# Active Queue Management For Fair Resource Allocation in Wireless Networks

Lachlan L. H. Andrew and Stephen V. Hanly

ARC Special Research Centre on Ultra-Broadband Information Networks

Department of Electrical and Electronic Engineering, University of Melbourne, Australia

l.andrew@ieee.org, hanly@ee.unimelb.edu.au

**Abstract**

This paper investigates the interaction between end-to-end flow control and MAC-layer scheduling on wireless links. We consider a wireless network with multiple users receiving information from a common access point; each user suffers fading, and a scheduler allocates the channel based on channel quality, but subject to fairness and latency considerations. We show that the fairness property of the scheduler is compromised by the transport layer flow control of TCP NewReno. We provide a receiver-side control algorithm, CLAMP, that remedies this situation. CLAMP works at a receiver to control a TCP sender by setting the TCP receiver's advertised window limit. CLAMP provides control over wireless link utilisation and queueing delay at the access point buffer, and it provides the scheduler control over the allocation of bandwidth between the users. The paper shows that physical and link layer effects have a strong impact on transport layer performance, and vice versa, both for TCP and for CLAMP-controlled TCP.

**Index Terms**

wireless communications, wireless networks, TCP, Transmission Control Protocol, active queue management, multiuser diversity, scheduling, flow control, access networks.

This work was supported by the Australian Research Council grant DP0557611.

# I. Introduction

TCP Reno (and its variants) is the dominant transport layer (layer 4) protocol for data transfers in the internet. In the last few years, increasing attention has been drawn to the performance of TCP across wireless networks, and in the present paper we focus on the interaction between TCP, which attempts to fill and overflow network buffers, and a lower-layer wireless scheduler that is trying to maximize throughput subject to fairness constraints. The model we use is applicable to fading channels in which the scheduler can exploit multi-user diversity [1]. We undertake this study by comparing TCP throughput with an alternative flow control algorithm that clamps TCP's bandwidth probing mechanism.

## A. Motivation

In this study, we investigate the potential impact of TCP flow control on a lower-layer scheduler, in terms of the fairness and throughput it can provide. Although it is already well known that TCP is unfair toward flows with long propagation delays, the studies that have concluded this fact, and provided analysis to support it, have focused on wireline networks. Wireless networks may be different for a number of reasons:

Firstly, in wireless networks, fairness is typically handled at a lower layer than TCP. The recent trend has been towards scheduled service at the wireless access points, where the traditional FCFS queue is replaced by a set of queues (perhaps per-receiver queues) with a scheduler allocating capacity between the different streams. A motivation for this approach is multi-user diversity: it is well known that schedulers can take advantage of channel knowledge to schedule users that have good channel conditions, and queue the packets of users who are in bad channel states until they can be re-scheduled in more favourable channel states [1].

Secondly, channel rates cannot be treated as deterministic quantities, and there is interaction between TCP and the link and MAC layers, where link rate adaptation and scheduling both respond to fluctuating channel conditions. TCP responds to buffer overflows which may be caused by these fluctuations.

Thirdly, queueing dynamics become important; significant queueing is required to average out the lower layer fluctuations, and the amount of buffering required impacts TCP's performance. It is necessary to take account of these issues in models of TCP performance.

In the present paper, we study the performance of TCP over a wireless network, using a model rich enough to incorporate the above features. The terminals are mobile receivers, downloading information from servers located anywhere in the Internet via a common access point (AP). The access point must schedule the packets as they arrive, but take into consideration the current channel conditions of each link, which are fading due to the mobility.

We assume in this paper that the primary cause of interaction between TCP and the lower layers is TCP's fluctuating window size. Since wireless channels are inherently random, we consider an alternative mechanism for window control that leads to fairly static window sizes (in equilibrium), with the window sizes determined by *average quantities* which are measured at the AP. This alternative control algorithm is itself a contribution of the present paper, and it is motivated by the large body of theoretical work on flow control for the Internet based on the concept of an explicit congestion price signal [2]. Another viewpoint of the results of the present paper is that it provides a study of the potential impact of flow control based on prices explicitly computed by the nodes, in the context of wireless networks with the above properties.

We were also motivated by some recent works that use the receiver-side advertised window (*AWND*) as a control mechanism ([3], [4]), rather than attempting to change the way the *CWND* is calculated at the sender. A benefit is that the modification can be implemented entirely within the wireless access network, without the need for widespread changes to routers and servers throughout the Internet.

### *B. Approach*

In this paper, we propose a novel flow control mechanism that we use to study the benefit of clamping the TCP window. This is an algorithm, CLAMP, that controls the receiver *AWND* (advertised window) based on feedback from the access point (AP). We consider its operation at the receiver side, in conjunction with TCP NewReno at the sender-side, and compare its performance against that of TCP NewReno without CLAMP.

The first step is to describe the clamping algorithm. This algorithm runs at the mobile device (the receiver), and acknowledgements are sent back to the sender containing the new values of *AWND*, as calculated by CLAMP. Since all versions of TCP interpret the *AWND* as an upper limit on the allowable window size, a mechanism to avoid overflow of the receiver buffer, this provides an effective method of control, provided that the *AWND* value is smaller than the *CWND* value calculated by the sender.

To provide a simple context in which to introduce CLAMP, we begin in Section III-B with a fluid model of a deterministic link. In this model, a fixed rate link is shared between a number of flows, and CLAMP is the agent for flow control. This model is simple enough to admit an analytical treatment, and provides insight into the design of CLAMP, the meaning of the various parameters, and the effect they have on the equilibrium achieved by the algorithm.

In Section V, we describe the details of the wireless network scenario that is the main focus of the present paper. A number of TCP flows are studied, which are assumed to be bottlenecked at the AP. Packet loss can occur from buffer overflow at the AP, or from packet loss across the wireless link. The AP implements multiuser diversity, using per-receiver queues, and the scheduler selects which queue to service. We have built a comprehensive simulator that models the interactions among the first four protocol layers (physical, link, network and transport). At layer 4, TCP NewReno is used at the sender-side, and the receivers have the option of using CLAMP to suppress the window oscillations of NewReno. We consider experiments in which either CLAMP is turned off in all mobiles, or CLAMP is switched on in all mobiles.

The conclusions of this study are presented in Section VI. Our main conclusions are that clamping the TCP window oscillations in the way described in this paper allows the lower-layer scheduler to achieve its aims, in terms of fairness between users. It also provides fairness between flows that share the same queue, irrespective of their round trip times. These are flows that the scheduler cannot distinguish. Thus, the well known unfairness of TCP toward flows that have long propagation delays is corrected by the clamping mechanism. CLAMPing the window in the way described in this paper also provides the AP some control over the queueing delays in its buffers.

## II. Related work

Most work on TCP over wireless channels has focused on the issue of packet loss, its detrimental effect on TCP [5], [6], and mechanisms at layer 1 and 2 to reduce packet loss rates [7]–[12]. Retransmissions can also be handled at layer 4 [13]. The net effect of these mechanisms is to provide a low packet loss rate at the expense of variable packet transmission times [9], [14]. Recent papers have focused on the rate and delay variation that results from a reliable link layer. Packet level burstiness can lead to packet losses at the access point buffers, and the mechanisms of delayed

ACKs [15], ACK regulation [16], [17], [18] attempt to overcome this problem. We also address this problem with CLAMP in the context of a cellular, fading scenario.

Several recent papers have studied the interaction between TCP and the MAC layer of the wireless LAN (IEEE 802.11) [19], [20]. Fairness is considered, and a recent paper [21] has highlighted potential instabilities that may result from the interaction between layers. In an experiment-based paper analyzing TCP over IEEE 802.11 ad-hoc networks, [22] shows the benefit of “clamping” the congestion window to a value that depends on the number of hops in the network. Recent work to reduce the contention at the MAC layer by generalizing the concept of delayed ACKs is reported in [23].

The 802.11 family of standards do not include channel-based scheduling, as will occur in cellular networks. The papers [16] and [17] consider channel-state based schedulers, as we do in the present paper. Other papers have considered the interactions between TCP and the MAC layer of cellular systems [24], [25], [26].

Another research thrust is to make TCP more resilient to wireless loss by distinguishing between wireless and congestion loss [27], [28]. A new sender-side protocol called “TPC Veno” tries to make this distinction and only halve the window on a congestion-related loss [29]. Recent proposals that modify the TCP sender-side also include TCP-Westwood [30], [31], and TCP-Jersey [32], which both involve bandwidth estimation at the sender to adjust the transmission rate after detected congestion; [32] also uses packet marking at the wireless routers to signify incipient congestion, so as to distinguish wireless packet loss from network congestion related loss. Other works include TCP-Probing [33], which suspends data transfer and uses a probing technique to ride out a lossy period, TCP Santa-Cruz [34], which adapts the *CWND* based on relative delays signalled in the acks from the receiver, and “wave and wait” [35], which sacrifices throughput, to avoid unnecessary losses and hence wasted energy.

It is also possible to consider tighter cross-layer coupling, in which the various layers try to collectively solve a joint optimization problem. There have been many recent papers in this area, see [36], and the references therein. In particular, we mention the rate based approaches in [37], [38], which base the layer 4 congestion signal on the queue size, as we do in the present paper.

There have been many recent papers concerning TCP over multi-hop wireless mesh networks [39], [40], [22], [18], [23]. This is an interesting direction for future work, but in the present paper, we address the fairness and stability issues associated with TCP in the context of a single hop wide-area wireless network, with reliable link layer and multi-user (variable rate) scheduling. In particular, we investigate the feasibility of a receiver-side enhancement to TCP to improve throughput and fairness in this scenario.

A precursor to the CLAMP protocol was proposed in [41], and it was later implemented in a GPRS performance enhancing proxy, with promising results, as reported in [42]. The performance of CLAMP over a single time-varying channel was studied in [43]. The present paper extends these results to multi-user wireless scheduling, with a more realistic model of wireless fading, and link rate adaptation. Other papers using the *AWND* to control the sender include Freeze-TCP [44] (for wireless), and [3], [4].

## III. System Topology and the CLAMPing algorithm

In this section, we describe the system topology that we study in this paper, and motivate and introduce a window flow control algorithm for this scenario that, in contrast to TCP, does not react to loss and does not lead to wide fluctuations in the window.

### *A. Topology and Notation*

The access network topology of interest is illustrated in Fig. 1. In the example scenarios studied in this paper, the access point maintains a separate queue for each user, to allow channel-state-aware scheduling between users, but users may be involved in multiple TCP sessions which then share the same queue, as is the case for user 1 in the figure. Alternatively, the access point may use only a single queue for all users (not depicted, but relevant for wireline applications, and for many existing wireless networks, such as 802.11a,b).

The service rate of a queue, $\mu_c$ bytes/sec, depends on the channel statistics of the users, and the scheduling policy at the wireless access point. Referring to Fig. 1, each flow, $i$, has a source node, $S_i$, and an associated round trip transmission delay, which includes all propagation and queueing delays, except for the queueing delay at the access router.

Each sending node implements sliding window flow control. Under the assumption that sources are greedy, the total number of packets and acknowledgements for flow $i$, in flight at any time, $t$, is equal to the window size, $w_i(t)$. The CLAMP algorithm selects $w_i(t)$ in a decentralized way, such

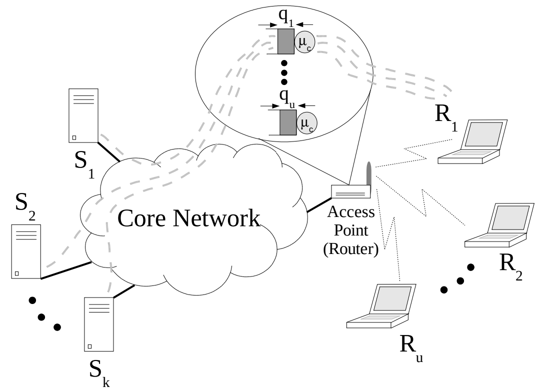



Fig. 1. System topology.

that each flow sharing the same queue obtains a proportional share of the service rate, $\mu_c$, and the equilibrium buffer occupancy of the queue, $q(t)$, can be controlled as described below.

### *B. Controlling the window: a fluid model*

Our aim is to introduce a model of the queueing at the AP, and a window control algorithm that responds to the queue size at the AP rather than buffer overflow events. The algorithm we propose has a similar objective to TCP Vegas: try and store a small number of bytes per flow at the bottleneck, *e.g.* $\tau$ bytes per flow. The aim is to avoid interaction between layer $4$ flow control and the layer $2$ wireless scheduler.

Let us focus for now on a single queue, and assume it receives a deterministic service rate of $\mu_c$ bytes/sec. In a wireless scenario, there could be any number of queues ($u$ queues depicted in Figure 1; here, we are focusing on just one of them, and assuming that the scheduler offers it a rate $\mu_c$ bytes/sec when it is backlogged). Assume that there is a single flow, and it has a propagation delay of $d$ seconds, so its bandwidth delay product is $\mu_c d$ bytes, excluding queueing delay at the AP. An alternative to TCP congestion-avoidance would be the following control, which we express in the form of a differential equation:

$$\begin{aligned} \frac{dw}{dt} &= \frac{1}{d}(\tau - bq(t)) \qquad (1)\\ q(t) &= \begin{cases} 0 & \text{if } w(t) < \mu_c d \\ w(t) - \mu_c d & \text{o.w.} \end{cases} \end{aligned}$$

where $b$ is a dimensionless constant, and $q(t)$ is the amount of the flow (in bytes) stored at the AP (a continuous-valued quantitied since this is a fluid model).

The algorithm (1) will increase $w(t)$ until $q(t)$ is nonzero (in Section IV we will introduce a slow-start mechanism to speed up this initial increase); when $q(t) > 0$ the link output rate is a constant $\mu_c$ bytes/sec. Since we are assuming window flow control (which implies self clocking occurs at the sender-side) the input rate to the queue must also be $\mu_c$ bytes/sec plus the rate of change of the window (positive or negative) that occured $d$ seconds earlier. When $q(t) > 0$, the only effect of the changing window size is to change the amount of fluid, $q(t)$, buffered in the queue; it will not affect the rate, $\mu_c$, allocated by the scheduler. The point of the flow control is to keep the "pipe" full, and maintain a target $\tau$ bytes in the queue, to avoid interaction with the scheduler.

In this paper, we like to think of the control as occuring at the receiver, so $w(t)$ is the amount of flow outstanding ("in the pipe") at time $t$ from the receiver's point of view. Thus the input rate to the queue at time $t$ is $\mu_c + \frac{dw}{dt}|_{t-d}$, where the time-lag $d$ reflects the time it takes for the change in the window to be seen in the arrival rate at the queue. Finally, since the rate of change of the queue is the difference between input and output rates, it follows that the queue size obeys the delayed differential equation:

$$\frac{dq}{dt} = \frac{1}{d}(\tau - bq(t-d)) \tag{2}$$

during periods when the queue is nonzero.

The above model is highly simplified, but it contains the basic form of the flow control we will define later. The assumption of a determinsitic service rate, $\mu_c$, is not realistic for wireless links, but we ignore this issue for now. The assumption that the flow can be modeled by a fluid, and that the window can vary in a continuous manner, is clearly highly idealized, and we will relax all these assumptions in the precise definition of the CLAMP protocol in Section IV.

For now, let us motivate the above flow control algorithm with the following observations. Firstly, if $q(t)$ is to converge, it will converge to the unique equilibrium

$$q(t) = \frac{\tau}{b} \tag{3}$$

The question of stability is resolved in the following theorem:

*Theorem 1:* The necessary and sufficient condition for the total queue size, $q(t)$, to converge under (1) is:

$$b < \frac{\pi}{2}. \tag{4}$$

*Proof:* Taking the unilateral Laplace transform of (2) gives

$$Q(s) = \frac{q(0^-)}{s + (b/d)e^{-ds}}. \tag{5}$$

The (infinite number of) poles of (2) determine the limiting behaviour of (1). For $d = 0$ the system has a single real pole located at $s = -\infty$, hence is stable for all $b \geq 0$. For $b \geq 0$ as $d$ is increased, an infinite number of poles will appear from infinity on the left half plane, and ultimately cross the imaginary axis. Applying the method shown in [45, p 26], we search for potential points where the poles cross the imaginary axis by solving:

$$W(\omega^2) = \omega^2 - (b/d)^2 = 0,$$

for $\omega^2$. The solution $\omega^2 = (b/d)^2$ indicates that the poles cross the imaginary at $s = \pm j(b/d)$. Equating the characteristic equation of (5) to zero and substituting in $\omega = b/d$ reveals that the first time two poles cross into the positive half of the imaginary axis occurs when $d = (\pi d)/(2b)$, i.e., the system will be stable for $b < \pi/2$. Note that if (2) is stable, then asymptotic stability of (1) follows, since $q(t)$ will not drop to zero if we start sufficiently close to equilibrium. ■

Note from (1) that $b$ is a "reactivity" parameter, that determines how quickly the control responds to the measured queue size. If it is too large, then the system is unstable, but if it is too small, then by (3), the queue size in equilibrium can be very large. We conclude that $b = 1$ is a safe choice. In this case, the system is stable, and the equilibrium queue size is $\tau$ bytes.

One oversimplification of this model is that it assumes per-flow queueing, when in fact several flows may share the same queue, as is the case for queue $1$ in Figure 1. For example, in our simple wireless scenario, several flows may be destined for the same mobile device, but the scheduler may not be able to distinguish these flows. In more complex multi-hop network scenarios, per-receiver queueing may be preferable to per flow queueing, so it is important to allow this possibility. In this case, our flow control algorithm must provide fair allocation to the flows sharing a queue, as the scheduler is unable to distinguish them.

Let us modify the basic flow control equation (1) to allow more than one flow to share the queue. In the following, we write the window evolution equation for one flow, $i$, that shares the queue:

$$\frac{dw_i}{dt} = \frac{1}{d_i}(\tau - p(q(t))\mu_i(t)) \tag{6}$$

where we have replaced $bq/\mu_c$ with the function

$$p(q) = \frac{bq - a}{\mu_c} \tag{7}$$

and $\mu_c$ with the rate, $\mu_i(t)$, achieved by flow $i$ at time $t$. In (7) the parameter $a$ is a constant, in units of bytes, that allows more flexibility in controlling the equilibrium queue size, as will be seen. Note that $\tau$ can be interpreted as an additive increase term, and $p(q(t))\mu_i(t)$ can be interpreted as a multiplicative decrease term, given that the decrease is proportional to the rate achieved by the flow.

This window update rule incorporates two important principles of fair, stable flow control. It is well known [46] that a simple "additive increase, multiplicative decrease" (AIMD) window update rule produces fair bandwidth allocation. This suggests the rate at which the window is updated should be the sum of a positive constant and a negative term proportional to the flow's rate. It has also been recently shown [47] that the update rate should be inversely proportional to the flow's round trip time in order for the flow control system to be stable. The algorithm above incorporates both of these principles, and fairness and stability are indeed achieved, as we show in this paper.

Using $k$ to denote the number of flows sharing the queue, and applying the same reasoning as for the single flow system, we obtain the delayed differential equation for the queue size:

$$\begin{aligned} \frac{dq}{dt} &= \sum_{i=1}^{k} \frac{dw_i}{dt}|_{t-d_i} && (8) \\ &= \sum_{i=1}^{k} \frac{1}{d_i}\left(\tau - (bq(t-d_i) - a)\frac{\mu_i}{\mu_c}\right) && (9) \end{aligned}$$

To get insight into the behaviour of this system of equations, note first that since we are assuming the queue does not empty (this is a stability issue that we address later) the following flow conservation law must hold:

$$\sum_{i=1}^{k} \mu_i(t) = \mu_c. \tag{10}$$

If the system is stable *i.e.* $q(t)$ converges to a constant, $q^*$, and each window size $w_i(t)$ converges to a corresponding constant, $w_i^*$, then it follows from (6) that at equilibrium $\mu_i(t) = \frac{\tau}{p(q^*)}$. Since this is a constant, independent of $i$, it follows from (10) that $\mu_i(t)$ must converge to $\frac{\mu_c}{k}$, and thus the system is fair, in the sense that it allocates an equal fraction of the total service rate to each flow. Assuming the system is at equilibrium, we obtain from (6) and (7) that

$$q^* = \frac{k\tau + a}{b} \tag{11}$$

The question of stability is resolved in the case of equal delays (*i.e.* $d_i = d$ for all $i$) by noticing that in this case (9) reduces to the simpler form:

$$\frac{dq}{dt} = \frac{1}{d}(k\tau + a - bq(t-d)) \tag{12}$$

(using (10) and this becomes identical to (2), if we replace the constant $k\tau + a$ with $\tau$. Since the value of this constant did not enter the stability analysis, Theorem 1 continues to apply, and we obtain the stability condition that $b < \frac{\pi}{2}$.

We do not have a complete analysis of stability in the case of unequal delays, but the numerical evidence we have obtained is that the system is stable when $b < \frac{\pi}{2}$. Preliminary analysis in the case of unequal delays can be found in [48]. The results we present in this paper, for the CLAMP protocol (as defined in Section IV), also support the hypothesis that the underlying equations are stable.

Stability justifies our assumption that the output rate of the queue is fixed at $\mu_c$ bytes/sec. Provided we start the system sufficiently close to equilibrium, the scheduler will always see a backlogged queue, and be able to allocate it a constant rate of $\mu_c$ bytes/sec. Although the output rate is constant, its *composition* in terms of the individual flow rates is time-varying, and converging to an equal share of the output rate.

One can interpret the value $p(q)$ in equation (7) as a *congestion price signal* that is used by the flow control equations (6) to adjust the window size. Note that it will be the AP that signals this price to the mobiles, and in this context the AP only needs to know $q(t)$ and the total rate $\mu_c$ (both of which it can measure, after suitable averaging to smooth out fluctuations) and the constants $a$ and $b$.

In the protocol implementation of CLAMP in Section IV, $\tau$ will be a parameter that is used in the mobile receiver, to calculate the value of *AWND*, whilst $a$ is a parameter that is used in the AP to compute a congestion price signal. We will use a fixed value of $\tau = 500$ bytes in all scenarios in the present paper. The parameter $a$ provides a tunable parameter to the AP to help target a suitable equilibrium queue size. We will illustrate the effectiveness of this control knob in Section V-E, and why it is particularly useful at a wireless AP. The viewpoint here is that $a$ is free to be chosen by the AP but it is a fixed parameter for the system once that choice has been made (it is possible that it can be varied on a slow timescale in response to changing network conditions, but we have not yet investigated this approach). Finally, as noted earlier, $b = 1$ is a safe choice, and one that we make for the remainder of this paper.

A disadvantage of the proposed method of control is that the equilibrium queue size, $q^*$, depends on the number of flows, k; indeed there is linear growth with respect to $k$, as given in (11), and the constant $\tau$ provides the rate of growth with $k$. It is an open research problem to devise a control signal that allows the AP to have complete control over the equilibrium queue size, without suffering this growth with the number of flows that share the same queue. In the context of the present paper, *i.e.* a single-hope wireless network from AP to mobiles, this is not a major problem. There will not usually be many flows in the same receiver-queue, and in any case, the AP can use the parameter $a$ to offset the effect of $k\tau$. For example, if a single flow requires significant queueing (to average out the channel fluctuations) then the value of $a$ may dominate the value of $k\tau$, for small values of $k$.

We propose to implement (6) in the mobile receivers, and this requires them to measure the rate in bytes/sec ($\mu_i$ for flow $i$) that they are receiving, and also that they receive the congestion price signal $p(q(t))$ from the AP. In a wireless scenario, it will be necessary to perform some averaging on the measured received rate to remove statistical fluctuations not accounted for above. It also appears that the mobile needs a precise measurement of each flow's propagation delay, which might not be known at the receiver. However, we remark that the normalization by $d_i$ in (6) is only a gain parameter, needed to ensure stability, and its precise value does not affect the equilibrium achieved. In practice, one can replace $d_i$ by the estimate $\frac{w_i(t)}{\mu_i(t)}$ (seconds), and this is what we use in the definition of the CLAMP protocol in Section IV.

All bets are off in the more realistic case that the scheduler offers a random and time-varying rate to each queue. In this case, one cannot assume that the output rates are constant, and nor that the queues will not empty. It is then hard to avoid interaction between layer $4$ flow control and layer $2$ scheduling. Unfortunately, realistic scenarios are hard to treat analytically, and for this reason we resort to simulation experiments in Section V.

# IV. The CLAMP Protocol

In Section V, we will implement the window clamping algorithm as a receiver-side modification to TCP, operating over a simulated wireless network. To do so, we need to recast the algorithm in terms of discrete packets, which arrive at possibly random time instants, and which are controlled at the sender by TCP NewReno. In order to implement CLAMP, the access network should be modified by inserting a software agent in both the AP and the mobile client, as described below.

## A. Access Router Agent

The software agent in the access router samples each user's queue length, and computes an exponential moving average, $q$ (a separate value for each queue). Precisely, when the $n$th packet arrives in the queue, the moving average $q(n)$ is computed via:

$$q(n) = \exp(-2I)q(n-1) + (1 - \exp(-2I))Q(n)$$

where $I$ is the interval of time between packet arrivals at the AP buffer, and $Q(n)$ is the current queue size at packet arrival $n$.

The scheduler also needs to know (or measure) the average rate in bytes per second that it could allocate to each user, if it had an unlimited supply of each users' packets *i.e.* if the AP were always backlogged with packets from all users. In the following, we consider a particular queue, with average (potential) rate $\mu_c$ bytes/sec for that user.

Given $q$, it evaluates the following price:

$$p_s(q) = \max((q-a)/\mu_c\,, 0), \tag{13}$$

which is the same as that described in equation (7), apart from an explicit non-negativity constraint, now included. This non-negativity constraint has no effect on the equilibrium calculated previously; the parameter $a$ (in bytes) still controls the equilibrium mean queue size, $q^*$, exactly as it did in Section III-B. We have found that performance of the packet based protocol is improved if non-negativity of the price is enforced. This is particularly so for the simulations of a wireless network in Section V, where the queue sizes and channel rates do fluctuate, and the price could otherwise go negative, resulting in "multiplicative increase". This was not an issue for the stability analysis of the fluid flow model, so for simplicity the non-negativity was not included there.

Each packet that is sent out contains $p(q)$, where $q$ is the current value of the moving average when the packet is sent.

### *B. Client Side Agent: window update algorithm*

The client agent's function is to receive the router's congestion measure, $p$, and set the receiver's TCP *AWND* value according to the algorithm described below, hence clamping the sender's TCP transmission window.

For simplicity, the algorithm will be described for a single flow. Let $t_k$ denote the time instant when the $k$th packet, of size $s_k$ bytes, is received by the receiving client. CLAMP calculates the change in window size (in bytes [49]) as

$$\Delta w(t_k) = \left[ \frac{\tau - p(q(t_k))\hat{\mu}(t_k)}{\hat{d}(t_k)} \right] (t_k - t_{k-1}) \tag{14}$$

where $\hat{\mu}$ is an estimate of the average received rate of the flow in bytes per second, $\hat{d}(t_k)$ is given below, and $\tau > 0$ (bytes) is a constant for all flows. Note that this is a discrete approximation to the fluid equation (6).

The factor in brackets is the rate of update of the window. The factor $\hat{d}(t_k)$ is a coarse approximation of the round trip time, which ensures the control loop has a gain with the appropriate scaling behaviour. In Section III-B, we showed that the fluid equations were stable with this scaling of the gain. In [48], we showed that without this scaling, the equations can be unstable in some circumstances.

The current received rate, $\hat{\mu}$, is estimated using a sliding window averaging function,

$$\hat{\mu}(t_k) = \frac{\sum_{i=k-\alpha}^{k} s_i}{t_k - t_{k-\alpha}}, \tag{15}$$

and

$$\hat{d}(t_k) = (1-\beta)\hat{d}(t_{k-1}) + \beta \frac{\text{AWND}(t_k)}{\hat{\mu}(t_k)}, \tag{16}$$

where the integer $\alpha$ and $\beta \in (0,1)$ are smoothing factors, and $\text{AWND}(t_k)$ is the actual advertised window (see below). These estimators were chosen for their simplicity; more sophisticated estimators may further improve the effectiveness of the algorithm. In the experiments in this paper, we use the values $\alpha = 1000$ and $\beta = 0.001$.

In equilibrium, $\hat{d}$ will be the RTT of the flow in seconds, i.e., propagation plus queueing delay. However, computing $\hat{d}(t_k)$ clearly does not require an explicit measurement of such a delay, which is an important consideration given that the algorithm must operate during the transient period. Before updating the window size, $w$, $\Delta w$ is clipped to a maximum value of $10$ kbytes, to avoid sudden increases after idle periods. It is also clipped below; the window cannot decrease by more than the number of bytes in the received packet to ensure that the right edge of the window is monotonic non-decreasing [49, p. 42]. Note that for notational ease we have omitted the index of the flow in the above notation. However, each flow will maintain its own $w$, $\hat{\mu}$, and $\hat{d}$.

The *AWND* value advertised to the sender is then

$$\text{AWND}(t_k) \equiv \min(w(t_k), \text{AWND}), \tag{17}$$

where the *AWND* on the right hand side of the assignment is the value received by the client side agent from the receiver's operating system. The assigned value on the left hand side is the one to be used in (16).

### *C. Receiver-side Slow Start*

It is important for CLAMP to allow TCP slow-start to take place at the start of a connection to allow TCP to fill the network pipe. We propose that CLAMP mimic TCP slowstart at the beginning of a connection, so as to find a reasonable estimate of the bandwidth delay product. TCP starts with a window size of one, and during its "slow start" phase, it increases its window by one segment for each acknowledgment received, doubling the window every RTT. Slow start terminates when buffer overflow causes a packet loss, and the window size is then halved.

To track this, a new CLAMP connection starts with $w(0) = MSS$, where $MSS$ is the connection's maximum segment size, and for each packet received, it sets

$$w(t_k) \leftarrow w(t_k) + MSS,$$

instead of using (14). CLAMP stops tracking slow start when $\Delta w(t_k) < 0$ in (14) or three duplicate acknowledgments are received, indicating packet loss. The receiver then sets $w(t_{k+1}) \leftarrow w(t_k)/2 \approx w(t_k - \text{RTT})$. The reason for this is that the observations at time $t_k$, which indicate congestion, reflect the *AWND* size one round trip time earlier. This terminates CLAMP's slow-start in a very similar way to the sender's TCP slowstart. The only difference is that it is slightly more conservative in that the condition $\Delta w(t_k) < 0$ can terminate slow-start, in addition to duplicate acknowledgements.

It is necessary to place an upper limit on the growth of the receiver window during slow-start, to counter the possibility that the sender-side slow-start terminates prior to the pipe filling up (due to a small value of the sender-side *ssthresh*). Therefore, in practice, a receiver-side threshold would also need to be used. In the present paper, we assume that the sender and receiver thresholds are larger than the bandwidth-delay product of our relatively low-rate flows, and hence do not consider the issue in our simulation.

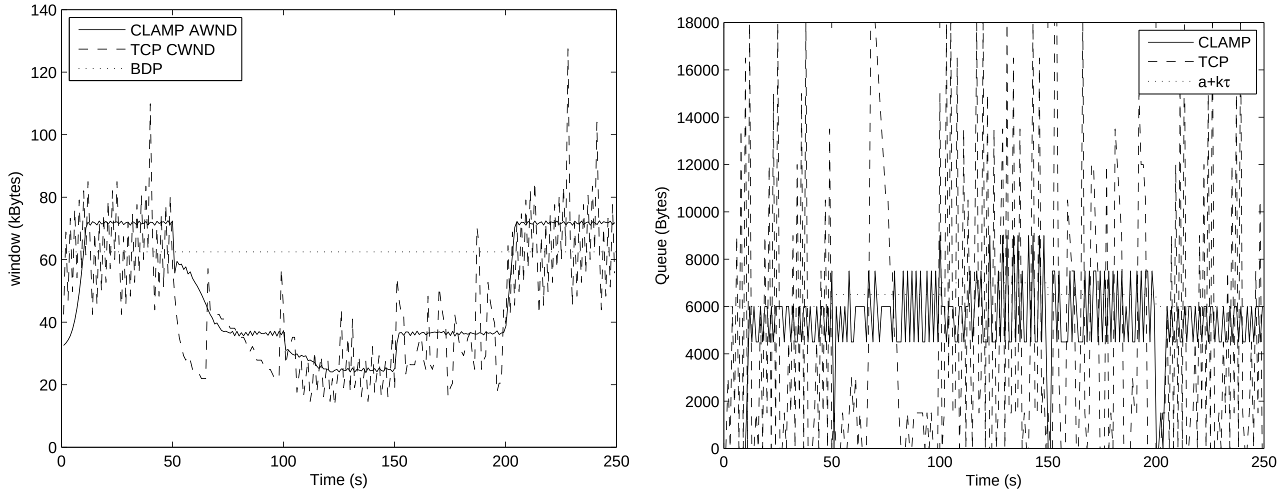
The CLAMP slowstart is only invoked at the start of a TCP connection to get an initial estimate of the bandwidth delay product. If a timeout occurs, TCP will go back to slowstart, but CLAMP will just continue with CLAMP "congestion avoidance", *i.e.* (14), with the view that it has a reasonable estimate of the bandwidth delay product. In other words, handling timeouts is left to TCP.

## V. Performance evaluation of CLAMP through simulation

In the following sections, we investigate the performance of TCP NewReno over the Internet, for a scenario in which all flows terminate in devices in the same network, all receiving information from a common access point (AP). In these experiments the last hop network is assumed to be the bandwidth bottleneck; this assumption is reasonable for many cellular networks, where the base station is connected to the high-speed network core. In some scenarios, a flow may be bottlenecked elsewhere in the Internet; congestion control is then provided by TCP NewReno [50].

The CLAMP protocol, and TCP NewReno, are both too complex to treat analytically, so we have developed an event driven simulator to model the physical, link and transport layers. Our implementation of TCP follows TCP NewReno [51], [52], including the so-called "bugfix". We study various network scenarios, and compare the performance of TCP NewReno with and without the CLAMP protocol at the receiver end of the connections.

With regard to CLAMP, it is no longer clear that it will be able to provide the fairness promised in the fluid analysis of Section III-B. Now, the protocol has to cope with random and time varying service rates, wireless packet loss, and the fact that it is limited to only receiver-side control; TCP NewReno is also operating at the sender-side and can potentially take control. A major focus is to see how much of the fluid analysis carries over to this more realistic scenario: can CLAMP reduce the interactions between layer $4$ flow control, and layer $2$ scheduling?




(a) Window dynamics

(b) Queue dynamics

Fig. 2. Deterministic link cross traffic from 50s to 200s and from 100s to 150s.

### A. *Window dynamics for a deterministic link*

Section III-B provides a mathematical fluid model to describe the equilibrium behaviour of CLAMP, but in this section we wish to illustrate the behaviour in a more dynamic environment in which flows come and go, and the flow control algorithm must be able to adapt to the fluctuations in system load.

In the first experiments, we consider a deterministic link shared by a number of receivers. A deterministic link could model a wireline network, with a number of users connected to a central hub. Alternatively, it could be a wireless network with sufficient diversity (frequency, time or space) that it can be treated as essentially error free, and constant rate. By presenting results for a constant bit rate link, we are able to see more clearly the dynamics of the CLAMP window control algorithm, and contrast it with that of TCP NewReno (without CLAMP at the receiver).

In Figure 2(a), we plot the dynamics of flow 1, starting at time $0$ and terminating at time 250 s. This flow initially gets all the bandwidth, which is $10$ Mb/s, but at time 50 s it is joined by flow 2, and at time 100 s by flow 3. Flow 2 leaves at time 150 s, and flow 3 leaves at time 200 s. All flows have a a propagation delay of $50$ msec, and share the same queue.

There are two experiments for this scenario, both with TCP NewReno at the senders, one with CLAMP switched on, and the other with CLAMP switched off. Since there is no wireless loss, and CLAMP prevents congestion loss, the *CWND* will exceed CLAMP's *AWND*, and need not be plotted. The plotted TCP (*CWND*) is the *CWND* from the experiment in which CLAMP is not used. The buffer at the AP has size 18,000 bytes, $a$ is 5500 bytes and $\tau$ is 500 bytes. With these values, the average queueing delays in the two experiments are approximately equal.

The window evolution from both experiments are depicted in Figure 2(a). It is clear from the figure that CLAMP tracks the dynamics of flow arrivals and departures in a similar way to TCP NewReno, but with much less fast time-scale fluctuations.

To appreciate the actual rates achieved, it is necessary to consider the queue evolution, which is depicted in Figure 2(b). Note that the queue for CLAMP tracks the queue size predicted by the fluid model quite closely, in terms of the given parameters $a$, $k$ and $\tau$. Note also that the queue suffers much less fluctuations than under TCP NewReno control. In both cases, however, the queueing delay is very small compared to the round trip time of $50$ msec. For example, the average queueing delay when the queue is $6000$ bytes, and the rate is $10$ Mb/s, is $4.8$ msec, which is much less than $50$ msec. Thus, the window evolution depicted in Figure 2(a) can be interpreted as a rate versus time plot as well. The figure shows how CLAMP reacts to the changes of load, decreasing the rate in proportion to the number of flows, with the queue size increasing (slightly) with the number of flows, as predicted in Section III-B. TCP's queueing fluctuations are much greater, but it also achieves a fair rate allocation between the flows, an example of the fairness of TCP when propagation delays are the same.

Figure 2(a) contrasts the window dynamics of CLAMP with that of TCP. The TCP dynamics are the well known additive increase and multiplicative decrease in the window size. The additive increase occurs once per RTT, and the multiplicative decrease occurs on the first packet loss within a RTT. With CLAMP, there is an additive increase term ($\tau$) but the multiplicative decrease term is multiplicative in the rate, not the window size, and both terms operate simultaneously and on every packet, thus getting averaged over a RTT. The net effect is a window size that varies more smoothly, as illustrated in this example, and the aim is to avoid buffers from emptying unnecessarily.

### *B. Wireless channel model and link layer*

We now consider performance when the last hop network is time-varying. The main scenario we have in mind is a cellular network, in which the receivers are highly mobile, and the base station plays the role of the AP. Our event driven simulator requires a different layer 1 and 2 model for the time-varying scenarios. The model used in these simulations aims to be as simple as possible, while describing the variable rate transmission characteristic of optimal wireless links, and providing a reasonable model for packet loss.

The multipath fading process between the base station and each receiver is a stationary, stochastic process according to Jakes' model, with $G = 20$ generators and Doppler frequencies of either 0.6 Hz or 60 Hz (1.8 GHz carrier, and mobile speeds of either 0.1 m/s or 10 m/s, respectively). That gives a channel of mean square magnitude

$$N(s) = SNR_{ave} \left| \sqrt{\frac{2}{G}} \sum_{n=1}^{G} \cos(\omega_n t) \exp\left[\frac{j\pi n}{G}\right] \right|^2,$$

where we take $SNR_{ave} = 4 (= 6 \text{ dB})$ as the mean signal to noise ratio, $\omega_n = 2\pi f_d \cos\left(\frac{\pi}{2G}(n + 0.5)\right)$ is the angular frequency of the $n$'th path, and $f_d$ is the Doppler frequency.

Frame transmissions over the physical channel occur in slots of $\tau_{\text{slot}} = 3\,\text{ms}$; a slot is the time period of one transmission attempt. A single 3ms slot can contain data from multiple packets, and packets can be split between slots. The coding rate for the slot is calculated at the start of each slot as follows. Let $N(s)$ be the SNR at the start of slot $s$, then $B(s)$ information bits are transmitted, where:

$$B(s) = \tau_{\text{slot}} \min\left(W_0 \log(1 + (1 - m)N(s)), C_{\max}\right), \tag{18}$$

where $C_{\max} = 10$ Mbps is the maximum possible rate that can be transmitted over the channel, $N(s)$ is the SNR at the start of the frame, $W_0 = 4.3$ MHz models the bandwidth, and $m$ is a "backoff margin" which is adaptively tuned such that the packet error rate after retransmissions is approximately a target, nominal, packet error rate, specified in the simulation.

In order to capture the bursty nature of errors we use a simple error model: if the SNR, $N(e)$, at the end of the frame is such that

$$B(s) > \tau_{\text{slot}} W_0 \log(1 + N(e)),$$

then an error is declared. This will occur if the channel deteriorates sufficiently during the slot that it can no longer support the selected rate. If the SNR is sufficiently high at both the start and end of a slot, then it is assumed to be sufficiently high throughout the slot. The backoff margin, $m$, is included to allow some degradation to occur without causing an error. We use a stochastic approximation method to find the correct backoff margin to get the targeted packet loss rate.

If $n$ consecutive attempts to a given destination fail, then all packets which would have been completed by that transmission are dropped. We use different values of $n$ for fast and slow fading as discussed in the experiments. Packets which are started in the initial 3ms slot, but would not be completed, are not dropped, but buffered to re-attempt in the next slot.

We will see later that this approach to link layer coding results in higher throughput over slowly varying channels, as compared to fast fading channels. This is due to the fact we are exploiting channel state information at the transmitter, and that information is less reliable in a fast fading environment, requiring a larger backoff margin.

### C. *The scheduler*

The AP maintains separate queues for each receiver (mobile device), and acts as an IP router, routing packets destined for the mobiles into the appropriate queue based on the address of the mobile. Thus, all TCP flows destined for the same mobile are put into the same output queue.

The scheduler is based on the principle of multi-user diversity. At each slot, a new scheduling decision is made, and the general preference is for scheduling a mobile is that it is in a good channel state. However, fairness must also be considered: some users may be in better locations than others, and thus get much higher rates, if only channel quality is considered.

In this study, we implement a scheduler along the lines of the proportionally fair scheduler [53] used in Qualcomm's HDR. At each slot, the scheduler picks the queue $i$ with the largest utility that has data to send in this slot. The utility, $U_i$, is calculated as

$$U_i := \frac{\mu_i}{r_i} \tag{19}$$

where $\mu_i$ is the current rate for the mobile $i$ at the beginning of the slot, and $r_i$ is the exponential moving average of the rate obtained by mobile $i$, with averaging time constant $100$ msec. Precisely, $r_i$ is obtained from

$$r_i(n) = 0.97 r_i(n-1) + 0.03\mu_i(n)$$

where $n$ indexes time, in slots. Thus, $0.03$ is the exponential smoothing factor, giving a time-constant of $100$ msec. We use an exponential moving average in order to be able to track changing channel statistics, and to allow the scheduler to react to the arrival and departure of flows. In the fast fading scenario (below) the coherence time of an individual user's channel is $16.67$ msec, and we want the

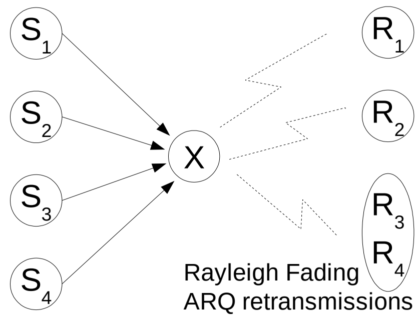



Fig. 3. Simulation network topology

exponential smoothing time constant to be significantly larger than this, to allow the scheduler to allocate slots to good channels, rather than in a round robin fashion. In the slow fading scenario, the scheduler will act more like a weighted round robin, with better channels getting higher weights. We chose a time-constant of $100$ msec, to give the scheduler the chance to schedule based on channel quality in the fast fading scenario, yet be able to adapt to changing dynamics of flow arrivals and departures.

### *D. Wireless network topology*

Fig. 3 depicts the topology of the time-varying wireless simulation experiments of this section. Due to space limitations, we have focused on a specific scenario, with a small number of long-lived TCP flows, and a range of different propagation delays.

Sources $S_1, S_2, \ldots, S_k$ represent servers which are TCP rate controlled, and the links from $S_i$ to $X$ model entire paths through the internet. Node $X$ is the wireless access router which contains the CLAMP router software agent. The wireless access router transmits to the TCP/CLAMP clients, $R_1$–$R_4$, over a common broadcast radio channel. Clients $R_3$ and $R_4$ are both running in the same physical device and so have a common queue at router $X$, and common radio propagation conditions, as described below. Clients $R_1$ and $R_2$ have separate queues, so in this experiment, there are three channels to be scheduled. All links are bidirectional, with characteristics as shown in Table I, and the wireless bandwidth is $W_0 = 4.3\,$MHz, with a maximum bit-rate of $10$ Mbits/sec. The packet loss rate is $10^{-3}$ which is achieved via the fade margin and link layer retransmissions. The number of retransmission attempts at the link layer is $n = 5$ in the slow fading scenario, and $n = 15$ in the fast fading scenario. Other simulation parameters are given in Table II.

In the following experiments, we will plot throughput (and other measures) against average

TABLE I

LINK CONFIGURATION

| Node 1 | Node 2 | Capacity | Delay | Pk loss rate |
|---|---|---|---|---|
| $S_i$ | $X$ | 100 Mb/s | $d_i/2$ | 0 |
| $X$ | $R_i$ | see text | 1 msec | $10^{-3}$ |

TABLE II

SIMULATION PARAMETERS

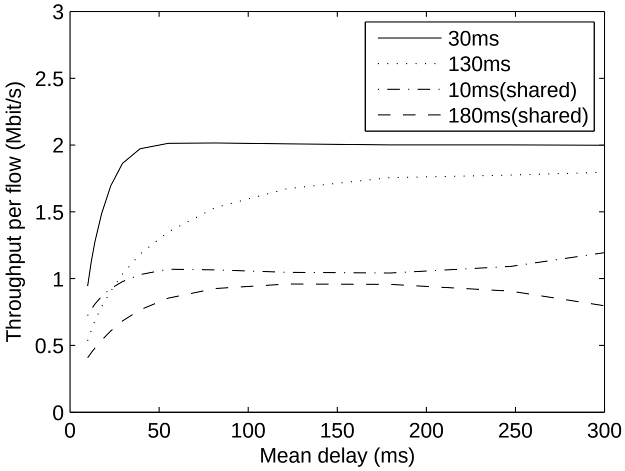
| Parameter | Value |
|---|---|
| TCP Packet Size | 1500 Bytes |
| CLAMP $\tau$ | 500 Bytes |
| CLAMP $b$ | 1 |
| CLAMP $a$ | Variable |
| $d_1$ | 30 ms |
| $d_2$ | 130 ms |
| $d_3$ | 10 ms |
| $d_4$ | 180 ms |

queueing delay, where the average is taken across all queues. For pure TCP Reno, the average queueing delay is controlled by the choice of buffer size at the AP; the queueing delay is increasing in the buffer size. For TCP Reno + CLAMP, the average queueing delay is controlled by the choice of the parameter $a$. It will be shown (see Figure 7) that queueing delay is an increasing function of the parameter $a$. By running simulations across a range of values of buffer size, for TCP, and $a$, for CLAMP, we are able to generate the following graphs.

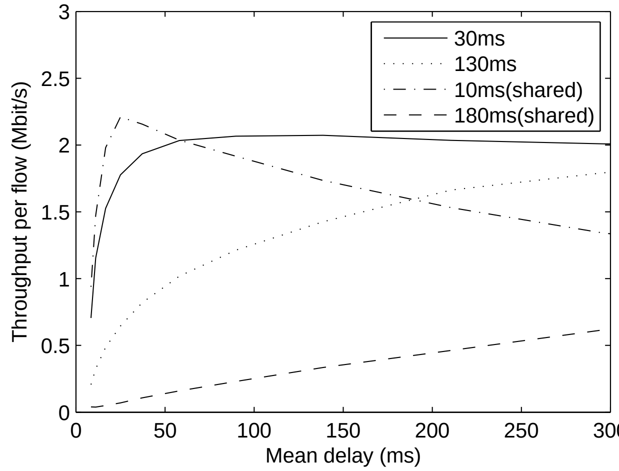
CLAMP does not use the AP buffer to control the flow rates, so the CLAMP buffer can be much larger than with TCP NewReno. In fact, in the results in this paper, we use a large buffer size of $6$ Mbytes for CLAMP, to avoid any chance of buffer overflow for the CLAMP protocol. Instead, the average queueing delay for CLAMP is limited by the choice of CLAMP parameters. In Section V-I, we will discuss the problem of choosing the appropriate parameters, both for CLAMP and for TCP.

### *E. Fair resource allocation*

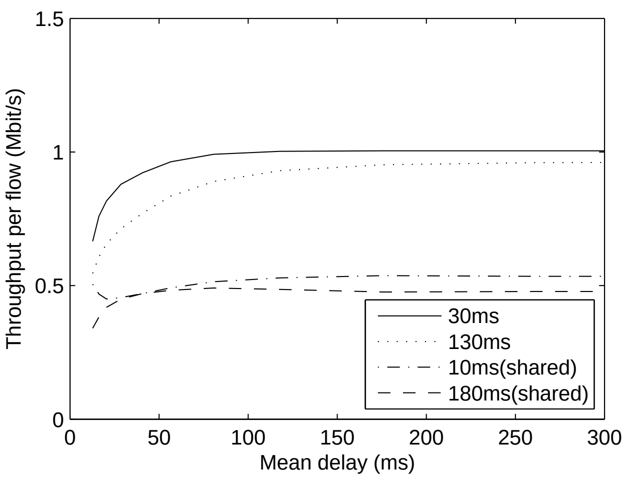
We are particularly interested in the fairness of resource allocation to the long-lived TCP flows, since the congestion-avoidance part of TCP tries to reach long-term equilibrium throughputs for these flows. Figure 4 plots the throughput of each flow against the mean queueing delay (averaged over all flows), both for CLAMP (figures 4(a) and 4(c)) and for pure TCP NewReno (figures 4(b) and 4(d)).



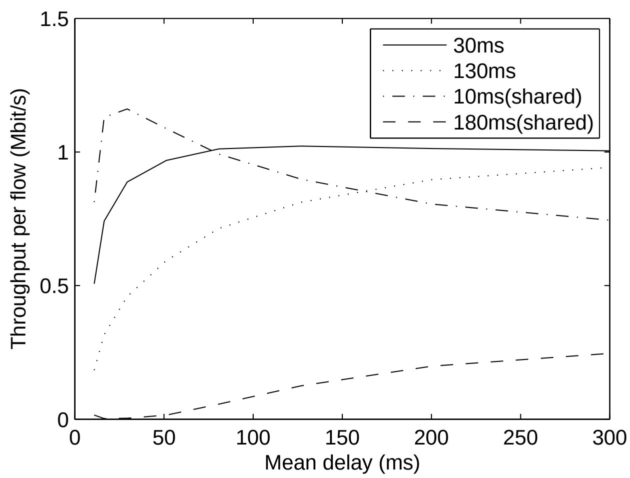

(a) CLAMP, 0.1 m/s




(b) TCP NewReno, 0.1 m/s




(c) CLAMP, $p_e = 10^{-4}$, 10 m/s




(d) TCP NewReno, 10 m/s

Fig. 4. Throughput per flow for CLAMP and TCP NewReno. Dot-dashed lines are for flows 3, dashed lines are for flow 4, both sharing a common queue at the AP.

In the following, we will denote flows by their propagation delays. For moderate average queueing delay ($50$ ms or less), TCP NewReno is unfair to flows with large propagation delays. RTT unfairness exists even for flows that do not share the same queue, such as the $30$ ms, and $130$ ms flows, where the $30$ ms flow has an unfair advantage. Ideally, the scheduler allocates equal rates to all queues. However, in both scenarios (slow and fast fading) the $30$ ms flow receives more bandwidth than the $130$ ms flow, and significantly more bandwidth at "desirable" queueing delays, such as $50$ msec (or less).

It is well known that TCP (both Reno and NewReno) are unfair to flows with long propagation delays [5]. This assumes that all flows share the same queue, are served in FCFS order, and get the same channel rate when serviced. Certainly, the $10$ ms and $180$ ms flows share the same queue and are served in FCFS order, so the fact that the $180$ ms flow is treated unfairly is well known, but the magnitude of the unfairness, as depicted in Figures 4(b) and 4(d), is striking. Our results here extend to flows that are in separate queues, in spite of a scheduler that is trying to allocate bandwidth fairly to the users (queues). The problem is that the scheduler assumes the queues are always backlogged, and TCP NewReno is unable to satisfy this assumption. Thus, the propagation delays have an impact on rate allocation in the TCP NewReno experiments.

The fairness situation is quite different for CLAMP. Section III-B predicted that CLAMP will keep the queues full, and share rates fairly between flows that share the same queue. In the high-speed mobility scenario depicted in Figure 4(c) this is shown to be approximately true. All three queues achieve approximately equal rates, and the $10$ ms and $180$ ms flows, sharing the same queue, also get an approximately equal rates.

There is a discrepancy in that there is *less* fair sharing between the $10$ ms and $180$ ms flows as the queueing delay increases, in Figure 4(c), which is not what the fluid model would predict. We investigated this, and found that the $10$ ms flow is experiencing more timeouts than the other flows, and significantly more between $25$ and $100$ ms of queueing delay. The $10$ ms flow experiences more variability in its RTT, hence the larger frequency of timeouts for this flow. As the queueing delay increases beyond $100$ ms, the frequency of timeouts for the $10$ ms flow decreases, and its rate improves. The “fairness” depicted at about $30$ ms is coincidental; the timeouts have brought the rate for the $10$ ms flow down to that of the $180$ ms flow, but were it not for the timeouts, the $10$ ms rate would be higher, and the $180$ ms rate would be lower.

CLAMP is less fair in the low-speed mobility scenario, Figure 4(a), but still much fairer than TCP NewReno. At an average queueing delay of $50$ ms, the $30$ ms flow gets a higher rate than the $130$ ms flow, and the shared flows get a combined rate between these two rates. Within the shared queue, the $10$ ms flow has a small advantage over the $180$ ms flow. At queueing delays larger than $250$ ms, the $10$ ms flow gets more rate than at lower delays. This is because CLAMP starts to lose control at high queueing delay (see Section V-G for an explanation and illustration).

We might expect that the fast fading scenario is more like the “fluid model” in that there is more averaging. The scheduler averages rates over a period of $100$ ms, and in the fast fading scenario, the channel fluctuations are much faster than this: the channel coherence time is $16.67$ ms. So we might expect fairness to be achieved when the average queueing delay is significantly larger than the coherence time *e.g.* at, say, $100$ ms of queueing delay. In this case, the queues rarely empty, and the rates achieved per flow are insensitive to the propagation delays.

In the slow fading scenario, the coherence time is significantly longer than the averaging period of the scheduler. In this case, the channel fluctuations do not average out over the queueing delay, and the queue can drain of packets. The larger the propagation delay of a flow, the more this affects the rate allocated to the flow. This effect reduces as the queueing delay increases. Once the queueing delay exceeds the maximum propagation delay of any flow sharing the queue, the queue will not drain, due to self-clocking.

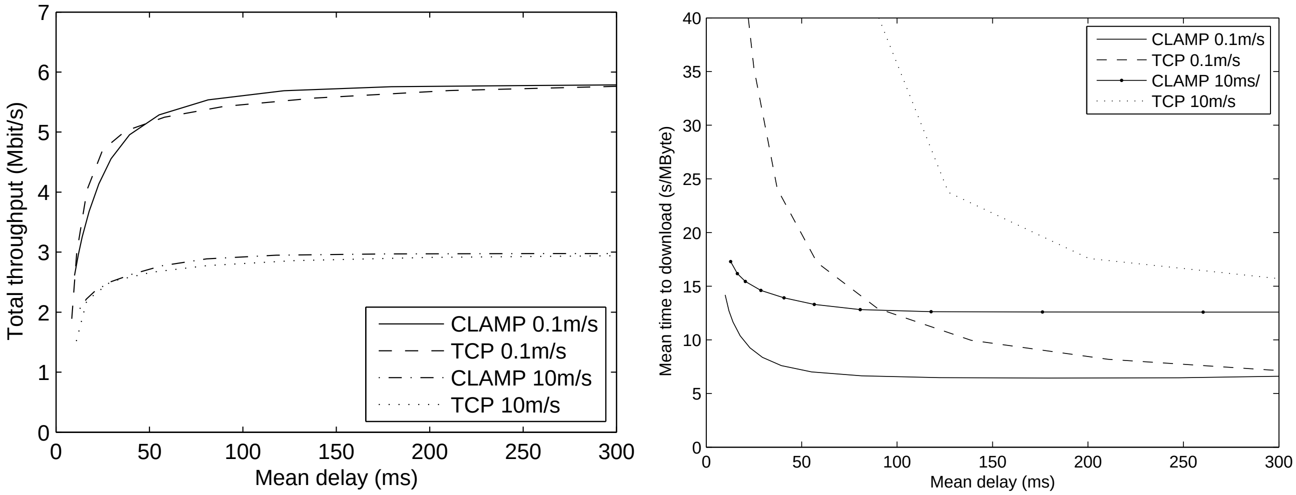
This analysis predicts that CLAMP will avoid interacting with the scheduler provided the channel fluctuations are much faster than the average queueing delay. The scheduler can then allocate capacity independently of propagation delays. This hypothesis is supported by the above simulation results.

Fast fluctuations happen automatically in high-speed mobility scenarios due to multipath fading, as considered here. However, it is also possible to induce fast fluctuations (at a controlled rate) even when the mobility is slow (or non-existent) using the technique of opportunistic beamforming [54]. CLAMP may be fruitfully combined with such techniques to provide fair allocation in a wireless network.

### *F. Total throughput and mean time to download*

Figure 4 reports on the individual rates per user to illustrate the fairness achieved by the two protocols. In Figure 5(a), the sum rate across flows is depicted. Together, these figures indicate that CLAMP's benefit is to provide fairness, rather than to increase total throughput; there is not much between the two protocols in terms of total throughput in the scenario investigated here.

To understand the throughput-delay characteristics of both TCP and CLAMP, recall that the scheduler operates most efficiently if the queues are always backlogged. However, if the queue feeding the optimal channel is empty, then the scheduler will have to serve a different queue, perhaps a queue with a much lower rate channel, reducing the overall average rate. Hence both CLAMP and TCP gain throughput when the delay increases, as the chance of the queues emptying decreases as the average queueing delay increases.



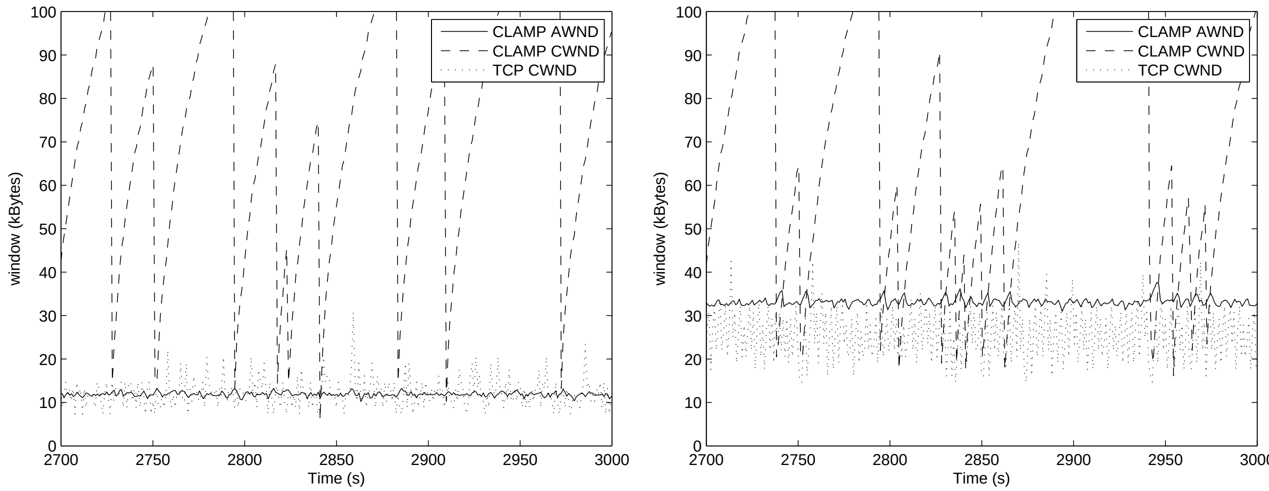

(a) Total throughput versus delay

(b) Time to download 1 Mbyte versus delay

Fig. 5. Performance comparison: CLAMP versus TCP

Moreover, the chance of the queue emptying depends on the propagation delay of the flow (or flows). The buffer of a flow with a small propagation delay, of the order of the time allocated by the scheduler (or less), will be much less likely to drain than the buffer of a flow with a large propagation delay. Thus, increasing the average queueing delay will benefit the throughput of flows with large propagation delays more than flows with small propagation delays. These facts are illustrated in Figures 4(a) - 4(d).

Throughput, in bits/sec, is only one measure of performance. Since CLAMP allows the scheduler to allocate rates more fairly between flows, we might expect that the average time to download a file of, say, 1 Mbyte, is less under CLAMP than under pure TCP, where the average time to download is dominated by the low rate flows. In Figure 5(b), the average time to download a file of size 1 Mbyte is plotted against mean queueing delay for both TCP NewReno, and CLAMP. As can be seen, at small to moderate mean queueing delays, CLAMP provides a very significant gain with respect to this measure of performance. For example, at 50 msec of average queueing delay, and at the mobile speed of 10.1 m/s, TCP files take on average more than twice as long to download, as they do under CLAMP. The average time to download is a better metric than throughput in capturing how users perceive performance, since users experience latency, not bitrate.




(a) 50 ms queueing delay

(b) 180 ms queueing delay

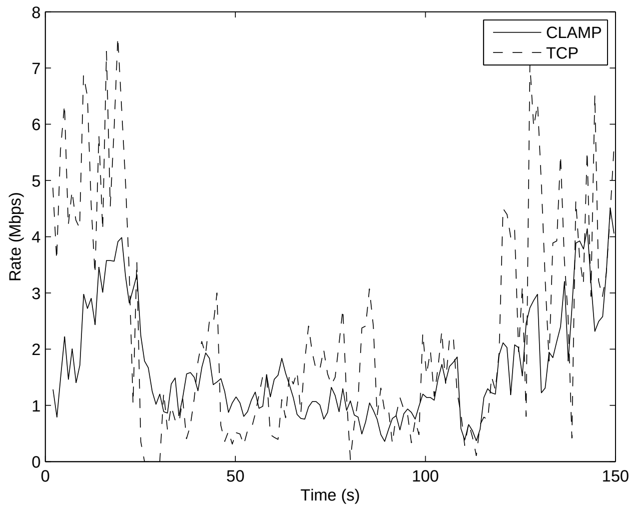
Fig. 6. Windows versus time for 30 ms flow

### *G. Window dynamics*

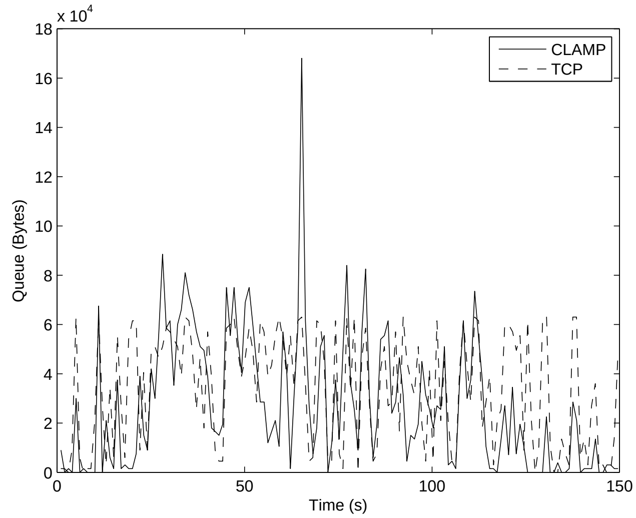
In this section, we plot the evolution of the window sizes over time for one of the flows in the above experiment. Figure V-G plots the window dynamics for the $30$ ms flow; space considerations prohibit plotting the other flows, but qualitatively they are similar. We illustrate with $300$ seconds of window evolution, starting after sufficient time has elapsed for the fade margin to have converged to the correct value.

We have chosen two different parameter settings for TCP and CLAMP, corresponding to average queueing delays of $50$ and $180$ ms, respectively. We observe that the CLAMP *AWND* is slightly greater than the mean of the TCP *CWND* in the TCP experiment. Thus, CLAMP achieves short-term fairness, as well as long-term fairness; the system is quite stable under CLAMP. Also plotted is the TCP congestion window for the CLAMP experiment.

For a queueing delay of $50$ms, the *AWND* is almost always smaller than the *CWND* in the CLAMP experiments, only losing control to TCP occasionally, and for very brief periods. The frequency and duration that CLAMP loses control, at $180$ ms of queueing delay, is higher, due to the higher bandwidth delay product: it becomes more likely that the *CWND* can drop below the *AWND* value at high queueing delay. However, for the sake of mice TCP flows (and other delay sensitive traffic) it is desirable to operate at low-moderate queueing delay ($50$ ms or less).




(a) Rate allocated to long-lived flow




(b) Queue dynamics

### *H. Flow Dynamics*

In Section V-A, we considered the window and queue dynamics as flows arrive and depart, but there was no fading. In Section V-E, we have considered the equilibrium behavior of the protocols under dynamic fading, but without the dynamics of flows arriving and departing. In this section, we consider both flow dynamics and fading: we study a scenario in which a long-lived TCP flow coexists with a number of short-lived TCP flows, which start and finish at random times.

In this experiment, the channel is the same as described previously, but there is only a single mobile user with multiple flows sharing the same queue at the AP. One flow is long-lived, and corresponds to a file transfer running in the background. In addition, the user also generates a series of short TCP flows, one after the other, with random periods of idleness between short flows. Each short flow is of size $4000$ bytes. This models a web browsing session in which the user makes a series of small requests from, say, a web server.

In Figure V-H, we plot the dynamics of the rate allocated to the long-lived flow, and the total queue size at the AP (a function of all the flows in the system).

We conclude that CLAMP and TCP both track the changing dynamics in a similar manner. The fluctuations in rate are less under CLAMP, but the queue fluctuations are quite similar (apart from a single spike in the queue size for CLAMP). Note that the spike could be avoided if we restricted the buffer size for CLAMP. The spike offsets the otherwise smaller values for the CLAMP queue size; parameters were chosen to get the same average queue size for CLAMP and TCP. The buffer size for TCP is $60$ Kbytes.

We conclude that CLAMP's aim of reducing queueing fluctuations is not really happening in this example. Note that the $4000$ byte flows never have time to reach equilibrium (they are always in slow-start) and so the cross traffic is quite bursty, and never CLAMP controlled. CLAMP cannot control these fluctuations and this is reflected in the plot of the queue dynamics.

### *I. Choosing the parameters*

To provide a basis for comparing TCP with TCP+CLAMP, we have plotted performance (above) as a function of average queueing delay, but average queueing delay is a function of the underlying parameters. A question arises as to how to set these parameters to obtain a particular desired point on, say, the throughput-delay tradeoff curve. For TCP NewReno, we vary the buffer size at the AP. For CLAMP, we hold the buffer size fixed and vary the parameter $a$. Unfortunately, for the fading scenarios considered above, we have no analytical formula to describe the throughput-delay curve. The problem of buffer dimensioning for time-varying channels is an open area of research, and beyond the scope of the present paper.

In Figure 7 we plot both the TCP buffer size, and the CLAMP parameter, $a$, against average queueing delay. This figure suggests that the problems of setting the buffer size (for TCP) and the parameter, $a$ (for CLAMP) are quite similar problems. There is no reason to suppose that one is more difficult than the other. The only significant difference is that the buffer size must always be a fixed constant. With CLAMP, there is the option that the parameter $a$ can be an adaptive parameter, adapting in real-time to changes in channel statistics, or traffic load (for example). Alternatively, it can be left as a fixed parameter, analogous to the buffer size. Adaptively tuning the parameter $a$ is a topic for future work; in the present paper, the parameter is held fixed during a particular experiment, although we plot the results of different experiments across a range of parameter values.

### *J. Lower data rate scenario*

With regard to parameter settings, it is of interest to see if the CLAMP parameters need to be re-tuned for every network scenario. We have already examined two fading rates (fast and slow) and here we consider a much lower channel rate. An important issue is the setting of the parameter $\tau$,

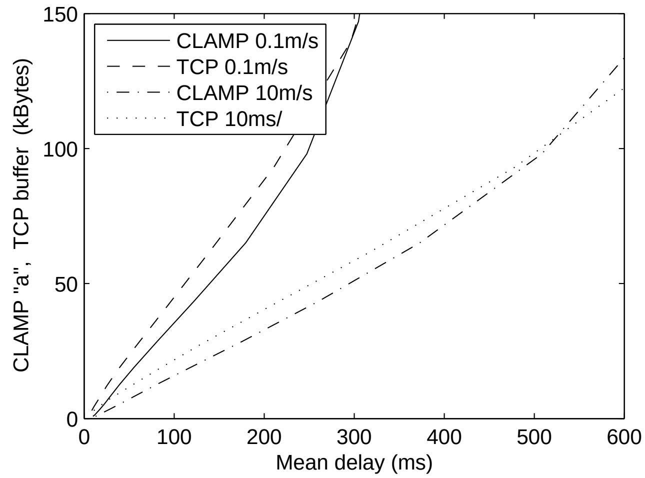
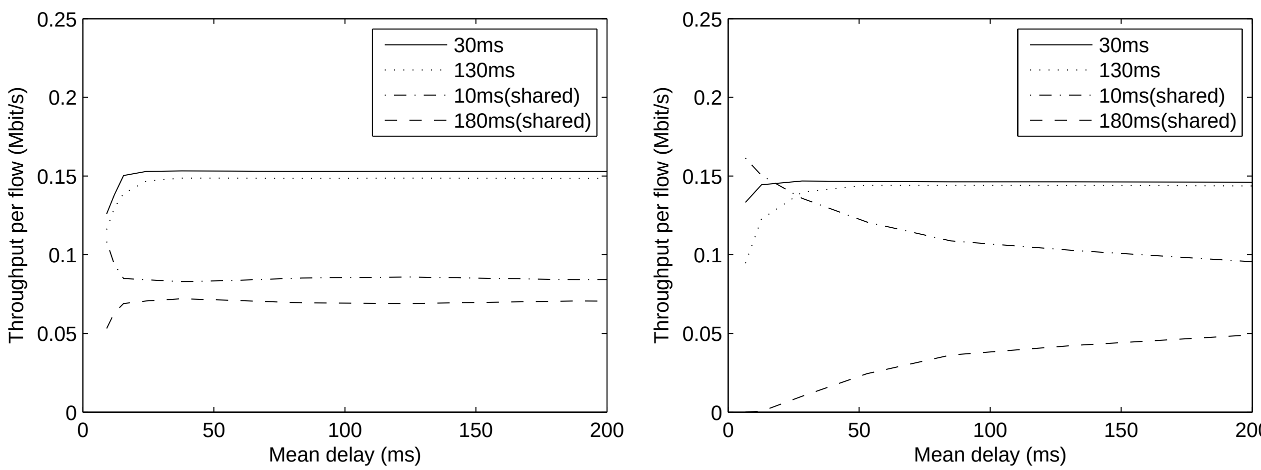



Fig. 7. TCP buffer size and CLAMP $a$ versus queueing delay




(a) CLAMP, 10 m/s

(b) TCP, 10 m/s

Fig. 8. Throughput per flow in lower data-rate scenario. Dot-dashed lines are for flows 3, dashed lines are for flow 4, both sharing a common queue at the AP.

which is set in the mobile receivers, and not likely to be tunable. In the high data rate experiments, we used $\tau = 500$ bytes, and we use the same value for the lower data rate scenario.

Identical experiments were undertaken for the same topology as described in Tables I, and II, except that the bandwidth was reduced to $W_0 = 620\,$kHz, and the maximum wireless link rate to $C_0 = 1.44$ Mbits/sec. Due to lack of space, we do not provide all the plots, but Figure 8 presents the results for fairness at high fading rate.

None of the general conclusions change: CLAMP does not increase the total throughput of the system, but it does allow the scheduler to provide fair service across users, and it provides fairness automatically to flows destined for the same receiver.

Although $\tau$ is fixed, there is flexibility in the CLAMP protocol for the $a$ parameter to be tuned to the conditions of the network. We observe that in each scenario (combination of data rate, and fading rate) total throughput versus delay has a fairly well defined “knee”, but the knee occurs at a different average delay in each case. It might be desirable to operate the system near the ‘knee” and this would require the AP to make a judicious choice of the parameter $a$.

We observe that the knees occur at lower delays in the lower data-rate scenario. It is not surprising that the delay depends on the statistics of the service process, and we can contrast this behavior with what happens in a non-fading, wireline scenario, when the output rates are deterministic. In that case, queueing delay should *increase* as the output rate of the queue decreases, because the average queue size to meet a particular utilization does not depend on the service rate (this holds for the M/M/1 queue, for example). The present wireless scenario is quite different: the amount of queueing required for a particular utilization depends on the rate of fading, and more particularly, on the whole distribution of the service process. In this case, the size of the average queue required to keep all queues from emptying increases with the data rate.

## VI. Conclusions

This paper has studied the interaction between congestion control and scheduling in fading, wireless channels. The scheduler attempts to allocate bandwidth to the users in such a way that a user will only be allocated rate when its channel is good, but subject to fairness constraints. We do observe considerable interaction between TCP NewReno and the scheduler, to the extent that rate allocations are determined, in part, by the propagation delays of the flows in the network.

Our approach was to provide a detailed packet level simulation of a wireless network with a single access point, sending data to a number of mobile receivers. We model the physical layer fading process, link layer rate adaptation, scheduling, and flow control. Each mobile is provided a separate queue at the AP, and access to the radio channel is provided by the scheduler. TCP NewReno does the flow control, subject to possible intervention by the proposed CLAMP receiver-side algorithm, which allows us to measure the impact of Reno’s window fluctuations on the performance of the scheduler.

CLAMP is based on a fluid model algorithm that is designed to avoid interaction between layers 2 and 4. The experiments are to test if this remains true in a more complicated model in which there is fading, and in which CLAMP can only intervene at the receiver-side. Experiments are mainly focused on a particular scenario with 3 mobile recievers, one of which is receiving two flows simultaneously. Parameters are selected to provide insight into the effect of propagation delays, and rates of fading, on performance.

For these experiments, we conclude that the fairness of the scheduler is compromised by TCP NewReno. Flows with large propagation delays get much worse performance, even if the scheduler is attempting to schedule them fairly. When we employ the receiver-side algorithm, CLAMP, to remove the window fluctuations, we find that the scheduler is able to achieve much better fairness than with pure TCP NewReno. As with TCP NewReno, the CLAMP throughput increases with AP queueing delay; when the average queue size is larger, the queues empty less often. Nevertheless, for moderate queueing delay, the scheduler is able to provide much more fair allocation under CLAMP than under TCP NewReno.

In the scenarios tested, better fairness is achievable when the fading is faster, but throughput is reduced at faster fading due to the increased fade margin required. Techniques such as incremental redundancy coding should alleviate this problem, but we did not investigate this. Inducing channel fluctuations in a controlled way [54] is a promising technique, and may allow CLAMP + scheduler to operate fairly even in slow fading scenarios.

Our approach has been to focus on a single hop wireless downlink, and in particular on the issues of time-varying channels, and channel-state-aware scheduling. Although we have made specific assumptions about the channels for the simulation, the CLAMP algorithm can in principle be applied to any wireless network with a single hop, and for that matter to the last hop of a wireline network.

A cleaner solution from a systems-level perspective would be a sender-side implementation, in which the feedback signals are sent back to the sender in the ACK's, who then can set an appropriate *CWND* value. In that case, the flow control is primarily achieved via *CWND*. There is no inherent reason why the CLAMP control cannot be effected at the sender side. However, this requires extending CLAMP to the scenario in which all routers in the network provide congestion-indication feedback to the source. There has been much work on price-based congestion control for the core Internet [55], [56], [57], and CLAMP provides one such approach, but others are also possible.

A long-term goal is to generalize CLAMP to handle multiple bottlenecks, and to incorporate the principles learned from studies such as the present one into a robust sender-side control that can handle the vagaries of time-varying, and lossy, wireless channels, as well as the scaling problems associated with high-speed networks.

# References


[1] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication.* Cambridge University Press, 2005.

[2] S. H. Low, F. Paganini, and J. C. Doyle, "Internet congestion control," *IEEE Control Systems Magazine*, vol. 22, pp. 28–43, Feb. 2002.

[3] N. T. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. N. Bershad, "Receiver based management of low bandwidth access links," in *Proc. IEEE INFOCOM*, pp. 245–254, 2000.

[4] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "Explicit window adaption: A method to enahance TCP performance," *IEEE/ACM Trans. Networking*, vol. 10, pp. 338–350, June 2002.

[5] T. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. Networking*, vol. 5, pp. 336–350, June 1997.

[6] T. V. Lakshman, U. Madhow, and B. Suter, "TCP/IP performance with random loss and bidirectional congestion," *IEEE/ACM Trans. Networking*, vol. 8, pp. 541–555, Oct. 2000.

[7] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, pp. 756–769, 1997.

[8] H. M. Chaskar, T. Lakshman, and U. Madhow, "TCP over wireless with link level error control: Analysis and design methodology," *IEEE/ACM Trans. Networking*, vol. 7, pp. 605–615, Oct. 1999.

[9] D. A. Eckhardt and P. Steenkiste, "Improving wireless LAN performance via adaptive local error control," in *Proc. IEEE Int. Conf. Netw. Protocls (ICNP)*, pp. 327–338, Mar. 2003.

[10] R. Ludwig, A. Konrad, A. Joseph, and R. Katz, "Optimizing the end-to-end performance of reliable flows over wireless links," *Kluwer/ACM Wireless Netw. J.*, vol. 8, pp. 289–299, Mar.-May 2002.

[11] M. Meyer, "TCP performance over GPRS," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, pp. 1248–1252, Mar. 2003.

[12] H.-K. Shiu, Y.-H. Chang, T.-C. Hou, and C.-S. Wu, "Performance analysis of TCP over wireless link with dedicated buffers and link level error control," in *IEEE International Conference on Communications*, pp. 3211– 3216, IEEE, 2001.

[13] K. Ratnam and I. Matta, "WTCP: an Efficient Mechanism for Improving TCP Performance over Wireless Links," in *Proc. IEEE Symposium on Computers and Communications*, pp. 74–78, June 1998.

[14] M. Sagfors, R. Ludwig, M. Meyer, and J. Peisa, "Queue management for TCP traffic over 3G links," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, pp. 1663–1668, Mar. 2003.

[15] J. Ma, J. Ruutu, and J. Wu, "An enhanced TCP mechanism - fast-TCP in IP networks with wireless links," *Wireless Networks (Kluwer)*, vol. 6, pp. 375–379, Nov. 2000.

[16] M. Chan and R. Ramjee, "TCP/IP performance over 3G wireless links with rate and delay variation," in *Proc. ACM Int. Conference on Mobile Coputing and Networking*, pp. 71–82, Sept. 2002.

[17] M. Chan and R. Ramjee, "Improving TCP/IP performance over third generation wireless networks," in *Proceedings of IEEE INFOCOM*, vol. 3, pp. 1893–1904, 2004.

[18] S. ElRakabawy, A. Klemm, and C. Lindemann, "TCP with adaptive pacing for multihop wireless networks," in *ACM International Symposium on Adhoc Networking and Computing*, pp. 288–299, May 2005.

[19] S. Pilosof, R. Ramjee, D. Raz, and P. Sinha, “Understanding TCP fairness over wireless LAN,” in *Proceeings of IEEE INFOCOM*, vol. 2, pp. 863–872, Mar. 2003.

[20] D. Leith, P. Clifford, D. Malone, and A. Ng, “TCP fairness in 802.11e WLANs,” *IEEE Commun. Lett.*, vol. 9, pp. 964–966, Nov. 2005.

[21] V. Rughunathan and P. Kumar, “A counterexample in congestion control of wireless networks,” in *Proc. International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pp. 290–297, Oct. 2005.

[22] V. Kawadia and P. Kumar, “Experimental investigations into TCP performance over wireless multihop networks,” in *Proc. ACM Sigcomm Workshop on Experimental Approaches to Wireless Network Design*, pp. 29–24, Aug. 2005.

[23] R. Oliveira and T. Braun, “A Dynamic Adaptive Acknowledgement Strategy for TCP over Multihop Wireless Networks,” in *Proc. of IEEE Infocom*, vol. 3, pp. 1863–1874, Mar. 2005.

[24] Y. Bai, A. Ogielski, and G. Wu, “Interactions of TCP and radio link ARQ protocol,” in *IEEE Vehic. Technol. Conf.*, vol. 3, pp. 1710–1714, 1999.

[25] Z. Kostic, X. Qiu, and L. F. Chang, “Interactions between TCP and RLP protocols in a cellular system,” *IEEE Vehic. Technol. Conf.*, vol. 3, pp. 2244–2248, May 2001.

[26] M. Malkowski and S. Heier, “Interaction between UMTS MAC scheduling and TCP flow control mechanisms,” in *Proc. International Conference on Communication Technology*, pp. 1373–1376, 2003.

[27] N. Samaraweera, “Non-congestion packet loss detection for TCP error recovery using wireless link,” *IEE Proceedings Communications*, vol. 146, no. 4, pp. 222–230, 1999.

[28] S. Biaz and N. Vaidya, ““de-randomizing” congestion losses to improve tcp performance over wired-wireless networks,” *IEEE/ACM Trans. Networking*, vol. 13, pp. 596 – 608, June 2005.

[29] P. Cheng and S. Liew, “TCP Veno: Enhancement for transmission over wireless access networks,” *IEEE J. Select. Areas Commun.*, vol. 21, pp. 216–228, Feb. 2003.

[30] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, “TCP Westwood: Bandwidth estimation for enhanced transport over wireless links,” in *Proc. ACM MOBICOM*, pp. 287–297, 2001.

[31] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, “TCP Westwood: End-to-end congestion control for wired/wireless networks,” *Wireless Networks (Kluwer)*, vol. 8, pp. 467 –479, Sept. 2002.

[32] K. Xu, Y. Tian, and N. Ansari, “TCP-jersey for wireless ip communications,” *IEEE J. Select. Areas Commun.*, vol. 22, pp. 747–756, May 2004.

[33] V. Tsaoussidis and H. Badr, “TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains,” in *Proc. International Conference on Network Protocols*, pp. 12–21, Nov. 2000.

[34] C. Parsa and J. Garcia-Luna-Aceves, “Improving TCP Congestion Control over Internets with Heterogeneous Transmission Media,” in *Proc. International Conference on Network Protocols*, pp. 213–221, Oct. 1999.

[35] V. Tsaoussidis, H. Badr, and R. Verma, “Wave & Wait Protocol (WWP): an Energy-saving Transport Protocol for Mobile IP-devices,” in *Proc. International Conference on Network Protocols*, pp. 301–308, Oct. 1999.

[36] X. Lin, N. Shroff, and R. Srikant, “A Tutorial on Cross-layer Optimization in Wireless Networks,” *IEEE J. Select. Areas Commun.*, vol. 24, pp. 1452–1463, Aug. 2006.

[37] A. Eryilmaz and R. Srikant, “Fair Resource Allocation in Wireless Networks Using Queue-length based Scheduling,” in *Proc. IEEE Infocom*, vol. 3, pp. 1794–1803, Mar. 2005.

[38] L. Ying, R. Srikant, E. Eryilmaz, and G. Dullerud, “Distributed Fair Resource Allocation in Cellular Networks,” *IEEE Trans. Automat. Contr.*, vol. 52, pp. 129–134, Jan. 2007.

[39] S. Bae, K. Xu, S. Lee, and M. Gerla, “Measured analysis of TCP behavior across multihop wireless and wired networks,” in *IEEE Globecom*, vol. 1, pp. 153–157, Nov. 2002.

[40] Z. Fu, H. Luo, P. Zerfos, L. Zhang, and M. Gerla, “The impact of multihop wireless channel on TCP performance,” *IEEE Trans. on Mobile Computing*, vol. 4, pp. 209–221, Mar. 2000.

[41] L. L. H. Andrew, S. V. Hanly, and R. G. Mukhtar, “Analysis of rate adjustment by managing inflows,” in *Proc. 4th Asian Control Conference*, (Singapore), pp. 47–52, 2002.

[42] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt, “Flow aggregation for enhanced TCP over wide-area wireless,” in *Proc. IEEE INFOCOM*, pp. 1754–1764, 2003.

[43] L. L. Andrew, S. V. Hanly, and R. Mukhtar, “CLAMP: a system to enhance the performance of wireless access networks,” in *Proc. IEEE Globecom*, vol. 7, pp. 4142–4147, Dec. 2003.

[44] T. Goff, J. Moronski, D. Phatak, and V. Gupta, “Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments,” in *IEEE INFOCOM*, pp. 1537–1545, 2000.

[45] J. E. Marshall, H. Górecki, K. Walton, and A. Korytowski, *Time-Delay Systems: Stability and Performance Criteria with Applications*. New York, NY: Ellis Horwood, 1992.

[46] D.-M. Chiu and R. Jain, “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks,” *Computer Networks and ISDN Systems*, vol. 17, pp. 1–14, 1989.

[47] F. Paganini, J. C. Doyle, and S. H. Low, “Scalable laws for stable network congestion control,” in *Proc. IEEE Conf. Decision Contr. (CDC)*, (Orlando, FL), pp. 185–90, 2001.

[48] L. L. H. Andrew, S. V. Hanly, and R. G. Mukhtar, “Clamp: Maximizing the performance of tcp over low bandwidth variable rate access links,” technical report, University of Melbourne. http://www.ee.unimelb.edu.au/staff/lha/abstract/clamp_tr-2004-02-01.pdf.

[49] Information Sciences Institute University of Southern California, “RFC 793: Transmission control protocol,” RFC 793, IETF, 1981.

[50] L. L. H. Andrew, S. V. Hanly, and R. G. Mukhtar, “CLAMP: Differentiated capacity allocation in access networks,” in *Proc. IEEE Int. Performance Computing and Communications Conf.*, (Phoenix), pp. 451–458, Apr. 2003.

[51] S. Floyd and T. Henderson, “The NewReno modification to TCP’s fast recovery algorithm,” RFC 2582, IETF, 1999.

[52] S. Floyd, T. Henderson, and A. Gurtov, “The NewReno modification to TCP’s fast recovery algorithm,” RFC 3782, IETF, 2004.

[53] E. Chaponniere, P. Black, J. Holtzman, and D. Tse, “Transmitter directed code division multiple access system using path diversity to equitably maximize throughput,” 2002.

[54] P. Viswanath, D. Tse, and R. Laroia, “Opportunistic beamforming using dumb antennas,” *IEEE Transactions on Information Theory*, vol. 48, pp. 1277–1294, June 2002.

[55] F. Kelly, A. Maulloo, and D. Tan, “Rate control in communication networks: Shadow prices, proportional fairness and stability,” *J. Op. Res. Soc.*, vol. 49, pp. 237–378, 1998.

[56] S. H. Low and D. E. Lapsley, “Optimization flow control I: Basic algorithm and convergence,” *IEEE/ACM Trans. Networking*, vol. 7, pp. 861–875, Dec. 1999.

[57] J. Mo and J. Walrand, “Fair end-to-end window-based congestion control,” *IEEE/ACM Trans. Networking*, vol. 8, pp. 556–567, Oct. 2000.