2 + p3 = 1$, the average TCP throughput can now be approximated by a weighted combination of the three types of loss events to be

$$\frac{p3(n_{ss} + n_{A3}) + p2n_{A2} + p1n_{A1} + n_{B1}}{p3(t_R + T_0 + t_{ss} + t_{A3}) + p2(t_R + t_{A2}) + p1t_{A1} + t_{B1}}. \tag{19}$$

If any of t_*, are less than 0, those respective epochs do not occur and we can use the above equation while setting the respective n_*, t_* to zero. In this paper, we infer parameters

[3] Using analysis similar to [19] and assuming adequate buffer so that there is no loss in slow start.

Table 1
Simulation and model parameters.

Item	Rate (Kb/s)	Delay (ms)	pkts	*TD*	*TO*	T_0	rtt	$p1$	$p2$	W_f	$\hat{T}$	$\hat{\mu}$
1	200	400	89713	401	1	1.76	616.2	0.998	0.000	22.00	440	25.0
2	200	380+e(20)	83426	498	1	1.71	579.3	0.639	0.357	21.38	442	25.0
3	200	350+e(50)	78827	489	12	1.79	595.8	0.599	0.367	21.24	461	25.0
4	200	300+e(100)	58348	496	114	1.92	606.0	0.339	0.279	18.95	517	25.0
5	$u(200, 20)$	400	82180	504	1	1.75	578.1	0.535	0.460	21.61	400	24.74
6	$u(200, 50)$	400	74840	517	29	1.80	579.9	0.510	0.403	20.52	400	23.34
7	$u(200, 75)$	400	62674	516	81	1.86	585.9	0.398	0.348	19.05	400	20.93
8	$u(200, 50)$	350+e(50)	70489	507	43	1.81	595.7	0.496	0.377	20.15	459	23.34
9	$u(200, 75)$	300+e(100)	53357	497	93	2.03	635.7	0.404	0.298	17.78	511	20.93

Table 2
Simulation and model throughput values.

Item	Simulator goodput	Model 1 [23] (accuracy)	Model 2 [22] (accuracy)	Model 3 (equation (19)) (accuracy)
1	199.8	228.5 (0.86)	201.9 (0.99)	199.8 (1.0)
2	185.4	208.0 (0.88)	186.0 (1.0)	186.0 (1.0)
3	175.1	195.5 (0.88)	177.2 (0.99)	180.9 (0.97)
4	129.4	145.3 (0.88)	153.7 (0.81)	137.0 (0.94)
5	182.5	205.2 (0.88)	184.6 (0.99)	181.3 (0.99)
6	166.2	186.0 (0.88)	174.6 (0.95)	165.2 (0.99)
7	139.2	158.4 (0.86)	163.4 (0.83)	137.2 (0.99)
8	156.5	174.6 (0.88)	166.5 (0.94)	160.2 (0.97)
9	118.4	134.0 (0.87)	142.6 (0.80)	125.0 (0.94)

such as loss probability, round trip time, and timeout durations from traces.

Table 1 lists the various parameters used by the different models for simulations with rate and delay variability. We use a packet size of 1000 bytes, a buffer of 10 which represents the product of the average bandwidth times average delay and we ensure that the source is not window limited. *TD* and *TO* denote the number of loss events that are of the triple duplicate and timeout type respectively and these values are used by models in [23] and [22]. The simulation is run for 3600 seconds. We simulate delay and rate variability with exponential and uniform distributions respectively ($u(a, b)$ in the table represents uniform distribution with mean a and standard deviation b while $e(a)$ represents an exponential distribution with mean a). The details of the simulation are presented in section 6.

Table 2 compares the throughput of simulation of different distributions for rate and delay variability at the server and the throughput predicted by the exact equation of the model in [23], the Poisson model in [22] and by equation (19). The accuracy of the prediction, defined as 1 minus the ratio of the difference between the model and simulation throughput value over the simulation throughput value, is listed in the parenthesis. As the last column shows, the match between our model and simulation is extremely accurate when the delay/rate variation is small and the match is still well over 90% even when the variation is large. The Poisson loss model used in [22] performs very well when the variability is low but, understandably, does not predict well when variability increases. The deterministic loss model seems to consistently overestimate the throughput.

From table 1, one can clearly see the impact of delay and rate variability. As the variability increases, the probability of double loss, $p2$, and three or more losses, $p3 = 1 - p2 - p1$, start increasing while the goodput of the TCP flow starts decreasing. For example, comparing case 1 to case 4, $p1$ decreases from 0.998 to 0.339 while $p3$ increases. Increases in $p2$ and $p3$ come about because when the product $\hat{T}\hat{\mu}$ decreases, a pipe that used to accommodate more packets suddenly becomes smaller causing additional packet losses. Given that $n_{A1}/t_{A1} > n_{A2}/(t_R + t_{A2}) > (n_{ss} + n_{A3})/(t_R + T_0 + t_{ss} + t_{A3})$, any solution that improves TCP performance must reduce the occurrence of multiple packet losses, $p2$ and $p3$. We present a solution that tries to achieve this in the next section.

5. Ack Regulator

In this section, we present our network-based solution for improving TCP performance in the presence of varying bandwidth and delay. The solution is designed for improving the performance of TCP flows towards the mobile host (for downloading-type applications) since links like HDR are designed for such applications. The solution is implemented at the wireless edge, specifically at the RNC, at the layer just above RLP/RLC. Note that, in order to implement the standard-based RLP/RLC, the RNC already needs to maintain a per-user queue. Our solution requires a per-TCP-flow queue, which should not result in significant additional overhead given the low bandwidth nature of the wireless environment. We also assume that the data and ack packets go through the same RNC; this is true in the case of 3G networks where the TCP flow is anchored at the RNC because of the presence of soft handoff and RLP.

We desire a solution that is simple to implement and remains robust across different implementations of TCP. To this end, we focus only on the congestion avoidance phase of TCP and aim to achieve the classic saw-tooth congestion window behavior even in the presence of varying rates and delays by controlling the buffer overflow process in the bottleneck link. We also assume for this discussion that every packet is acknowledged (the discussion can be easily modified to account for delayed acks where single ack packets acknowledge multiple data packets).

Our solution is called the Ack Regulator since it regulates the flow of acks back to the TCP source. The intuition behind the regulation algorithm is to avoid any buffer overflow loss until the congestion window at the TCP source reaches a pre-determined threshold and beyond that, allow only a single buffer overflow loss. This ensures that the TCP source operates mainly in the congestion avoidance phase with congestion window exhibiting the classic saw-tooth behavior. Before we present our solution, we describe two variables that will add in the presentation of our solution.

ConservativeMode: mode of operation during which each time an ack is sent back towards the TCP source, there is buffer space for at least two data packets from the source.

Note that if TCP operates in the congestion avoidance phase, there would be no buffer overflow loss as long as the algorithm operates in conservative mode. This follows from the fact that, during congestion avoidance phase, TCP increases its window size by at most one on reception of an ack. This implies that on reception of an ack, TCP source sends either one packet (no window increase) or two packets (window increase). Therefore, if there is space for at least two packets in the buffer at the time of an ack being sent back, there can be no packet loss.

AckReleaseCount: the sum of total number of acks sent back towards the source and the total number of data packets from the source in transit towards the RNC due to previous acks released, assuming TCP source window is constant.

AckReleaseCount represents the number of packets that can be expected to arrive in the buffer at the RNC assuming that the source window size remains constant. Thus, buffer space equal to AckReleaseCount must be reserved whenever a new ack is sent back to the source if buffer overflow is to be avoided.

Figure 6 shows the data and ack flow and the queue variables involved in the Ack Regulator algorithm, which is presented in figure 7. We assume for now that the AckReleaseCount and ConservativeMode variables are as defined earlier. We later discuss how these variables are updated. The Ack Regulator algorithm runs on every transmission of a data packet (deque) and every arrival of an ack packet (enque). The instantaneous buffer availability in the data queue is maintained by the BufferAvail variable (line 2). BufferAvail is then

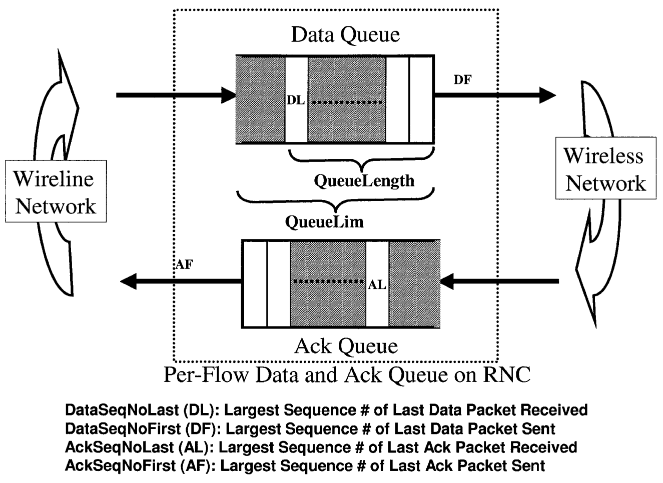

Data Queue
DL
DF
Wireline Network
Wireless Network
QueueLength
QueueLim
AF
AL
Ack Queue
Per-Flow Data and Ack Queue on RNC
DataSeqNoLast (DL): Largest Sequence # of Last Data Packet Received
DataSeqNoFirst (DF): Largest Sequence # of Last Data Packet Sent
AckSeqNoLast (AL): Largest Sequence # of Last Ack Packet Received
AckSeqNoFirst (AF): Largest Sequence # of Last Ack Packet Sent

Figure 6. Ack Regulator implementation.

```
On Enque of Ack/Deque of data packet:
1.   AcksSent = 0;
2.   BufferAvail = QueueLim − QueueLength;
3.   BufferAvail = AckReleaseCount + ConservativeMode;
4.   if (BufferAvail ⩾ 1)
5.     if (AckSeqNoLast − AckSeqNoFirst < BufferAvail)
5.1        AcksSent+ = AckSeqNoLast − AckSeqNoFirst;
5.2        AckSeqNoFirst = AckSeqNoLast;
       else
5.3        AckSeqNoFirst+ = BufferAvail;
5.4        AcksSent+ = BufferAvail;
5.5    Send acks up to AckSeqNoFirst;
```

Figure 7. Ack Regulator processing at the RNC.

reduced by the AckReleaseCount and the ConservativeMode variables (line 3).

Depending on the value of the ConservativeMode variable (1 or 0), the algorithm operates in two modes, a conservative mode or a non-conservative, respectively. In the conservative mode, an extra buffer space is reserved in the data queue to ensure that there is no loss even if TCP congestion window is increased by 1, while, in the non-conservative mode, a single packet loss occurs if TCP increases its congestion window by 1. Now, after taking AckReleaseCount and ConservativeMode variables into account, if there is at least one buffer space available (line 4) and, if the number of acks present in the ack queue AckSeqNoLast − AckSeqNoFirst is lesser than BufferAvail, all those acks are sent to the source (lines 5.1, 5.2); otherwise only BufferAvail number of acks are sent to the source (lines 5.3, 5.4).

Note that the actual transmission of acks (line 5.5) is not presented here. The transmission of AcksSent acks can be performed one ack at a time or acks can be bunched together due to the cumulative nature of TCP acks. However, care must be taken to preserve the duplicate acks since the TCP source relies on the number of duplicate acks to adjust its congestion window. Also, whenever three or more duplicate acks are sent back, it is important that one extra buffer space be reserved to account for the fast retransmission algorithm. Additional buffer reservations of two packets to account for the

```
1. Initialize ConservativeMode = 1; $\alpha = 2$
2. On Enque of ack packet:
   if (DataSeqNoLast − AckSeqNoFirst > $\alpha$*QueueLim)
      ConservativeMode = 0;
3. On Enque and Drop of data packet:
   Conservative Mode = 1;
4. On Enque/Deque of data packet:
   if (((DataSeqNoLast − AckSeqNoFirst) < $\alpha$*QueueLim/2)
      OR (DataQueueLength == 0))
      ConservativeMode = 1;
```

Figure 8. ConservativeMode updates.

```
1. Initialize AckReleaseCount = 0;
2. On Enque of Ack/Deque of data packet:
   (after processing in figure 7)
   AckReleaseCount+ = AcksSent;
3. On Enque of data packet:
   if (AckReleaseCount > 0)
      AckReleaseCount−;
4. On Deque of data packet:
   if (DataQueueLength == 0)
      AckReleaseCount = 0;
```

Figure 9. AckReleaseCount updates.

Limited Transmit algorithm [15] can also be provided for, if necessary.

We now present the algorithm (figure 8) for updating the ConservativeMode variable which controls the switching of the Ack Regulator algorithm between the conservative and the non-conservative modes. The algorithm starts in conservative mode (line 1). Whenever a targeted TCP window size is reached (in this case, 2*QueueLim), the algorithm is switched into non-conservative mode (line 2). TCP Window Size is approximated here by the difference between the largest sequence number in the data queue and the sequence number in the ack queue. This is a reasonable approximation in our case since the wireless link is likely the bottleneck and most (if not all) of the queuing is done at the RNC. When operating in the non-conservative mode, no additional buffer space is reserved. This implies that there will be single loss the next time the TCP source increases it window size. At the detection of the packet loss, the algorithm again switches back to the *conservative* mode (line 3). This ensures that losses are of the single loss variety as long as the estimate of AckReleaseCount is conservative. Line 4 in the algorithm results in a switch back into conservative mode whenever the data queue length goes to zero or whenever the TCP window size is halved. This handles the case when TCP reacts to losses elsewhere in the network and the Ack Regulator can go back to being conservative. Note that, if the TCP source is ECN capable, instead of switching to non-conservative mode, the Ack Regulator can simply mark the ECN bit to signal the source to reduce its congestion window, resulting in no packet loss.

We finally present the algorithm for updating the AckReleaseCount variable in figure 9. Since AckReleaseCount estimates the expected number of data packets that are arriving and reserves buffer space for them, it is important to get an accurate estimate. An overestimate of AckReleaseCount would result in unnecessary reservation of buffers that will not be occupied, while an underestimate of AckReleaseCount can lead to buffer overflow loss(es) even in conservative mode due to inadequate reservation.

With the knowledge of the exact version of the TCP source and the round trip time from the RNC to the source, it is possible to compute an exact estimate of AckReleaseCount. However, since we would like to be agnostic to TCP version as far as possible and also be robust against varying round trip times on the wired network, our algorithm tries to maintain a conservative estimate of AckReleaseCount. Whenever we send acks back to the source, we update AckReleaseCount by that many acks (line 2). Likewise, whenever a data packet arrives into the RNC from the source, we decrement the variable while ensuring that it does not go below zero (line 3).

While maintaining a non-negative AckReleaseCount in this manner avoids underestimation, it also can result in unbounded growth of AckReleaseCount leading to significant overestimation as errors accumulate. For example, we increase AckReleaseCount whenever we send acks back to the source; however, if TCP is reducing its window size due to loss, we cannot expect any data packets in response to the acks being released. Thus, over time, AckReleaseCount can grow in an unbounded manner. In order to avoid this scenario, we reset AckReleaseCount to zero (line 4) whenever the data queue is empty. Thus, while this reset operation is necessary for synchronizing the real and estimated AckReleaseCount after a loss, it is not a conservative mechanism in general since a AckReleaseCount of zero implies that no buffer space is currently reserved for any incoming data packets that are unaccounted for. However, by doing the reset only when the data queue is empty, we significantly reduce the chance of the unaccounted data packets causing a buffer overflow loss. We discuss the impact of this estimation algorithm of AckReleaseCount in section 6.6.

Finally, we assume that there is enough buffer space for the ack packets in the RNC. The maximum number of ack packets is the maximum window size achieved by the TCP flow (α*QueueLim in our algorithm). Ack packets do not have to be buffered as is, since storing the sequence numbers is sufficient (however, care should be taken to preserve duplicate ack sequence numbers as is). Thus, memory requirement for ack storage is very minimal.

6. Simulation results

In this section, we present detailed simulation results comparing the performance of TCP Reno and TCP Sack, in the presence and absence of the Ack Regulator. First, we study the effect of variable bandwidth and variable delay using different distributions on the throughput of a single long-lived TCP flow. Next, we present a model for 3G1X-EVDO (HDR) system (which exhibits both variable rate and variable delay), and evaluate the performance of a single TCP flow in the HDR

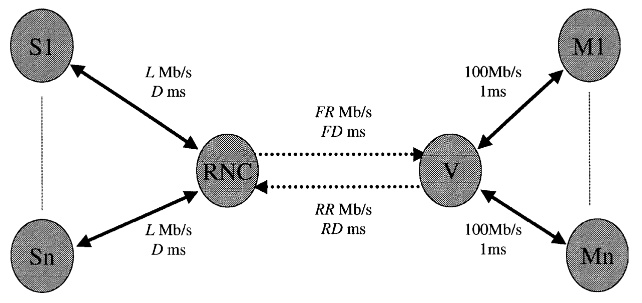


Figure 10. Simulation topology.

environment. Then, we present the performance of multiple TCP flows sharing a single HDR wireless link. Finally, we briefly discuss the impact of different parameters affecting the behavior of Ack Regulator.

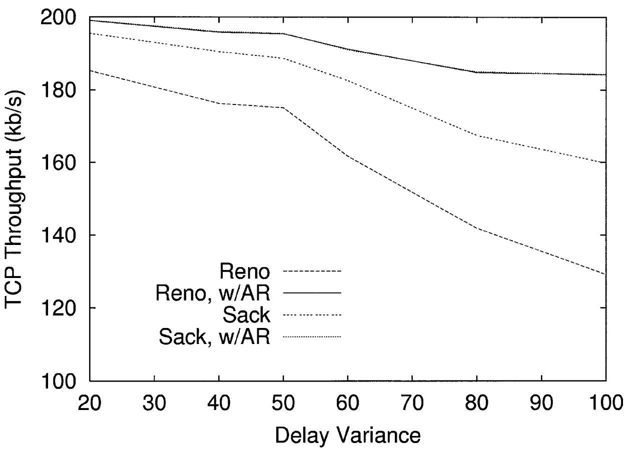
All simulations are performed using ns-2. A number of changes are made to ns-2 before the target system can be simulated. In order to simulate variable bandwidth and delay, a new delay object VarLinkDelay is added. VarLinkDelay inherits from the LinkDelay object. The initialization of VarLinkDelay object allows the user to specify a specific distribution (e.g., uniform, exponential, normal, lognormal, etc.) plus the desired mean and standard deviation. The recv() methods computes the packet transmission time (a function of the current bandwidth) and propagation time (a function of the current delay). In order to ensure that packets are delivered in order, packet arrival time is computed as the maximum of the scheduled time and the arrival time of the previous packet. Ack Regulator is implemented based on the DropTail object. The modified DropTail object accepts parameters that makes it implements Ack Regulator functions if the parameter *ar_enable_* is turned on.

The simulation topology used is shown in figure 10. S_i, $i = 1..n$, corresponds to the set of TCP source nodes sending packets to a set of the TCP sink nodes M_i, $i = 1..n$. Each set of S_i, M_i nodes form a TCP pair. The RNC is connected to the M_i nodes through a V (virtual) node for simulation purposes. L, the bandwidth between S_i and the RNC, is set to 100 Mb/s and D is set to 1 ms except in cases where D is explicitly varied. The forward wireless channel is simulated as having rate *FR* and delay *FD*, and the reverse wireless channel has rate *RR* and delay *RD*.

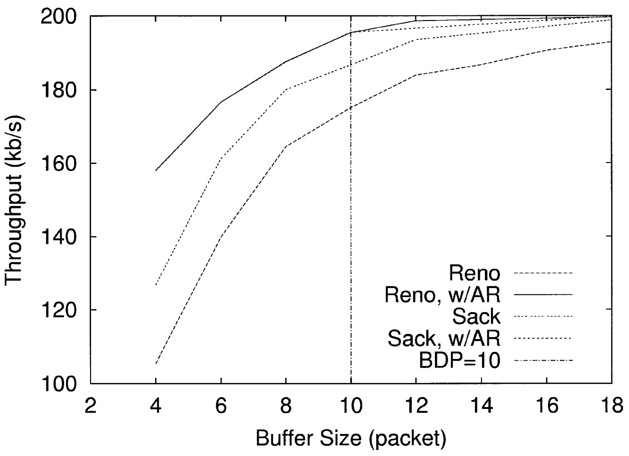
Each simulation run lasts for 3600 s (1 h) unless otherwise specified and all simulations use packet size of 1 Kb. TCP maximum window size is set to 500 Kb. Using such a large window size ensures that TCP is never window limited in all experiments except in cases where the window size is explicitly varied.

6.1. Variable delay

In this section, the effect of delay variation is illustrated by varying *FD*, the forward link delay. Without modification, the use of a random link delay in the simulation will result in out-of-order packets since packet transmitted later with lower delay can overtake packets transmitted earlier with higher delay. However, since delay variability in our model is caused by factors that will not result in packet reordering (e.g., processing time variation) and RLP delivers packet in sequence, the simulation code is modified such that packets cannot reach the next hop until the packet transmitted earlier has arrived. This modification applies to all simulations with variable link delay.



(a) Delay variability.



(b) Different buffer size.

Figure 11. Throughput with variable delay $e(x) + 400 - x$.

Figure 11(a) shows throughput for a single TCP flow ($n = 1$) for $FR = 200$ Kb/s and $RR = 64$ Kb/s. *FD* has an exponential distribution with a mean that varies from 20 ms to 100 ms, and $RD = 400$ ms – mean(*FD*) so that average $FD + RD$ is maintained at 400 ms. The buffer size on the bottleneck link for each run is set to 10, the product of the mean throughput of (200 Kb/s or 25 pkt/s) and mean link delay (0.4 s). This product will be referred to as the bandwidth-delay product (BDP) in later sections. Additional delay distributions like uniform, normal, lognormal, and Poisson were also experimented with. Since the results are similar, only plots for an exponential delay distribution are shown.

As expected, when the delay variation increases, throughput decreases for both TCP Reno and TCP Sack. By increasing the delay variance from 20 to 100, throughput of TCP Reno decreases by 30% and TCP Sack decreases by 19%. On the other hand, TCP Reno and TCP Sack flows which are Ack Regulated are much more robust and its throughput decreases

Table 3
Parameters from simulation for variance = 100.

Item	Rate (Kb/s)	*TD*	*TO*	$p1$	$p2$	$p3$	W_f
Reno	129	496	114	0.34	0.3	0.38	19
Reno+AR	184	302	8	0.98	0.0	0.02	24
Sack	160	434	4	0.99	0.0	0.01	19
Sack+AR	184	302	8	0.97	0.0	0.03	24

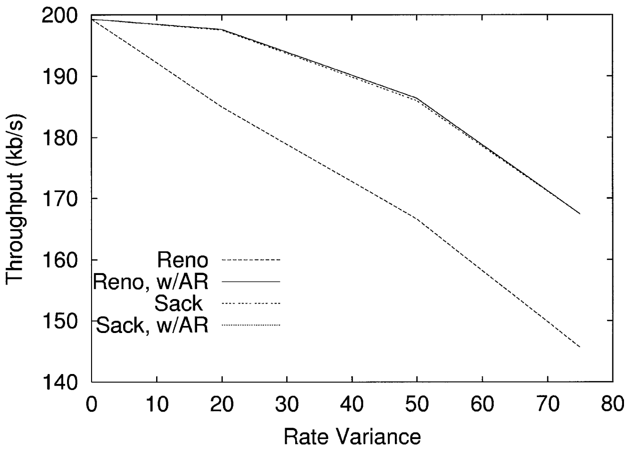
by only 8%. Relatively to one another, *Ack Regulator performs up to* 43% *better than TCP Reno and* 19% *better than TCP Sack*. Another interesting result is that *Ack Regulator delivers the same throughput irrespective of whether the TCP source is Reno or Sack*. This is understandable given the fact that the Ack Regulator tries to ensure that only single buffer overflow loss occurs and in this regime, Reno and Sack are known to behave similarly. This property of Ack Regulator is extremely useful since for a flow to use TCP Sack, both the sender and receiver needs to be upgraded. Given that there are still significant number of web servers that have not yet been upgraded to TCP Sack [25], deployment of Ack Regulator would ensure excellent performance irrespective of the TCP version running.

Figure 11(b) shows how throughput varies with buffer size with the same set of parameters except for *FD*, which is now fixed with a mean of 50 ms (exponentially distributed). Even with a very small buffer of 5 packets (0.5 BDP), Ack Regulator is able to maintain a throughput of over 80% of the maximum throughput of 200 Kb/s. Thus, *Ack Regulator delivers robust throughput performance across different buffer sizes*. This property is very important in a varying rate and delay environment of a wireless system, since it is difficult to size the system with an optimal buffer size, given that the BDP also varies with time. For a buffer of 4 packets, the improvement over TCP Reno and Sack is about 50% and 24%, respectively. As buffer size increases, the throughput difference decreases. With buffer size close to 20 packets (2 BDP), TCP Sack performs close to Ack Regulated flows, while improvement over TCP Reno is about 4%.

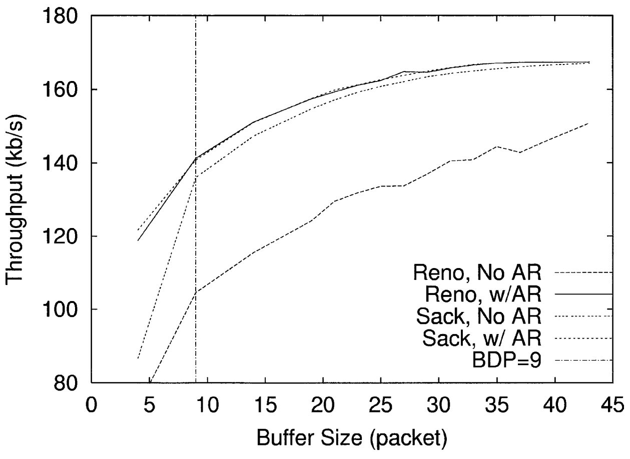
Finally, in table 3, we list parameter values from the simulation for delay variance of 100. First, consider Reno and Reno with Ack Regulator (first two rows). It is clear that Ack Regulator is able to significantly reduce the conditional probability of multiple losses $p2$ and $p3$ as well as absolute number of loss events (*TD* and *TO*) resulting in substantial gains over Reno. Next, consider Sack and Sack with Ack Regulator (last two rows). In this case, we can see that Sack is very effective in eliminating most of the timeout occurrences. However, Ack Regulator is still able to reduce the absolute number of loss events by allowing the congestion window to grow to higher values (24 vs. 19), resulting in throughput gains.

6.2. *Variable bandwidth*

In this section, we vary the link bandwidth, *FR*. Figure 12(a) shows throughput for a single TCP flow. *FR* is uniformly distributed with a mean of 200 Kb/s and the variance is varied from 20 to 75. $FD = 200$ ms, $RR = 64$ Kb/s and



(a) Delay variability.



(b) Different buffer size.

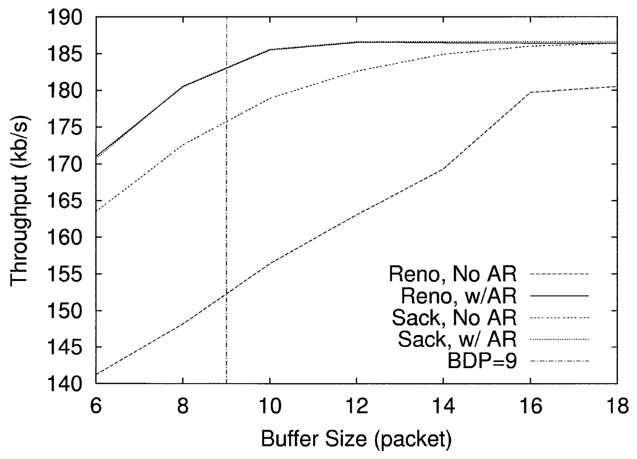
Figure 12. Throughput with variable bandwidth $u(200, x)$.

$RD = 200$ ms. The buffer size on the bottleneck link for each run is 10. Again, we have experimented with other bandwidth distributions, but, due to lack of space, only uniform distribution is shown. Note that, with variable rate, the maximum throughput achievable is different from the mean rate. For uniform distribution, a simple closed form formula for the throughput is simply

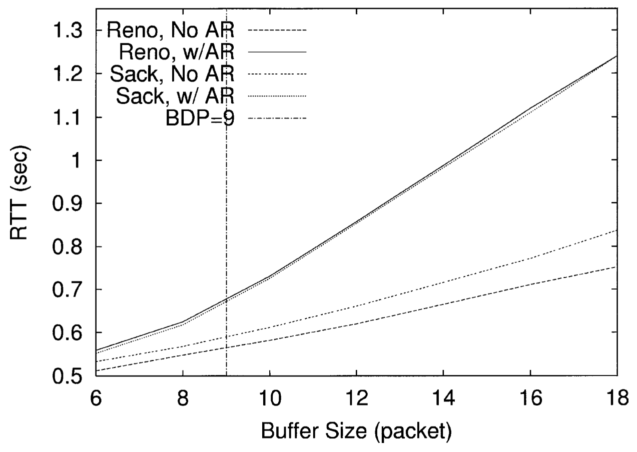
$$\frac{1}{\int_a^b 1/x \, \mathrm{d}x} = \frac{1}{\ln b - \ln a}$$

where b is the maximum rate and a is the minimum rate.

When the rate variance increases, throughput of TCP Reno decreases as expected. Compared to TCP Reno, Ack Regulator improves the throughput by up to 15%. However, TCP Sack performs very well and has almost the same throughput as Ack Regulated flows. Based on the calculations for maximum throughput discussed before, it can be shown that all flows except Reno achieve maximum throughput. This shows that if rate variation is not large enough, TCP Sack is able to handle the variability. However, for very large rate variations (e.g., rate with lognormal distribution and a large variance), the performance of TCP Sack is worse than when Ack Regulator is present.



(a) Throughput vs. buffer size.



(b) rtt vs. buffer size.

Figure 13. Throughput and rtt for $u(200, 50)$, $350 + e(50)$.

Figure 12(b) shows how the throughput varies with buffer size. Note that with a lower throughput, bandwidth delay product is smaller than 10 packets. Again, Ack Regulated TCP flows perform particularly well when the buffer size is small. With buffer size of 5, the improvement over TCP Sack is 40%.

6.3. Variable delay and bandwidth

In this section, we vary both the bandwidth and delay of the wireless link. *FR* is uniformly distributed with a mean of 200 Kb/s and variance of 50, *DR* is exponentially distributed with a mean of 50 ms, $RR = 64$ Kb/s and $RD = 350$ ms. The maximum achievable throughput is 186.7 Kb/s. The BDP is therefore about 9 packets.

Figure 13(a) shows the throughput for a single TCP flow with the buffer size ranging from 7 to 20. The combination of variable rate and delay has a large negative impact on the performance of TCP Reno and it is only able to achieve 70–80% of the bandwidth of Ack Regulated flows when the buffer size is 6 packets. Even with a buffer size of 18 packets, the throughput difference is more than 5%. Throughput of TCP Sack is about 5–10% lower than Ack Regulator, until the buffer size reaches 18 packets (about 2 BDP).

Table 4
HDR data rates for a one user system.

Rate (Kb/s)	Probability	Rate (Kb/s)	Probability
38.4	0.012	614.4	0.069
76.8	0.002	921.6	0.066
102.6	0.008	1228.8	0.234
153.6	0.005	1843.2	0.185
204.8	0.015	2457.6	0.358
307.2	0.045		

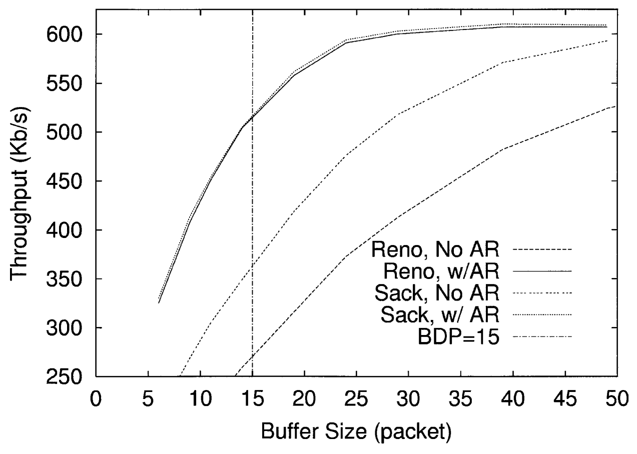
One of the cost of using the Ack Regulator is the increase in average round trip time (rtt). The average rtt values for all 4 types of flows are shown in figure 13(b) for different buffer sizes. TCP Reno has the lowest rtt followed by TCP Sack and the rate of rtt increase with buffer size is comparable. With Ack Regulator, rtt increase is comparable with unregulated flows for buffer size less than 9 (1 BDP). For larger buffer sizes, since Ack Regulator uses $\alpha = 2$ times buffer size to regulate the acks in conservative mode, rtt increases faster with buffer size than regular TCP, where only the data packet buffer size contributes to rtt. For example, with buffer size of 9, Ack Regulated flows have a rtt 15% larger and with buffer size of 18, the rtt is 48% larger compared to TCP Sack. This effect can be controlled by varying the α parameter of the Ack Regulator.

6.4. Simulation with High Data Rate

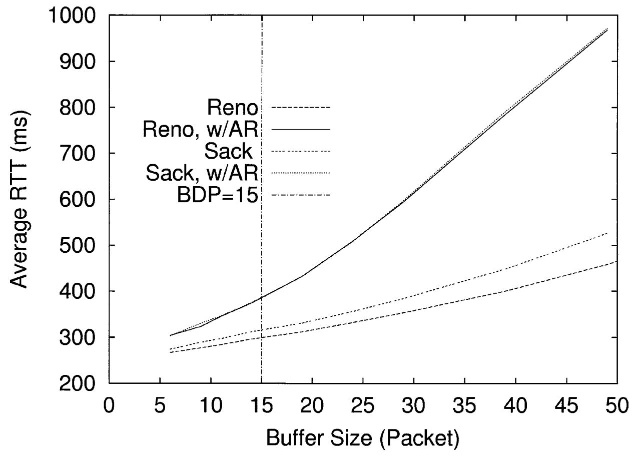
High Data Rate (HDR) [6] is a Qualcomm proposed CDMA air interface standard (3G1x-EVDO) for supporting high speed asymmetrical data services. One of the main ideas behind HDR is the use of channel-state based scheduling which transmits packets to the user with the best signal-to-noise ratio. The actual rate available to the selected user depends on the current signal-to-noise ratio experienced by the user. The higher the ratio, the higher the rate available to the user. In addition, in order to provide some form of fairness, a Proportional Fair scheduler is used which provides long-term fairness to flows from different users. We use Qualcomm's Proportional Fair scheduler [6] in our simulation with an averaging window of 1000 time slots, where each slot is 1.67 ms. While the HDR system results in higher raw throughput, the rate and delay variation seen is substantial.

In this section, we model a simplified HDR environment in ns-2 using a new queue object *HDR*, focusing on the layer 3 scheduling and packet fragmentation. The implementation of the HDR object is based on PacketSFQ, which allows per-flow queuing. The fading model for the wireless link used is based on Jake's Rayleigh fading channel model [17] with a base 4 dB signal-to-noise ratio.[4] The channel model gives us the instantaneous signal-to-noise ratio. Using table 2 in [6] which lists the rate achievable for a given signal-to-noise ratio assuming a frame error rate of less than 1%, the achievable bandwidth distribution (with one user) for our simulation is shown in table 4.

[4] The simulations in [11] used a slightly different HDR model and a base value of 0 dB.

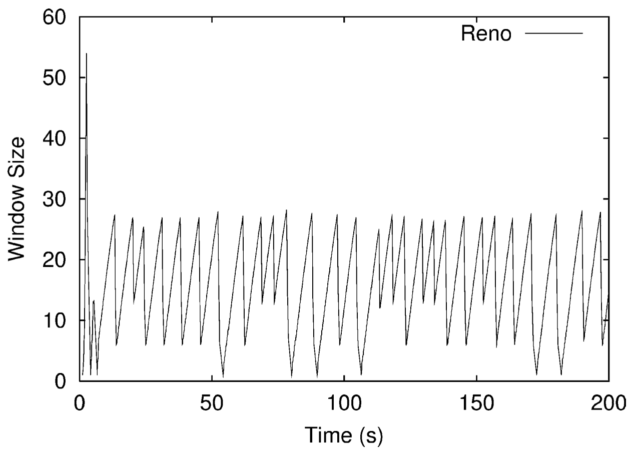


(a) Throughput.

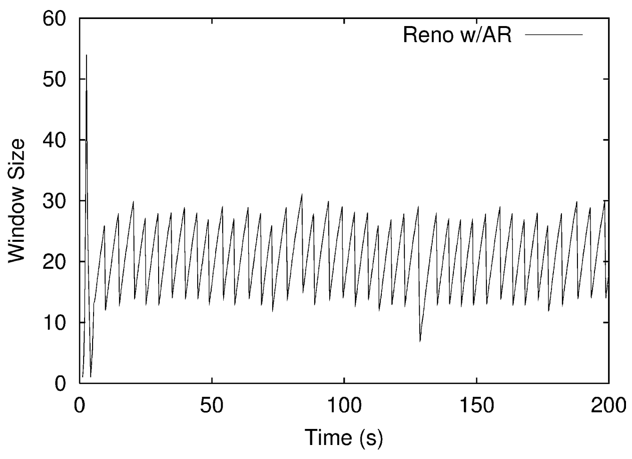


(b) rtt.

Figure 14. Throughput/rtt with HDR, single flow.



(a) Reno.



(b) Reno w/AR.

Figure 15. TCP window evolution without and with AR.

The simulation settings are as follows. *FR* is a variable that has a bandwidth distribution of table 4, due to the variations of the fading conditions of the channel. Based on the guidelines from [27], *FD* is modeled as having a uniform distribution with mean 75 ms and variance 30 and *RD* is modeled as having a uniform distribution with mean 125 ms and variance 15. These are conservative estimates. We expect delay variations in actual systems to be higher (for example, note the ping latencies from our experiment in section 3). The uplink in a HDR system is circuit-based and *RR* is set to be 64 Kb/s which ensures that there is no bandwidth constraint in the reverse link for returning Acks.

Figure 14(a) shows how throughput for a single TCP flow varies with buffer size. Assuming an average bandwidth of 600 Kb/s and a link delay of 200 ms, BDP is 15 packets.

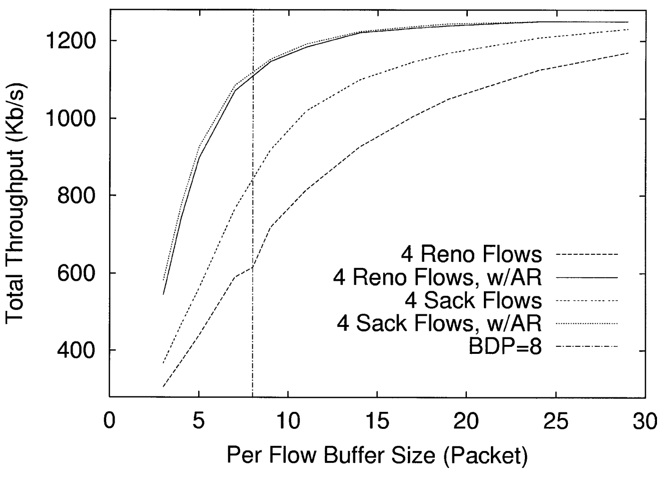
Again, the performance of TCP Reno flows that are Ack Regulated is significantly better than plain TCP Reno over the range of buffer size experimented, with improvements from 16% to 103%. TCP Sack flows also performs worse than Ack Regulated flows up to buffer size of 20. The improvement of Ack Regulator over TCP Sack ranges from 3% to 57%.

As mentioned earlier, one of the costs of using the Ack Regulator is increase in average rtt. The average rtt for all 4 types of flows are shown in figure 14(b) with buffer size varying from 5 to 50. The effect is similar to the rtt variation with buffer size seen in section 6.3.

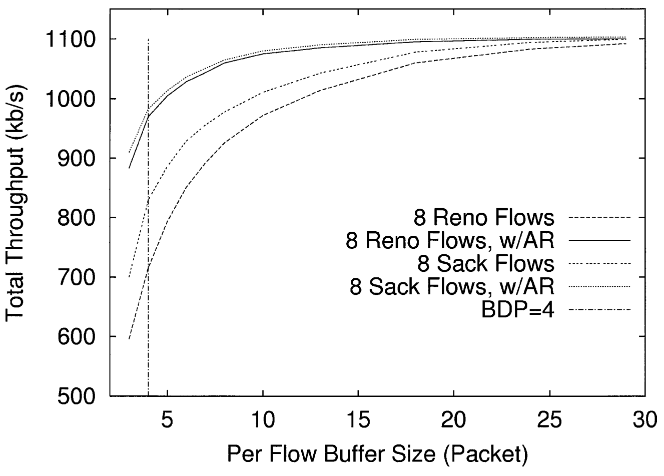
An interesting way to illustrate the benefits of Ack Regulator is to look at figure 15, which shows how the TCP congestion window for a single TCP Reno flow without and with Ack Regulator. From the two figures, it can be observed that Ack Regulator changes TCP window evolution to be *closer to the classic saw-tooth behavior*. The higher throughput achieved by Ack Regulator can be explained as follows. First, the number of multiple packet drops that result in sharp decrease in window size is substantially reduced. This effect is more important for TCP Reno than TCP Sack. Next, the average TCP window size is higher and the number of loss events is reduced. By allowing TCP to operate at a higher average window and with less backoff, throughput increases. This improvement applies to both TCP Reno and Sack.

6.5. *Multiple TCP flows*

In this simulation, the number of flows (n) sharing the bottleneck link is increased to 4 and 8. Per-flow buffering is provided for each TCP flow. For 4 flows, using mean rate of 300 Kb/s, 1 KB packet and rtt of 0.2 s, BDP is 8 packets per



(a) 4 TCP flows.



(b) 8 TCP flows.

Figure 16. Throughput with HDR, multiple flows.

flow. For 8 flows, using mean rate of 140 Kb/s, 1 KB packet and rtt of 0.2 s, BDP is 3.5 or 4 packets per flow.

As the number of TCP flows increases, the expected rate and delay variation seen by individual flows also increases. Thus, even though the total throughout of the system increases with more users due to channel-state based scheduling, the improvement is reduced by the channel variability.

Figure 16(a) shows the throughput for 4 TCP flows. The improvement of Ack Regulator over TCP Sack increases compared to the single TCP case. For example, the gain is about 30% with per-flow buffer size of 8 (BDP). For Reno the gain is even greater. With per-flow buffer size of 8, the improvement is 87%. Similar result can also be observed for the case of 8 TCP flows as shown in figure 16(b). For both TCP Reno and Sack, the gain is about 36% and 19%, respectively, for per-flow buffer size of 4. From the figure, it can seen that, *for TCP Sack and Reno to achieve close to maximum throughput without Ack Regulator, at least three times the buffer requirements of Ack Regulator is necessary* (buffer requirements for acks in the Ack Regulator is negligible compared to the 1 KB packet buffer since only the sequence number needs to be stored for the acks). This not only increases the cost of the RNC, which needs to support thousands of active flows, it also has the undesirable side-effects of large rtt's that was noted in section 3.

With multiple TCP flows, the issue of throughput fairness naturally arises. One way to quantify how bandwidth is shared among flows is to use the fairness index described in [16]. This index is computed as the ratio of the square of the total throughput to n times the square of the individual flow throughput. If all flows get the same allocation, then the fairness index is 1. As the differences in allocation increases, fairness decreases. A scheme which allocates bandwidth to only a few selected users has a fairness index near 0.

For the case of 4 TCP flows, TCP Reno has larger fairness indexes than TCP Sack; adding Ack Regulator reduces the fairness index slightly. All fairness indexes lie between 0.971 to 1.000. The results are similar for the case of 8 TCP flows except that the fairness indexes are lower. The values range from 0.928 to 0.998. Increasing the number of TCP flows reduces the fairness among TCP flows, even in the presence of the proportional fair scheduler. One way to increase fairness is to reduce the number of slots in the averaging window of the proportional fair scheduler but this may also result in reduced overall throughput.

6.6. *Parameters affecting the performance of Ack Regulator*

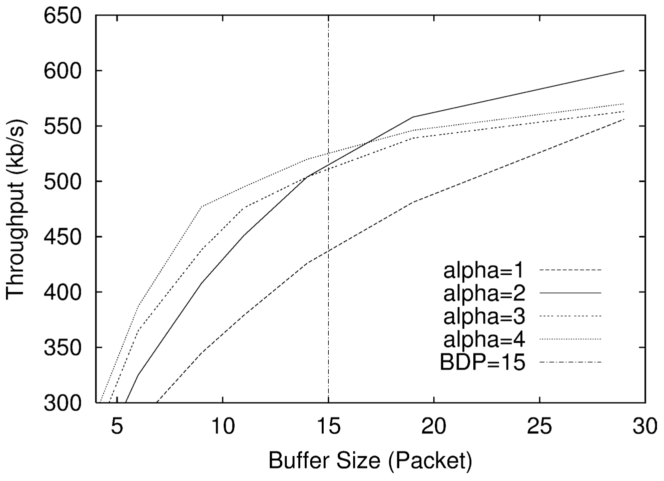
In this section, we investigate the impact of a number of parameters that govern the operation of the Ack Regulator. The simulation setting of the HDR link in section 6.4 is used parameters that govern the operation of the Ack Regulator. The simulation setting of the HDR link in section 6.4 is used and only TCP Reno is simulated.

6.6.1. *Effect of α*

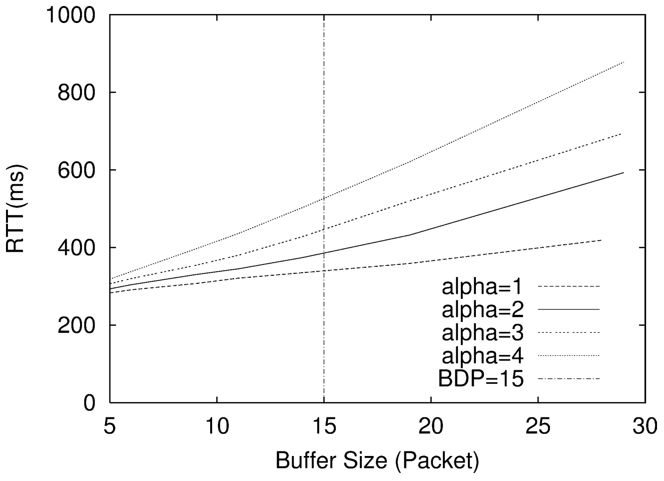
Recall that the parameter α is used to control when the Ack Regulator enters or leaves conservative mode. In conservative mode, buffer space is reserved so that the chance of packet loss is minimal and Ack Regulator goes into non-conservative mode when the window size is greater than α times buffer size. Increasing α means that loss is delayed until the window size is much larger, thereby increasing bandwidth. However, a larger α also implies that the rtt will be larger, as there are more ack packets in the queue. The increase in rtt will not reduce throughput significantly unless the sender is window limited. Thus, a larger α usually trades rtt for bandwidth and vice versa.

Figures 17(a), (b) shows the TCP throughput and rtt of flows with different values of α. It is clear that as α is increased from 1 to 4, rtt increases because more acks are held back for larger α.

However, the throughput results are different. When α increases from 1 to 2, throughput increases for the same buffer size. When $\alpha \geqslant 3$, throughput decreases for larger buffer sizes (>15). The decrease in throughput is caused by the accumulation of sufficiently large amount of duplicate acks that are sent to the TCP sender. In fast recovery mode, the sender's usable congestion window is given by $cwnd + ndup$, where $cwnd$ is the sender's congestion window and $ndup$ the num-



(a) Throughput.



(b) Delay.

Figure 17. TCP throughput and delay with different α.

ber of duplicate acks [12]. Since the Ack Regulator forwards additional duplicate acks without reserving buffer, if the sum of the current window size and the total number of duplicate acks received by the TCP sender exceeds the sender's previous window size, the sender sends new packet for each additional duplicate ack received. These new packets may be dropped at the RNC causing multiple packet losses. As a result, without adding a mechanism in the Ack Regulator to handle excessive number of duplicate acks, α should be kept small (< 3) in order to avoid accumulating too many duplicate acks.

One way around this tradeoff between rtt and bandwidth is to make α dependent on the buffer size so that when the buffer is small, a larger α is used (resulting in higher throughput) and when the buffer is large, a smaller α is used (resulting in same throughput but smaller rtt). Maintaining a target of α times buffer size to be around two times BDP seems to be a good region of operation. We are currently investigating such an adaptive algorithm that estimates the variable BDP and chooses α accordingly.

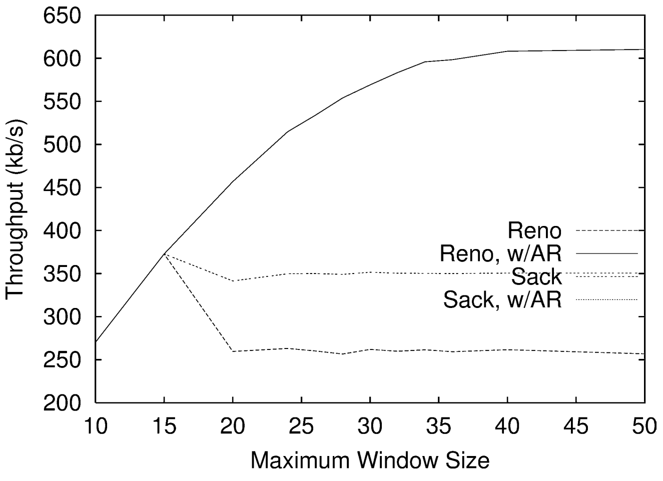


Figure 18. HDR throughput vs. maximum TCP window size.

6.6.2. Effect of maximum congestion window size

In all the previous experiments, the maximum congestion window size was set to 500 KB. This value was never reached, resulting in no impact on the results. In this section, the maximum congestion window size is varied and the impact of window limitation is studied. The buffer size is set to one BDP (15 KB).

Figure 18 shows how TCP throughput is affected by maximum congestion window size for the various TCP flows. For window size less than 15 KB, all flows have similar performance because TCP is window limited and has a throughput of window size over rtt (and incurs no buffer overflow loss). As the window size limit is increased beyond this value, TCP throughputs for Reno and Sack start dropping since now the congestion window exhibits the classic saw-tooth behavior. For Reno, the throughput drops significantly due to the possibility of multiple packet drops. Throughput of TCP Sack also drops but to a smaller degree because of its ability to handle multiple losses. The throughput of Ack Regulated flows continue to increase with increasing congestion window size since Ack Regulator is able to handle the delay and bandwidth variation. Beyond window size of 40, throughput stabilizes and is limited by the average delay and channel rate. For TCP Reno and Sack flows, throughput also stabilizes beyond congestion window size of 20. This is because due to packet loss and timeout, larger congestion window sizes are seldom attained even though the values are allowed.

6.7. Summary of results and discussion

In this section, we first summarize the results from the simulation experiments and then briefly touch upon other issues.

We first started with experiments using a wireless link with variable delay. We showed that Ack Regulator delivers performance up to 43% better than TCP Reno and 19% better than TCP Sack when the buffer size was set to one BDP. We then examined the impact of a wireless link with variable rate. We saw that when the rate variance increases, throughput of TCP Reno decreases as expected. Compared to TCP Reno, Ack Regulator improves the throughput by up to 15%. However, TCP Sack performs very well and has almost the same throughput as Ack Regulated flows as long variable delay and variable rate. We found that this combination had a large negative impact on the performance of both TCP Reno and Sack (up to 22% and 10% improvement, respectively, for Ack Regulated flows). We then considered a specific wireless link standard called HDR which exhibits both variable delay and variable rate. The gains of Ack Regulator over normal TCP flows were even greater in this case, with Ack Regulator improving TCP Reno performance by 16–103% and TCP Sack by 3–57%. We then evaluated the impact of multiple TCP flows sharing the HDR link. The gains of Ack Regulator were as expected (with 30–87% improvements) when the buffer size is set to one BDP.

In general, we showed that Ack Regulator delivers the same high throughput irrespective of whether the TCP flow is Reno or Sack. We further showed that Ack Regulator delivers robust throughput performance across different buffer sizes with the performance improvement of Ack Regulator increasing as buffer size is reduced.

We only considered TCP flows towards the mobile host (for downloading-type applications) since links like HDR are designed for such applications. In the case of TCP flows in the other direction (from the mobile host), Ack Regulator can be implemented, if necessary, at the mobile host to optimize the use of buffer on the wireless interface card.

Finally, Ack Regulator cannot be used if the flow uses end-to-end IPSEC. This is also true for all performance enhancing proxies. However, we believe that proxies for performance improvement are critical in current wireless networks. In order to allow for these proxies without compromising security, a split security model can be adopted where the RNC, under the control of the network provider, becomes a trusted element. In this model, a VPN approach to security (say, using IPSEC) is used on the wireline network between the RNC and the correspondent host and 3G authentication and link-layer encryption mechanisms are used between the RNC and mobile host. This allows the RNC to support proxies such as the Ack Regulator to improve performance without compromising security.

7. Conclusion

In this paper, we comprehensively evaluated the impact of variable rate and variable delay on TCP performance. We first proposed a model to explain and predict TCP's throughput over a link with variable rate and delay. Our model was able to accurately (better than 90%) predict throughput of TCP flows even in the case of large delay and rate variation. Based on our TCP model, we proposed a network based solution called *Ack Regulator* to mitigate the effect of rate and delay variability. The performance of Ack Regulator was evaluated extensively using both general models for rate and delay variability as well as a simplified model of a 3rd Generation high speed wireless data air interface. Ack Regulator was able to improve the performance of TCP Reno and TCP Sack by up to 100% without significantly increasingly the round trip time. We also showed that Ack Regulator delivers the same high throughput irrespective of whether the TCP source is Reno or Sack. Furthermore, Ack Regulator also delivered robust throughput performance across different buffer sizes. Given the difficulties in knowing in advance the achievable throughput and delay (and hence the correct BDP value), a scheme, like Ack Regulator, which works well for both large and small buffers is essential. In summary, Ack Regulator is an effective network-based solution that significantly improves TCP performance over wireless links with variable rate and delay.

References

[1] E. Altman, K. Avrachenkov and C. Barakat, A stochastic model of TCP/IP with stationary random loss, in: *Proceedings of SIGCOMM 2000* (2000) pp. 231–242.

[2] F. Baccelli and D. Hong, TCP is max-plus linear, in: *Proceedings of SIGCOMM 2000* (2000) pp. 219–230.

[3] A. Bakre and B.R. Badrinath, Handoff and system support for indirect TCP/IP, in: *Proceedings of 2nd Usenix Symposium on Mobile and Location-Independent Computing* (April 1995) pp. 11–24.

[4] H. Balakrishnan et al., Improving TCP/IP performance over wireless networks, in: *Proceedings of ACM Mobicom* (November 1995) pp. 2–11.

[5] H. Balakrishnan, V.N. Padmanabhan and R.H. Katz, The effects of asymmetry on TCP performance, in: *Proceedings ACM/IEEE Mobicom* (September 1997) pp. 77–89.

[6] P. Bender et al., A bandwidth efficient high speed wireless data service for nomadic users, IEEE Communications Magazine (July 2000) pp. 70–77.

[7] P. Bhagwat et al., Enhancing throughput over wireless LANs using channel state dependent packet scheduling, in: *Proceedings IEEE INFOCOM'96* (1996) pp. 1133–1140.

[8] K. Brown and S. Singh, M-TCP: TCP for mobile cellular networks, ACM Computer Communications Review 27(5) (1997) 19–43.

[9] A. Canton and T. Chahed, End-to-end reliability in UMTS: TCP over ARQ, in: *Proceedings of Globecomm* (2001) pp. 3473–3477.

[10] R. Chakravorty et al., Flow aggregation for enhanced TCP over wide-area wireless, in: *Proceedings of INFOCOM* (2003) pp. 1754–1764.

[11] M. Chan and R. Ramjee, TCP/IP performance over 3G wireless links with rate and delay variation, in: *Proceedings of ACM Mobicom* (2002) pp. 71–82.

[12] K. Fall and S. Floyd, Simulation-based comparisons of Tahoe, Reno and SACK TCP, ACM Computer Communication Review 26(3) (1996) 5–21.

[13] P.M. Garrosa, Interactions between TCP and channel type switching in WCDMA, Master of Science Thesis, Chalmers University, Polytechnical University of Madrid (January 2002).

[14] G. Holland and N.H. Vaidya, Analysis of TCP performance over mobile ad hoc networks, in: *Proceedings of ACM Mobicom'99* (1999) pp. 219–230.

[15] H. Inamura et al., TCP over 2.5G and 3G wireless networks, draft-ietf-pilc-2.5g3g-07 (August 2002).

[16] R. Jain, *The Art of Computer Systems Performance Analysis* (Wiley, 1991).

[17] W.C. Jakes (ed.), *Microwave Mobile Communications* (Wiley, 1974).

[18] F. Khafizov and M. Yavuz, TCP over CDMA2000 networks, Internet Draft, draft-khafizov-pilc-cdma2000-00.txt

[19] T.V. Lakshman and U. Madhow, The performance of networks with high bandwidth-delay products and random loss, IEEE/ACM Transactions on Networking (June 1997) pp. 336–350.

[20] R. Ludwig et al., Multi-layer tracing of TCP over a reliable wireless link, in: *Proceedings of ACM Sigmetrics* (1999) pp. 144–154.

[21] R. Ludwig and R.H. Katz, The Eifel algorithm: Making TCP robust against spurious retransmissions, ACM Computer Communications Review 30(1) (2000) 30–36.

[22] V. Misra, W. Gong and D. Towsley, Stochastic differential equation modeling and analysis of TCP windowsize behavior, in: *Proceedings of Performance'99* (1999).

[23] Modeling TCP throughput: a simple model and its empirical validation, in: *Proceedings of SIGCOMM 1998* (1998) pp. 303–314.

[24] P. Narvaez and K.-Y. Siu, New techniques for regulating TCP flow over heterogeneous networks, in: *Proceedings of LCN'98* (1998) pp. 42–51.

[25] J. Padhye and S. Floyd, On inferring TCP behavior, in: *Proceedings of SIGCOMM 2001* (2001) pp. 287–298.

[26] S. Paul et al., An asymmetric link-layer protocol for digital cellular communications, in: *Proceedings of INFOCOM 1995* (1995) pp. 1053–1062.

[27] QUALCOMM, Delays in the HDR System (June 2000).

[28] Third Generation Partnership Project, RLC Protocol Specification (3G TS 25.322:) (1999).

[29] TIA/EIA/cdma2000, Mobile Station – Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular Systems, Washington, Telecommunication Industry Association (1999).

[30] TIA/EIA/IS-707-A-2.10, Data Service Options for Spread Spectrum Systems: Radio Link Protocol Type 3 (January 2000).

[31] S. Karandikar et al., TCP rate control, ACM Computer Communication Review (January 2000) pp. 44–58.

[32] 3G Partnership Project, Release 99.

Mun Choon Chan received the B.S. degree from Purdue University, West Lafayette, IN, in 1990 and the M.S. and Ph.D. degrees from Columbia University, NY, in 1993 and 1997, respectively, all in electrical engineering. From 1991 to 1997, he was a member of the COMET Research Group, working on ATM control and management. Since 1997, he has been a Member of the Technical Staff at Bell Laboratories, Lucent Technologies, Holmdel, NJ. His current research include wireless data networking and network management. Dr. Chan has published over 20 technical papers and holds 3 patents. He is a member of the ACM and IEEE, and served on the Technical Program Committee of IEEE INFOCOM 2000–2003.
E-mail: munchoon@lucent.com

Ramachandran Ramjee received his B.Tech. in computer science and engineering from the Indian Institute of Technology, Madras, and his M.S. and Ph.D. in computer science from University of Massachusetts, Amherst. He has been at Bell Labs, Lucent Technologies since 1996, where he is currently a Distinguished Member of Technical Staff. His research interests include protocols, architecture, and performance issues in wireless and high speed networks. Dr. Ramjee is an associate editor of IEEE Transactions on Mobile Computing and a technical editor of IEEE Wireless Communications Magazine. He has published over 25 technical papers and holds 9 U.S. patents.
E-mail: ramjee@bell-labs.com