

PAQ: A Starvation-Resistant Alternative to Proportional Fair

Soshant Bali, Sridhar Machiraju, and Hui Zang
Sprint

1 Adrian Court, Burlingame, CA 94010, USA
Email: {Soshant.Bali,Machiraju,Hui.Zang}@sprint.com

Abstract—Proportional Fair (PF) is a frequently used channel-aware scheduling algorithm in 3G wireless networks. However, recent work by us and others has shown that, in practice, PF suffers from significant robustness issues that can unnecessarily starve “well-behaved” users. In this paper, we analyze these issues with the goal of developing an alternative scheduling algorithm more robust than PF. We start by identifying two scenarios in which PF can cause starvation. We analyze both scenarios and develop mechanisms that prevent such starvation. Then, we combine these mechanisms to propose our Parallel Adaptive Quantile-based (PAQ) scheduling algorithm. We use simulation experiments with synthetic and measurement-based traces of wireless channel conditions to show that PAQ is not only robust but also achieves comparable or better throughput and fairness than PF.

Index Terms—Proportional fair, quantile-based scheduler, starvation, robust scheduler

I. INTRODUCTION

Scheduling algorithms are a key component of wireless networks. Unlike in wireline networks, channel conditions in wireless networks vary with time. *Channel-aware opportunistic scheduling* algorithms exploit these variations in channel quality to improve per-user and/or system throughput. Many channel-aware algorithms have been proposed [4], [6], [11], [14], [15] in the past. Proportional Fair (PF) [8] is one such algorithm that has been widely deployed in cellular data networks, especially in many 3G networks such as CDMA-based EVDO networks [8].

Though widely used, PF can unnecessarily starve well-behaved users for significant periods of time. PF can cause starvation, for instance, when one user receives maliciously crafted “on-off” flows [2]. PF can also starve a user accidentally, for example, when another user experiences wide variation in channel quality. Our primary goal, in this paper, is to gain insights into why PF is not robust and use these insights to develop a scheduling algorithm that, unlike PF, is robust and yet, achieves comparable or better system/user throughput as well as fairness.

We start by using prior work [2], [9] and simulations to show that PF-induced starvation is due to one of two reasons - “on-off” traffic patterns and certain types of natural channel variations. For each of these, we identify the reason why PF can induce starvation and propose mechanisms to prevent it. The specific mechanisms we propose are the use of a parallel scheduler instance, which was introduced in [1], with a simple

adaptive initialization and a practical, adaptive quantile-based algorithm inspired by PF. Using extensive simulation experiments, we show that our proposed mechanisms do eliminate starvation in each of the respective scenarios. We combine these orthogonal mechanisms to construct a composite PAQ (Parallel Adaptive Quantile-based) scheduling algorithm. We use simulations to show that PAQ is superior to PF - it is robust and achieves comparable or better throughput and fairness than PF.

The PF-centric nature of our work is justified for two reasons. First, a robust alternative to PF is of great practical interest given the widespread use of PF. Second, by deconstructing and analyzing PF, we develop mechanisms that are fairly general and can be used with non-PF schedulers. Indeed, PAQ itself is obtained by combining the use of a parallel scheduler instance with a quantile-based scheduling algorithm. Previously proposed algorithms are not suitable for one or more reasons. For instance, solutions using a strict time-based threshold [12] to limit starvation do not distinguish between the starvation of a well-behaved user experiencing “good” channel conditions - an undesirable outcome - from the starvation of a user experiencing fading - a desirable outcome. Approaches using some form of delay-throughput tradeoff [14] do not prevent vulnerability (of TCP and UDP flows) to malicious “on-off” flows and, also, compromise on channel-awareness (for instance, see [13]).

The rest of this paper is organized as follows. In Section II, we briefly discuss and analyze the PF algorithm. We also introduce the two scenarios in which PF can induce starvation. In Section III, we describe our experimental methodology including our simulation setup and measurements of wireless channels. In Section IV, we analyze each of the scenarios, identify why PF can cause starvation and develop mechanisms that prevent it. We use simulation experiments to show the efficacy of our proposed mechanisms. We combine the solution mechanisms to develop our new scheduling algorithm, PAQ. We conclude and outline future work in Section V.

II. OVERVIEW OF THE PF ALGORITHM

We start by providing a brief overview of the PF algorithm. Then, we discuss the two scenarios in which user performance is severely degraded due to PF-induced starvation.

A. The PF Algorithm

We refer to end-users as Access Terminals (ATs, in short). The PF algorithm maintains per-AT queues at the base station and transmits data to ATs in time slots of fixed size, for example, 1.67ms in EVDO networks. Each AT reports its current *achievable rate* (of downlink transmission) on a per-slot basis to the base station. In the EVDO system, there are 10 unique achievable data rates [3], [8]. A naive channel-aware algorithm would allocate a slot to the user with the highest achievable rate. Since this can permanently starve users with bad channel conditions, PF does the following. Assume that there are n ATs in the system. Denote the achievable data rate reported by user i in time slot t to be $R_i[t]$ ($i = 1 \dots n$). PF maintains $A_i[t]$, the exponentially-weighted *mean achieved rate* of AT i .

$$A_i[t+1] = \begin{cases} A_i[t](1-\alpha) + \alpha R_i[t] & \text{if } t \text{ allocated to } i \\ A_i[t](1-\alpha) & \text{otherwise} \end{cases}$$

PF allocates the slot of time instant t to the AT with the highest $\frac{R_i[t]}{A_i[t]}$ ratio. Typically, α is chosen to be 0.001 [8]. This ensures that $A_i[t]$ remains roughly constant over time under stationary channel conditions. Under reasonably general conditions, PF maximizes the sum of the logarithms of per-AT throughput when the channel conditions of ATs are independent [10]. Moreover, if their channel conditions are identically distributed, PF ensures that all ATs are allocated equal number of slots in the long term.

We note that $A[t]$ can be viewed as a function of the average achievable rate as well as the fraction of slots allocated (it increases by an amount proportional to the achievable rate when allocated a slot). The latter implies the presence of a desirable “feedback loop”, namely, when an AT is not allocated a slot, the value of $A[t]$ reduces thereby making it more likely for that AT to be allocated a slot.

B. PF-induced Starvation

There are two different scenarios in which PF can starve an AT experiencing *good* conditions for a large period of time thereby causing serious performance degradation in the form of high UDP jitter, spurious TCP timeouts, etc.

1) **“On-Off” Traffic:** To explain the first scenario, we observe that PF would allocate an AT, which has a much larger $\frac{R_i[t]}{A_i[t]}$ ratio than other ATs, consecutive slots until the ratios cross-over. It turns out that an AT can have its $\frac{R_i[t]}{A_i[t]}$ ratio grow arbitrarily large simply by not receiving any traffic. To see why, note that $\frac{R_i[t]}{A_i[t]}$ is inversely proportional to $A_i[t]$ and recall that $A_i[\cdot]$ reduces (via multiplication with $1-\alpha$) when AT i is not allocated slot t *irrespective of whether AT i is backlogged or not*. Recently [2], we showed that an AT receiving “on-off” traffic could potentially starve other ATs during the beginning of each “on” period. The resulting starvation can increase jitter by up to 1 second and reduce TCP goodput (due to spurious timeouts) by up to 1 second and 30%.

In many wireless networks, ATs disconnect from the base station after a few seconds of inactivity to conserve battery power. For example, if an AT does not receive data for 12–13

seconds in an EVDO network, it is disconnected and stops reporting its achievable data rates to the base station. Upon reconnection, $A[t]$ needs to be (re-)initialized. Often, the initial value is close to 0 so that short TCP flows can experience smaller completion times without affecting long-lived flows. However, as with “on-off” traffic, a small initial value of $A[t]$ can also starve other ATs.

2) **Natural Channel Variations:** The second scenario in which PF can induce starvation occurs due to natural variations in the achievable rate (also, see [9]). For instance, consider a system with 2 ATs - AT1 and AT2. Assume that both ATs always have the same achievable rate R except from slot 0 when AT2’s achievable data rate drops to 0 due to fading. During this time period AT2 will be starved - a desirable result of PF’s channel awareness. At the same time, $A_2[t]$ converges towards 0. Hence, when AT2 stops experiencing fading at slot θ , $\frac{R_2[\theta]}{A_2[\theta]}$ will be very large and AT1 will be starved even though it is experiencing a constant achievable rate - an undesirable side-effect of PF’s channel awareness. In the next section, we use measurement-based traces of channel conditions to show that this would be a serious problem.

III. METHODOLOGY

In the rest of the paper, we propose solutions to prevent PF-induced starvation and combine them to develop a robust alternative to PF. To benchmark them, we conducted simulation experiments using the *ns-2* framework. We describe our simulation setup now. Our basic goal was to simulate a reasonable but simple approximation of the path to ATs - a wireless link (characterized by time-varying data rates) governed by PF that connects to a wired network with typical round trip times. We used a base station to which one or more ATs were associated with. Wired sources of data to the ATs were connected to the base station across a 100Mbps link. Since the wireless uplink was used only for TCP acknowledgments, we did not model the reverse link scheduler. By default, we ensured that there was a round trip time of 250ms between the source and all ATs. To reflect the operation of commercial EVDO networks, the base station split IP packets to fit into 1.67ms time slots at the current achievable data rate of an AT.

The achievable data rates of ATs in our simulations were either synthetically generated or based on measurement-based traces. The synthetically derived data rates were generated using a Matlab implementation of a wireless channel model. We assumed a reference base station and six neighboring base stations. ATs were “almost” stationary (speed < 10kmph) and always within the cell boundary of the reference base station. Their location was uniformly distributed in angle and distance from the reference base station. This is similar to the model used in prior work [8]. We modeled path loss and fast fading (Rayleigh fading using Jake’s model) to determine the signal strength at the ATs. Signals from the six neighboring base stations constituted the interference; we assumed that white noise was negligible compared to interference from the other base stations. We converted the resulting Signal Interference

Noise Ratio (SINR) time series to achievable data rates using the EVDO conversion table [3]. We also collected traces of per-slot achievable data rates of ATs in a commercial EVDO network. To collect these 30-minute long traces, we used the Qualcomm CAIT software [7] on a stationary AT and a mobile AT moving at an average speed of 40mph.

Our performance metrics were along three dimensions - throughput, fairness and starvation. Our throughput metrics were per-AT throughput and/or system throughput. We measured fairness using a simple metric - the difference in the fraction of slots allocated to backlogged ATs. We measured starvation by counting the number of TCP timeouts (if applicable) and *starvation duration*. The latter is defined as the time between two consecutive slots allocated to an AT. We examined the maximum and/or distribution of starvation durations, We estimated these metrics using multiple runs of each experiment. By default, we used 20 runs and ran each experiment for 100s. Unless specified otherwise, we used measurement-based traces of achievable rate.

IV. PROPOSED SOLUTION MECHANISMS

In this section, we identify the primary reason why PF is not robust in each of the above scenarios and develop solution mechanisms. We also use simulations to understand their robustness and whether or not this comes at a cost in terms of throughput or fairness especially compared to PF.

A. Combating “On-Off” Behavior

PF is vulnerable to “on-off” traffic primarily because it reduces $A[t]$ (by multiplying it with $1 - \alpha$) if an AT is not allocated a slot irrespective of whether it has data to receive or not. A naive solution is to freeze the value of $A[t]$ when an AT is not backlogged. However, a frozen $A[t]$ value does not adapt to changes in the number of backlogged ATs or channel conditions (see Section II-A). Hence, an AT with a recently unfrozen $A[t]$ can have a ratio that is much lower or higher than other ATs thereby causing starvation. A backlog-unaware algorithm, which always considers ATs to be backlogged, is also not desirable since it would allocate slots to ATs with no data to receive and hence, would not be work conserving.

1) **Parallel PF:** We propose the following *Parallel PF* (PPF) algorithm, which was first introduced in [1]. PPF uses a backlog-unaware scheduler instance only to remove the undue advantage an “on-off” user receives at the beginning of all possible “on” periods. A normal instance of PF drives slot allocation. An additional parallel instance of PF that assumes all ATs are backlogged executes simultaneously. We use $A_p[t]$ to refer to its $A[t]$ values. When a previously idle AT becomes backlogged, all $A[t]$ values are reset to the corresponding $A_p[t]$ values. Notice that as long as an idle AT does not become backlogged, PPF is equivalent to PF. Also, when a previously idle AT becomes backlogged, $A[t]$ values are reset to $A_p[t]$ so that differences in achieved throughput of backlogged and idle ATs are forgotten.

As discussed previously, the initial value of $A[\cdot]$ is usually very small and can cause starvation. Initializing $A[t]$ is

inherently hard because it should reflect the average channel conditions without any knowledge of past conditions. Hence, we also use an adaptive- α strategy with PPF that uses information as it becomes available. Specifically, during the slot $\tau \leq T$ after an AT becomes active, we adapt α to the desired $\frac{1}{T}$ by using $\alpha = \frac{\tau}{T}$. It is easy to show that, until time slot T , $A[t]$ would be a (simple) mean of the achieved rate which becomes the usual exponential mean from time slot T onwards. T is set to 1000 in our simulations to achieve the α value used in typical implementations (see [2]).

2) **Results:** We conducted experiment 1 of Table 1 to test if PPF is vulnerable to “on-off” traffic patterns. We recreated our laboratory-based experiments (see Fig. 1 in [2]) using two ATs - AT1 and AT2. AT1 receiving a long-lived TCP flow and AT2 received a (malicious) “on-off” UDP flow consisting of 225KB bursts sent at various inter-burst time periods. The results (Fig. 1 (Left)) clearly show that, with PPF, the TCP goodput is not affected by the “on-off” flow. In fact, by not causing the UDP flow to have low $A[t]$ values, PPF better implements channel-awareness. This is the reason why the TCP goodput with PPF is slightly higher than the goodput with PF and a constant-rate flow consisting of periodically-sent UDP packets. To measure the effectiveness of the adaptive- α strategy, we also conducted experiments with a large “off” duration of 15 seconds. $A[\cdot]$ was initialized to 0 before each “on” period. We found that all spurious timeouts were eliminated and TCP goodput improved on average by 15%.

Previously [2], we used laboratory experiments to show that web browsing sessions, which consist of short TCP flows followed by idle periods, resemble “on-off” behavior and can accidentally cause starvation. Hence, we replaced AT2 in the above experiments with 20 ATs, each of which received short-lived TCP flows once every 30 seconds on average (2 in Table 1). The short-lived TCP flows transferred between 50 and 200KB of data. We found that PPF prevented all spurious timeouts.

It is well known that PF allocates equal number of slots to ATs with independent, identically distributed channel conditions. This is easy to show using symmetry arguments [5], [15]. Similar symmetry arguments can be used to show that PPF allocates equal number of slots to backlogged ATs experiencing identical channel conditions. Due to lack of space, we do not show results confirming this.

B. Quantile-based Scheduler

The main reason why PF can starve an AT experiencing “good” conditions is its flawed use of the $\frac{R[t]}{A[t]}$ ratio as a measure of how good channel conditions are. For instance, depending on the variability in $R[t]$, the ratio can take extreme values (as well as result in unfairness [5] under heterogeneous channel conditions). We illustrate this using a simple experiment (3 in Table 1) with 2 backlogged ATs. AT1’s and AT2’s achievable rates were based on the trace of a stationary and a mobile AT respectively. We plot the inverse distribution of starvation duration of AT1 and AT2 in Fig. 1 (Middle) for the entire experiment which lasted for 1800 seconds. Since

TABLE I
VARIOUS EXPERIMENTS: THE PERCENTAGE-BASED RESULTS INDICATE IMPROVEMENT COMPARED TO PF.

	Goal	Description	Result
1	PPF with malicious "on-off" traffic	2 ATs - long-lived TCP, "on-off" UDP	No timeouts; TCP Goodput improves (Fig. 1 (Left))
2	PPF robustness to accidental starvation	1 AT with long-lived TCP flow, 20 ATs receiving short TCP flows every 30s	PPF throughput +27% (1.07Mbps to 1.36Mbps)
3	Starvation with PF and Q	2 ATs with long-lived UDP flows; achievable rate based on measurements	Q is fair (+8%); system throughput -2.5%; Fig. 1 (Middle)
4	Starvation with AQ - different β	Same as above	Fig. 1 (Right); β between 0.001 and 0.01 is ideal
5	PAQ robustness to "on-off"	2 ATs - long-lived UDP of varying rate and "on-off" UDP	Fig. 2 (Fig. 1(Left) shows results when long-lived flow is TCP)
6	PAQ	3 ATs - 2 long-lived (non-identical channel conditions), "on-off" UDP	PAQ is fair +10.9%; system throughput -4.2%

AT2's achievable rate was highly variable, so was the $\frac{R_2[t]}{A_2[t]}$ ratio. Consequently, AT2 was starved when the ratio was low, which is a desirable outcome of channel-aware scheduling. However, when AT2's ratio was high, AT1 was starved even when it experienced good channel conditions - an undesirable side-effect of PF's channel-awareness.

1) **The Q Algorithm:** A measure of channel quality that is more appropriate than $\frac{R[t]}{A[t]}$ is the quantile of $R[t]$ in the distribution of $R[\cdot]$. Indeed, quantile-based algorithms have been proposed in prior work [4], [12], [13] not only because quantile is a better metric but also because such algorithms are provably fair as long as channel conditions are independent (identical distribution is not required unlike with PF).

Quantile-based scheduling has typically been viewed as impractical because we have to maintain the entire probability distribution of $R[\cdot]$. For instance, some previously-proposed algorithms [4], [13] maintain the previous m (for some m) samples of achievable rate. We observe that, in practice, there are only a few distinct achievable data rates (10 in the case of EVDO) and hence, we need to maintain the probability estimates p_i of $P(R[\cdot] = R_i)$ for the n distinct achievable rates R_1, \dots, R_n . Moreover, by borrowing one of PF's strengths - the use of exponential averaging - these probabilities can be adaptively maintained with little state. Hence, our proposed **Q algorithm** sets $p_i[t+1]$ to be $p_i[t] \cdot (1 - \alpha)$ and adds α if $R[t] = R_i$. Since the distribution is discrete, Q chooses, as the quantile value, a number drawn *uniformly at random* between $\sum_1^{i-1} p_j[t]$ and $\sum_1^i p_j[t]$ when $R[t] = R_i$.

To determine the efficacy of Q in reducing starvation duration, we conducted the above experiment (3 in Table 1) with Q. The inverse distribution of starvation duration plotted in Fig. 1 (Middle) shows that Q reduced starvation by a significant amount. Nevertheless, AT1 continued to experience starvation over 500ms because AT2's channel conditions were not stationary; as AT2 moved towards a base station, its achievable rate steadily increased and Q consistently chose a very high quantile value.

2) **The AQ Algorithm:** Recall from Section II-A that PF has a desirable feedback loop, namely, $A[\cdot]$ reduces in value when an AT is starved. However, this is not effective in preventing starvation since the reduction is coupled to α ,

which has to be small enough that $A[t]$ is roughly constant when channel conditions *are* stationary. Hence, we propose that these two functions of α be decoupled and propose an *Adaptive Q (AQ)* algorithm to achieve robustness under non-stationary channel conditions as demonstrated above. AQ continues to use α to estimate the distribution of $R[\cdot]$. To prevent starvation, AQ maintains an exponential average (with parameter $0 < \beta < 1$) $f[t]$ of the fraction of slots *not* allocated to each AT. This is added to the current quantile value and the AT with the maximum combined value is scheduled.

We repeated the above experiment with different values of β (4 in Table 1) and plotted the results in Fig. 1 (Right). AQ succeeds in decoupling the tolerable starvation timescale ($\frac{1}{\beta}$) from the timescale of stationarity ($\frac{1}{\alpha}$). Also, we empirically observed that AQ allocates equal number of slots. The cost of robustness was a reduction in system throughput that was small enough (2.5% for $\beta = 0.1$).

C. The PAQ Algorithm

We develop an algorithm that is robust to "on-off" traffic and natural channel variations by combining the parallel instance and the AQ algorithm. We start by determining the robustness of AQ to "on-off" traffic (5 in Table 1). We move from the inverse distribution of slot-based starvation durations to the maximum packet jitter (excess delay over the minimum delay) experienced by a well-behaved AT receiving a long-lived constant-rate flow as a result of a competing burst. The jitter results for various rates are plotted in Fig. 2. Though the use of quantile values makes AQ more robust than PF, it is still vulnerable especially when the data rate is not low. This is because $f[t]$, the average fraction of slots not allocated to an AT, increases even when the AT is idle.

To solve the above problem, we propose the *Parallel Adaptive Quantile (PAQ)* algorithm. PAQ uses a parallel AQ instance that is backlog-unaware and maintains $f_p[t]$ values. The $f[t]$ values are reset to $f_p[t]$ whenever an idle AT becomes re-backlogged. The parallel instance does not need a separate set of $p_i[t]$ values. In Fig. 1 (Left) and 2, we compare the goodput of a long-lived TCP and jitter of a long-lived UDP flow respectively with PF, AQ and PAQ in the presence of "on-off" traffic. AQ controls starvation better than PF but is

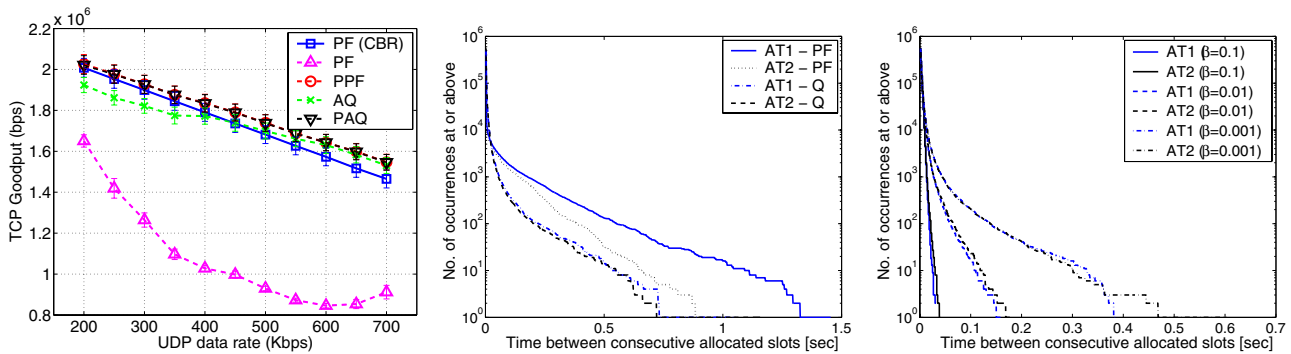


Fig. 1. (Left) TCP goodput to an AT when another AT receives “on-off” traffic at various rates. As in [2], the inter-burst times decreased from 9s to 2.57s. Goodput decrease due to PF is similar to that seen in [2] but higher due to differences in TCP timeout algorithms in *ns-2* and practical implementations. (Middle) Plot illustrating that the Q algorithm reduces starvation due to natural channel variations. (Right) Starvation experienced with AQ for various β values.

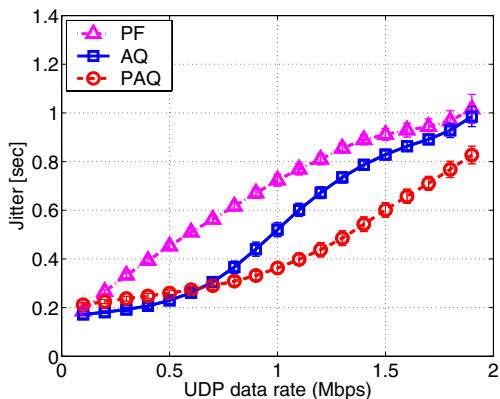


Fig. 2. Expected value of maximum jitter caused by a burst of 225 1000-byte packets on a long-lived UDP flow. AQ and PAQ perform similarly at low data rates but PAQ is more robust at higher data rates.

not perfect. PAQ, however, is fully robust.

Often, there is an inherent tradeoff involving throughput, fairness and robustness when we choose between PF and the other algorithms. To demonstrate this, we conducted experiments with 3 ATs (6 in Table 1) one of which received an “on-off” flow and the others were always backlogged. The channel conditions were not identically distributed. PF and PPF were unfair whereas AQ and PAQ were fair to the backlogged ATs with the average improvement in fairness being about 10.9%. Recall that fairness is the difference in slots allocated to backlogged ATs. The system throughput with PF was about 4–5% higher than AQ, PAQ and PPF. Thus, PAQ’s robustness and fairness come at an acceptably small cost in average system throughput.

V. CONCLUSIONS AND FUTURE WORK

We focused on starvation-related robustness issues that arise with PF in practice. We analyzed the two scenarios in which these issues arise and developed mechanisms to prevent starvation in each of them. Then, we combined them into a single Parallel Adaptive Quantile-based (PAQ) scheduling

algorithm. Using extensive simulation-based experiments, we showed that PAQ is much more robust and fair compared to PF while providing almost the same system throughput. In future work, we intend to extend PAQ to the case where multiple ATs can be scheduled in a time slot.

REFERENCES

- [1] S. Bali, S. Machiraju, and H. Zang. Beyond Proportional Fair: Designing Robust Wireless Schedulers (Extended Abstract). In *Proc. of IFIP Networking*, 2007.
- [2] S. Bali, S. Machiraju, H. Zang, and V. Frost. A Measurement Study of Scheduler-based Attacks in 3G Wireless Networks. In *Proc. of Passive and Active Measurement (PAM) Conference*, 2007.
- [3] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, 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] T. Bonald. A Score-based Opportunistic Scheduler for Fading Radio Channels. In *Proc. of European Wireless*, 2004.
- [5] S. Borst. User-level Performance of Channel-aware Scheduling Algorithms in Wireless Data Networks. In *Proc. of IEEE INFOCOM*, 2003.
- [6] S. Borst and P. Whiting. Dynamic Rate Control Algorithms for HDR Throughput Optimization. In *Proc. of IEEE INFOCOM*, 2001.
- [7] CDMA Air Interface Tester. http://www.cdmatech.com/download_library/pdf/CAIT.pdf, 2006.
- [8] A. Jalali, R. Padovani, and R. Pankaj. Data Throughput of CDMA-HDR: A High Efficiency-high Data Rate Personal Communication Wireless System. In *Proc. of IEEE VTC*, volume 3, pages 1854–1858, May 2000.
- [9] T.E. Klein, K.K. Leung, R. Parkinson, and L.G. Samuel. Avoiding Spurious TCP Timeouts in Wireless Networks by Delay Injection. In *Proc. of IEEE GLOBECOM*, 2004.
- [10] H.J. Kushner and P.A. Whiting. Convergence of Proportional-Fair Sharing Algorithms Under General Conditions. *IEEE Trans. Wireless Communications*, 3:1250–1259, July 2004.
- [11] X. Liu, E. K. P. Chong, and N. B. Shroff. A Framework for Opportunistic Scheduling in Wireless Networks. *Computer Networks*, 4:451–474, March 2003.
- [12] D. Park, H. Seo, H. Kwon, and B. G. Lee. A New Wireless Packet Scheduling Algorithm based on the CDF of User Transmission Rates. In *Proc. of IEEE GLOBECOM*, 2003.
- [13] S. Patil and G. de Veciana. Measurement-based Opportunistic Feedback and Scheduling for Wireless Systems. In *Proc. of Annual Allerton Conference on Communication, Control and Computing*, 2005.
- [14] 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.
- [15] P. Viswanath, D. Tse, and R. Laroia. Opportunistic Beamforming using Dumb Antennas. *IEEE Trans. Information Theory*, 48:1277–1294, June 2002.