Improving TCP/IP Performance over Third Generation Wireless Networks

Mun Choon Chan and Ramachandran Ramjee
Bell Labs
Lucent Technologies
munchoon,ramjee@bell-labs.com

***Abstract*— As third generation (3G) wireless networks with high data rate get widely deployed, optimizing TCP performance over these networks would have a broad and significant impact on data application performance. One of the biggest challenges in optimizing TCP performance over the 3G wireless networks is adapting to the significant delay and rate variations over the wireless channel. In this paper, we make two main contributions. First, we present a Window Regulator algorithm that uses the receiver window field in the acknowledgment packets to convey the instantaneous wireless channel conditions to the TCP source and an ack buffer to absorb the channel variations, thereby maximizing long-lived TCP performance. It improves the performance of TCP Sack by up to 100% over a simple drop-tail algorithm for small buffer sizes at the congested router. Second, we present a wireless channel and TCP-aware scheduling and buffer sharing algorithm that reduces the latency of short TCP flows by up to 90% while still exploiting user diversity, thus allowing the wireless channel to be utilized efficiently.**

I. Introduction

Third generation (3G) wide-area wireless networks, based on the CDMA technology [5], are increasingly being deployed throughout the world. While voice and, to some extent, short messaging service have been the predominant applications on the low bandwidth wireless networks to date, the support for high speed data in 3G networks with bandwidths up to 2.4Mbps should enable the widespread growth of wireless data applications. Since the vast majority of data applications use the TCP/IP protocols, optimizing TCP performance over these networks would have a broad and significant impact on the user-perceived data application performance.

TCP performance over wireless networks have been studied over the last several years. Early research [1], [2] showed that wireless link losses have dramatic adverse impact on TCP performance due to the difficulty in distinguishing congestion losses from wireless link losses. These results have been one of the main motivations behind the use of extensive local retransmission mechanisms in 3G wireless networks [14], [15]. While these local retransmission mechanisms solve the impact of wireless link losses on TCP performance, they also result in unavoidable variations in packet transmission delay (due to local retransmissions) as observed by TCP.

In addition to these delay variations, 3G wireless links use channel-state based scheduling mechanisms [4] that result in significant rate variations. The basic idea behind channel state scheduling is to exploit user diversity. As wireless channel quality of different users vary due to fading, the total cell throughput can be optimized if scheduling priority is given to the user with higher channel quality. For example, Qualcomm's proportional fair scheduler [3] exploits this idea while providing for long-term fairness among different users. Thus, while these scheduling mechanisms maximize overall link-layer throughput, they also can result in significant variations in instantaneous individual user throughput as observed by TCP.

Only recently, researchers have begun investigating the impact of these wireless link rate and delay variations on TCP [6], [13]. These variations cause TCP performance degradation due to the difficulty in estimation of the appropriate throughput (i.e. congestion window size and round trip time) of the end-to-end path at the TCP source. When the source over-estimates the available throughput, it causes multiple and frequent packet drops at the congested router buffer, resulting in poor TCP throughput. In [6], we propose a network-based solution called the Ack Regulator to address this problem.

Ack Regulator determines the available buffer space at the congested router and the expected number of data packets arriving at the router, and manages the release of acks to the TCP source so as to prevent an undesired overflow of the buffer. While Ack Regulator was shown to increase the throughput of long-lived TCP flows, it has a few drawbacks due to the limitation of not being able to modify headers of TCP packets. First, Ack Regulator needs to estimate the number of data packets in transit from the source - errors in this estimation, for example due to variations on the wired network, could lead to multiple-packet drops resulting in lowered throughput. Second, since it intentionally causes single packet drops to force TCP source to go into congestion avoidance phase, it inherently cannot achieve maximum goodput. Finally, it also does not address the performance of short-lived flows such as HTTP.

In this paper, we make two important contributions. First, we design a network-based solution called the *Window Regulator* that maximizes TCP performance for any given buffer size at the congested router. Second, we present a scheduling and buffer sharing algorithm that reduces the latency for short flows while exploiting user diversity, thus allowing the wireless channel to be utilized efficiently.

The proposed Window Regulator algorithm uses the receiver window field in the acknowledgment packets to convey the instantaneous wireless channel conditions to the TCP source and an ack buffer to absorb the channel variations, thereby maximizing long-lived TCP performance. Window Regulator ensures that the source TCP *operates in the window-limited region* resulting in a congestion loss-free operation. While the receiver window field of the ack packets have been used for ensuring fairness and regulating flows in wired networks [16], we show that these schemes do not perform as well over wireless links with variation. We show that the Window Regulator results in highest goodput and maximum TCP performance for even small values of the buffer size, reasonably large wired latencies and small amount of packet losses. For example, it improves the performance of TCP Sack by up to 100% over a simple drop-tail policy for small buffer sizes at the congested router. The use of a small buffer for long flows is important when we consider the impact of having both short and long flows sharing the buffer since the buffer needs to have space to be able to absorb the burst of packets from the short flows.

While minimizing short flow latency is important for the user perceived performance of applications like HTTP, any short flow differentiation scheme has to take into account the wireless channel condition in order to take advantage of user diversity. We show that a scheduling algorithm that provides differentiation but does not fully exploit user diversity can have the adverse effect of increasing short flow latencies and decreasing long flow throughput at the same time. We present a wireless channel and TCP-aware buffer sharing and scheduling algorithm that decreases the latency of short TCP flows by up to 90% while still exploiting user diversity, thus allowing the wireless channel to be utilized efficiently.

The rest of the paper is organized as follows. In Section II, we review related work. In Section III, we present our architecture. In Section IV, we start with a simple model to analyze receiver window-based algorithms and their impact on TCP performance over wireless links with variation. The model serves as motivation for the design of our window regulator algorithm. We compare its performance to several algorithms, including the Ack Regulator, through extensive simulations in Section V. In Section VI, we present our buffer-sharing and scheduling algorithm for differentiation of short flows and discuss its performance. We finally present our conclusions in Section VII.

II. Related Work

The vast majority of related work on TCP performance over wireless networks have concentrated on reducing the impact of TCP mis-reacting to wireless losses as congestion losses [1], [2] that result in poor throughput. As mentioned earlier, link layer retransmission in 3G wireless links [14], [15] have effectively reduced the loss rate of wireless links to well under 1%, thereby minimizing the impact of loss on TCP performance.

Recently, there have been several studies that examine the impact of wireless link variations on TCP performance [6], [9], [11], [13]. Large delay variations resulting in delay spikes can cause spurious timeouts in TCP where the TCP source incorrectly assumes that a packet is lost while the packet is only delayed; this unnecessarily forces TCP into slow start, adversely impacting TCP performance. In [13], the authors propose enhancements to the TCP timer calculations to better track the round trip time of the connection, thereby avoiding spurious timeouts. The authors in [9], [11] present several recommendations for TCP hosts such as enabling the timestamp option and the use of large windows, for improving performance over wireless networks. As discussed in the introduction, in [6], the authors present the Ack Regulator solution for avoiding multiple packet drops at the congested router for long-lived flows.

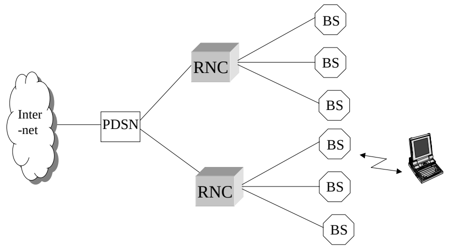


Fig. 1. Wireless network architecture

The use of receiver-window field in the ack packets to throttle TCP source is not new. In [16], the authors implement a receiver window-based technique with ack pacing at the bottleneck router to reduce burstiness and ensure fairness among different flows. In [10], the authors use the receiver window field to better manage TCP throughput over a connection that spans both IP and ATM networks. However, since these solutions do not explicitly cater to significant rate and delay variations, they do not perform as well over wireless networks.

Differentiation for short flows over long flows in wired networks have been studied [7], [8]. The basic idea is to identify short flows heuristically through the use of a simple threshold for bytes transmitted and another threshold for idle period and then give priority to short flows. The authors in [7], [8] use RED with different weights for short and long flows to provide differentiation. However, tuning RED for wireless links that exhibit significant variation will be hard. Furthermore, wireless networks already employ per-user buffering in order to implement reliable link layers with local retransmissions; thus utility of an algorithm like RED is reduced since we already have per-user state information available.

Differentiation for short flows in wireless networks have also been studied [17], [18]. In [18], Foreground-Background (FB) scheduling is used to schedule flows within a user and Proportional Fair (PF) scheduling is used to schedule packets across users. No new algorithm is proposed for inter-user scheduling in place of PF. In [17], the goal is to minimize the average stretch (ratio of actual job completion time over minimum job completion time) over all jobs. One drawback is that it requires advance knowledge of all job sizes which may not be available.

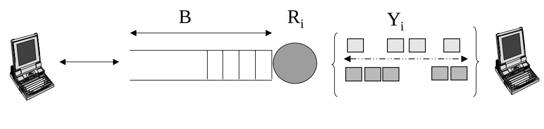


Fig. 2. Simple model of the wireless network

III. Architecture

A simplified view of the third generation wireless access network architecture is shown in Figure 1. The base stations are managed by a Radio Network Controller (RNC). The RNC performs handoffs and terminates the Radio Link Protocol (RLP), that is responsible for improving the reliability of the wireless link through link-layer retransmissions. The Packet Data Service Node (PDSN) terminates PPP, performs the function of a Mobile IP Foreign Agent and interfaces to the public Internet. In this architecture, the RNC receives IP packets encapsulated in PPP from the PDSN. These IP packets are fragmented into smaller radio frames using the RLP protocol and transmitted to the base station. The base station then schedules the transmission of the packet over the air. In the case of a wireless frame loss, the RLP protocol performs retransmission of the radio frames. In this architecture, the RNC maintains a per-user packet buffer and drop packets during congestion when the per-user buffer is full.

For the algorithms discussed in this paper, we assume that the RNC can be extended to distinguish between different TCP flows based on the IP addresses and port numbers inside the packet headers. However, we still use the same per-user buffer limit and ensure that all the flows belonging to a single user share the per-user buffer judiciously. The RNC must carefully choose how much buffer is to be allocated to a single user. Placing a strict upper limit on the maximum buffer allocated to a single user is necessary because of several reasons: a) during handoffs, the per-user buffers have to be either quickly moved from one RNC/base station to another or flushed; large buffers can result in long handoff latencies and/or wasted bandwidth on the access network; b) Scalability and cost considerations also place a limit on the buffer size as the RNC must scale to the order of hundred thousand users or more; c) stale data (for example, user clicking "reload" on a browser or terminating a FTP flow) will still be sent over the wireless link; large buffers imply larger amount of stale data, wasting limited wireless bandwidth.

IV. Window Regulator

Consider a simple model of a single flow in a wireless network shown in Figure 2. The base stations and the RNC are collapsed into a single bottleneck link with the per-flow buffer size at the bottleneck node set to B. For simplicity of analysis, let us assume that there are no losses or re-ordering on the wired or the wireless links and let the latency on the wired network be insignificant compared to wireless delays. We will relax these assumptions later when we evaluate the performance of Window Regulator in Section V.

In order to maximize TCP throughput, we need to ensure that there is always at least one packet available in the bottleneck node to be transmitted (no underflow) and there is no packet loss due to buffer overflow (this leads to retransmission, which is wasted bandwidth, and reduction in congestion window size or timeout that could then lead to underflow). In order to understand the performance of the Window Regulator, we focus on the arrival events on the data queue.

Consider the arrival of the i^{th} packet. Let Y_i represent the number of packets in the "wireless pipe" when packet i arrives. Y_i is a function of the varying rates and delays on the forward and reverse directions on the wireless link. Finally, let $N_i(\leq B)$ be the number of packets in the bottleneck link when packet i arrives. Since we assume that wired network latency is insignificant, all data packets are buffered in the bottleneck link and all acknowledgments are acknowledged immediately with no outstanding packets in the wired network. The sum of data packets in the buffer and in the "wireless pipe" equals the TCP window size, W_i.

$$W_i = N_i + Y_i + 1 \qquad (1)$$

Equation 1 is always true since wired network latency is assumed to be insignificant. In Section V, we study the impact of relaxing this assumption on the throughput of the different algorithms.

Now, for TCP to operate without buffer underflow, the window size, W_i, must obey

$$\forall i \quad Y_i + 1 < W_i \qquad \text{(no underflow)} \qquad (2)$$

Equation 2 states that the window size must be greater than the instantaneous number of packets in the wireless pipe and there must be at least one packet in the buffer for there to be no underflow. In general, this is a necessary but not sufficient condition. However, since we assume that Equation 1 is true, $N_i \geq 1$ when there is no underflow and thus Equation 2 is also a sufficient condition for no underflow.

For there to be no overflow (packet drop), from Equation 1, $B \geq W_i - Y_i$. In other words,

$$\forall i \quad Y_i + B \geq W_i \qquad \text{(no overflow)} \qquad (3)$$

Similarly, Equation 3 is necessary and sufficient to prevent overflow.

The difficulty in choosing an appropriate TCP window size is in adapting to the variations in Y_i while ensuring that equations 2 and 3 are not violated.

Next, we consider three Window Regulator algorithms that set the receiver window size, W^r, in the ack packets in order to manage the window size at the TCP source. For the purpose of this discussion, we assume that each packet is acked. However, the algorithms can be generalized to handle acks that represent more than one packet as well.

A. *Window Regulator-Static (WRS)*

The first algorithm, called the Window Regulator-Static (WRS), is a common algorithm used in wired routers for guaranteeing bandwidth and ensuring fairness (for example, Packeteer [16]). The receiver window size, W^r, is set statically

```
On Enque of each Ack
    set $W^r$ = B
    transmit the Ack to the source
```

Fig. 3. Window Regulator-Static

to the buffer allocated for that flow. That is, Note that, this algorithm is very conservative as it easily satisfies equation 3 (no overflow) since $W_i = B$ and $\forall i Y_i \geq 0$ as the number of packets in the wireless pipe cannot be negative. However, depending on the size of the buffer in relation to the variation of the wireless pipe, the queue can be idle, resulting in loss of throughput. In other words, the utilization of the queue, Q_{WRS}, of this algorithm is approximated by

$$Q_{WRS} = \frac{1}{k}\Sigma^k{}_{i=1} 1\{B > Y_i + 1\} \quad (4)$$

where k is the total number of packets arrived and $1\{B > Y_i + 1\}$ is the delta function:

$$1\{B > Y_i + 1\} = \begin{cases} 1, & B > Y_i + 1 \\ 0, & \text{otherwise} \end{cases}$$

The approximation is exact if the arrival process is Poisson (PASTA). For the case of bursty arrivals, which is the expected case here, Equation 4 is an upper bound.

B. Window Regulator-Dynamic (WRD)

One simple way to extend the WRS algorithm is to track the changes in Y_i and convey this in each ack packet flowing back to the sender. We call this the Window Regulator-Dynamic (WRD) algorithm and it operates as shown in Figure 4, where Y is the current estimate of the size of the wireless pipe.

```
On Deque of each Data packet
    $Y = Y + 1$
On Enque of each Ack
    $Y = Y - 1$
    set $W^r$ = $Y$ + B
    transmit the Ack to the source
```

Fig. 4. Window Regulator-Dynamic

Assuming that wired latency is insignificant, if a data packet i arrives due to this ack departure, then $W_i = W^r$ and $Y_i = Y$. Note that this algorithm might end up reducing the receiver window size W_i compare to W_{i-1} as Y is a varying quantity. However, we never reduce the receiver window size between consecutive acks by more than one packet (reception of an ack reduces the packets in the wireless pipe by one since we assume every packet is acknowledged). Thus, transmitting an ack with a reduced window size does not shrink the window but just freezes the window from advancing (no new packets will be transmitted in response to an ack with reduced window). In the implementation, if packets are not individually acknowledged, Y can be easily measured as the difference between the sequence number of the packet being transmitted and the sequence number of the ack being received[1].

[1]Care has to be taken to handle retransmissions and duplicates.

This algorithm is also conservative as it satisfies equation 3 (no overflow) but it uses a higher window size compared to WRS. Since both these algorithms operate in the window limited region of TCP (where throughput is given by W/RTT, where W is the window size and RTT is the round trip time), the throughput of WRD is as good or better than the throughput of WRS since

$$\text{If} \quad \frac{W}{RTT} \leq R \quad \text{then} \quad \frac{W}{RTT} \leq \frac{W+k}{RTT + k/R} \forall k \geq 0$$

where R is the average rate of the connection. Window size of WRS is B and window size of WRD is $B + Y$. Y is the difference in window size between WRD and WRS and is always non-negative.

However, this algorithm can also result in underflow when the whole buffer is drained before the reception of an ack (causing a sudden large increase in the number of packets in the wireless pipe). The utilization of the queue, Q_{WRD}, of this algorithm is approximated by

$$Q_{WRD} = \frac{1}{k}\Sigma^k{}_{i=1} 1\{Y_i + B > Y_{i+1}\} \quad (5)$$

The equation can be rewritten as

$$Q_{WRD} = \frac{1}{k}\Sigma^k{}_{i=1} 1\{B > Y_{i+1} - Y_i\} \quad (6)$$

In other words, the number of packets in the buffer must be larger than the number of packets transmitted between two packet arrivals for there to be no underflow in this algorithm. Comparing equation 6 to equation 4, we can again see that the *utilization of WRD is always greater than or equal to the utilization of WRS* since $Y_i \geq 0, \forall i$.

C. Window Regulator with ack Buffer (WRB)

One of the problems with the WRD algorithm is that when $Y_{i+1} - Y_i$ increases beyond B and the buffer drains completely, the congested node may not have any acks to provide feedback to the TCP source. One way to overcome this is to maintain an ack buffer in the reverse direction. As mentioned earlier, when the window size is reduced by one in the WRD algorithm, the TCP source does not transmit any packet. Thus, this feedback is not used by the TCP source (other than to reset its timers for this packet). If instead, this ack is stored in an ack buffer, we can use it to indicate any increase in size of the wireless pipe as soon as it occurs and thereby allow the transmission of a data packet from the source. We call this the Window Regulator with ack Buffer (WRB) algorithm and it operates as shown in Figure 5. B_a is the size of the ack buffer and is set to 0 initially.

Note that in the WRB algorithm, B_a will always increase until it converges to some value Y_{max} and $W_{i+1} \geq W_i, \forall i$.

Equation 2 is obviously true since $\forall i$, $Y_{max} \geq Y_i$ and Equation 3 is true because the wired delay is insignificant.

The queue utilization, Q_{WRB}, of this algorithm is approximated by

$$Q_{WRB} = \frac{1}{k}\Sigma^k{}_{i=1} 1\{Y_i + B + B_a > Y_{i+1}\} \quad (7)$$

```
On Deque of each Data packet
    $Y = Y + 1$
    if there is an Ack stored in the ack buffer, then
        $W^r$ = Y + B + $B_a$
        transmit Ack to the source
    endif
On Enque of each Ack, set
    $Y = Y - 1$
    $W^r$ = Y + B + $B_a$
    if (new Ack AND ($W^r <$ last transmitted value of $W^r$))
        store Ack in the Ack Buffer
        $B_a = B_a + 1$
    else
        transmit Ack to the source
    endif
```

Fig. 5. Window Regulator with ack Buffering

If we do not limit the size of the ack buffer (since acks consume very little memory and do not impact the latency of new flows), then B_a will grow large enough to absorb the maximum variation on the wireless link. Therefore, $Pr(Y_i + B + B_a > Y_{i+1})$ approaches one. Thus, if the flow lasts long enough, the *WRB algorithm achieves the maximum utilization of 1*. The queue utilization Q_{WRB} is

$$Q_{WRB} = \frac{1}{k}\Sigma^k{}_{i=1}1 = 1 \tag{8}$$

In order to handle packet losses in the network, ack is also released whenever the data queue is empty. The ack release mechanism works in the following way. On deque of a data packet or enque of an ack packet, if the data queue is detected to be empty the first time, 2 acks are sent. On subsequent enque or deque, if the data queue remains empty, the number of ack released is double until all acks are released. This process resets when the data queue becomes non-empty. A similar reset mechanism is also found in the Ack Regulator [6] to handle packet loss.

We next study the throughput and impact of various parameters on these algorithms through extensive simulation.

V. Performance of long-lived TCP flows

In this section, we study the performance of the three Window Regulator algorithms through extensive simulation and compare its performance with those of Ack Regulator [6] and a simple drop-tail policy.

All simulations are performed using ns-2 with modifications that implement HDR scheduling and variable link delays. The simulation topology used is shown in Figure 6. $S_i, i = 1..n$ corresponds to the set of TCP source nodes sending packets to a set of the mobile 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 router $N1$, is set to 100Mb/s and D is set to 1ms except in cases where D is explicitly varied. The forward wireless channel is simulated with a model for 3G1X-EVDO (HDR) system (which exhibits

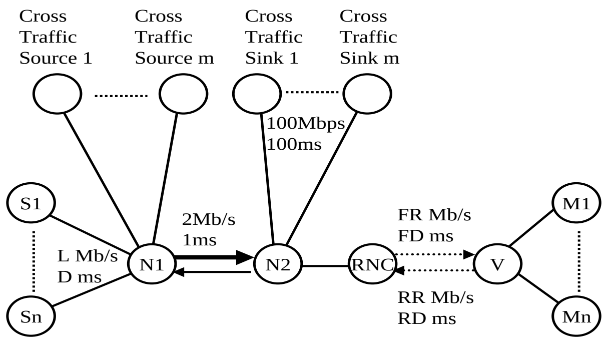


Fig. 6. Simulation Topology

both variable rate and variable delay), and the reverse wireless channel has rate $RR = 64kbps$ and delay RD. The fading model for the wireless link used is based on a Rayleigh fading channel model and the base signal-to-noise ratio (SNR) is set to +4dB. FD is modeled as having a uniform distribution with mean 75ms and variance 30 and RD is modeled as having a uniform distribution with mean 125ms and variance 15. These are conservative values and were used in [6]. Even in the presence of variable delays, we ensure that packets are delivered in order. Figure 6 also shows a set of nodes used to generate cross traffic and are used in Section V-D, where the impact of loss is investigated. The links between the cross traffic sinks and the router $N2$ have bandwidth of 100Mbps and delay of 100ms. All other links have link speed of 100Mbps and delay of 1ms.

Unless otherwise mentioned, we use TCP Sack with the timestamp option for long-lived flows that last at least 1000 seconds in each simulation run. All simulations use packet size of 1KB and the queue size is set to 20 packets. TCP maximum window size is set to 500KB. Using such a large window size ensures that TCP is never window limited by the TCP source in all experiments.

We next evaluate the performance of the five algorithms, i.e. three Window Regulator algorithms (WRS, WRD, WRB), Ack Regulator (AR), and Drop Tail (DT).

A. Throughput vs Buffer Size (Single User)

In this section, the effect of RNC buffer size on throughput is presented for the case of a single user with a long-lived TCP flow. Figure 7(a) and (b) plot the throughput performance of the TCP flow using the five algorithms for TCP Reno and TCP Sack, respectively.

First, observe that the performance of the algorithms are similar for TCP Reno and TCP Sack, with the throughput for TCP Sack slightly higher overall for all the algorithms. Let us now focus on the performance of TCP Sack (Figure 7(b)). As shown in the figure, drop tail performs poorly as the variations over the wireless link cause significant throughput degradation. Interestingly, the WRS algorithm also has very low throughput for a given buffer size and even underperforms DT in some cases. The improvement of WRS over DT ranges from -28%

(for small buffer sizes) to 10%. Recall, that the WRS algorithm is a common algorithm used in wired routers for fairness and regulating flows; setting the receiver window size statically to the buffer size for small values result in significant under utilization of the link as the full buffer of packets is not able to absorb the variations over the wireless link.

The Ack Regulator performs better than WRS and DT but since it can only use the technique of holding back acks to signal the source to slow down during buffer buildup, it does not achieve the maximum throughput gains. Improvement of AR over DT ranges from 1% to 120%. The WRD algorithm delivers close to maximum throughput for reasonable buffer sizes (> 15) but for small buffer sizes, it can lead to under utilization of the link as the node may not have any acks to provide feedback during sudden increases in the number of packets in the wireless channel. Improvement of WRD over DT ranges from 2% to 138%, with the biggest improvement occurring between buffer sizes of 5 to 15 packets.

As expected, *the WRB algorithm outperforms all the other algorithms and delivers the highest throughput ranging with improvement over DT ranging up to 360% for very small buffer sizes and close to 100% improvement when one bandwidth-delay product worth of buffer (15) is used.*

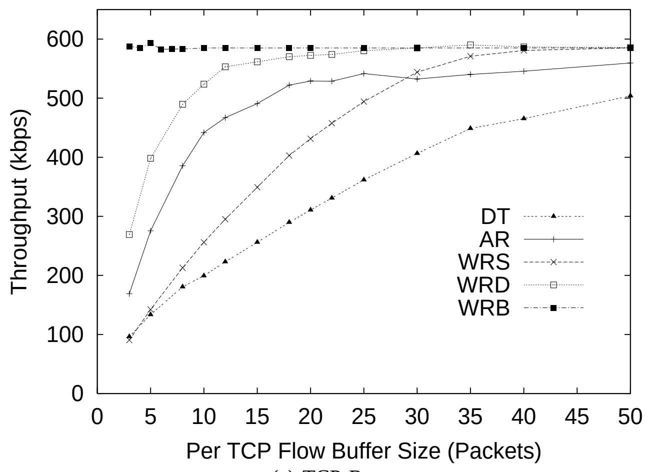
B. Throughput vs Buffer (Multiple Users)

Figure 8(a) and (b) show the effect of buffer size on throughput for 4 and 8 users respectively. The results are similar to the one user case in terms of relative performance of the various algorithms, except for the case of AR and WRD, where AR now outperforms WRD. In terms of absolute performance, the improvement ratio comes down (but is still substantial) when the number of users (flows) are increased since the likelihood of the buffer being empty is reduced even under the drop tail policy.

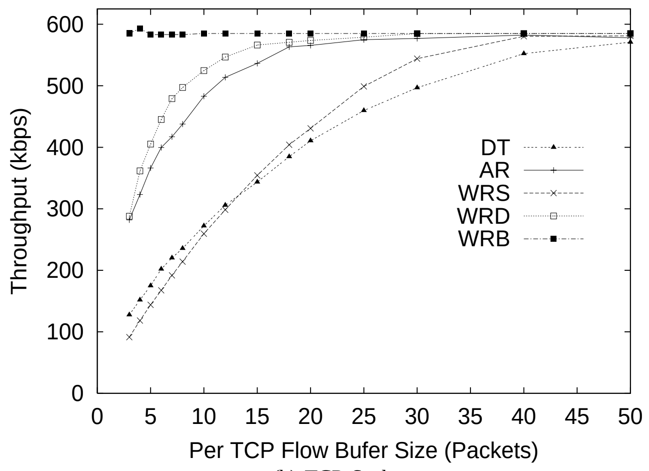
For four users, performance of DT and WRS is almost indistinguishable. The improvement of WRD over DT is up to 88% and the improvement of AR over DT is up to 103%. Again, WRB performs the best and improves throughput over DT by up to 259%.

The result for eight users is similar to that of four users except that the gap between AR and WRD widens. One of the reasons is that as the number of users increases, the variation in Y, the size of the wireless pipe increases. As a result, the likelihood of an underflow increases, resulting in decrease in throughput. Again, WRB performs the best with throughput improvements over DT ranging up to 122%.

We have also investigated the trade-off between throughput and Round Trip Time (RTT) over multiple users but due to lack of space will only briefly present the results here. AR has the worst throughput-RTT trade-off due to buffering of acks. WRS and DT exhibit fairly similar behavior and achieve slightly better trade-off than AR. WRB always operates with high RTT but also always achieves high throughput. WRD achieves the best trade-off among all the algorithms.



(a) TCP Reno



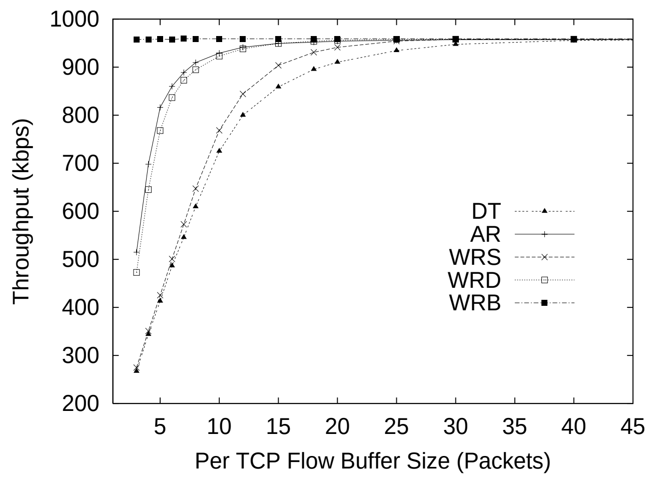
(b) TCP Sack

Fig. 7. Throughput vs. Queue Length for a single user/flow (TCP Reno and Sack)

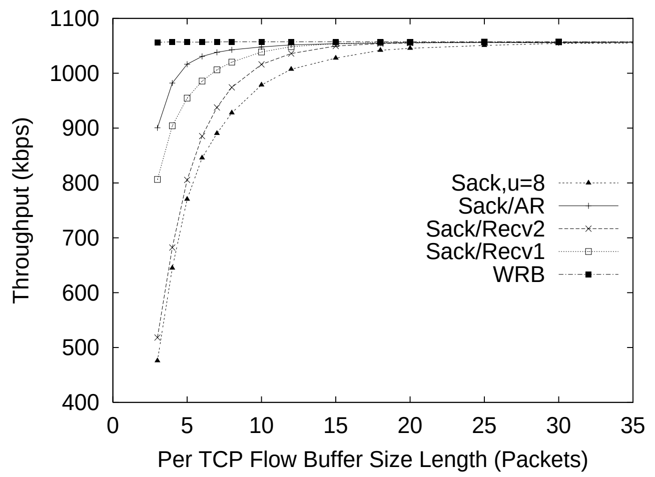
C. Throughput vs Wired Latency

In all previous measurements, the latency on the wired network is assumed to be small (D=1ms). In this section, we study the effect of a larger wired latency on throughput by varying D. Recent measurement on the Internet (http://amp.nlanr.net/AMP) shows that most round trip times are less than 200ms and rarely go above 400ms. In this section, D is vary from 1ms to 500ms, which is equivalent to varying the wired round trip time from 2ms to 1000s.

Figure 9 shows how the throughput of the different algorithms vary with increasing wired latency for 1 user/flow. The performance of all algorithms are expected to drop as the wired latency increases since TCP throughput is inversely proportional to RTT. For round trip wireline latency less than 70ms, WRS is better than DT. Beyond that point, since WRS is operating at the window limited region and the TCP window remains fixed at 20 packets, throughput of WRS degrades inversely with RTT and performs worse than DT which can utilize a larger window. For latency below 40ms, AR performs



(a) 4 Users



(b) 8 Users

Fig. 8. Aggregate Throughput vs. Queue Length for TCP Sack with multiple users/flows

very well. For larger wired latencies, the estimation algorithm in AR becomes less accurate and buffer loss begins to occur more frequently. Beyond latency of 120ms, improvement of AR over DT is less than 10% and beyond 200ms the throughput is very close to DT. WRD is fairly robust with respect to increase in wired latency up to 200ms. One impact of larger latency on WRD and WRB is that more packets are being buffered in the wired network, increasing the chance of buffer underflow and lower throughput. With a round trip latency of 200ms, WRB is still about 25% better than DT. However, the throughput degrades rapidly beyond latency of 200ms. With latency larger than 400ms, WRB has lower throughput than AR and DT. Again, this is due to the fact that WRB is operating at the window limited region. The performance of WRD is similar to WRB.

To summarize, the performance of AR degrades rapidly at round trip latency larger than 40ms but its performance is never worse than DT. On the other hand, both WRD and WRB perform well with round trip latency below 200ms but their performance degrade rapidly at larger latencies since they

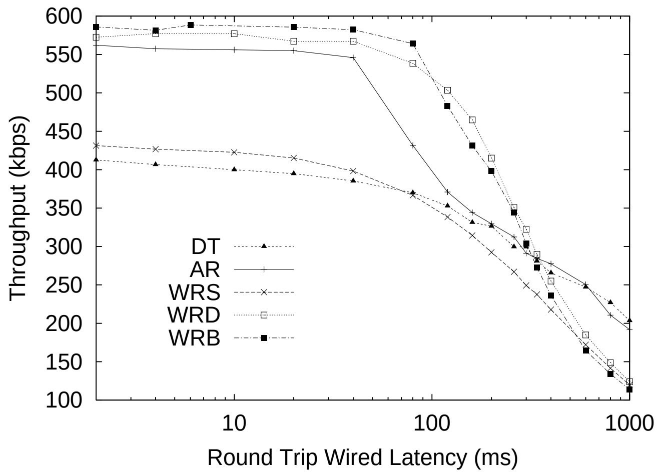


Fig. 9. Throughput vs. Round Trip Wired Latency

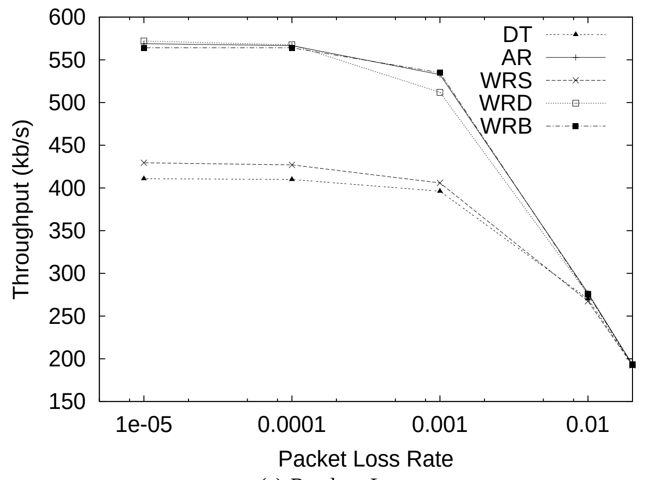
are operating in the window limited region. Beyond 400ms round trip time on the wired network, they perform worse than DT. Since the window sizes of all the three algorithms (AR, WRD and WRB) are a function of the buffer size available in the RNC, increasing the RLP buffer size will allow these algorithms to maintain their high throughput for even larger wired latencies.

D. Impact of Loss

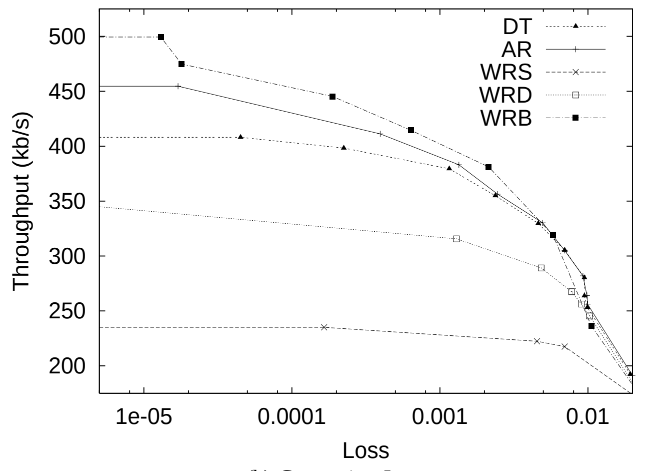
In all the simulations so far, we have assumed zero loss in the rest of the network. In this section, we study the impact of loss in two different ways. First we simulate the effect of random loss and second we simulate the effect of congestion loss. In these experiments, we consider the case of a single mobile user, $M1$, performing a download from source, $S1$. Each simulation runs for 10000s.

The amount of random loss is varied in the link between the virtual node V and the mobile device $M1$ by using the random loss error module available in ns-2. The packet loss rate is varied from 10^{-5} to 10^{-2}. Figure 10(a) shows how throughput varies with increasing amount of packet loss. AR, WRB and WRD continue to perform well for very small amount of loss but the throughput starts to decrease at loss rate of 10^{-3}. With loss rate 10^{-2} or greater, all algorithms have the same low performance since random loss is now the dominant factor determining TCP throughput. Note that random loss can occur in the wireline network as well if RED is enabled on the routers.

In order to generate congestion loss, 4 FTP sessions using TCP Sack are generated from the cross traffic sources in Figure 6 to the cross traffic sinks. The bottleneck link is the link going from router $N1$ to $N2$ and has a link bandwidth of 2Mbps with delay of 1ms. The packet buffer on router $N1$ is set to 100. Different congestion conditions are simulated by varying the link bandwidth between the cross traffic sources and the router $N1$ from 300kbps to 500kbps. The impact of congestion loss on performance is shown in Figure 10(b). The loss rate plotted in the figure is the loss rate experienced only by the traffic going from S1 to M1.



(a) Random Loss



(b) Congestion Loss

Fig. 10. Throughput vs. Loss

A number of observations can be made. Both WRB and AR perform better than DT with congestion loss rate below 10^{-3} though the difference in performance decreases fairly rapidly from 10^{-5} to 10^{-3}. On the other hand, WRD performs poorly with respect to congestion loss and performs worse than DT for even very small amount of loss. Similar result is true for WRS. The results for congestion loss, which is different from random loss, can be explained as follows.

During congestion buildup, the buffer in router $N1$, which can buffer up to 100 packets, increases the RTT by up to 400ms in the worst case. With the increase in RTT, the throughput of WRD, as shown in Section V-C, decreases rapidly. In fact, throughput of WRD decreases to below 400 kbps before any packet loss happens and is caused solely by the increase in wired latency due to congestion. The same is true for WRS which performs even worse as it operates with a smaller window. Interestingly, the performance of AR and WRD do not degrade as significantly. This is due to the fact that these schemes have an ack buffer that can provide fast feedback. Recall that for both AR and WRD, whenever

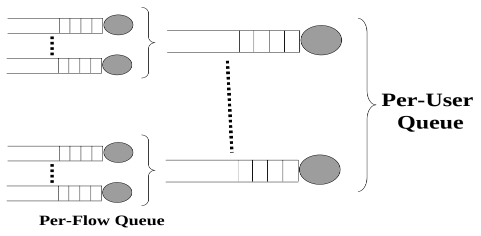


Fig. 11. Queueing Structure of Scheduler

the data queue (going towards the mobile device) is empty, the reset mechanism is enabled and more acks are released. This provides quick feedback to the TCP source and high throughput is maintained even in the presence of congestion losses.

E. Comparison Summary

We conclude this section by summarizing the results of the evaluation. Simulation results show that DT and WRS cannot adapt to the large rate and delay variation in the wireless channel and TCP throughput suffers as a result. AR adapts reasonably well to the large variation but does not perform well when the wired latency is significant as estimation errors cause throughput degradation. WRD performs well in terms throughput and robustness against reasonable wired latencies but performs poorly in the presence of congestion loss. Finally, WRB is the best algorithm in terms of maximizing overall throughput and is robust against reasonably large wired latencies and packet loss.

VI. Short Flow Differentiation

In the previous section, the problem of improving the performance of long-lived TCP flows is addressed. However, it is well known that Internet traffic consists of a small number of long-lived flows that make up a large part (in bytes) of the total traffic and a large number of the flows (in count) that are short-lived. This is especially true with the popularity of applications such as web browsing. As a result, optimizing the third generation wireless data system for short-lived TCP flows is also important. The main difference between the performance goal for between long and short flows is that in the former case the goal is to maximize throughput and in the latter case, the performance goal is to minimize the average transfer latency.

In this paper, we consider a two level hierarchical queueing system, as shown in Figure 11 where the first level consists of per-flow queue for a given user and the second level consists of a per-user queue. Within each first level queue, an intra-user scheduler selects a packet to be sent first among all the flows of that respective user. At the second level queue, a 3G scheduler selects a packet among different users to be sent over the wireless channel. For example, in HDR, the first level scheduler is a First-in-First-out (FIFO) scheduler and the second level scheduler is a proportional fair queueing (PF) scheduler [3].

The main contribution of this section is for inter-user scheduling where we show that a straight forward application of flow differentiation without fully exploiting user diversity does not improve short flow latency and at the same time also reduces overall throughput. We propose a scheduling algorithm for the inter-user scheduler that integrates the elements of flow differentiation into the PF algorithm such that short flow latency can be reduced without sacrificing system throughput.

A. Scheduling

In this section, we first briefly describe our intra-user scheduler called Short Flow Priority (SFP). We then present three inter-user scheduling algorithms.

In SFP, only two classes are defined and strict priority is implemented between the classes. Therefore, if packets from the higher priority class are present, they will always be scheduled first. A flow is identified by the information in the packet header (e.g. Type of Service bits in the IP header or the 4 tuples, namely source IP address, destination IP address, source TCP port and, destination TCP port) and a flow is classified as either a short or long flow by the amount of bytes sent so far by the scheduler. Initially all flows are classified as short flows and a counter keeps track of the total number of bytes sent so far for each flow. When that counter increases beyond a pre-defined threshold, the flow is re-classified as a long flow. This simple reclassification scheme reduces the likelihood of starving long flows of the same user because SFP would eventually promote short flows to long flows. A flow is also re-classified from a long flow to a short flow if the flow is idle (no packet arrival) for a certain amount of time, called the *Reset Duration.* This reclassification has two advantages. First, for interactive applications like telnet, such resets will allow a telnet session to be classified as a short flow even though the total amount in bytes of a telnet session is large. Second, for the case where web traffic from a mobile terminates on a proxy or uses persistent connections (HTTP 1.1), it is important to re-classify the flow as short since the idle period likely indicates that the data belongs to a new Web download. A strict priority buffer eviction is used, where high priority packets always evict lower priority packets if the buffer is full (except for the packet being served). We found that the use of RED/RIO schemes as in [7], [8] requires tuning of parameters which can be difficult. The use of a strict priority eviction policy is simple and provide sufficient differentiation.

Next, we consider the inter-user scheduler. There are three parameters that can be taken into account by such a scheduler, namely: exploiting user diversity in order to improve overall throughput, maintaining long term fairness, and minimizing short flow latency. Flows to a given user are classified as being a long or short flow based on the first packet in the queue. We consider three different schedulers that take into account different aspects of the above three parameters.

1) PF scheduler: PF is the scheduler used in HDR [3]. In order to understand how PF works we first need to understand the concept of user diversity which is central to how PF improves channel throughput. Consider the model where there are N active users sharing a wireless channel. The channel condition seen by each user varies independently. Better channel conditions translate into higher user rate and vice versa. Each user continuously sends its measured channel condition back to the centralized PF scheduler which resides at the base station. If the channel measurement feedback delay is relatively small compared to the channel rate variation, the scheduler has a good enough estimate of all the users' channel condition when it schedules a packet to be transmitted to the user. Since channel condition varies independently among different users, user diversity can be exploited by selecting the user with the best condition to transmit in different time slots. This approach can increase system throughput substantially compared to a round-robin scheduler. However, such a rate maximizing scheme can be very unfair and users with relatively bad channel conditions can be starved. Hence, the mechanism used in PF is to weight the current rate achievable by a user by the average rate received by a user. The decision of the PF scheduler is to select the user with the largest $max_i \frac{R_i}{A_i}$, where R_i is the rate achievable by user i and A_i is the average rate of user i. The average rate is computed over a time window as a moving average:

$$\begin{aligned} A_i(t+1) &= (1-\alpha)A_i(t) + \alpha R_i && \text{if scheduled} \\ A_i(t+1) &= (1-\alpha)A_i(t) && \text{if not scheduled} \end{aligned}$$

Since PF takes into account the average rate, if traffic from user i is predominantly short web traffic that arrives infrequently and the other users have only long-lived TCP flows (or a mixture of long and short flows), PF does a good job of giving priority to the short flows from user i since A_i is smaller and will be given higher priority compared to the other users with long-lived flows. On the other hand, in the presence of a large number of users with long flows, short flows from users with a mixture of short and long flows (where the average rates, A_is, are high) does not get priority.

2) PF-SP scheduler: One possible way to improve short flow latency over the PF scheduler is to include a notion of priority in the PF scheduler so that users with short flows are given higher priority than users with long flows. We call this the *PF-SP* (Proportional Fair with Strict Priority) scheduler. PF-SP always prefers short flows over long flows. In PF-SP, we select the user with the highest instantaneous rate among users with short flows. When there are only long flows in the system, the default PF is run. The PF-SP is summarized in Figure 12. PF-SP gives strict priority to short flow across all users and,

```
Let set of short flow user be $S$
Let set of long flow user be $L$
if $S$ is non empty
  select the user $i$ with the largest $R_i$
else if $L$ is non empty
  select the user $i$ with the largest $\frac{R_i}{A_i}$
Update $A_i$ for all users
```

Fig. 12. The PF-SP Algorithm

in a channel with no variation, can be expected to provide the lowest latency for short flows. In PF-SP, by selecting the user with the highest rate among all users with short flows, some user diversity is also exploited (limited to users with short flows only instead of over all flows) and short flow latency is expected to be minimized, at the expense of fairness among users. The average rate for each user is maintained over all short and long flows.

3) PF-RP scheduler: One of the problems of the PF-SP scheduler is that it always prefers short flow over long flow, independent of channel conditions. Short flows by definition cannot be always present in the queue as long flows dominate in terms of byte count. As a result, the amount of diversity available to PF-SP could be reduced in comparison to PF.

```
Run the PF and let user $i$ be selected
Run the PF-SP and let user $j$ be selected
if $R_i > R_j$
    select user $i$
else
    select user $j$
update $A_i$ for all users
```

Fig. 13. The PF-RP Algorithm

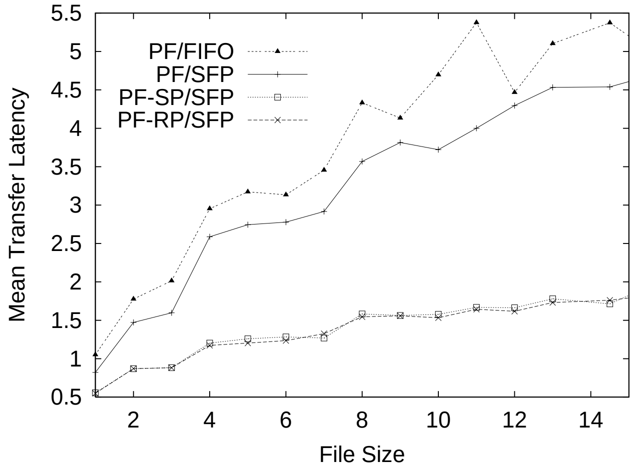
We propose an algorithm called *PF-RP: Proportional Fair with Rate Priority* which attempts to strike a better balance between minimizing short flow latency, exploiting user diversity, and providing fairness among users. In PF-RP, in each time slot, both PF and PF-SP are run logically. The selection from PF-SP is used if the user selected has a higher R_i than the user selected from PF; otherwise the user selected from PF is used. This allows us to exploit diversity across all users (with both long and short flows) while retaining differentiation for short flows. The algorithm is summarized in Figure 13.

Since PF is used except in cases when using PF-SP improves the channel utilization, PF-RP has the property that it can decrease short flow latencies while at the same time providing higher throughput. However, PF-RP algorithm sacrifices fairness to users with only long flows but that is necessary by definition in any mechanism that provides differentiation to short flows. As in the case of PF-SP, the average rate for each user is maintained over all short and long flows and provides some measure of overall fairness to users with little or no short flows.

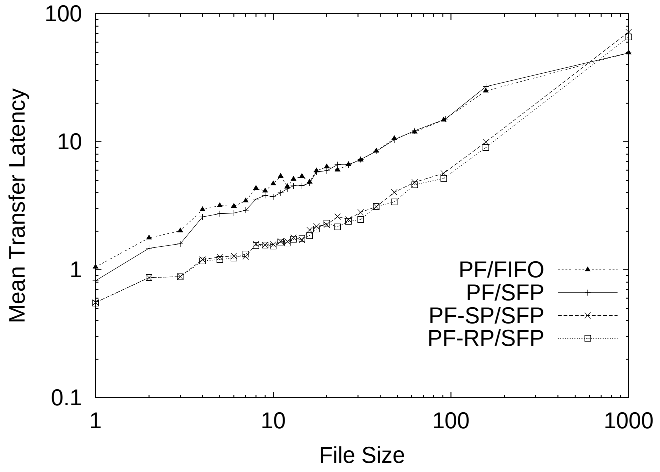
The amount of improvement provided by PF-RP depends on the channel conditions. If the user with short flow has very bad channel conditions, PF-RP will not force the transmission of packets from the short flow since it would cause too much throughput degradation. On the other hand, if the user with short flow has good channel condition, short flow latency will be reduced significantly but with the risk of unfairness to other users. However, note that the SFP algorithm at the first level would eventually promote short flows to long flows (after the threshold bytes are transferred) and thus unfairness cannot be persistent. The latency reduction for short flows in PF-RP is also a function of user diversity and improves when there are more users with short flows.

B. Performance

Using the same ns-2 simulation setup as before, we calculate the flow completion time distribution for various file sizes for



(a) Small Files Only



(b) All Files

Fig. 14. Download Latency for 4 users, Different Base SNR

one user and compare the three different inter-user scheduling mechanisms with the SFP intra-user scheduler: PF, PF-SP, and PF-RP. As a baseline, we compare these three algorithms with the standard PF inter-user/FIFO intra-user scheduler. The parameters of the web traffic model used is shown in Table I. FTP sessions are used as background traffic, and arrive at a constant interval of 200 and 80 seconds for medium and heavy load conditions respectively. The FTP file size is fixed at 1000 packets (or 1MB since packet length is 1KB). Flows are classified as short if the file size is below 15KB and the reset duration is set to 1 second. The averaging parameter α used in calculating A_i is set to 0.001.

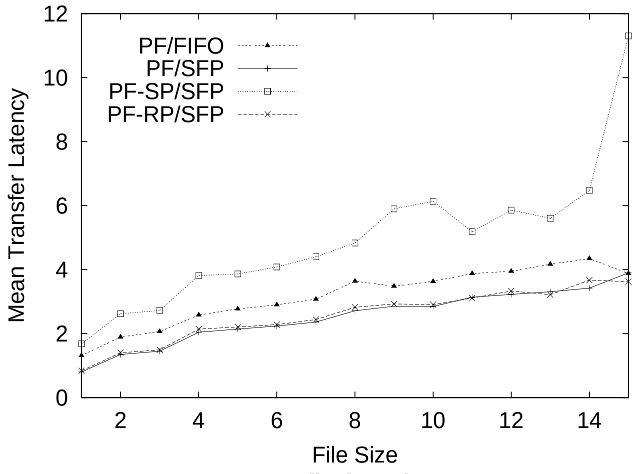
Figures 14 and 15 show a series of plots that demonstrate the strength and weaknesses of the three inter-user scheduling schemes presented in Section VI-A. Flow latencies using the PF/FIFO is provided as a basis for comparison. Figure 14 shows the results for all four scheduling algorithms but with different base SNR among users. Two users have base SNR of $+4$dB and two users have base SNR of -4dB. In addition, user 1 (with base SNR of +4dB) only has the web traffic

Web Model Elements	Attributes	Distribution and values
Web Page	Time interval between Pages (s) Number of Web objects per pages	Exponential, mean=15 Exponential, mean=5
Web Object	Time interval between Web objects (s) object size	Exponential, mean=0.01 Bounded Pareto, mean = 12, shape = 1.2, max=200

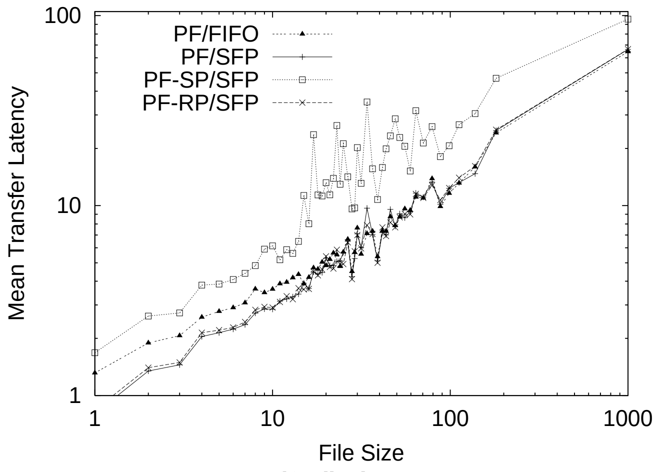
FTP Model Elements	Attributes	Distribution and values
File Size	Time interval between FTP Session (s) FTP File Size	Constant, mean=200/80 (medium/heavy traffic) Constant, mean=1000

TABLE I

TRAFFIC MODEL PARAMETERS



(a) Small Files Only



(b) All Files

Fig. 15. Download Latency for 4 users, Same Base SNR

component in Table I while users 2 to 4 have the medium load FTP traffic component.

As expected, the short flow latency of PF/FIFO has the highest latencies for short flows. PF/SFP shows latency improvement for flows under 15KB by reducing the latency by 17% and has similar performance as PF/FIFO after that. By emphasizing fairness across users over flow differentiation, PF cannot take advantage of the better channel condition of the short flow user even if doing so increases the overall channel utilization. On the other hand, both PF-SP and PF-RP have the capability to exploit the difference in channel condition and reduce short flow latencies significantly. The improvement in latency in PF-SP and PF-RP is significant and is true for all file sizes less than 200 packets and the negative impact is only felt by the 1000 packet FTP flows. The improvement of short flow latency is 54% over PF/FIFO.

Figure 15 shows the results for all four scheduling algorithms with the same base SNR of -4dB and uses the web and medium FTP load as shown in Table I. The results show surprisingly that the greedy approach of PF-SP performs the worst for all file sizes and increases both the short and long flow latencies. The average increase in short flow latency is 93% (from 1.48s to 2.86s). One explanation for this result is that PF-SP forces short flows to be scheduled immediately even when the channel condition is bad. Delaying the transmission and waiting for a better channel condition (and thus higher bandwidth) could actually help improve latency. In addition, the amount of diversity (number of candidate user to choose from) available is much smaller in PF-SP since diversity is exploited only across users with short flows when short flows are present. The average number of backlogged short flow queues, representative of user diversity, for PF-SP is 1.64. On the other hand, for PF and PF-RP, the average number of backlogged queues is 3.01. Therefore, as a result of limiting the effect of user diversity by selecting from a smaller pool of users, PF-SP performs poorly.

From the simulation results, in the wireless channel simulated, PF/SFP always performs better than the default PF/FIFO. On the other hand, depending on the overall user channel conditions, PF-SP may be better or worse than PF. For the channel conditions simulated, PF-RP is the most robust.

VII. CONCLUSION

In this paper, we make two important contributions. First, we proposed a network-based solution called the *Window Regulator* that maximizes TCP performance in the presence of channel variations for any given buffer size at the congested router. We analyzed the performance of various versions of the Window Regulator schemes and make the following observations. Window Regulator-Static (WRS), a common algorithm used in wired routers, performs poorly. The Window Regulator with ack Buffer (WRB) scheme, which explicitly adapts to the wireless channel conditions and also performs ack regulation, improves the throughput by up to 100% over a drop-tail scheme. WRB also delivers robust performance gains even with reasonably large wired latencies and small amount of packet losses.

Next, we presented a scheduling and buffer sharing algorithm that reduces the latency for short flows while exploiting user diversity, thus allowing the wireless channel to be utilized efficiently. We found that the proposed PF-RP/SFP scheme provides robust performance over different user channel conditions and improves latency up to 54% over PF/FIFO.

The open question remains as to what is the best way to balance all three parameters: diversity, fairness, and preference for short flows. Answering this question would require a better understanding of the trade-off and comparison metric. We are exploring this issue as part of future work.

REFERENCES

[1] A. Bakre and B.R. Badrinath, “Handoff and System Support for Indirect TCP/IP,” in proceedings of Second Usenix Symposium on Mobile and Location-Independent Computing, Apr 1995.

[2] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz, “Improving TCP/IP Performance over Wireless Networks,” in proceedings of ACM Mobicom, Nov 1995.

[3] P. Bender et al. , “A Bandwidth Efficient High Speed Wireless Data Service for Nomadic Users,” in IEEE Communications Magazine, Jul 2000.

[4] P. Bhagwat, P. Bhattacharya, A. Krishna, and S. Tripathi, ”Enhancing Throughput over Wireless LANs Using Channel State Dependent Packet Scheduling,” in Proc. IEEE INFOCOM’96, pp. 1133-40, March 1996.

[5] TIA/EIA/cdma2000, ”Mobile Station - Base Station Compatibility Standard for Dual-Mode Wideband Spread Spectrum Cellular Systems”, Washington: TIA, 1999.

[6] M. C. Chan and R. Ramjee, “TCP/IP Performance over 3G wireless links with rate and delay variation,” in Proceedings of ACM Mobicom’02.

[7] X. Chen and J. Heidemann, ”Preferential Treatment for Short Flows to Reduce Web Latency”, USC/ISI Technical Report ISI-TR-548 (Oct 2001) to appear in Computer Networks.

[8] L. Guo and I. Matta, ”The war between mice and elephants,” in Proceedings of ICNP’01

[9] H.Inamura et.al. , ”TCP over 2.5G and 3G Wireless Networks,” draft-ietf-pilc-2.5g3g-07, August 2002.

[10] L. Kalampoukas , A. Varma, and K. K. Ramakrishnan, “Explicit window adaptation: a method to enhance TCP performance,” IEEE/ACM Transactions on Networking,June 2002

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

[12] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden and A. Joseph, “Multi-layer Tracing of TCP over a Reliable Wireless Link,” in Proceedings of ACM SIGMETRICS 1999.

[13] R. Ludwig and R. H. Katz “The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions,” in ACM Computer Communications Review, Vol. 30, No. 1, January 2000.

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

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

[16] S. Karandikar, S. Kalyanaraman, P. Bagal, B. Packer, “TCP rate control,” ACM Computer Communication Review, Jan 2000.

[17] N. S. Joshi, S. R. Kadaba, S. Patel and G. S. Sundaram, ”Downlink Scheduling in CDMA Data Networks,” in Proceedings of Mobicom 2000.

[18] Z. Shao and U. Madhow, ”Scheduling Heavy-tailed Data Traffic over the Wireless Internet,” in Proceedings of VTC’2002.