

A Measurement Study of Scheduler-Based Attacks in 3G Wireless Networks

Soshant Bali¹, Sridhar Machiraju², Hui Zang², and Victor Frost¹

¹ University of Kansas
{sbali,frost}@ittc.ku.edu
² Sprint ATL
{Machiraju,Hui.Zang}@sprint.com

Abstract. Though high-speed (3G) wide-area wireless networks have been rapidly proliferating, little is known about the robustness and security properties of these networks. In this paper, we make initial steps towards understanding these properties by studying Proportional Fair (PF), the scheduling algorithm used on the downlinks of these networks. We find that the fairness-ensuring mechanism of PF can be easily corrupted by a malicious user to monopolize the wireless channel thereby starving other users. Using extensive experiments on commercial and laboratory-based CDMA networks, we demonstrate this vulnerability and quantify the resulting performance impact. We find that delay jitter can be increased by up to 1 second and TCP throughput can be reduced by as much as 25 – 30% by a single malicious user. Based on our results, we argue for the need to use a more robust scheduling algorithm and outline one such algorithm.

1 Introduction

Today, the mobile Internet is one of the fastest-growing segments of the Internet. One of the main reasons for this is the rapid adoption of high-speed (3G) wide-area wireless networks. The two main 3G standards are Evolution Data Optimized (EV-DO) [1] and High-Speed Downlink Packet Access (HSDPA). The increasing use of these networks to access the Internet makes it important that these are well-engineered, robust and secure. Much work has been done on designing these networks [3,7,5] and the algorithms used in them, especially, the wireless scheduling algorithms [4,11,13]. However, most prior work has focused on improving system performance assuming cooperative scenarios without malicious users.

In this paper, we make some initial steps towards understanding the important issue of 3G robustness from the security viewpoint. We study the scheduling algorithm since it plays a vital role in deciding system performance and user experience. Some prior work [9] have studied vulnerabilities associated with FIFO schedulers that are common on wired IP networks. Since most 3G networks use the Proportional Fair (PF) algorithm [7] for *downlink* scheduling, we focus on it in this paper. PF has been widely deployed because it is simple and increases system throughput by being channel-aware, i.e., it schedules data transmission

to users with good wireless conditions over those who are experiencing *fading* (bad channel conditions). Under general conditions, PF maximizes the product of throughputs received by all users [13]. Moreover, when all users have identical and independent fading characteristics, PF is known to be fair in the long term.

The main finding of our study is that the fairness-ensuring mechanism of PF can easily be corrupted by a malicious user to monopolize the wireless channel thereby starving other users. Such scheduler-based attacks are possible because PF does not distinguish between users with outstanding data and those without. Using extensive measurements on a commercial CDMA network and on a similar laboratory setup, we show that the performance degradation due to such attacks is severe - they can increase “jitter” by up to 1 second and cause frequent spurious TCP timeouts. We also show that the latter can increase flow completion times and decrease TCP goodput by up to 30%. Our findings are important not only because we tackle mechanisms used in 3G networks and possibly, other future wireless networks, but also because our work (re-)emphasizes the need to consider security while designing network algorithms.

This paper is organized as follows. Section 2 provides an overview of the PF scheduling algorithm and describe how it can be attacked. In Section 3, we conduct initial experiments on a commercial network and motivate the need to move to a more controlled experimental environment. In Section 4, we use experiments in our laboratory to quantify the impact of PF-induced attacks on UDP and TCP-based applications. We also discuss other important issues including a possible replacement for PF. We conclude and outline future work in Section 5.

2 The PF Algorithm and Starvation

As with any managed wireless network, access to the wireless channel in 3G networks is controlled by Base Stations (BSs) to which mobile devices or Access Terminals (ATs) are associated. Our focus is on Proportional Fair (PF) - the scheduling algorithm [13] used to schedule transmissions on the downlink in most 3G networks. In these networks, downlink transmission is slotted. For example, in CDMA-based EV-DO networks, slot size is 1.67ms. BSs have per-AT queues and employ PF to determine the AT to transmit to, in a time slot.

The inputs to PF are the current channel conditions reported on a per-slot basis by each AT. Specifically, each AT uses its current Signal-to-Noise Ratio (SNR) to determine the required coding rate and modulation type and hence, the *achievable rate* of downlink transmission. In the EV-DO system, there are 10 unique achievable data rates (in Kilobits per second) - 0, 38.4, 76.8, 153.6, 307.2, 614.4, 921.6, 1228.8, 1843.2 and 2457.6. Assume that there are n ATs in the system. Denote the achievable data rate reported by AT i in time slot t to be R_t^i ($i = 1 \dots n$). For each AT i , the scheduler also maintains A_t^i , an exponentially-weighted average rate that user i has achieved, i.e.,

$$A_t^i = \begin{cases} A_{t-1}^i(1 - \alpha) + \alpha R_t^i & \text{if slot allocated} \\ A_{t-1}^i(1 - \alpha) & \text{otherwise} \end{cases}$$

Slot t is allocated to the AT with the highest ratio $\frac{R_t^i}{A_{t-1}^i}$. α is usually around 0.001 [7] (we verified this using measurements [2]). Thus, an AT will be scheduled less often when it experiences fading and more often when it does not. Under general conditions, PF maximizes the product of per-AT throughputs [13]. Moreover, if all ATs have identical and independent fading characteristics, all ATs are allocated equal number of slots in the long term. If different ATs experience non-identical wireless conditions, unequal slot allocation may result [4] in the long term. Long-term fairness is not our focus here. We show how malicious ATs can cause short-term unfairness that is severe enough to degrade application performance.

The basic observation behind our work is that a malicious AT can influence the value of its $\frac{R_t}{A_{t-1}}$ ratio thereby affecting the slot allocation process. An AT can do this simply by receiving data in an *on-off* manner. To see why, consider an AT that receives no data for several slots. Its A_t would slowly reduce and approach zero. After several inactive slots, when a new packet destined for that AT arrives at the base station, that AT has a low value of A_t and is likely to get allocated the slot because its ratio is very high. This AT keeps getting allocated slots until its A_t increases enough. During this period, all other ATs are starved. A few Prior work [8] has observed excessive delays with PF. To our knowledge, no prior work has considered on-off traffic or explored how malicious users can exploit it.

Starvation due to on-off behavior occurs because PF reduces A_t during the off periods. This implies that PF “compensates” for slots that are not allocated even when an AT has no data to receive! In the rest of the paper, we study how a malicious AT (and a cooperating data source to that AT) can exploit this vulnerability, namely, the inability of PF to distinguish between an AT to which no data needs to be sent and one that is not allocated a slot.

3 Experimental Setup

We conduct our initial experiments on a commercial EV-DO network in USA. Our ATs are IBM T42 Thinkpad laptops running Windows XP equipped with commercially-available PCMCIA EV-DO cards. The laptops have 2GHz processors and 1GB of main memory. All ATs connect to the same base station and sector. Data to the ATs is sourced from Linux PCs with 2.4GHz processors and 1GB of memory. All of these PCs are on the same subnet and about 10 – 11 hops away from the ATs.

For our first experiment, we use two ATs - AT1 and AT2. AT1 receives a long-lived periodic UDP packet stream consisting of 1500-byte packets with an average rate of 600Kbps. AT2 is assigned the role of a malicious AT and hence, receives traffic in an on-off pattern from its sender. Specifically, it receives a burst of 250 packets of 1500 bytes every 6 seconds. We plot the “jitter” experienced by AT1 in Figure 1. Since our ATs are not time-synchronized with the senders, jitter is calculated as the excess one-way delays over the minimum delay. Well-defined increases in jitter are observed whenever a burst is sent to AT2. In contrast, a base station employing fair queueing would cause almost no increase in “jitter” as

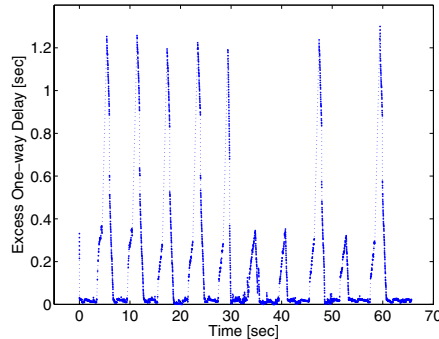


Fig. 1. “Jitter” caused by a malicious AT in a commercial EV-DO network

long as the wireless link capacity is not exceeded. These results clearly show that AT1 experiences extraordinary increase in “jitter”. We observe similar results with other parameter settings (results not shown). With all of them, however, the jitter increases vary from 300ms to 1 second. The variability is likely due to traffic to other ATs and queueing effects at other hops. Hence, to understand and quantify the attack scenarios better, we move to a more controlled laboratory setup that recreates actual network conditions while eliminating unknowns such as cross-traffic. We describe this setup now.

In our laboratory, we use commercially available equipments that are widely used in commercial networks to recreate the EV-DO network including the Base Station, the Radio Network Controller (RNC) and the Packet Data Serving Node (PDSN) (see [7]). The links between the Base Station, RNC and PDSN are 1Gbps Ethernet links. The Base Station serves 3 sectors of which we use one. Our ATs and senders are the same as before. We collect *tcpdump* [12] traces at the senders and ATs. Due to the peculiarities of PPP implementation on Windows XP, the timestamps of received packets are accurate only to 16ms. However, these inaccuracies are small enough to not affect our results. For TCP-based experiments, we use *tcptrace* [12] to analyze the sender-side traces. There are three main differences with a commercial network. First, we use lower power levels than commercial networks due to shorter distances and in the interest of our long-term health. Since our goal is not to characterize fading and PF’s vulnerability does not depend on channel characteristics, this does not affect the validity of our results. Second, we can control the number of ATs connected to the base station. Third, the number of hops from the senders to the ATs is only 3. This eliminates the additional hops on normal Internet paths and queueing effects on those hops. We discuss the impact of this in Section 4.2. Moreover, this is realistic in networks that use split-TCP or TCP-proxy [14].

Our laboratory setup poses a few challenges; we describe two of the more important challenges now. First, even though we conduct our experiments in the laboratory, the wireless conditions varied significantly. Hence, we conducted up to 30 runs of each experiment (a particular parameter setting) to calculate a

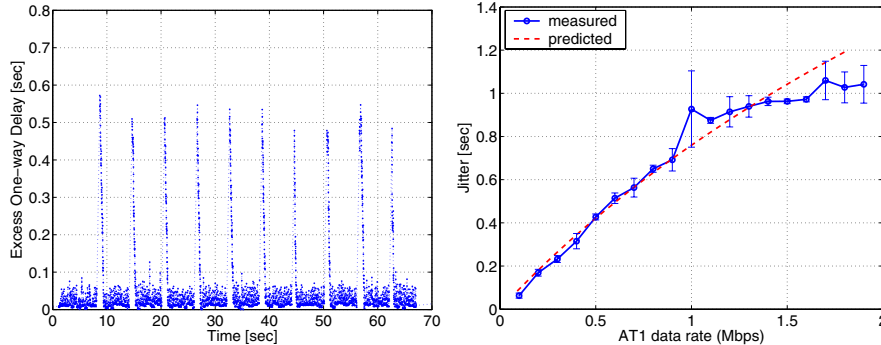


Fig. 2. (Left) Results of “jitter” experiment performed in the lab setup. We plot the excess of (unsynchronized) delays. (Right) The maximum amount of “jitter” - measured and predicted - that can be caused as a function of the data rate of the long-lived flow to AT1. As noted before, fair queuing would cause negligible “jitter” if channel capacity is not exceeded.

good estimate of the required performance metric with a small enough confidence interval. We also interleaved the runs of the different parameter settings used to plot a figure so that they all experienced the same wireless conditions on average. A second challenge is that ATs become disassociated with the base station after around 12 seconds of inactivity. Also, the initial few packets sent to an inactive AT encountered large delays due to channel setup and other control overhead. To prevent our ATs from becoming inactive, we use a low rate background data stream of negligible overhead.

4 Experimental Results

In this section, we use our laboratory setup to quantify the severity of PF-based attacks. We first focus on non-reactive UDP-based applications and then, on TCP-based applications. We conclude this section by discussing how common traffic patterns can also trigger PF-induced starvation and briefly discuss a preliminary replacement for PF.

4.1 UDP-Based Applications

In Figure 2, we plot the results of a laboratory experiment similar to that of Figure 1. We send a long-lived UDP flow of average rate 600Kbps to AT1 and bursts of 150 packets to AT2 every 6 seconds. The results mirror the behavior observed in the commercial network, namely, large “jitter” whenever a burst is sent. Notice the reduction in the variability of results due to the absence of other ATs and queuing at other hops.

Recall that the PF algorithm compares the ratios of all ATs to allocate slots. Intuitively, the “jitter” of AT1 depends on the value of A_t of AT1 just before

a burst to AT2 starts. This can be analytically derived. Due to lack of space, we skip the derivation and only provide the final expression for the “jitter” J experienced by AT1 when both ATs experience unchanging wireless conditions and hence, have constant achievable data rates $R1_t = R1$ and $R2_t = R2$ in every time slot t . We also assume that $A1_T = \beta1_T R1$ and $A2_T = \beta2_T R2$ are the moving averages for AT1 and AT2 in time slot T , the last slot before a burst to AT2 starts. Then (the derivation can be found in [2]),

$$J = \left\lceil \frac{\log\left(\frac{1}{1+\beta1_T-\beta2_T}\right)}{\log(1-\alpha)} \right\rceil \quad (1)$$

In Figure 2 (Right), we plot the predicted value of “jitter” assuming $R1 = 1.8\text{Mbps}$ and $\beta2_T = 0$. We compare this with experiments in which we vary the rate of AT1’s flow from 100Kbps to 2Mbps. For each experiment, we calculate the maximum “jitter” experienced by AT1 and plot them in Figure 2. Comparing the results of these experiments with $\beta2_T = 0$ makes sense because the bursts are separated long enough that AT2’s A_t is close to zero. It is clear that the experimental results closely follow the analytically predicted values. Also, the jitter experienced by AT1 increases almost linearly with the *entire* data rate to AT1. Thus, an AT1 with a single VoIP application of 100Kbps may experience only 100ms increase in “jitter” whereas additional concurrent web transfers by this VOIP user would cause larger “jitter”. As another example, an AT receiving a medium-rate video streams of 0.6Mbps could experience a jitter increase of more than 0.5 seconds. This can cause severe degradation in video quality.

4.2 Effect on TCP Flows

We now show that TCP-based applications are also susceptible to PF-induced attacks. We start by conducting an experiment in which we replace the UDP flow to AT1 with a long-lived TCP transfer of 20MB. As before, we send an on-off UDP stream to AT2 in which every burst consists of 150 1500-byte packets once every 3 seconds - an average rate of 600Kbps. We analyze sender-side *tcpdump* [12] traces with *tcptrace* [10] and plot the TCP sequence number of the bytes transmitted versus time of the flow to AT1 in Figure 3 (Left). The SYN packet is marked at time 0. The black dots represent transmitted packets (x-value is time of transmission and y-value is the sequence number). We see periodic retransmissions (blobs of small Rs) every 3 seconds corresponding to each burst of the flow to AT2. This demonstrates how a malicious user can easily cause TCP timeouts to other users.

TCP timeouts in the above experiment could be caused due to one of two reasons. The first reason is that AT1 is starved long enough that its buffer overflows and some packets are dropped. The second reason is that the buffer is large enough but AT1’s packets are delayed long enough that TCP experiences a *spurious timeout*. It turns out that per-AT buffers in EV-DO base stations are usually 80 to 100KB in size, which is larger than the default receiver window size of 64KB in Linux and Windows (other versions use 32KB and 16KB [6]). We also verified this using sender side *tcpdump* traces. Due to lack of space, we do not explore scenarios involving multiple flows to the same AT and timeouts caused by the resulting

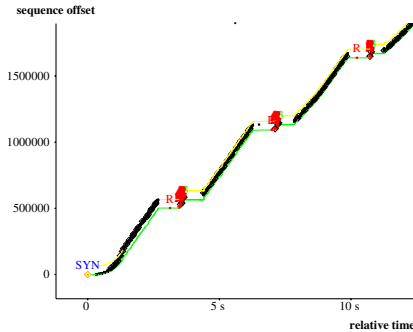


Fig. 3. Results of *tcptrace* analysis of AT1. Timeouts are caused whenever AT2 received a burst.

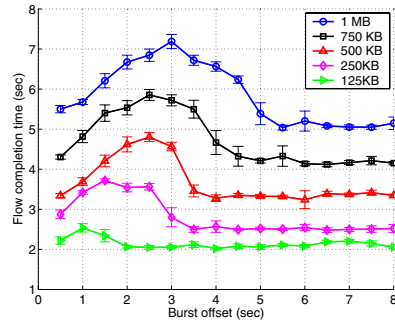


Fig. 4. Increase in flow completion time for short TCP flows. 95% confidence intervals are plotted.

packet losses. It might be argued that our laboratory setup causes more spurious timeouts because we have fewer hops than typical Internet paths. In fact, our setup reflects the common practice of wireless providers in using split-TCP or TCP proxies [14]. Moreover, as wireless speeds go up, delays are only going to decrease.

Short Flows. We now study the impact on TCP performance due to spurious timeouts caused by a malicious user. We first consider short TCP flows for which flow completion times are the suitable performance metric. We conduct experiments as before but replace the UDP flow to AT1 with TCP transfers ranging from 125KB to 1MB. Since short flows spend a significant fraction of time in slow start, A_t is likely to be small early on. Hence, the starvation duration is likely to depend on the offset of the burst from the start time of the TCP flow. To understand this better, we conduct experiments for various values of the burst offsets. For each offset and flow size, we run 30 experiments and plot the average flow completion time in Fig. 4. We make four observations. First, for a large enough offset, the burst has no impact because the TCP flow is already complete. Second, the probability of a timeout increases as the offset increases. This confirms our intuition that, during slow start, A_t of AT1 is smaller and hence, starvation duration is smaller. Maximum performance impact is realized when the offset is 2 – 3 seconds. This is observed when we plot the average number of retransmissions too (figure not shown due to lack of space). Third, the inverted-U shape shows that the probability of a timeout decreases when the burst starts towards the end of the flow. Fourth, for downloads of 250KB and above, there is a 25 – 30% increase in flow completion time. Note, however, that A_t depends on the *total* data rate to AT1. Hence, if AT1 receives other data flows simultaneously, its A_t would be larger and more timeouts may result.

Long Flows. Next, we study long-lived TCP flows for which the suitable performance measure is goodput. Consequently, we start a long TCP flow to AT1. Our malicious AT, AT2, receives on-off traffic in the form of periodic bursts. To understand how AT2 can achieve the maximum impact with minimal overhead,

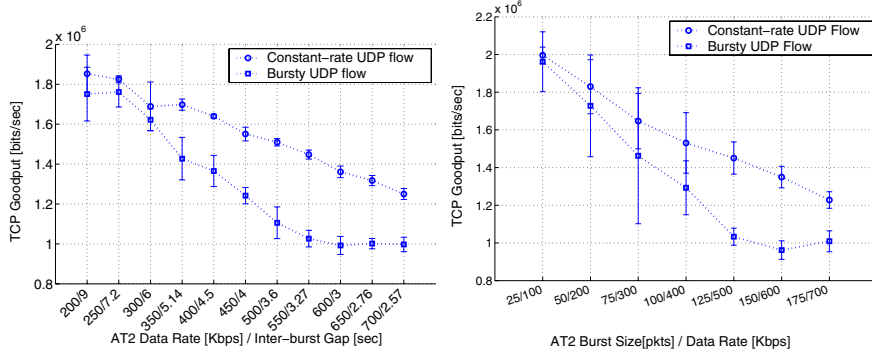


Fig. 5. Plots illustrating the reduction in TCP goodput as a function of the burst size (Left) and burst frequency (Right) of an on-off UDP flow. Note that the y-axis on the plots does not start at 0.

we conduct experiments with various burst sizes and frequencies. Since the average rate to AT2 changes based on the burst size and frequency, we cannot compare one experiment to another. Instead, we compare each experiment with an experiment in which AT2 receives a constant packet rate UDP stream of the same average rate. The TCP goodput achieved with such well-behaved traffic captures the effect of the additional load. Any further reduction in goodput that we observe with on-off UDP flows essentially captures the performance degradation due to unnecessary timeouts. We plot the average TCP goodput achieved in our experiments with on-off and well-behaved UDP flows to AT2 in Figure 5. In the Left plot, we vary the inter-burst gap for a burst size of 150 1500-byte packets. As expected, the slope of goodput with well-behaved UDP flows is almost linear with slope close to -1 . The performance impact of malicious behavior is clearly shown with the maximum reduction in goodput when the inter-burst gap is around 3 – 3.5 seconds. In this case, the goodput reduces by about 400Kbps - almost 30%. Larger gaps cause fewer timeouts and smaller gaps cause bursts to be sent before AT2's A_t has decayed to a small enough value. In the Right plot, we vary the burst size for a 3-second inter-burst gap. We find that bursts of 125 – 150 packets cause the largest reduction in goodput of about 25 – 30%.

4.3 Discussion

Starvation-driven spurious timeouts can also be triggered accidentally by benign users with typical user behavior (also, see [8]). We illustrate such a scenario using an experiment in which AT2 periodically downloads a 500KB file via TCP (to model HTTP transfers). AT1 receives a long-lived TCP flow. We plot the round-trip times and TCP sequence numbers of AT1 in Fig. 6. We see large RTT increases corresponding to the slow start phase of AT2's periodic downloads. In the TCP sequence plot (right), we see that many of these cause timeouts. For this experiment, AT2 is at a spot where it had a smaller average achievable rate

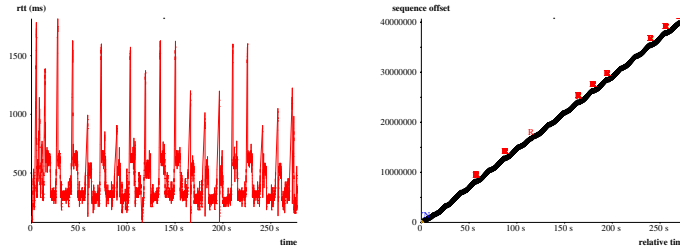


Fig. 6. Starvation caused by periodic short TCP flows to AT2 on a long-lived TCP flow to AT1. (Left) RTT over time. (Right) TCP sequence number over time.

Table 1. Performance improvement with our preliminary solution

Scenario	AT1 (TCP)	AT2	Perf. Metric	Improvement
I	long flow	periodic UDP burst	goodput	82.40%
II	short flow 1MB	periodic UDP burst	completion time	28.10%
III	short flow 500KB	periodic UDP burst	completion time	26.64%
IV	long flow	periodic TCP download 50KB	goodput	8.03%

(921.6Kbps) so that it can cause significant starvation to AT1 even when it is in the slow start phase. In future work, we intend to thoroughly explore the kinds of user behavior that can accidentally trigger starvation and to determine how frequently they occur in practice.

To rectify the vulnerability detailed in this paper, PF needs to be modified. A naive strategy is to stop updating A_t when an AT has no data to receive. This does not work because A_t needs to reflect the total number of active users *and* the recent channel conditions of that AT - both of which might change. For similar reasons, we cannot use the A_t values of ATs already receiving data either. Investigating a complete solution is out of the scope of this paper. Here we briefly discuss a promising candidate. In our proposed algorithm, we run two instances of PF - one of which assumes that all ATs always have data to receive. Actual slot allocation is governed by the second instance, which resets its A_t values with those of the first instance whenever an idle AT begins to receive data.

We explore the potential of the above solution using *ns-2* simulations of four scenarios corresponding to those explored previously and summarize the results in Table 1. In the simulation we use traces of achievable rate collected from the commercial EV-DO network. AT2 assumes the role of a malicious user in scenarios I, II, and III and of a benign user who accidentally causes AT1 to timeout in scenario IV. Since the timeout calculation in *ns-2* results in a larger value than Linux, we had to use an achievable data rate of 300Kbps for AT2 to simulate the accidental timeouts observed in Figure 6. In Table 1, we list the performance improvement, which is the decrease in completion times of a short flow or increase in TCP goodput of a long flow for AT1, of our proposed algorithm over PF in the four scenarios. We see that our solution is more robust to malicious behavior. In

fact, we verified that it eliminates virtually all spurious timeouts that PF may introduce. A detailed investigation into alternative scheduling algorithms and their performance is in the longer version of this paper [2].

5 Conclusions and Future Work

We showed that the Proportional Fair (PF) algorithm, which is used in many 3G networks, can be easily corrupted by malicious users to starve users. Using extensive measurements, we showed that such starvation can occur in deployed systems and lead to severe performance degradation including jitter of more than a second and spurious TCP timeouts. The latter can reduce TCP goodput and increase flow completion times by about 30%. As future work, we intend to extend our study to scenarios with more than two ATs, where we expect the results to follow our analytical predictions as in this paper. We briefly explored a promising algorithm to overcome PF's vulnerability. We intend to explore this more thoroughly in the future.

References

1. Telecommunications Industry Association. CDMA 2000: High Rate Packet Data Air Interface Specification (TIA-856-A), 2004.
2. S. Bali, S. Machiraju, H. Zang, and V. Frost. On the Performance Implications of Proportional Fairness (PF) in 3G Wireless Networks. Technical Report RR06-ATL-040624, Sprint ATL, 2006.
3. P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhusayana, and A. Viterbi. CDMA/HDR: A Bandwidth-efficient High-speed Wireless Data Service for Nomadic Users. *IEEE Communications Magazine*, 38:70–77, July 2000.
4. S. Borst. User-level Performance of Channel-aware Scheduling Algorithms in Wireless Data Networks. In *Proc. of IEEE INFOCOM*, 2003.
5. Mun Choon Chan and Ramachandran Ramjee. Improving TCP/IP Performance over Third Generation Wireless Networks. In *Proc. of IEEE INFOCOM*, 2004.
6. Carl Harris. Windows 2000 TCP Performance Tuning Tips <http://rdweb.cns.vt.edu/public/notes/win2k-tcpip.htm>.
7. A. Jalali, R. Padovani, and R. Pankaj. Data Throughput of CDMA-HDR: A High Efficiency-high Data Rate Personal Communication Wireless System. *Proc. of IEEE Vehicular Technology Conference*, 3:1854–1858, May 2000.
8. T. Klein, K. Leung, and H. Zheng. Enhanced Scheduling Algorithms for Improved TCP Performance in Wireless IP Networks. In *Proc. of GLOBECOM*, 2004.
9. A. Kuzmanovic and E.W. Knightly. Low-rate TCP-targeted Denial of Service Attacks: The Shrew vs. The mice and Elephants. In *Proc. of SIGCOMM*, 2003.
10. Shawn Ostermann. tcptrace, <http://jarok.cs.ohiou.edu/software/tcptrace>.
11. S. Shakkottai and A. Stolyar. Scheduling Algorithms for a Mixture of Real-time and Non-real-time Data in HDR. In *Proc. of ITC-17*, September 2001.
12. tcpdump. <http://www.tcpdump.org>.
13. P. Viswanath, D. Tse, , and R. Laroia. Opportunistic Beamforming using Dumb Antennas. *IEEE Transactions on Information Theory*, 48:1277–1294, June 2002.
14. W. Wei, C. Zhang, H. Zang, J. Kurose, and D. Towsley. Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks. In *Proc. of PAM*, 2006.